რომან სამხარაძე

Visual C#.NET

C# ენის საფუძვლები
Windows-პროგრამების შექმნა
პროგრამირების ახალი ასპექტები
მონაცემთა ბაზები. SQL Server

საქართველოს ტექნიკური უნივერსიტეტი

რომან სამხარაძე

Visual C#.NET

დამტკიცებულია
სახელმძღვანელოდ სტუ-ს
სარედაქციო-საგამომცემლო
საბჭოს მიერ

თბილისი
2008

სახელმძღვანელო წარმოადგენს მეორე, გადამუშავებულ გამოცემას. მასში გადმოცემულია Microsoft Visual Studio .NET გარემოში პროგრამების შემუშავების საკითხები. დაწვრილებითაა განხილული C# ენის საფუძვლები, მონაცემთა ტიპები, მმართველი ოპერატორები, სტრუქტურირებული ტიპები. გადმოცემულია ობიექტზე ორიენტირებული პროგრამირების პრინციპები: ინკაფსულაცია, პოლიმორფიზმი და მემკვიდრეობითობა. აღწერილია ინდექსატორებთან, თვისებებთან, ინტერფეისებთან, დელეგატებთან და მოვლენებთან მუშაობის მექანიზმები. განხილულია განსაკუთრებული სიტუაციები, ფაილებთან და კატალოგებთან მუშაობის საკითხები. სახელმძღვანელოში დამატებულია კოლექციები, შესრულების ნაკადები, გლობალიზაციასთან, გრაფიკასთან და მონაცემების დაშიფვრასთან დაკავშირებული საკითხები, აგრეთვე, დინამიური სტრიქონები. დაწვრილებითაა გადმოცემული SQL-სერვერის ბრძანებების სისტემა, მოყვანილია მონაცემთა ბაზებთან მომუშავე პროგრამების მაგალითები. წიგნში უხვადაა მაგალითები, ამოცანები და ტესტები.

განკუთვნილია დამწყები პროგრამისტებისათვის, კომპიუტერული სისტემებისა და ქსელების სპეციალობის ბაკალავრებისა და მაგისტრებისათვის.

პედაგოგიკის მეცნიერებათა დოქტორის, სრული პროფესორის გ. ჩაჩანიძის საერთო რედაქციით

რეცენზენტი: ტექნიკის მეცნიერებათა დოქტორი, სრული პროფესორი გ. სურგულაძე

© გამომცემლობა “ტექნიკური უნივერსიტეტი”, 2008

ISBN 99940-57-27-8

ყველა უფლება დაცულია. ამ წიგნის მასალები არანაირი ფორმით არ უნდა იყოს გამოყენებული ავტორის წერილობითი ნებართვის გარეშე.

წიგნს ვუძღვნი
ჩემს შვილებს ანას და საბას
და მეუღლეს ლიას

მადლობას ვუხდი

ჯეონეთის დირექტორს ალექსანდრე ქევხიშვილს და მენეჯერს ია ლასურაშვილს გაწეული ტექნიკური დახმარებისათვის.

პროგრამისტ ზურაბ ნონიაშვილს ფასდაუდებელი რჩევებისა და თანადგომისათვის.

სარჩევი

წინასიტყვაობა 11
I ნაწილი. C# ენა 12
თავი 1. შესავალი 12
პროგრამირების ისტორიის მოკლე მიმოხილვა 12
C# ენის კავშირი .NET Framework გარემოსთან 13
ობიექტზე ორიენტირებული პროგრამირება 14
ინკაფსულაცია 14
პოლიმორფიზმი 15
მემკვიდრეობითობა 15
თავი 2. C# ენის საფუძვლები 16
პირველი პროგრამა 16
კომენტარები 22
C# ენის საბაზო ტიპები 22
მონაცემთა ჩვეულებრივი ტიპები 22
მთელრიცხვა ტიპები 23
მცურავწერტილიანი ტიპები 24
decimal ტიპი 24
char ტიპი (სიმბოლო) 25
bool ტიპი 27
ლიტერალები 27
ცვლადები და მათი ინიციალიზება 28
ცვლადის ინიციალიზება 28
ცვლადების დინამიური ინიციალიზება 29
ოპერატორები 29
არითმეტიკის ოპერატორები 29
ინკრემენტისა და დეკრემენტის ოპერატორები 31
შედარებისა და ლოგიკის ოპერატორები 32
მინიჭების ოპერატორი 34
ოპერატორების პრიორიტეტები 35
გამოსახულებები. მათემატიკური მეთოდები 36
ტიპების გარდაქმნა 38
ტიპების დაყვანის ოპერატორი (cast operator) 39
ტიპების გარდაქმნა გამოსახულებებში 41
თავი 3. მმართველი ოპერატორები 43
ამორჩევის ოპერატორები 43
if ოპერატორი 43
ჩადგმული if ოპერატორი 43
პროგრამული კოდის ბლოკები. ხილვადობის უბანი 44
switch ოპერატორი 46
იტერაციის ოპერატორები 49
for ოპერატორი 49
მაგალითები 53
while ციკლი 54

do-while ციკლი 54
გადასვლის ოპერატორები 55
break ოპერატორი 55
continue ოპერატორი 56
goto ოპერატორი 57
ტერნარული ოპერატორი 57
თავი 4. მასივები, ბიტობრივი ოპერაციები და ჩამოთვლები 59
მასივები 59
ერთგანზომილებიანი მასივები 59
ორგანზომილებიანი მასივები 61
გაუსწორებელი (“დაკბილული”) მასივები 63
Length თვისება 65
foreach ციკლი 66
ბიტობრივი ოპერატორები 68
&, |, ^ და ~ ბიტობრივი ოპერატორები 68
ძვრის ოპერატორები 72
ჩამოთვლები 73
თავი 5. კლასები, ინკაფსულაცია, მეთოდები 75
კლასის გამოცხადება 75
ინკაფსულაცია 76
ობიექტების შექმნა 78
მიმართვითი ტიპის მნიშვნელობის მინიჭება 79
მეთოდები 80
მეთოდიდან მართვის დაბრუნება 82
პარამეტრების გამოყენება 83
კონსტრუქტორები 85
დესტრუქტორები და "ნაგვის შეგროვება" 87
this საკვანძო სიტყვა 88
სტრუქტურები 90
შემთხვევითი რიცხვების გენერატორი 91
თავი 6. მეთოდები უფრო დაწვრილებით. პოლიმორფიზმი 93
მეთოდისათვის ობიექტის გადაცემა 93
მეთოდისათვის არგუმენტების გადაცემის ხერხები 94
ref და out მოდიფიკატორები 96
ref მოდიფიკატორი 96
out მოდიფიკატორი 97
params მოდიფიკატორი 99
მეთოდის მიერ ობიექტის დაბრუნება 101
მეთოდის გადატვირთვა. პოლიმორფიზმი 102
კონსტრუქტორების გადატვირთვა 105
გადატვირთვადი კონსტრუქტორის გამოძახება this სიტყვის გამოყენებით 107
რეკურსია 108
static მოდიფიკატორი 109
თავი 7. ოპერატორების გადატვირთვა, თვისებები და ინდექსატორები 112
ოპერატორების გადატვირთვა 112

ოპერატორის მეთოდი ... 112
ბინარული ოპერატორების გადატვირთვა ... 113
უნარული ოპერატორების გადატვირთვა ... 114
შედარების ოპერატორების გადატვირთვა ... 116
ოპერატორების გადატვირთვისას არსებული შეზღუდვები ... 117
თვისებები ... 117
ინდექსატორები ... 119
ერთგანზომილებიანი ინდექსატორები ... 120
მრავალგანზომილებიანი ინდექსატორები ... 124
თავი 8. მემკვიდრეობითობა. ვირტუალური მეთოდები ... 126
მემკვიდრეობითობის საფუძვლები ... 126
protected მოდიფიკატორი ... 128
კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა ... 129
ცვლადების დამალვა მემკვიდრეობითობის დროს ... 132
კლასების მრავალდონიანი იერარქიის შექმნა ... 135
მიმართვები წინაპარი და მემკვიდრე კლასების ობიექტებზე ... 138
ვირტუალური მეთოდები ... 139
აბსტრაქტული კლასები ... 146
sealed საკვანძო სიტყვა ... 150
object კლასი ... 150
თავი 9. ინფორმაციის შეტანა-გამოტანა ... 151
შეტანა-გამოტანის ნაკადების კლასები ... 151
ბაიტების ნაკადები ... 151
სიმბოლოების ნაკადები ... 152
ორობითი ნაკადები ... 154
FileStream კლასი ... 154
ფაილის გახსნა და დახურვა ... 154
ფაილიდან ბაიტების წაკითხვა ... 155
ფაილში ბაიტების ჩაწერა ... 157
ფაილში სიმბოლოების შეტანა-გამოტანა ... 158
StreamWriter კლასი ... 159
StreamReader კლასი ... 159
ორობითი მონაცემების წაკითხვა და ჩაწერა ... 160
BinaryWriter კლასი ... 160
BinaryReader ნაკადი ... 160
ფაილებთან პირდაპირი მიმართვა ... 163
ფაილის შესახებ ინფორმაციის მიღება ... 164
კატალოგებთან მუშაობა ... 168
ფაილების არჩევა ... 171
თავი 10. განსაკუთრებული სიტუაციები ... 173
განსაკუთრებული სიტუაციების დამუშავების საფუძვლები ... 173
try და catch ბლოკები ... 174
try და catch ბლოკების გამოყენების მაგალითები ... 177
რამდენიმე catch ბლოკის გამოყენება ... 181
ყველა განსაკუთრებული სიტუაციის დაჭერა ... 182

ჩადგმული try ბლოკები ... 182
განსაკუთრებული სიტუაციის იძულებით გენერირება ... 183
finally ბლოკი ... 184
checked და unchecked ოპერატორები ... 186
გამართვა ... 188
წყვეტის წერტილების შექმნა ... 188
ცვლადების მნიშვნელობების ნახვა ... 189
პროგრამის კოდის შესრულება ბიჯობრივ რეჟიმში ... 192
თავი 11. დელეგატები, მოვლენები, ინტერფეისები და სახელების სივრცე ... 195
დელეგატები ... 195
დელეგატების მრავალმისამართიანობა ... 198
მოვლენები ... 200
ფართოსამაუწყებლო მოვლენა ... 202
ინტერფეისები ... 204
ინტერფეისების რეალიზაცია ... 205
კლასების მემკვიდრეობითობა და ინტერფეისის რეალიზება ... 206
ინტერფეისული მიმართვები ... 208
ინტერფეისის თვისებები ... 210
ცხადი რეალიზება ... 212
სახელების სივრცე ... 213
სახელების სივრცის გამოცხადება ... 214
using დირექტივა ... 215
ჩადგმული სახელების სივრცე ... 216
ავტომატურად განსაზღვრული სახელების სივრცე ... 218
თავი12. სტრიქონები ... 219
სტრიქონები ... 219
სტრიქონებთან სამუშაო მეთოდები ... 219
სტრიქონების მასივები ... 228
სტრიქონების უცვლელობა ... 229
დინამიური სტრიქონები ... 230
დინამიური სტრიქონის შექმნა ... 230
StringBuilder კლასის თვისებები და მეთოდები ... 231
II ნაწილი. C# ენის სხვა შესაძლებლობები ... 235
თავი 13. თარიღი, დრო და დროის ინტერვალები ... 235
თარიღი და დრო ... 235
დროის ინტერვალები ... 244
თავი 14. კოლექციები ... 249
დინამიური მასივები ... 249
ჰეშ-ცხრილები ... 259
დახარისხებული სიები ... 262
რიგები ... 267
სტეკები ... 268
ბიტების მასივები ... 270
თავი 15. შესრულების ნაკადები ... 273
შესრულების ნაკადის განსაზღვრა ... 273

შესრულების ნაკადის შექმნა ... 274
შესრულების ნაკადის პრიორიტეტები ... 278
შესრულების ნაკადის მდგომარეობები ... 280
შესრულების ნაკადის ლოკალური მონაცემები ... 282
შესრულების ნაკადების მართვა ... 284
Timer კლასი ... 284
Join() მეთოდი ... 286
lock საკვანძო სიტყვა ... 287
Interlocked კლასი ... 291
Monitor კლასი ... 293
Mutex კლასი ... 295
თავი 16. საბაზო ბიბლიოთეკის სხვა კლასები ... 298
გლობალიზაცია ... 298
გრაფიკა ... 302
გეომეტრიული ფიგურების ხატვა ... 306
ფიგურების შევსება ფერით ... 310
სტრიქონების ხატვა ... 311
გამოსახულებასთან მუშაობა ... 312
მონაცემების დაშიფვრა და გაშიფვრა ... 315
სიმეტრიული კრიპტოგრაფია ... 315
ასიმეტრიული კრიპტოგრაფია ... 318
III ნაწილი. მონაცემთა ბაზები ... 319
თავი 17. შესავალი მონაცემთა ბაზებში ... 319
მონაცემთა ბაზების აგების პრინციპები ... 319
მონაცემთა ბაზების ტიპები ... 320
თავი18. SQL-სერვერი ... 322
SQL-მოთხოვნის ტიპები ... 322
Query Analyzer უტილიტთან მუშაობა ... 322
მონაცემთა ტიპები ... 325
მონაცემთა ბაზისა და ცხრილების შექმნა ... 326
SELECT ბრძანება ... 328
შედარებისა და ლოგიკის ოპერატორები ... 330
სპეციალური ოპერატორები IN, BETWEEN, LIKE, IS NULL ... 332
აგრეგატული ფუნქციები ... 334
ჩანაწერების დახარისხება ... 336
რამოდენიმე ცხრილიდან ჩანაწერების ამორჩევა ... 336
ჩადგმული მოთხოვნები ... 337
ბმული მოთხოვნები (ქვემოთხოვნები) ... 338
EXISTS ოპერატორი ... 340
ANY და ALL ოპერატორები ... 342
UNION ბრძანება ... 342
ჩანაწერების ჩამატება, წაშლა და შეცვლა ... 344
ტრანზაქცია ... 345
მონაცემთა ბაზასთან მიმართვა Visual Studio .NET-ის საშუალებებით ... 349
თავი 19. ADO.NET ბიბლიოთეკა ... 353

შესავალი 353
ADO.NET კლასები 353
მონაცემებთან სამუშაო კლასები და ობიექტები 354
მართვადი კლასები და ობიექტები 354
მონაცემთა ბაზასთან მუშაობა ვიზუალური და არავიზუალური კომპონენტების გამოყენებით 355
SELECT-მოთხოვნის შესრულება ADO.NET კლასების საშუალებით 365
INSERT, UPDATE და DELETE მოთხოვნების შესრულება ADO.NET კლასების საშუალებით 369
ტრანზაქციებთან მუშაობა ADO.NET საშუალებებით 376
ჩანაწერების ფილტრაცია და დახარისხება 377
DataTable ობიექტებს შორის ბმების განსაზღვრა 380
დანართი 1. ამოცანები თავების მიხედვით 383
დანართი 2. C# ენის ტესტები 402
ტესტების პასუხები 531
ლიტერატურა 545

წინასიტყვაობა

Visual Studio .NET წარმოადგენს დაპროგრამების თანამედროვე ტექნოლოგიას. მისი საშუალებით შესაძლებელია სხვადასხვა ტიპის პროგრამა-დანართების შემუშავება, როგორიცაა, Windows და Web-პროგრამები, ქსელური პროგრამები, მონაცემთა ბაზები და ა.შ. იგი განკუთვნილია ქსელში ჩართულ კომპიუტერებთან და მოწყობილობებთან სამუშაოდ.

წიგნის მიზანია მკითხველს შეასწავლოს C# ენა და ობიექტზე ორიენტირებული პროგრამირების საფუძვლები. იგი განკუთვნილია დამწყები პროგრამისატებისათვის, თუმცა გადმოცემულმა მასალამ შეიძლება პროფესიონალებიც დააინტერესოს.

აუცილებელი პროგრამული უზრუნველყოფაა - Windows XP - SP/2, Visual Studio.NET.

წიგნში აღწერილია Microsoft Visual Studio .NET 2008 სისტემა.

წიგნი არის ზემოთ აღნიშნული საკითხების ქართულ ენაზე გადმოცემის ერთ-ერთი პირველი მცდელობა, ამიტომ იგი არ არის დაზღვეული ხარვეზებისაგან. ავტორი მადლიერებით მიიღებს წიგნში გადმოცემული მასალის დახვეწისა და სრულყოფის მიზნით გამოთქმულ შენიშვნებსა და მოსაზრებებს. ისინი შეგიძლიათ მომაწოდოთ მისამართებზე: romanis@geo.net.ge, r_samxaradze@geomail.ge, rsamkharadze@gmail.com.

საქართველოს ტექნიკური უნივერსიტეტის
ინფორმატიკისა და მართვის სისტემების ფაკულტეტის
კომპიუტერული ინჟინერიის დეპარტამენტის
სრული პროფესორი, ტექნიკის მეცნიერებათა დოქტორი

რომან სამხარაძე

I ნაწილი. C# ენა

თავი 1. შესავალი

პროგრამირების ისტორიის მოკლე მიმოხილვა

ამჟამად დაპროგრამების ენები გამოიყენება ისეთი მრავალფეროვანი ამოცანების გადასაწყვეტად, როგორიცაა მონაცემთა საინფორმაციო სისტემების მართვა, რთული მათემატიკური და ეკონომიკური ამოცანების გადაწყვეტა, მედიცინა და ა.შ. C# ენა, რომელიც შეიმუშავა Microsoft კომპანიამ, მთლიანად პასუხობს პროგრამირების თანამედროვე სტანდარტებს და განკუთვნილია .NET Framework ტექნოლოგიის განვითარების უზრუნველყოფისათვის. იგი არის დაპროგრამების მძლავრი ენა განკუთვნილი Windows გარემოში მომუშავე თანამედროვე კომპიუტერული სისტემებისთვის, რომლებიც იყენებენ ინტერნეტ-ტექნოლოგიებს.

დაპროგრამების ენებს შორის არსებობს კავშირი. ყოველი ახალი ენა ადრე შექმნილი ენებისაგან მემკვიდრეობით იღებს გარკვეულ თვისებებს. კერძოდ, C# ენამ ბევრი სასარგებლო თვისება C, C++ და Java ენებისაგან მემკვიდრეობით მიიღო.

C ენა შეიქმნა ნიუ-ჯერსის შტატის ქალაქ მიურეი-ჰილის Bell Laboratories კომპანიის სისტემური პროგრამისტის დენის რიჩის მიერ 1972 წელს. თითქმის მთლიანად ამ ენაზე დაიწერა Unix ოპერაციული სისტემის ბირთვი. შემდგომში პროგრამების ზომისა და სირთულის ზრდამ საკმაოდ გააძნელა მათთან მუშაობა. გამოსავალი იყო სტრუქტურულ პროგრამირებაზე გადასვლა. სწორედ C გახდა 1980 წლებიდან ყველაზე ხშირად გამოყენებადი სტრუქტურული ენა.

პროგრამირების განვითარებასთან ერთად კვლავ დადგა დიდი ზომის პროგრამებთან მუშაობის პრობლემა. აუცილებელი გახდა ახალი მიდგომების შემუშავება. ერთ-ერთი მათგანია ობიექტზე ორიენტირებული პროგრამირება (ოობ). იგი იძლევა დიდი ზომის პროგრამებთან ეფექტური მუშაობის შესაძლებლობას. შესაბამისად, შეიქმნა C ენის ობიექტზე ორიენტირებული ვერსია, რომელსაც 1983 წლიდან C++ დაერქვა. იგი შემუშავებულ იქნა იმავე Bell Laboratories კომპანიაში ბიარნ სტრაუსის მიერ. C++ მთლიანად მოიცავს C ენას და შეიცავს ობიექტზე ორიენტირებული პროგრამირების შესაძლებლობებს. 1990 წლიდან იწყება C++ ენის მასობრივი გამოყენება და იგი ხდება ყველაზე პოპულარული დაპროგრამების ენებს შორის.

დაპროგრამების ენების განვითარების საქმეში მნიშვნელოვანი მიღწევა იყო Java ენის შემუშავება Sun Microsystem კომპანიაში. მისი ავტორები იყვნენ ჯეიმს გოსლინგი, პატრიკ ნოტონი, კრის ვორტი, ედ ფრანკი და მაიკ შერიდანი. მასზე მუშაობა დაიწყო 1993 წლიდან. Java არის სტრუქტურული ობიექტზე ორიენტირებული ენა, რომელმაც C++ ენიდან აიღო სინტაქსი და სტრატეგია. ინტერნეტის ფართო გავრცელებამდე პროგრამების უმრავლესობის კომპილირება ხდებოდა კონკრეტული პროცესორისა და ოპერაციული სისტემისათვის. ინტერნეტის განვითარებასთან ერთად გაჩნდა სხვადასხვა პროცესორებისა და ოპერაციული სისტემის მქონე კომპიუტერების დაკავშირების შესაძლებლობა. ამან წინა პლანზე წამოსწია ერთი პლატფორმიდან მეორეზე პროგრამების ადვილად გადატანის პრობლემა. მის გადასაწყვეტად საჭირო

გახდა ახალი ენის შემუშავება. სწორედ ასეთი ენა გახდა Java.

Java თავიდანვე შეიქმნა როგორც პლატფორმაზე დამოუკიდებელი ენა. მას შეეძლო პლატფორმათაშორისი გადატანადი კოდის შექმნა, რაც გახდა მისი სწრაფი გავრცელების მიზეზი. გადატანადობა მიიღწევა პროგრამის საწყისი კოდის შუალედურ ენაში ტრანსლირების გზით, რომელსაც ბაიტ-კოდი ეწოდება. შემდეგ ეს შუალედური ენა სრულდება Java ვირტუალური მანქანის მიერ (Java Virtual Machine, JVM). შედეგად, Java-პროგრამა შეიძლება შესრულდეს ნებისმიერ პლატფორმაზე, რომელსაც Java ვირტუალური მანქანა აქვს.

Java ენისაგან განსხვავებით C და C++ ენებში პროგრამის საწყისი კოდის კომპილაცია ხორციელდება შესრულებად კოდში, რომელიც დაკავშირებულია კონკრეტულ პროცესორთან და ოპერაციულ სისტემასთან. ასეთი პროგრამის გასაშვებად სხვადასხვა პლატფორმაზე აუცილებელია პროგრამის საწყისი კოდის კომპილირება თითოეული პლატფორმისათვის. ეს კი შრომატევადი და ძვირადღირებული პროცესია. ამიტომაც, C# ენაში გადმოტანილი იქნა Java ენაში გამოყენებული მიდგომა - შუალედური ენის გამოყენება.

მართალია, Java ენამ გადაჭრა ერთი პლატფორმიდან მეორეზე პროგრამების გადატანასთან დაკავშირებული ბევრი პრობლემა, მაგრამ, მას აქვს რამდენიმე ნაკლი. მაგალითად, იგი ვერ უზრუნველყოფს რამდენიმე დაპროგრამების ენის ურთიერთქმედებას, ე.ი. არ უზრუნველყოფს მრავალენობრივ პროგრამირებას. მრავალენობრივი პროგრამირების ქვეშ იგულისხმება დაპროგრამების სხვადასხვა ენაზე დაწერილი კოდის ერთად მუშაობის უნარი. ეს შესაძლებლობა მეტად მნიშვნელოვანია დიდი პროგრამების შემუშავებისას, აგრეთვე, ცალკეული კომპონენტების შექმნისას, რომელთა გამოყენება შესაძლებელი იქნება მრავალ დაპროგრამების ენასა და სხვადასხვა ოპერაციულ სისტემაში.

Java ენის ერთ-ერთი სერიოზული ნაკლია, აგრეთვე, Windows პლატფორმის პირდაპირი უზრუნველყოფის არქონა, რომელიც ამჟამად მსოფლიოში ყველაზე გავრცელებული ოპერაციული სისტემაა. Java-პროგრამების შესრულება Windows გარემოში შესაძლებელია იმ შემთხვევაში თუ დაყენებულია (დაინსტალირებულია) Java ვირტუალური მანქანა.

ამ პრობლემის გადასაწყვეტად Microsoft კომპანიამ 1990 წლების ბოლოს შეიმუშავა C# ენა. მისი ავტორია ანდერს ჰელსბერგი. C# ენა მჭიდროდაა დაკავშირებული C, C++ და Java ენებთან. იგი აგებულია C++ ენაში განსაზღვრულ ობიექტურ მოდელზე, C ენიდან აღებულია სინტაქსი, ოპერატორები და საკვანძო სიტყვები, Java ენიდან კი - შუალედური ენის გამოყენება. C# ენის ერთ-ერთი მნიშვნელოვანი სიახლეა პროგრამული უზრუნველყოფის კომპონენტების ჩადგმული უზრუნველყოფა. ფაქტობრივად C# ენა შექმნილია, როგორც კომპონენტებზე ორიენტირებული ენა, რომელიც მოიცავს ისეთ ელემენტებს, როგორიცაა თვისებები, მეთოდები და მოვლენები. მაგრამ, ყველაზე მნიშვნელოვანი სიახლე C# ენაში არის მრავალენობრივ გარემოში მისი მუშაობის უნარი.

C# ენის კავშირი .NET Framework გარემოსთან

.NET Framework განსაზღვრავს გარემოს, რომელიც უზრუნველყოფს პლატფორმაზე დამოუკიდებელი პროგრამა-დანართების (Application) შემუშავებასა და

შესრულებას. იგი, აგრეთვე, უზრუნველყოფს პროგრამების გადატანადობას. შედეგად, Windows-პროგრამები შეგვიძლია სხვა პლატფორმებზე ვამუშავოთ.

C# ენა იყენებს .NET Framework გარემოს ორ მთავარ ნაწილს. პირველია ენაზე დამოუკიდებელი შესრულების გარემო (Common Language Runtime, CLR). იგი მართავს ჩვენი პროგრამის შესრულებას და წარმოადგენს .NET Framework ტექნოლოგიის ნაწილს, რომელიც უზრუნველყოფს პროგრამების გადატანადობასა და პროგრამირებას რამდენიმე ენის გამოყენებით. მეორეა - კლასების ბიბლიოთეკა .NET, რომელიც პროგრამას საშუალებას აძლევს მიმართოს შესრულების გარემოს, მაგალითად, მონაცემების შეტანა-გამოტანისათვის.

მოკლედ განვიხილოთ CLR სისტემის მუშაობა. ფაილს, რომელიც შეიცავს C#-პროგრამის საწყის კოდს აქვს .cs გაფართოება. ამ ფაილის კომპილირებას მანქანურ კოდებში ასრულებს პროგრამა, რომელსაც კომპილატორი ეწოდება. C#-პროგრამების კომპილირების შედეგად მიიღება ფაილი, რომელიც შეიცავს შუალედურ ენას - MSIL (Microsoft Intermediate Language, MSIL). იგი შედგება გადასატანი ინსტრუქციების ნაკრებისაგან, რომელიც არაა დამოკიდებული კონკრეტული პროცესორის ინსტრუქციების ნაკრებზე. CLR სისტემა ახდენს შუალედური კოდის ტრანსლირებას შესრულებად კოდში პროგრამის გაშვების დროს. MSIL ენაში კომპილირებული პროგრამა შეიძლება ნებისმიერ ოპერაციულ სისტემაში შესრულდეს.

MSIL ენა შესრულებად კოდში გარდაიქმნება JIT კომპილატორის მიერ (just in time - საჭირო მომენტში). C#-პროგრამის გაშვებისას CLR სისტემა ააქტიურებს JIT კომპილატორს, რომელიც MSIL ენას გარდაქმნის მოცემული პროცესორის შიდა კოდად. ამასთან, პროგრამის ნაწილების გარდაქმნა საჭიროებისდა მიხედვით სრულდება.

ობიექტზე ორიენტირებული პროგრამირება

ჩვენს გარშემო ბევრი ობიექტია. მაგალითად, ავტომობილი, თვითმფრინავი, მატარებელი, მაღაზია, ავადმყოფი, ექიმი, სტუდენტი, გეომეტრიული ფიგურა და ა.შ. მსგავსი ობიექტები შეგვიძლია დავაჯგუფოთ კლასებში. მაგალითად, სხვადასხვა ავტომობილს ბევრი საერთო თვისება აქვს, ამიტომ ისინი შეგვიძლია ავტომობილის საერთო კლასში გავაერთიანოთ. ასევე სხვადასხვა ტიპის თვითმფრინავს ბევრი საერთო თვისება აქვს, ამიტომ ისინი შეგვიძლია თვითმფრინავის საერთო კლასში გავაერთიანოთ და ა.შ.

ობიექტზე ორიენტირებულ ენებში პროგრამები ორგანიზებულია მონაცემების გარშემო, ანუ ჩვენ განვსაზღვრავთ მონაცემებს და იმ პროგრამებს, რომლებიც ამ მონაცემებთან მუშაობენ. C# ენა ეფუძნება ობიექტზე ორიენტირებული პროგრამირების (ოოპ) სამ პრინციპს პრინციპს: ინკაფსულაცია, პოლომორფიზმი და მემკვიდრეობითობა. მოკლედ განვიხილოთ თითოეული მათგანი.

ინკაფსულაცია

ინკაფსულაცია არის დაპროგრამების მექანიზმი, რომელიც აერთიანებს პროგრამის კოდსა და იმ მონაცემებს, რომლებთანაც ეს კოდი მუშაობს, აგრეთვე, გამიჯნავს მათ (კოდსა და მონაცემებს) სხვა პროგრამების მხრიდან მიმართვებისაგან. ამით ხდება მათი დაცვა არასწორი გამოყენებისაგან. ობიექტზე ორიენტირებულ ენაში პროგრამის კოდი და მონაცემები ერთმანეთს ისე უკავშირდება, რომ ქმნიან ერთ ავტონომიურ სტრუქტურას, რომელსაც ობიექტი ეწოდება.

ობიექტის შიგნით პროგრამის კოდი და მონაცემები სხვა ობიექტებისათვის შეიძლება იყოს დახურული (private) ან ღია (public). დახურულ (პრივატულ) კოდთან და მონაცემებთან მიმართვა შეუძლიათ მხოლოდ ამავე ობიექტში აღწერილ კოდებს. ეს იმას ნიშნავს, რომ დახურულ კოდსა და მონაცემებს ვერ მივმართავთ პროგრამის სხვა ნაწილიდან, რომელიც ობიექტის გარეთაა მოთავსებული. ღია (საერთოწვდომის) კოდთან და მონაცემებთან მიმრთვა შესაძლებელია პროგრამის ნებისმიერი ნაწილიდან. როგორც წესი, ღია კოდი გამოიყენება ობიექტის პრივატულ ელემენტებთან მიმართვისათვის.

კლასი არის სტრუქტურა, რომელიც იყენებს რა ინკაფსულაციის პრინციპს, განსაზღვრავს კოდს და იმ მონაცემებს, რომლებთანაც ის მუშაობს. კლასი გამოიყენება ობიექტების აღსაწერად და შესაქმნელად. ობიექტები კლასის ეგზემპლარებს წარმოადგენენ. კლასის შემადგენელ კოდსა და მონაცემებს კლასის წევრები ეწოდებათ, კლასის მიერ განსაზღვრულ მონაცემებს კი – ეგზემპლარის ცვლადები. კლასში განსაზღვრულ პროგრამის კოდებს მეთოდები ეწოდებათ.

პოლიმორფიზმი

პოლიმორფიზმი არის დაპროგრამების მექანიზმი, რომელიც მსგავს ობიექტებს საშუალებას აძლევს ერთი ინტერფეისის გამოყენებით მიმართონ სხვადასხვა მეთოდებს. სხვა სიტყვებით, რომ ვთქვათ, ერთი და იგივე სახელი შეიძლება რამდენიმე მეთოდს ჰქონდეს, რომლებიც ერთსა და იმავე მოქმედებებს სხვადასხვა ტიპის მონაცემებზე ასრულებენ. ამ შემთხვევაში არგუმენტის ტიპის მიხედვით სრულდება შესაბამისი მეთოდის გამოძახება. კონკრეტულ მეთოდს არგუმენტის ტიპის მიხედვით ირჩევს კომპილატორი. ამრიგად, პოლიმორფიზმი საშუალებას გვაძლევს რამდენიმე სახელის ნაცვლად გამოვიყენოთ ერთი. პოლიმორფიზმის უზრუნველყოფა ხდება მეთოდების გადატვირთვის საშუალებით.

მემკვიდრეობითობა

მემკვიდრეობითობა არის დაპროგრამების მექანიზმი, რომლის საშუალებითაც ერთ ობიექტს მემკვიდრეობით გადაეცემა მეორე ობიექტის ელემენტები. ამავე დროს დაცულია იერარქიული სტრუქტურა მიმართული ზევიდან ქვევით. მაგალითად, კლასი ლომი არის კატისებრთა კლასის ნაწილი, რომელიც თავის მხრივ ეკუთვნის მტაცებლები კლასს. ეს უკანასკნელი კი არის ცხოველები კლასის ნაწილი. ცხოველები კლასს აქვს ისეთი მახასიათებლები, როგორიცაა ტყეში ცხოვრება და ა.შ. იგივე მახასიათებლები შეგვიძლია გამოვიყენოთ მტაცებლები კლასის მიმართ, რომელსაც დამატებით აქვს თავისი სპეციფიკური მახასიათებლები, როგორიცაა ნადირობის უნარი და ა.შ., რომელიც მას სხვა ცხოველებისაგან განასხვავებს. აღნიშნული მახასიათებლების მატარებელია კატისებრთა კლასი. იგი დამატებით შეიცავს მისთვის დამახასიათებელ მახასიათებლებს, რომლებიც მას სხვა მტაცებლებისაგან განასხვავებს. დაბოლოს, ლომი კლასი ყველა ზემოთ აღნიშნული მახასიათებლების მატარებელია და დამატებით შეიცავს მხოლოდ მისთვის დამახასიათებელ სპეციფიკურ მახასიათებლებს.

მემკვიდრეობითობის გამოყენება ობიექტს საშუალებას აძლევს განსაზღვროს ის ელემენტები, რომლებიც მხოლოდ მისთვისაა დამახასიათებელი. ობიექტი მშობელი (წინაპარი) კლასისაგან მემკვიდრეობით იღებს საერთო ელემენტებს. ამიტომ, მემკვიდრე კლასის აღწერისას აღარ ხდება საჭირო მისი ყველა ელემენტის აღწერა.

თავი 2. C# ენის საფუძვლები

პირველი პროგრამა

C# ენაზე შევადგინოთ უმარტივესი პროგრამა, რომელიც შეასრულებს ორი მთელი რიცხვის შეკრებას.

```
{
//      პროგრამა 2.1
//      ეს არის ჩვენი პირველი პროგრამა
int ricxvi1, ricxvi2, jami;

ricxvi1 = 2;
ricxvi2 = 3;
jami    = ricxvi1 + ricxvi2;
}
```

როგორც ვხედავთ C# ენაზე შედგენილი პროგრამა იწყება გამხსნელი ფიგურული ფრჩხილით { და მთავრდება დამხურავი ფიგურული ფრჩხილით }. პროგრამის მეორე და მესამე სტრიქონებში მოთავსებულია კომენტარები, რომელებიც // სიმბოლოებით იწყება (კომენტარებს მოგვიანებით განვიხილავთ). მომდევნო სტრიქონში ხდება ცვლადების გამოცხადება. int სიტყვა (integer - მთელი) იწყებს ცვლადების გამოცხადებას და მიუთითებს, რომ ისინი მთელ რიცხვებს წარმოადგენენ. მას მოსდევს ცვლადების სახელები ანუ იდენტიფიკატორები, რომლებიც ერთმანეთისაგან მძიმეებით გამოიყოფა. წერტილ-მძიმე ამთავრებს ცვლადების გამოცხადებას, აგრეთვე, ერთმანეთისაგან გამოყოფს ოპერატორებსა და მეთოდებს (მეთოდი არის მცირე ზომის პროგრამა, რომელიც გარკვეულ მოქმედებებს ასრულებს). ცვლადების სახელები (იდენტიფიკატორები) აუცილებლად უნდა იწყებოდეს სიმბოლოთი და უმჯობესია შევარჩიოთ შინაარსიდან გამომდინარე. C# ენაში ნებისმიერი ცვლადი გამოცხადებული უნდა იყოს მის გამოყენებამდე.

პროგრამის მეექვსე სტრიქონში ricxvi1 ცვლადს ენიჭება მთელი რიცხვი - 2. “=” სიმბოლო წარმოადგენს მინიჭების ოპერატორს. იგი ახდენს მის მარჯვნივ მოთავსებული მნიშვნელობის გადაწერას მის მარცხნივ მოთავსებულ ცვლადში. იგივე ეხება პროგრამის მეშვიდე სტრიქონს. მერვე სტრიქონში jami ცვლადს ენიჭება ricxvi1 და ricxvi2 ცვლადების მნიშვნელობების ჯამი.

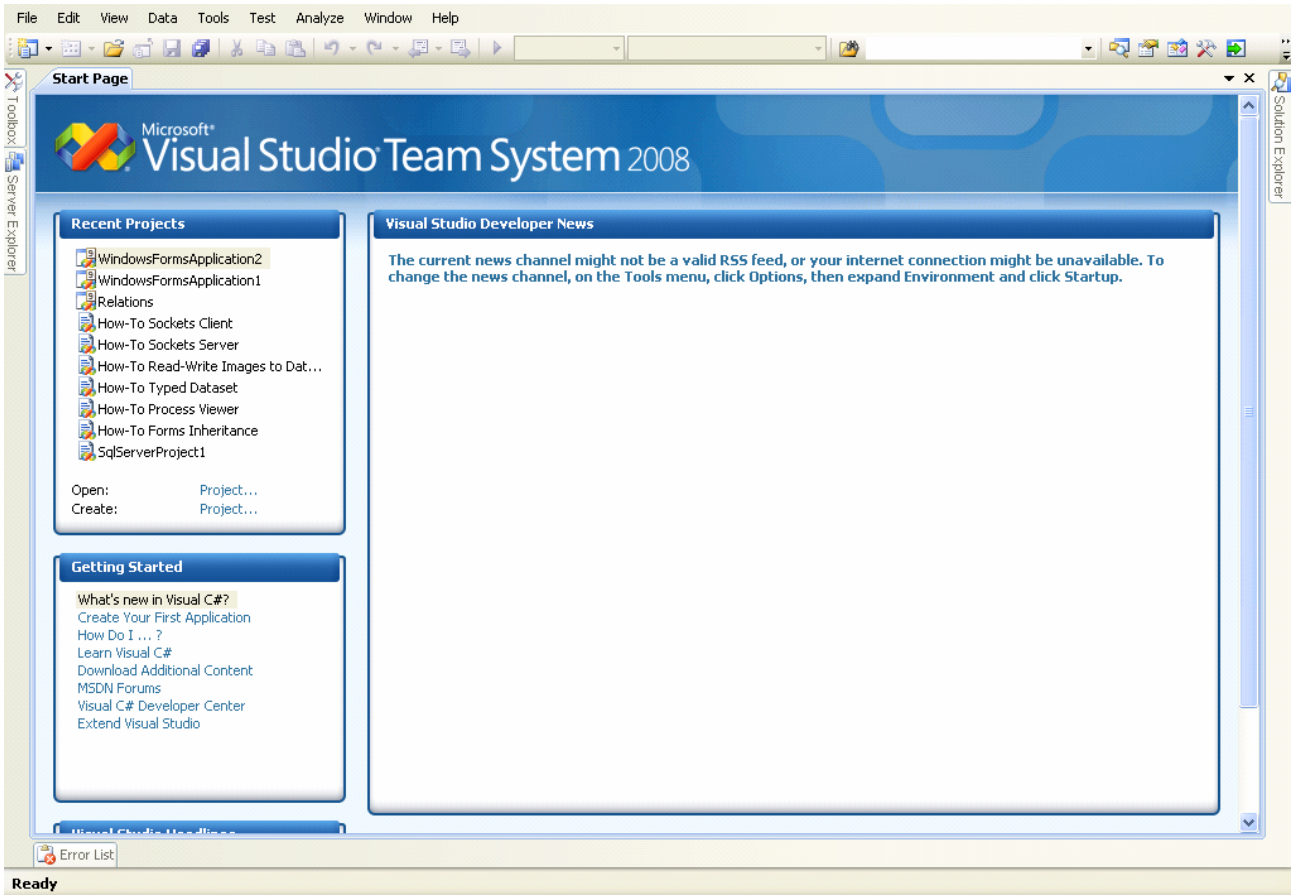
თვალსაჩინოების მიზნით პროგრამის თითოეულ სტრიქონში მოთავსებულია თითო ოპერატორი. თუმცა ერთ სტრიქონში შეიძლება რამდენიმე ოპერატორის მოთავსება. მთავარია, რომ ისინი ერთმანეთისაგან წერტილ-მძიმეებით იყოს გამოყოფილი.

ერთ-ერთი ნაკლი, რომელიც ამ პროგრამას აქვს ის არის, რომ სხვადასხვა რიცხვების შესაკრებად მოგვიწევს პროგრამის კოდის შეცვლა, რაც არასწორია. გარდა ამისა, პროგრამას შედეგი ეკრანზე არ გამოაქვს. ამ პრობლემების გადასაწყვეტად გამოვიყენებთ ვიზუალურ კომპონენტებს, კერძოდ, მონაცემების შესატანად გამოვიყენებთ textBox კომპონენტს, გამოსატანად - label კომპონენტს, პროგრამის (მეთოდის) შესასრულებლად კი - button კომპონენტს. მათი გამოყენებით პროგრამა 2.1. შემდეგ სახეს მიიღებს:

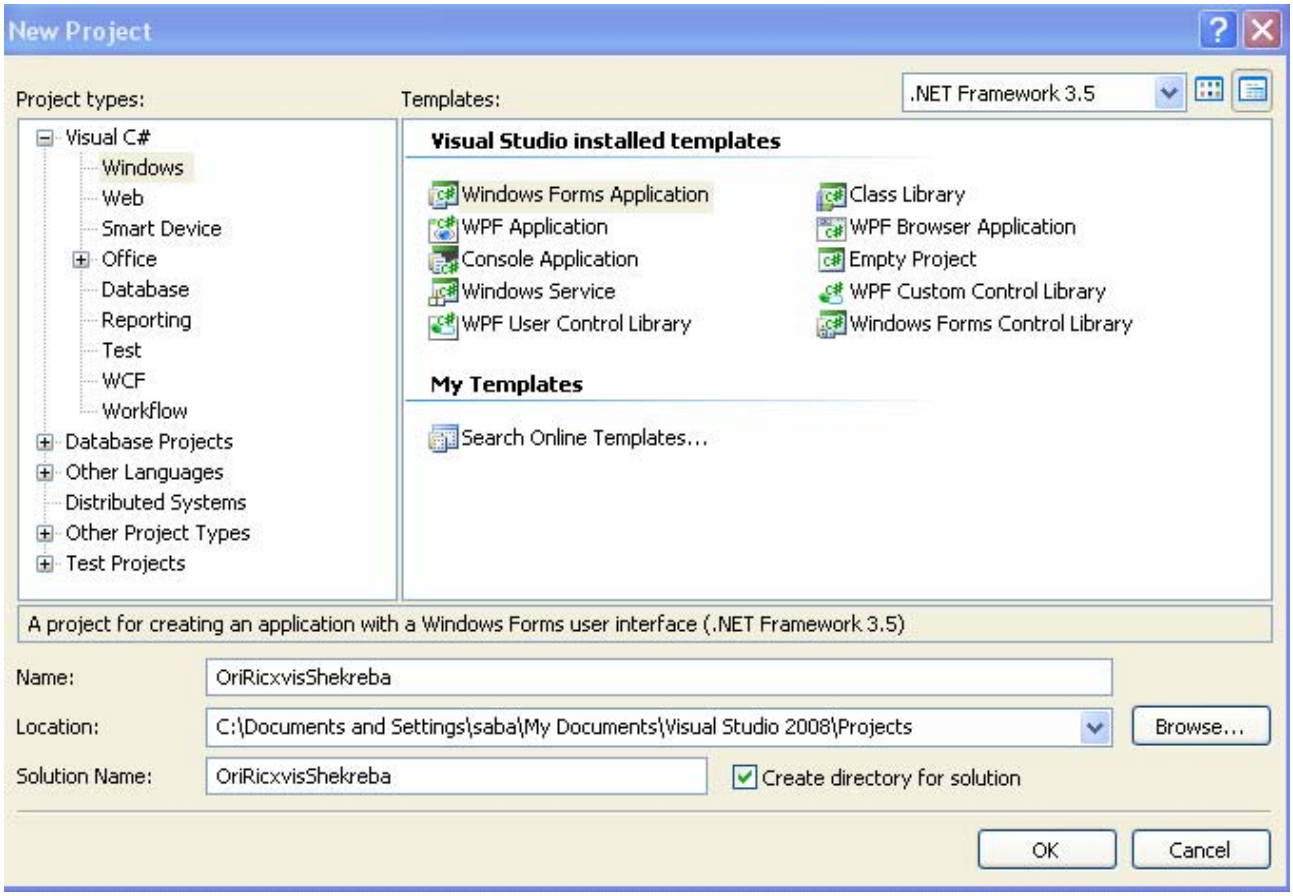
```
{
//      პროგრამა 2.2
//      ორი რიცხვის შეკრების პროგრამა
int ricxvi1, ricxvi2, jami;

ricxvi1 = Convert.ToInt32(textBox1.Text);
ricxvi2 = Convert.ToInt32(textBox2.Text);
jami    = ricxvi1 + ricxvi2;
label5.Text = jami.ToString();
}
```

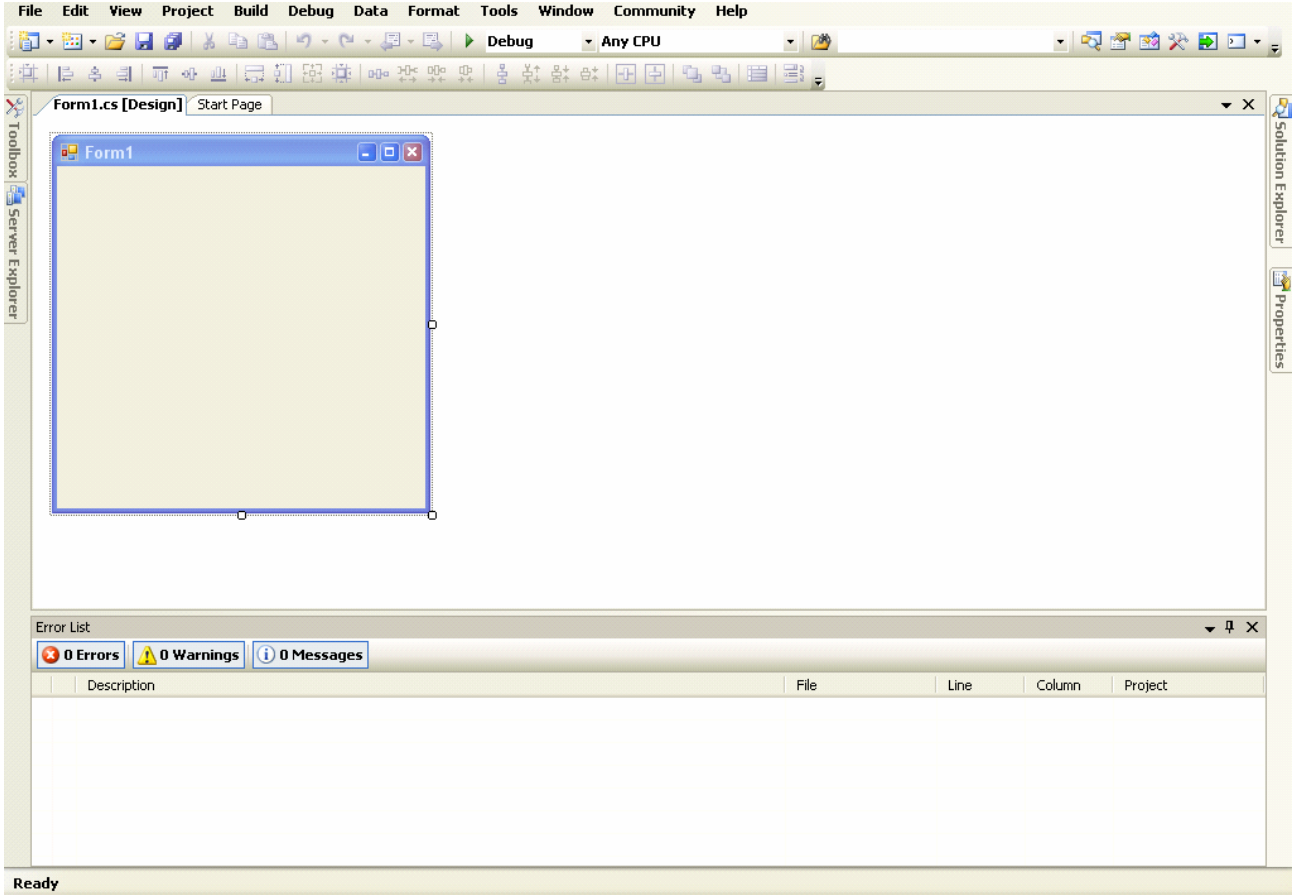
როგორც პროგრამიდან ჩანს ricxvi1 ცვლადს ენიჭება textBox1 კომპონენტში შეტანილი მნიშვნელობა (მაგალითად, 5), რომელიც Convert.ToInt32 მეთოდის საშუალებით მთელ რიცხვად გარდაიქმნება. საქმე ის არის, რომ textBox1 კომპონენტში შეტანილი მონაცემი მიენიჭება ამავე კომპონენტის Text თვისებას, რომელსაც სტრიქონული ტიპი აქვს. ამიტომ, ამ მონაცემს აქვს არა რიცხვითი, არამედ სტრიქონული ტიპი. Convert.ToInt32 მეთოდი თავის არგუმენტს, ჩვენს შემთხვევაში ესაა textBox1.Text, გარდაქმნის რიხცვად. შედეგად, textBox1 კომპონენტში შეტანილი მონაცემი, რომელიც სინამდვილეში სტრიქონია, მთელ რიცხვად გარდაიქმნება. ასეთი გარდაქმნა საჭიროა იმიტომ, რომ ricxvi1 არის მთელი ტიპის მქონე ცვლადი, ამიტომ მას უნდა მიენიჭოს მხოლოდ მთელი რიცხვი. ეს იმას ნიშნავს, რომ მინიჭების ოპერატორის მარცხნივ მოთავსებულ ცვლადს და მარჯვნივ მოთავსებულ გამოსახულებას ერთნაირი ტიპები უნდა ჰქონდეთ. იგივე ეხება მეექვსე და მეშვიდე სტრიქონებს.



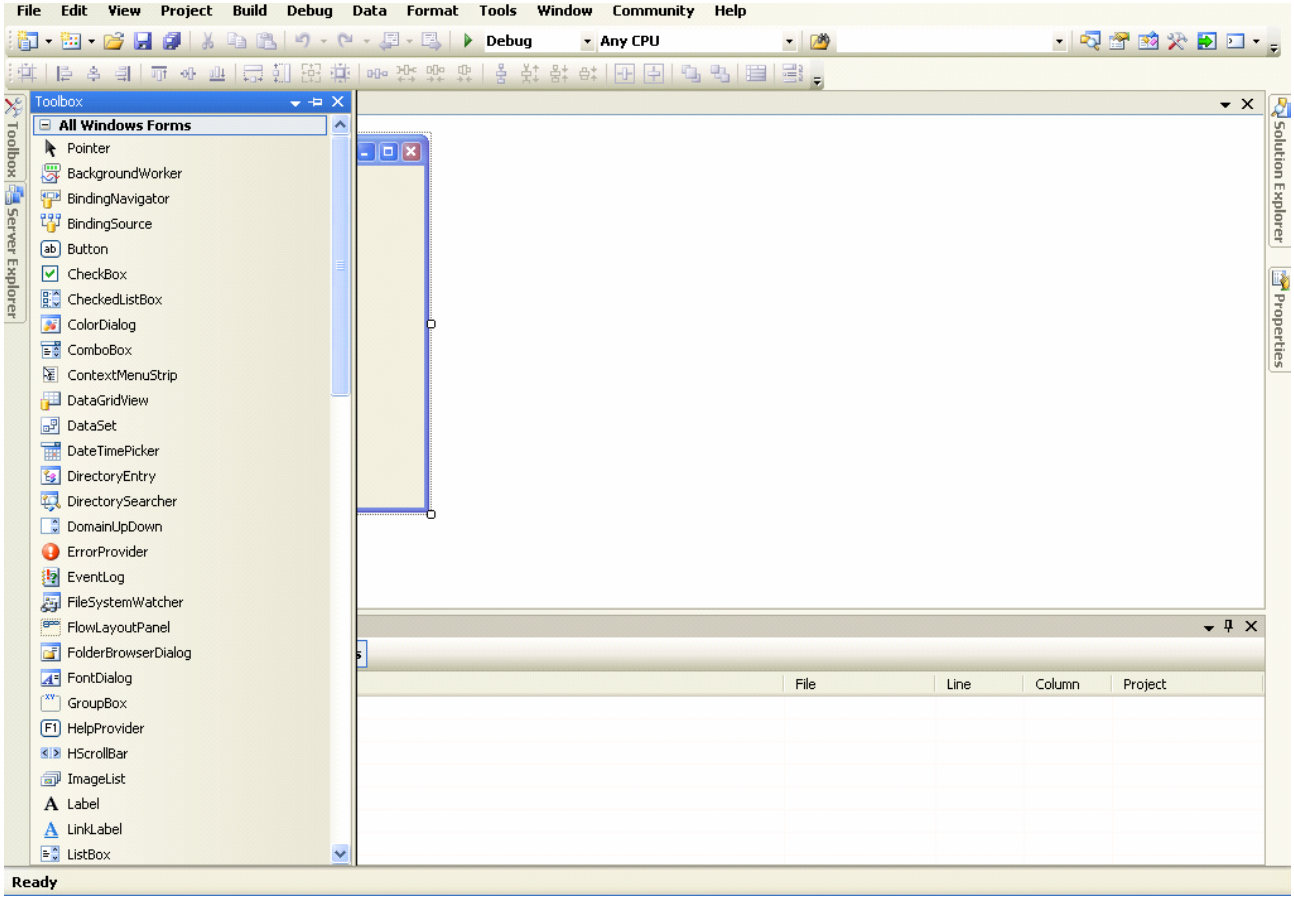
ნახ. 2.1.


New Project
Project types:
Templates:
.NET Framework 3.5
Visual C#
Windows
Web
Smart Device
Office
Database
Reporting
Test
WCF
Workflow
Database Projects
Other Languages
Distributed Systems
Other Project Types
Test Projects
Visual Studio installed templates
Windows Forms Application
WPF Application
Console Application
Windows Service
WPF User Control Library
Class Library
WPF Browser Application
Empty Project
WPF Custom Control Library
Windows Forms Control Library
My Templates
Search Online Templates...
A project for creating an application with a Windows Forms user interface (.NET Framework 3.5)
Name:
OriRicxvisShekreba
Location:
C:\Documents and Settings\saba\My Documents\Visual Studio 2008\Projects
Browse...
Solution Name:
OriRicxvisShekreba
Create directory for solution
OK
Cancel

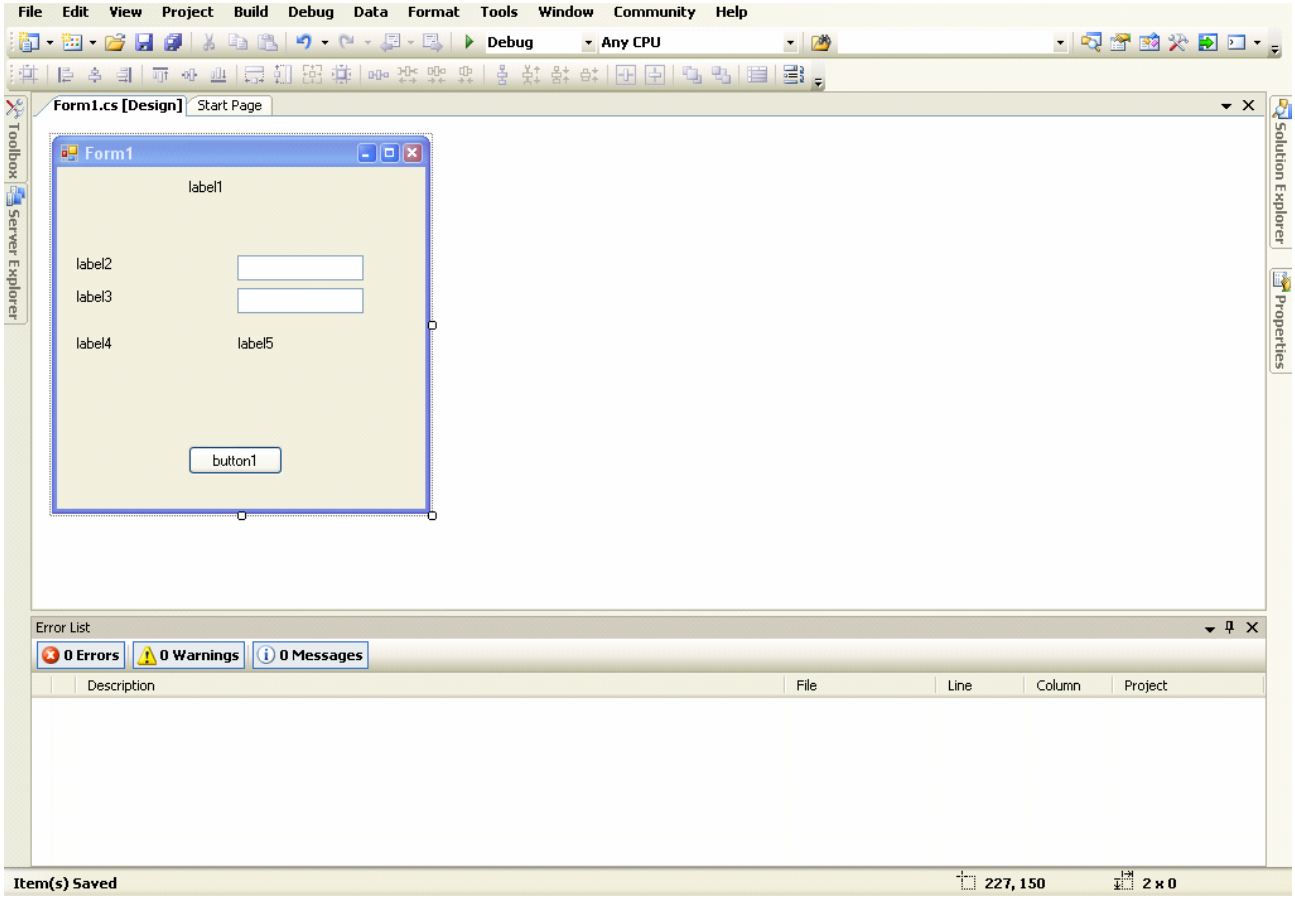
ნახ. 2.2.


File Edit View Project Build Debug Data Format Tools Window Community Help
Debug
Any CPU
Form1.cs [Design]
Start Page
Form1
Toolbox
Server Explorer
Solution Explorer
Properties
Error List
0 Errors
0 Warnings
0 Messages
Description
File
Line
Column
Project
Ready

ნახ. 2.3.


File Edit View Project Build Debug Data Format Tools Window Community Help
Debug
Any CPU
Toolbox
All Windows Forms
Pointer
BackgroundWorker
BindingNavigator
BindingSource
Button
CheckBox
CheckedListBox
ColorDialog
ComboBox
ContextMenuStrip
DataGridView
DataSet
DateTimePicker
DirectoryEntry
DirectorySearcher
DomainUpDown
ErrorProvider
EventLog
FileSystemWatcher
FlowLayoutPanel
FolderBrowserDialog
FontDialog
GroupBox
HelpProvider
HScrollBar
ImageList
Label
LinkLabel
ListBox
Solution Explorer
Properties
Server Explorer
File
Line
Column
Project
Ready

ნახ. 2.4.


File Edit View Project Build Debug Data Format Tools Window Community Help
Debug
Any CPU
Form1.cs [Design]
Start Page
Form1
label1
label2
label3
label4
label5
button1
Toolbox
Server Explorer
Solution Explorer
Properties
Error List
0 Errors
0 Warnings
0 Messages
Description
File
Line
Column
Project
Item(s) Saved
227, 150
2 x 0

ნახ. 2.5.

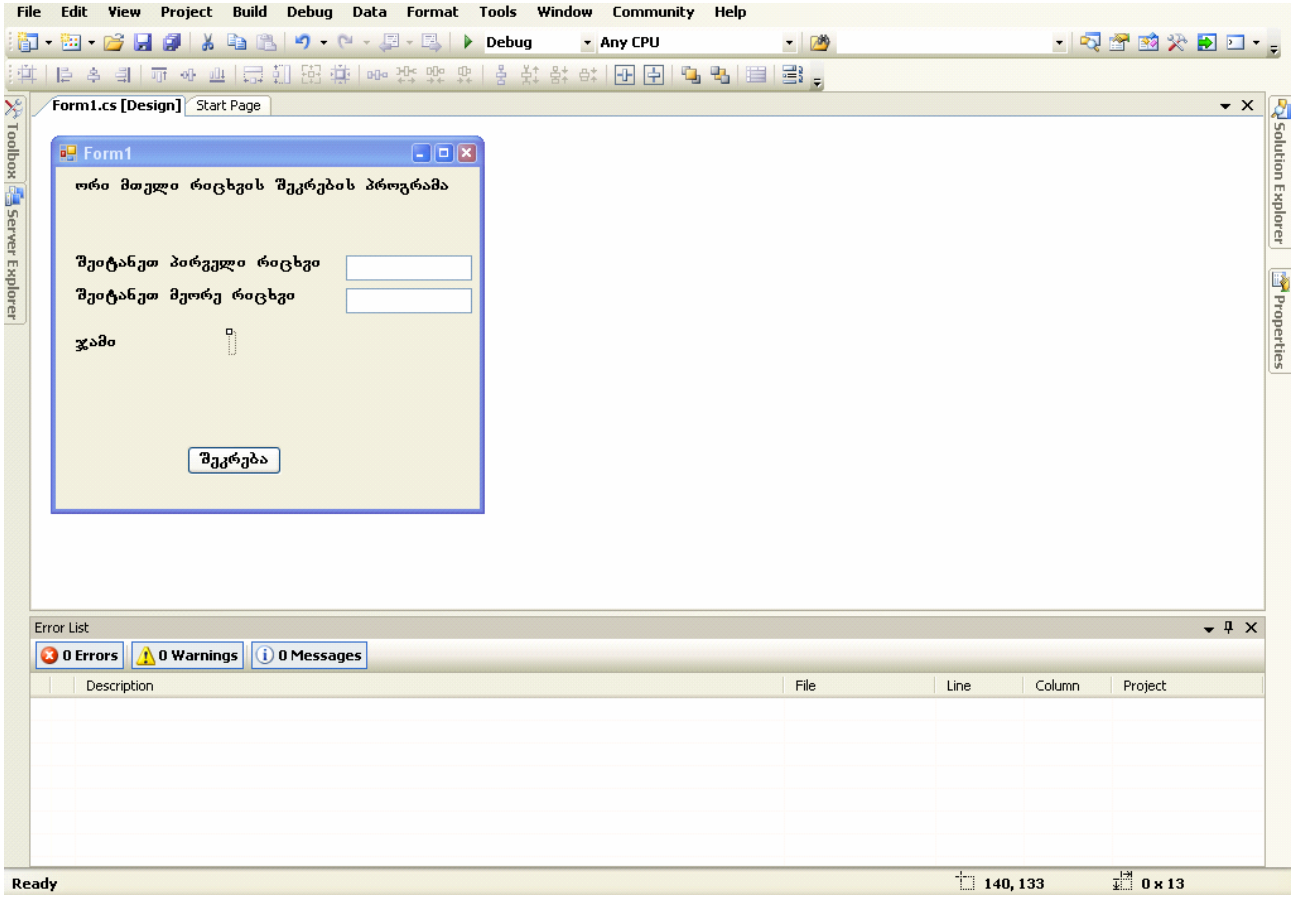
რაც შეეხება მერვე სტრიქონს, აქ მინიჭების ოპერატორის მარცხნივ მოთავსებულია label5 კომპონენტი. მის Text თვისებაში ვწერთ შედეგს, კერძოდ, jami ცვლადის მნიშვნელობას. რადგან Text თვისებას აქვს სტრიქონული ტიპი, ამიტომ jami ცვლადის ტიპიც უნდა გარდავქმნათ მთელი რიცხვიდან სტრიქონად. სწორედ ამას აკეთებს ToString() მეთოდი.

ახლა შევასრულოთ ეს პროგრამა. ამისათვის ჯერ გავუშვათ Microsoft Visual Studio .NET 2008 სისტემა. გაიხსნება Microsoft Development Environment ფანჯარა (ნახ. 2.1). Recent Projects განყოფილების Create ვაჭერთ Project მიმართვას. გაიხსნება New Project ფანჯარა (ნახ. 2.2). Project Types სიაში მოვნიშნოთ Windows ელემენტი. Templates ზონაში მოვნიშნოთ Windows Application ელემენტი. Name ველში შეგვიძლია შევიტანოთ პროექტის სახელი, მაგალითად, OriRicxvisShekreba. შემდეგ ვაჭერთ Browse კლავიშს და მოვნიშნავთ იმ კატალოგს, რომელშიც უნდა შეიქმნას OriRicxvisShekreba ქვეკატალოგი (შეგვიძლია წინასწარ შევქმნათ ის კატალოგი, რომელშიც მოვათავსებთ აღნიშნულ ქვეკატალოგს). ვაჭერთ Open კლავიშს. ბოლოს ვაჭერთ Ok კლავიშს. თუ კატალოგს არ ავირჩევთ, მაშინ OriRicxvisShekreba ქვეკატალოგი შეიქმნება Visual Studio Projects კატალოგში. ეკრანზე გამოჩნდება ფორმა, რომლის სახელია Form1 (ნახ. 2.3). ფორმის მარცხნივ მოთავსებულია ვიზუალური კომპონენტების პანელი (ToolBox). მასზე მოვათავსოთ ისარი (კურსორი, მიმთითებელი). გამოჩნდება ინსტრუმენტების პანელი (ნახ. 2.4). გავხსნათ Windows Forms პანელი. ავირჩიოთ textBox კომპონენტი. ამისათვის, მასზე მოვათავსოთ მიმთითებელი, დავაჭიროთ თაგვის მარცხენა კლავიშს, შემდეგ, მიმთითებელი მოვათავსოთ ფორმაზე საჭირო ადგილას და ისევ დავაჭიროთ თაგვის მარცხენა კლავიშს. შედეგად, კომპონენტი ფორმაზე გამოჩნდება მითითებულ ადგილზე. მისი სახელი იქნება textBox1 (ნახ. 2.5). ასეთი გზით ფორმაზე მოვათავსოთ კიდევ ერთი textBox კომპონენტი - textBox2, ერთი Button კომპონენტი და ხუთი label კომპონენტი. ფორმა მიიღებს ნახ. 2.5-ზე ნაჩვენებ სახეს. ვიზუალური კომპონენტები ფორმაზე შეგვიძლია განვათავსოთ ჩვენი მოსაზრებებიდან და გემოვნებიდან გამომდინარე. აქ შემოდის დიზაინის საკითხებიც, კერძოდ, ფორმაზე კომპონენტები ისე უნდა იყოს განლაგებული, რომ მათთან მუშაობა მოხერხებული იყოს.

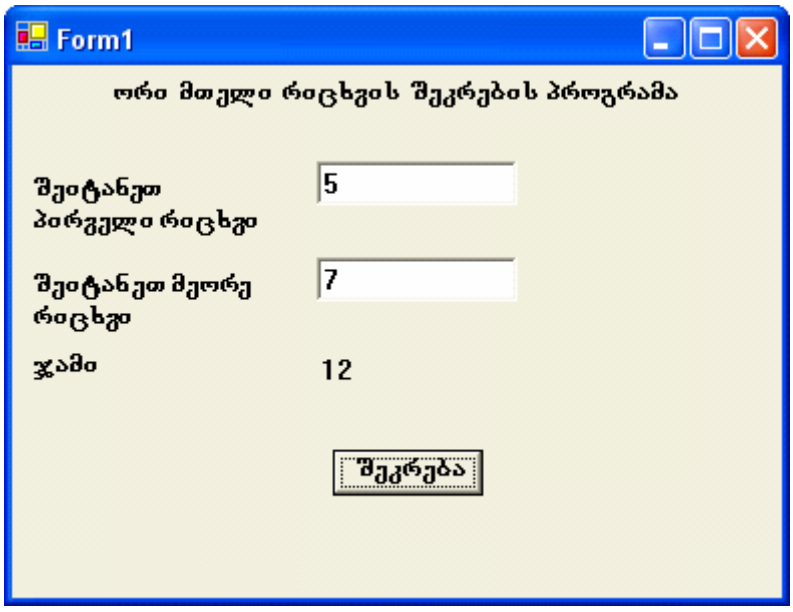
მოვნიშნოთ label1 კომპონენტი და მის Text თვისებაში, რომელსაც ვიპოვით ფანჯრის მარჯვენა მხარეს მოთავსებული თვისებების (Properties) ფანჯარაში, შევიტანოთ ტექსტი: “ორი მთელი რიცხვის შეკრების პროგრამა” და დავაჭიროთ კლავიატურის Enter კლავიშს. შემდეგ, ამავე ფანჯრის Font თვისებაში ვირჩევთ ქართულ შრიფტს, ზომას და სტილს. ანალოგიურად ვიქცევით დანარჩენი კომპონენტე-ბისთვისაც. მივიღებთ ნახ. 2.6-ზე ნაჩვენებ ფორმას.

ფორმაზე კომპონენტების განლაგების შემდეგ ორჯერ სწრაფად ვაწკაპუნებთ button1 კლავიშზე. გაიხსნება პროგრამის კოდის ზონა, რომელშიც შეგვაქვს პროგრამა 2.2. პროგრამის შეტანის შემდეგ მის შესანახად უნდა დავაჭიროთ ინსტრუმენტების პანელის Save All კლავიშს. ფორმაზე გადასასვლელად ვაჭერთ Shift+F7 კლავიშებს (+ ნიშნავს კლავიშებზე ერთდროულ დაჭერას), ფორმიდან პროგრამის კოდზე გადასასვლელად კი - F7 კლავიშს. პროგრამის შესასრულებლად გამოიყენება F5 კლავიში. თუ პროგრამა უშეცდომოდ შევიტანეთ, მაშინ ეკრანზე გამოჩნდება ფორმა (ნახ. 2.7). textBox1 და textBox2 კომპონენტებში შევიტანოთ მთელი რიცხვები და დავაჭიროთ button1 კლავიშს, რომლის სათაურში ჩანს სიტყვა “შეკრება”. label5 კომპონენტში გამოჩნდება შეტანილი რიცხვების ჯამი. ამის შემდეგ კვლავ შეგვიძლია შევიტანოთ სხვა რიცხვები და დავაჭიროთ “შეკრება” კლავიშს და ა.შ. ბოლოს, პროგ-

რამის დასამთავრებლად ვაჭერთ ფორმის ზედა მარჯვენა კუთხეში მოთავსებულ კლავიშს.



ნახ. 2.6.



ნახ. 2.7.

შევნიშნოთ, რომ C# ენაში განსხვავებულია ზედა და ქვედა რეგისტრის სიმბოლოები. მაგალითად, განსხვავებულია სახელები Computer და computer. განვიხილოთ პროგრამა:

```
{
//      პროგრამა 2.3
//      C# ენაში განსხვავებულია ზედა და ქვედა რეგისტრის სიმბოლოები
int ricxvi1;
Ricxvi1 = 5;   //      შეცდომა! Ricxvi1 ცვლადი არ არის გამოცხადებული
}
```

როგორც მაგალითიდან ჩანს, გამოცხადებული იყო ცვლადი სახელით - ricxcvi1, ხოლო გამოიყენება სახელი - Ricxvi1. C# ენისთვის ეს ორი სხვადასხვა სახელია.

C# პროგრამის ფაილებს აქვთ .cs გაფართოება (ტიპი). პროგრამის კოდის ნაწილი ავტომატურად იქმნება Visual C#.NET სისტემის მიერ.

კომენტარები

C# ენაში არსებობს ერთსტრიქონიანი და მრავალსტრიქონიანი კომენტარი. ერთსტრიქონიანი კომენტარი // სიმბოლოებით იწყება. მრავალსტრიქონიანი კომენტარი /* სიმბოლოებით იწყება და */ სიმბოლოებით მთავრდება. როგორც წესი, კომენტარები შეიცავს პროგრამის სხვადასხვა ნაწილების განმარტებას, ცვლადებისა და მეთოდების დანიშნულებას და ა.შ. კომენტარების გამოყენება აადვილებს პროგრამის გარჩევას. პროგრამის შესრულების დროს ხდება კომენტარების გამოტოვება. ქვემოთ მოყვანილია პროგრამა, რომელიც ორივე სახის კომენტარს შეიცავს.

```
{
//      პროგრამა 2.4                    ეს არის ერთსტრიქონიანი კომენტარი
/*
ეს არის ჩვენი                          ეს არის მრავალსტრიქონიანი კომენტარი
პირველი პროგრამა
*/
int ricxvi1, ricxvi2, jami;

ricxvi1 = 2;
ricxvi2 = 3;
jami = ricxvi1 + ricxvi2;
}
```

C# ენის საბაზო ტიპები

მონაცემთა ჩვეულებრივი ტიპები

C# ენაში არსებობს ტიპების ორი ძირითადი კატეგორია: ჩვეულებრივი ტიპები (მარტივი ტიპები) და მიმართვითი ტიპები. ჩვეულებრივი ტიპის ცვლადები შეიცავენ კონკრეტულ მნიშვნელობებს, მიმართვითი ტიპის ცვლადები კი - ობიექტების მისამართებს. მიმართვით ტიპებს კლასების შესწავლისას განვიხილავთ. ჩვეულებრივი ტიპები მოყვანილია ცხრილში 2.1.

ცხრილი 2.1. მონაცემთა ჩვეულებრივი ტიპები

ტიპი	აღწერა	ბიტების რაოდენობა	მნიშვნელობების დიაპაზონი
bool	მნიშვნელობა true/false (ჭეშმარიტი/მცდარი)		
byte	8 ბიტიანი უნიშნო მთელი რიცხვი	8	0 - 255
char	სიმბოლო	16	0 - 65535
decimal	ციფრული ტიპი საფინანსო გამოთვლებისათვის	128	1E-28 - 7.9E+28
double	ორმაგი სიზუსტის რიცხვი მცურავი წერტილით	64	5E-324 - 1.7E+308
float	ერთმაგი სიზუსტის რიცხვი მცურავი წერტილით	32	1.5E-45 - 3.4E+38
int	მთელი რიცხვი	32	-2147483648 - 2147483647
long	გრძელი მთელი რიცხვი	64	-9223372036854775808 - 9223372036854775807
sbyte	8 ბიტიანი ნიშნიანი მთელი რიცხვი	8	-128 - 127
short	ნიშნიანი მოკლე მთელი რიცხვი	16	-32768 - 32767
uint	უნიშნო მთელი რიცხვი	32	0 - 4294967295
ulong	უნიშნო გრძელი მთელი რიცხვი	64	0 - 18446774073709551615
ushort	უნიშნო მოკლე მთელი რიცხვი	16	0 - 65535

მთელრიცხვა ტიპები

მთელრიცხვა ტიპებია: char, byte, sbyte, short, ushort, int, uint, long და ulong. აქედან sbyte, short, int და long ტიპები გამოიყენება ნიშნიანი მთელი რიცხვების აღსაწერად, ხოლო byte, uint, ulong და ushort კი უნიშნო მთელი რიცხვების წარმოსადგენად. განსხვავება ნიშნიან და უნიშნო მთელ რიცხვებს შორის შემდეგში მდგომარეობს. ნიშნიან რიცხვებში უფროსი ბიტი (ორობითი თანრიგი) გამოიყენება ნიშნის მისათითებლად. თუ ეს ბიტი ნულის ტოლია, მაშინ რიცხვი დადებითია, თუ ერთის ტოლია, მაშინ - უარყოფითი. უნიშნო რიცხვებში უფროსი ბიტი არ გამოიყენება ნიშნის წარმოსადგენად და შედის რიცხვის შემადგენლობაში. ამიტომ, უნიშნო რიცხვების აბსოლუტური მნიშვნელობა ორჯერ აღემატება ნიშნიანი რიცხვის მნიშვნელობას. ამის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

```
{
//      პროგრამა 2.5
//      პროგრამაში ხდება ნიშნიან და უნიშნო რიცხვებთან მუშაობის დემონსტრირება
sbyte ricxvi1;                  //      ricxvi1 იცვლება დიაპაზონში -128 ÷ 127
byte ricxvi2;                   //      ricxvi2 იცვლება დიაპაზონში 0 - 255
```

```
ricxvi1 = Convert.ToSByte(textBox1.Text);
ricxvi2 = 255;
label1.Text = ricxvi1.ToString();
label2.Text = ricxvi2.ToString();
}
```

მცურავწერტილიანი ტიპები

ორი ასეთი ტიპი არსებობს: float და double. ისინი გამოიყენება წილადების წარმოსადგენად. ამასთან, double ტიპი უფრო ხშირად გამოიყენება. ამის ერთ-ერთი მიზეზი ის არის, რომ მათემატიკური მეთოდების უმრავლესობა ამ ტიპს იყენებს. მაგალითად, ისეთ მეთოდებს, როგორიცაა Sqrt(), Sin(), Cos() და ა.შ. აქვთ double ტიპის არგუმენტები და გასცემენ ამავე ტიპის შედეგს. ქვემოთ მოყვანილ პროგრამაში ხდება წილად რიცხვებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 2.6
//      პროგრამაში ხდება წილადებთან მუშაობის დემონსტრირება
double wiladi1, wiladi2, shedegi;

wiladi1 = 5.5;
wiladi2 = Convert.ToDouble(textBox1.Text);
shedegi = wiladi1 + wiladi2;
label1.Text = shedegi.ToString();
}
```

decimal ტიპი

C++ და Java ენებისაგან განსხვავებით C# ენაში შემოტანილია decimal ტიპი. ამ ტიპის ცვლადები გამოიყენება ფინანსური გამოთვლების შესრულების დროს. ქვემოთ მოყვანილია პროგრამა, რომელიც ახდენს მოგებიდან პროცენტის გამოთვლას.

```
{
//      პროგრამა 2.7
//      მოგებიდან პროცენტის გამოთვლა
decimal mogeba, procenti, shedegi;

mogeba   = Convert.ToDecimal(textBox1.Text);
procenti = 10.0m;
shedegi  = mogeba * procenti / 100.0m;
label2.Text = shedegi.ToString();
}
```

პროგრამის მეშვიდე და მერვე სტრიქონებში გამოყენებულია m სუფიქსი, რომელიც თან ერთვის decimal ტიპის ლიტერალს (მუდმივას). ToDecimal() მეთოდი textBox1 კომპონენტში შეტანილ მონაცემს გარდაქმნის საფინანსო ტიპის მონაცემად, რომელიც mogeba ცვლადს ენიჭება. პროგრამის უკანასკნელ სტრიქონში შედეგი label2 კომპონენტში გამოგვაქვს.

char ტიპი (სიმბოლო)

როგორც ცნობილია დაპროგრამების ენებში სიმბოლოების წარმოსადგენად გამოიყენება 8 ბიტი. C# ენაში სიმბოლოს წარმოსადგენად გამოიყენება Unicode სიმბოლოების ნაკრები, რომლის თითოეული სიმბოლო 16 ბიტს იკავებს. ქვემოთ მოყვანილია პროგრამა, რომელშიც ხდება სიმბოლოებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 2.8
//      პროგრამაში ხდება სიმბოლოებთან მუშაობის დემონსტრირება
char simbolo1, simbolo2, simbolo3;

simbolo1 = 'რ';
simbolo2 = textBox1.Text[0];
//      textBox1 კომპონენტში უნდა შევიტანოთ ერთი სიმბოლო
simbolo3 = Convert.ToChar(textBox2.Text);
label2.Text = simbolo1.ToString();
label2.Text = simbolo2.ToString();
label3.Text = simbolo3.ToString();
}
```

პროგრამის მეხუთე სტრიქონში simbolo1 ცვლადს ენიჭება 'რ' სიმბოლო. მეექვსე სტრიქონში simbolo2 ცვლადს ენიჭება textBox1 კომპონენტში შეტანილი სტრიქონის პირველი სიმბოლო. მეშვიდე სტრიქონში simbolo3 ცვლადს ენიჭება textBox2 კომპონენტში შეტანილი სიმბოლო. ბოლოს ხდება ამ სიმბოლოების გამოტანა label კომპონენტებში.

char ტიპის ცვლადებს შეგვიძლია, აგრეთვე, მივანიჭოთ მმართველი სიმბოლოები. მმართველი სიმბოლო ეს არის სიმბოლო, რომელიც გამოიყენება გარკვეული მოქმედებების შესასრულებლად. ეს მოქმედებებია: ახალ სტრიქონზე გადასვლა, ჰორიზონტალური ტაბულაცია, სტრიქონის დასაწყისში გადასვლა და ა.შ. ცხრილში 2.2 მოყვანილია მმართველი სიმბოლოები.

ცხრილი 2.2. მმართველი სიმბოლოები.

მმართველი სიმბოლო	აღწერა
\’	ერთმაგი ბრჭყალი
\”	ორმაგი ბრჭყალი
\\	ირიბი დახრილი ხაზი
\0	Null (სიმბოლო, რომლის ნომერია 0)
\a	ზარი
\b	უკან დაბრუნება (BackSpace)
\f	გვერდის გადაფურცვლა
\n	ახალ სტრიქონზე გადასვლა
\r	სტრიქონის დასაწყისში გადასვლა
\t	ჰორიზონტალური ტაბულაცია
\v	ვერტიკალური ტაბულაცია

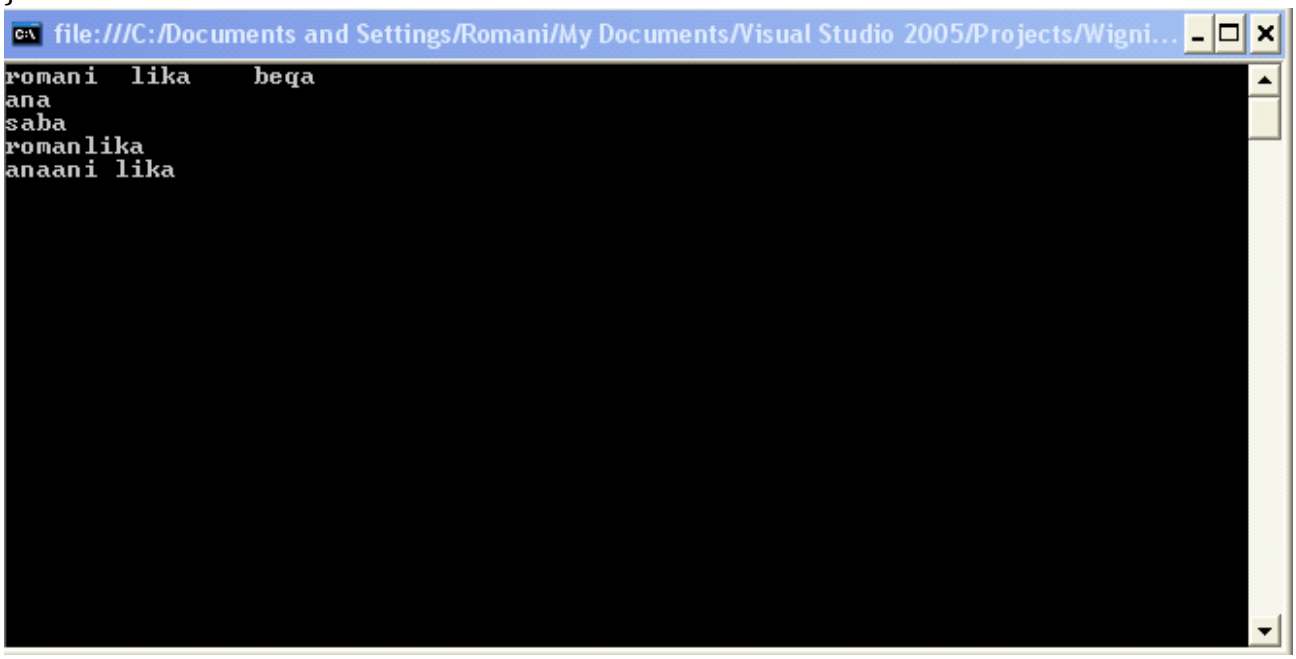
მაგალითი.

```
label1.Text = "საბა \n ანა";
```

ამ სტრიქონის შესრულების შედეგად label1 კომპონენტის პირველ სტრიქონში გამოჩნდება “საბა”, მეორე სტრიქონში კი - “ანა”.

მმართველი სიმბოლოების გამოყენების შედეგები უფრო მკაფიოდ ჩანს კონსოლურ პროგრამა-დანართებთან მუშაობის დროს. კონსოლური პროგრამის შესაქმნელად ნახ. 2.2-ზე ნაჩვენები ფანჯრის Templates ზონაში უნდა მოვნიშნოთ Console Application ელემენტი, Name: ველში შევიტანოთ პროგრამის სახელი, კატალოგი ავირჩიოთ Browse კლავიშის საშუალებით და დავაჭიროთ OK კლავიშს. გახსნილ ფანჯარაში შეგვაქვს ჩვენი პროგრამის კოდი ისე, როგორც ეს ქვემოთაა ნაჩვენები.

```
//      პროგრამა 2.9
//      პროგრამაში ხდება მმართველ სიმბოლოებთან მუშაობის დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
namespace Char_Console_2
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("რომანი\tლიკა\tბექა");
         Console.WriteLine("ანა\nსაბა\a");
         Console.WriteLine("რომანი\bლიკა");
         Console.WriteLine("რომანი ლიკა\rანა");
         Thread.Sleep(5000);
      }
   }
}
```



ნახ. 2.

პროგრამაში Thread.Sleep(5000); მეთოდი ახდენს ეკრანზე შედეგების დაყოვნებას 5 წამის განმავლობაში. როგორც პროგრამის შესრულების შედეგებიდან ჩანს (ნახ. 2.) Console.WriteLine("რომანი\tლიკა\tბექა"); მეთოდის შესრულების შედეგად შესრულდება გამოსატანი სტრიქონის ჰორიზონტალური ტაბულაცია. "რომანი" სტრიქონი გამოჩნდება პირველი პოზიციიდან. შემდეგ სრულდება ჰორიზონტალური ტაბულაცია. შედეგად მოხდება პირველი რვა პოზიციის გამოტოვება და მომდევნო "ლიკა" სტრიქონი გამოჩნდება მე-9 პოზიციიდან. შემდეგ ისევ შესრულდება ჰორიზონტალური ტაბულაცია, ანუ გამოიტოვება მომდევნო რვა პოზიცია და "ბექა" სტრიქონი გამოჩნდება მე-17 პოზიციიდან. ბოლოს WriteLine() მეთოდი ასრულებს მომდევნო, მეორე სტრიქონზე გადასვლას. არსებობს, აგრეთვე, Write() მეთოდი, რომელიც სტრიქონის გამოტანის შემდეგ არ ახდენს მომდევნო სტრიქონზე გადასვლას.

Console.WriteLine("ანა\nსაბა\a"); მეთოდის შესრულების შედეგად ეკრანის მეორე სტრიქონში გამოჩნდება "ანა" სტრიქონი. შემდეგ მოხდება მომდევნო, მესამე სტრიქონზე გადასვლა, რადგან ამ სტრიქონს მოსდევს მომდევნო სტრიქონზე გადასვლის სიმბოლო. მესამე სტრიქონში გამოჩნდება "საბა" სტრიქონი და გავიგებთ კომპიუტერის დინამიკის ხმას.

Console.WriteLine("რომანი\bლიკა"); მეთოდის შესრულების შედეგად "რომანი" სტრიქონის უკანასკნელი სიმბოლო წაიშლება და შემდეგ გამოჩნდება “ლიკა” სტრიქონი.

Console.WriteLine("რომანი ლიკა\rანა"); მეთოდის შესრულების შედეგად ჯერ გამოჩნდება "რომანი ლიკა" სტრიქონი, შემდეგ მოხდება ამავე სტრიქონის დასაწყისში გადასვლა და "ანა" სტრიქონის გამოტანა. შედეგად მიიღება "ანაანი ლიკა" სტრიქონი.

bool ტიპი

ბულის ტიპის ცვლადები იღებენ ორ მნიშვნელობას true (ჭეშმარიტი) და false (მცდარი). ქვემოთ მოყვანილია პროგრამა, რომელშიც ცვლადებს ენიჭებათ true და false მნიშვნელობები.

```
{
//      პროგრამა 2.10
//      პროგრამაში ხდება ლოგიკურ ცვლადებთან მუშაობის დემონსტრირება
bool b1, b2;

b1 = true;
b2 = Convert.ToBoolean(textBox1.Text);
label1.Text = b1.ToString();
label2.Text = b2.ToString();
}
```

ToBoolean მეთოდი textBox1 კომპონენტში შეტანილ სტრიქონს გარდაქმნის ლოგიკურ მონაცემად. ამ შემთხვევაში textBox1 კომპონენტში უნდა შევიტანოთ true ან false.

ლიტერალები

ლიტერალი (მუდმივა) არის ფიქსირებული მნიშვნელობა. მაგალითად, რიცხვი 50 არის ლიტერალი. ლიტერალი შეიძლება იყოს ნებისმიერი ტიპის მნიშვნელობა, მაგალითად, მთელი რიცხვები 5, -10, მცურავწერტილიანი რიცხვები 23.54, 1.09, სიმბოლოები 'Q', 'რ', სტრიქონი “C# თანამედროვე ენაა” და ა.შ.

ლიტერალების გამოყენებისას უნდა დავიცვათ შემდეგი წესები. მთელრიცხვა ლიტერალის ტიპი შეიძლება იყოს int, uint, long ან ulong. ეს დამოკიდებულია რიცხვის სიდიდეზე. შეგვიძლია ტიპის ცხადად მითითებაც ლიტერალისთვის სუფიქსის დამატების გზით. მაგალითად, 10L (ან 10l) ნიშნავს, რომ ამ ლიტერალის ტიპია long. უნიშნო მთელრიცხვა ლიტერალისთვის (ტიპი uint) გამოიყენება u ან U სუფიქსი, მაგალითად, 20U. გრძელი უნიშნო მთელრიცხვა ლიტერალის განსაზღვრისათვის ვიყენებთ UL ან ul სუფიქსებს, მაგალითად, 40UL. float ტიპის ლიტერალს ერთვის f ან F სუფიქსი, მაგალითად, 35.41F. decimal ტიპის ლიტერალს ერთვის m ან M სუფიქსი, მაგალითად, 8.75M.

ქვემოთ მოყვანილია პროგრამა, რომელშიც ხდება ლიტერალებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 2.11
//      პროგრამაში ხდება ლიტერალებთან მუშაობის დემონსტრირება
uint u1;
long l1;
float f1;
decimal d1;
char c1;
string s1;

u1 = 150U; label2.Text = u1.ToString();
l1 = 100L; label3.Text = l1.ToString();
f1 = 150F; label4.Text = f1.ToString();
d1 = 250M; label5.Text = d1.ToString();
c1 = 'B';  label6.Text = c1.ToString();
s1 = "C# თანამედროვე ენაა"; label7.Text = s1;
}
```

ცვლადები და მათი ინიციალიზება

ცვლადი არის მეხსიერების სახელდებული უბანი, რომელსაც მნიშვნელობა ენიჭება. ეს მნიშვნელობა შეიძლება შეიცვალოს პროგრამის მუშაობის პროცესში. ოპერატორს, რომლის საშუალებითაც ხდება ცვლადების გამოცხადება, შემდეგი სინტაქსი აქვს:

ტიპი ცვლადის_სახელი;

სადაც, ტიპი ცვლადის ტიპია, ცვლადის_სახელი - კი მისი სახელი. ნებისმიერი ცვლადი გამოცხადებული უნდა იყოს მის გამოყენებამდე, წინააღმდეგ შემთხვევაში ადგილი ექნება შეცდომას. ამასთან, ცვლადს უნდა მიენიჭოს მხოლოდ შესაბამისი ტიპის მნიშვნელობები. მაგალითად, bool ტიპის ცვლადს უნდა მიენიჭოს true ან false მნიშვნელობა და არა წილადი ან სხვა.

ცვლადის ინიციალიზება

ცვლადისთვის მნიშვნელობის მინიჭებას მისი გამოცხადების დროს ცვლადის

ინიციალიზება ეწოდება. ცვლადების ინიციალიზების ოპერატორის სინტაქსია:
ტიპი ცვლადის_სახელი = მნიშვნელობა;
მნიშვნელობის ტიპი უნდა ემთხვეოდეს ცვლადის ტიპს. ცვლადს მნიშვნელობა უნდა მივანიჭოთ მის გამოყენებამდე. ეს შეიძლება გავაკეთოთ ან ცვლადის გამოცხადებისას ან გამოცხადების შემდეგ მინიჭების ოპერატორის გამოყენებით. მოყვანილი პროგრამა ახდენს ცვლადების ინიციალიზაციის დემონსტრირებას.

```
{
//      პროგრამა 2.12
//      პროგრამაში ხდება ცვლადების ინიციალიზაციის დემონსტრირება
int i3, i1 = 25, i2 = 30;
char c1 = 'A';
double d1 = 150.75;
label2.Text = i1.ToString() + "   " + d1.ToString() + "   " + c1;
}
```

ცვლადების დინამიური ინიციალიზება

ცვლადების ინიციალიზება შესაძლებელია, აგრეთვე, დინამიურად, ე.ი. პროგრამის შესრულების დროს. ამ შემთხვევაში ცვლადის ინიციალიზება სრულდება არა პროგრამის დასაწყისში, არამედ მის ნებისმიერ ადგილში. მოყვანილი პროგრამა ახდენს ამის დემონსტრირებას.

```
{
//      პროგრამა 2.13
//      პროგრამაში ხდება ცვლადების დინამიური ინიციალიზების დემონსტრირება
int simagle = 5, sigane, sigrdze;
sigane  = Convert.ToInt32(textBox1.Text);
sigrdze = Convert.ToInt32(textBox2.Text);
// moculoba ცვლადის ინიციალიზება  ხდება დინამიურად
int moculoba = simagle * sigane * sigrdze;
label1.Text = moculoba.ToString();
}
```

ოპერატორები

ოპერატორი არის სიმბოლო, რომელიც კომპილატორს ატყობინებს თუ რომელი სპეციფიკური ოპერაცია უნდა შესრულდეს. C# ენაში არსებობს ოპერატორების ოთხი ძირითადი კლასი: არითმეტიკის, ლოგიკის, შედარებისა და ბიტობრივი.

არითმეტიკის ოპერატორები

არითმეტიკის ოპერატორები შეგვიძლია გამოვიყენოთ როგორც მთელი, ისე წილადი რიცხვების მიმართ. ისინი მოყვანილია ცხრილში 2.2.

მოვიყვანოთ ზოგიერთი განმარტება. როცა გაყოფის ოპერატორს ვიყენებთ მთელი რიცხვების მიმართ, მაშინ შედეგი იქნება მთელი რიცხვი, ნაშთი კი უარიყოფა. მაგალითად, 20/6 გაყოფის შედეგი იქნება მთელი რიცხვი 3. ნაშთის მისაღებად უნდა გამოვიყენოთ % ოპერატორი. მაგალითად, 20%6 ოპერაციის შედეგი იქნება 2. %

ოპერატორის გამოყენებით შეგვიძლია დავადგინოთ რიცხვი კენტია თუ ლუწი. მაგალითად, R%2 გამოსახულების გამოთვლისას თუ ნაშთი 1-ის ტოლია, მაშინ R რიცხვი კენტია, თუ ნაშთი 0-ის ტოლია, მაშინ R რიცხვი ლუწია. ანალოგიურად შეგვიძლია დავადგინოთ ერთი რიცხვი მეორის ჯერადია თუ არა.

ცხრილი 2.2. არითმეტიკის ოპერატორები

ოპერატორი	მნიშვნელობა
+	შეკრება
-	გამოკლება (და უნარული მინუსი)
*	გამრავლება
/	გაყოფა
%	მოდულით გაყოფა (ნაშთის გამოყოფა)
++	ინკრემენტი
--	დეკრემენტი

მოყვანილ პროგრამებში ხდება არითმეტიკის ოპერატორების გამოყენების დემონსტრირება მთელი რიცხვებისათვის.

```
{
//      პროგრამა 2.14
//      პროგრამაში ხდება არითმეტიკის ოპერაციების მუშაობის დემონსტრირება მთელი რიცხვებისათვის
int i1, i2, jami, sxvaoba, namravli, ganayofi, nashti;

i1 = Convert.ToInt32(textBox1.Text);
i2 = Convert.ToInt32(textBox2.Text);
jami     = i1 + i2;
sxvaoba  = i1 - i2;
namravli = i1 * i2;
ganayofi = i1 / i2;
nashti   = i1 % i2;
label1.Text = jami.ToString();
label2.Text = sxvaoba.ToString();
label3.Text = namravli.ToString();
label4.Text = ganayofi.ToString();
label5.Text = nashti.ToString();
}
```

ქვემოთ მოყვანილ პროგრამაში ხდება არითმეტიკის ოპერატორების მუშაობის დემონსტრირება წილადი რიცხვებისათვის.

```
{
//      პროგრამა 2.15
//      პროგრამაში ხდება არითმეტიკის ოპერატორების მუშაობის დემონსტრირება წილადი რიცხვებისათვის
double d1, d2, jami, sxvaoba, namravli, ganayofi, nashti;

d1 = Convert.ToDouble(textBox1.Text);
```

```
d2 = Convert.ToDouble(textBox2.Text);
jami     = d1 + d2;
sxvaoba  = d1 - d2;
namravli = d1 * d2;
ganayofi = d1 / d2;
nashti   = d1 % d2;
label1.Text = jami.ToString();
label2.Text = sxvaoba.ToString();
label3.Text = namravli.ToString();
label4.Text = ganayofi.ToString();
label5.Text = nashti.ToString();
}
```

ინკრემენტისა და დეკრემენტის ოპერატორები

ინკრემენტის ოპერატორი (++) ერთით ზრდის თავისი ოპერანდის (არგუმენტის) მნიშვნელობას. ოპერანდი არის ცვლადი. დეკრემენტის ოპერატორი (--) ერთით ამცირებს თავისი ოპერანდის მნიშვნელობას. მაგალითად, ოპერატორი

```
x = x + 1;
```

ასრულებს ისეთივე მოქმედებებს, როგორსაც ოპერატორი

```
x++; ან ++x;
```

ანალოგიურად, ოპერატორი

```
x = x - 1;
```

ასრულებს იგივე მოქმედებებს, რასაც ოპერატორი

```
x--; ან --x;
```

როგორც ვხედავთ ეს ოპერატორები შეიძლება მიეთითოს როგორც ოპერანდამდე (პრეფიქსი), ისე ოპერანდის შემდეგ (პოსტფიქსი). ახლა ვნახოთ რაში მდგომარეობს მათ შორის განსხვავება. თუ ინკრემენტის ან დეკრემენტის ოპერატორი მითითებულია ოპერანდის წინ, მაშინ ჯერ მოხდება ოპერანდის მნიშვნელობის გაზრდა ან შემცირება, ხოლო შემდეგ მისი გამოყენება გამოსახულებაში. თუ ინკრემენტის ან დეკრემენტის ოპერატორი მითითებულია ოპერანდის შემდეგ, მაშინ ჯერ მოხდება ამ ოპერანდის გამოყენება გამოსახულებაში და შემდეგ მისი მნიშვნელობის გაზრდა ან შემცირება.

ქვემოთ მოყვანილ პროგრამაში ხდება ინკრემენტის ოპერატორის გამოყენების დემონსტრირება.

```
{
//      პროგრამა  2.16
//      პროგრამაში ხდება ++ ოპერატორის მუშაობის დემონსტრირება
int incr, shedegi;

incr = 5;
shedegi = ++incr;                                   // incr    = 6, shedegi = 6
label1.Text = incr.ToString();
label2.Text = shedegi.ToString();
incr = 5;
shedegi = incr++;                                   // shedegi = 5, incr    = 6
label3.Text = incr.ToString();
label4.Text = shedegi.ToString();
```

}

თავდაპირველად incr ცვლადს ენიჭება მნიშვნელობა 5. მომდევნო სტრიქონში ამ ცვლადის მარცხნივ მოთავსებულია ++, ამიტომ მისი მნიშვნელობა ჯერ გაიზრდება ერთით და გახდება 6-ის ტოლი, შემდეგ კი მიენიჭება shedegi ცვლადს. შემდეგ, incr ცვლადს კვლავ ენიჭება მნიშვნელობა 5. მომდევნო სტრიქონში ++ მოთავსებულია incr ცვლადის მარჯვნივ. ამიტომ მისი მნიშვნელობა ჯერ მიენიჭება shedegi ცვლადს, შემდეგ კი გაიზრდება ერთით და გახდება 6-ის ტოლი.

ქვემოთ მოყვანილ პროგრამაში ხდება დეკრემენტის ოპერატორის გამოყენების დემონსტრირება.

```
{
//      პროგრამა  2.17
//      პროგრამაში ხდება -- ოპერატორის მუშაობის დემონსტრირება
int decr, shedegi;

decr = 5;
shedegi = --decr;                                   // decr    = 4, shedegi = 4
label1.Text = decr.ToString();
label2.Text = shedegi.ToString();
decr = 5;
shedegi = decr--;                                   // shedegi = 5, decr    = 4
label3.Text = decr.ToString();
label4.Text = shedegi.ToString();
}
```

შედარებისა და ლოგიკის ოპერატორები

შედარების ოპერატორები ადარებენ ორ მნიშვნელობას და გასცემენ bool ტიპის მნიშვნელობას true ან false. ისინი მოყვანილია ცხრილში 2.3. ლოგიკის ოპერატორები ასრულებენ ლოგიკურ ოპერაციებს bool ტიპის მნიშვნელობებზე. ისინი მოყვანილია ცხრილში 2.4.

C# ენაში == და != ოპერატორები შეიძლება გამოვიყენოთ ყველა ობიექტის მიმართ მათი შედარებისათვის ტოლობაზე ან უტოლობაზე. შედარების დანარჩენი ოპერატორები შეგვიძლია გამოვიყენოთ მხოლოდ მონაცემების ჩამოთვლადი ტიპების მიმართ, რომლებიც მოწესრიგებულია თავიანთ სტრუქტურაში. ასეთია მაგალითად, რიცხვების მოწესრიგებული სტრუქტურა 1, 2, 3 და ა.შ., ან ანბანის მიხედვით დალაგებული სიმბოლოები. შედეგად, შედარების ყველა ოპერატორი შეგვიძლია გამოვიყენოთ მონაცემთა ყველა რიცხვითი ტიპის მიმართ. bool ტიპის მნიშვნელობების შედარება შეიძლება მხოლოდ ტოლობაზე ან უტოლობაზე.

ცხრილში 2.5 მოყვანილია ლოგიკის ოპერატორების ჭეშმარიტების ცხრილი. როგორც ცხრილიდან ჩანს & ოპერატორი გასცემს true მნიშვნელობას მხოლოდ მაშინ, როცა მისი ორივე ოპერანდის მნიშვნელობაა true. წინააღმდეგ შემთხვევაში, იგი გასცემს false მნიშვნელობას. | ოპერატორი გასცემს true მნიშვნელობას, თუ მისი ერთ-ერთი ოპერანდის მნიშვნელობაა true. წინააღმდეგ შემთხვევაში, იგი გასცემს false მნიშვნელობას. ! ოპერატორი გასცემს თავისი ოპერანდის ლოგიკურად საწინააღმდეგო მნიშვნელობას. ^ ოპერატორი გასცემს false მნიშვნელობას თუ მის ორივე ოპერანდს ერთნაირი მნიშვნელობა აქვს, წინააღმდეგ შემთხვევაში გასცემს true მნიშვნელობას.

ცხრილი 2.3. შედარების ოპერატორები

ოპერატორი	მნიშვნელობა
==	ტოლია
!=	არ არის ტოლი
>	მეტია
<	ნაკლებია
>=	მეტია ან ტოლი
<=	ნაკლებია ან ტოლი

ცხრილი 2.4. ლოგიკის ოპერატორები

ოპერატორი	მნიშვნელობა
&	AND (და)
|	OR (ან)
^	XOR (გამომრიცხავი ან)
&&	Short-circuit AND (სწრაფი და ოპერატორი)
||	Short-circuit OR (სწრაფი ან ოპერატორი)
!	NOT (არა)

ცხრილი 2.5. ლოგიკის ოპერატორების ჭეშმარიტების ცხრილი

B1	B2	B1 & B2	B1 | B2	B1 ^ B2	!B1
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

მოყვანილ პროგრამაში ხდება ლოგიკის ოპერატორების მუშაობის დემონსტრირება.

```
{
//      პროგრამა  2.18
//      პროგრამაში ხდება ლოგიკის ოპერატორების მუშაობის დემონსტრირება
bool b1, b2, b3, b4, b5, b6;

b1 = true;b2 = false;
b3 = b1 & b2;                   //      b3 = false
b4 = b1 | b2;                   //      b4 = true
b5 = b1 ^ b2;                   //      b5 = true
b6 = !b1;                       //      b6 = false
label1.Text = b3.ToString();
label2.Text = b4.ToString();
label3.Text = b5.ToString();
```

```
label4.Text = b6.ToString();
}
```

მოყვანილ პროგრამაში ხდება შედარების ოპერატორების მუშაობის დემონსტრირება.

```
{
//      პროგრამა  2.19
//      პროგრამაში ხდება შედარების ოპერატორების მუშაობის დემონსტრირება
int ricxvi1 = 5, ricxvi2 = 10;
bool shedegi;

shedegi = ricxvi1 > 0;                          //      shedegi = true
label10.Text = shedegi.ToString();
shedegi = ( ricxvi1 < 2 ) && ( ricxvi2 > 7 );   //      shedegi = false
label11.Text = shedegi.ToString();
shedegi = ( ricxvi1 == 3 ) || ( ricxvi2 != 5 ); //      shedegi = true
label12.Text = shedegi.ToString();
}
```

არსებობს, აგრეთვე, ლოგიკის სწრაფი ოპერატორები && და ||. მათი გამოყენებისას პროგრამა უფრო სწრაფად სრულდება. თუ && ოპერატორის შესრულებისას პირველმა ოპერანდმა მიიღო false მნიშვნელობა, მაშინ შედეგი იქნება false მიუხედავად მეორე ოპერანდის მნიშვნელობისა. თუ || ოპერატორის შესრულებისას პირველმა ოპერანდმა მიიღო true მნიშვნელობა, მაშინ შედეგი იქნება true მიუხედავად მეორე ოპერატორის მნიშვნელობისა. ასეთ შემთხვევებში აღარ არის საჭირო მეორე ოპერანდის შეფასება, რადგან მისი მნიშვნელობა საბოლოო შედეგს ვერ შეცვლის, და შედეგად პროგრამა უფრო სწრაფად სრულდება.

მინიჭების ოპერატორი

მისი სინტაქსია:

```
ცვლადი = გამოსახულება;
```

ცვლადს და გამოსახულებას ერთნაირი ტიპი უნდა ჰქონდეს. მინიჭების ოპერატორი ცვლადს ანიჭებს გამოსახულების მნიშვნელობას.

მინიჭების ოპერატორი საშუალებას იძლევა, აგრეთვე, შევქმნათ მინიჭებების მიმდევრობა, მაგალითად,

```
int ricxvi1, ricxvi2, ricxvi3;
ricxvi1 = ricxvi2 = ricxvi3 = 50;
```

აქ სამივე ცვლადს ენიჭება რიცხვი 50. ასეთი მინიჭება საშუალებას გვაძლევს რამდენიმე ცვლადს ერთდროულად მივანიჭოთ ერთი მნიშვნელობა.

მინიჭების შედგენილი ოპერატორი

მინიჭების შედგენილ (შემოკლებულ) ოპერატორში არითმეტიკის ოპერატორები მოთავსებულია მინიჭების ოპერატორის მარცხნივ. მისი სინტაქსია:

```
ცვლადი ოპერატორი= გამოსახულება;
```

სადაც, ოპერატორი არის არითმეტიკის ან ლოგიკის ოპერატორი. მაგალითად,

გამოსახულება
jami = jami + 25;
შედგენილი ოპერატორის გამოყენებით შეგვიძლია ასე ჩავწეროთ
jami += 25;

+= ოპერატორების წყვილი მიუთითებს, რომ jami ცვლადს უნდა მიენიჭოს მნიშვნელობა jami + 25.

შედგენილი (შემოკლებული) ოპერატორებია:

+= -= *= /= %= &= |= ^=

მინიჭების შედგენილი ოპერატორები მინიჭების ჩვეულებრივ ოპერატორთან შედარებით უფრო კომპაქტურია. ეს განსაკუთრებით იგრძნობა მაშინ, როცა გრძელ სახელებს ვიყენებთ. ამ შემთხვევაში სახელის შეტანა ერთხელ გვიწევს. მეორეც, მათი გამოყენება აჩქარებს კოდის კომპილაციას, რადგან ოპერანდის შეფასება მხოლოდ ერთხელ ხდება.

ოპერატორების პრიორიტეტები

ცხრილში 2.6 ნაჩვენებია C# ენის ყველა ოპერატორის პრიორიტეტი. ოპერატორები დალაგებულია პრიორიტეტების კლების მიხედვით. ცხრილის ერთ სტრიქონში ოპერატორებს თანაბარი პრიორიტეტები აქვთ.

როგორც ცხრილიდან ჩანს, *, / და % ოპერატორებს უფრო მაღალი პრიორიტეტი აქვთ, ვიდრე + და - . მოვიყვანოთ მაგალითები, რომლებშიც ხდება ამ ოპერატორების პრიორიტეტების დემონსტრირება. განვიხილოთ გამოსახულება:

y = x1 + x2 * x3 - x4 / x5 ;

იგი შემდეგნაირად გამოითვლება. გამოსახულებაში პირველია + ოპერატორი. სანამ ეს ოპერატორი შესრულდება კომპილატორი ამოწმებს მის მარჯვნივ რომელი ოპერატორია მოთავსებული. ესაა *, რომელსაც უფრო მაღალი პრიორიტეტი აქვს, ვიდრე + ოპერატორს. შემდეგ, კომპილატორი ამოწმებს * ოპერატორის მარჯვნივ რომელი ოპერატორია მოთავსებული. ესაა -, რომელსაც უფრო დაბალი პრიორიტეტი აქვს, ვიდრე * ოპერატორს. ამიტომ პირველ რიგში შესრულდება * ოპერატორი. რადგან + და - ოპერატორებს თანაბარი პრიორიტეტი აქვთ და + რიგით პირველია გამოსახულებაში, ამიტომ შემდეგ შესრულდება + ოპერატორი. სანამ კომპილატორი შეასრულებს - ოპერატორს, იგი ამოწმებს მის მარჯვნივ რომელი ოპერატორია მოთავსებული. ესაა / ოპერატორი, რომელსაც უფრო მაღალი პრიორიტეტი აქვს ვიდრე - ოპერატორს. ამიტომ, ჯერ შესრულდება / ოპერატორი, შემდეგ კი - ოპერატორი.

ახლა განვიხილოთ მეორე გამოსახულება:

y = (x1 + x2) * x3 - (x4 + x5) / x6 ;

აქ ჯერ შესრულდება პირველ მრგვალ ფრჩხილებში მოთავსებული გამოსახულება. შემდეგ, * ოპერატორი. შემდეგ, მეორე მრგვალ ფრჩხილებში მოთავსებული გამოსახულება. შემდეგ, / ოპერატორი და ბოლოს, - ოპერატორი.

როგორც ვხედავთ, პრიორიტეტების ცოდნა საჭიროა იმისათვის, რომ სწორად ჩავწეროთ გამოსახულება და შესაბამისად, მივიღოთ სწორი შედეგი.

გამოსახულებები. მათემატიკური მეთოდები

გამოსახულება არის ოპერატორების, ცვლადებისა და მეთოდების კომბინაცია. მაგალითად,

```
y = x + z - 5;
y = x - Math.Sin(x) + 10.97;
```

პროგრამების კითხვადობის გაუმჯობესების მიზნით გამოსახულებებში ტაბულაციის სიმბოლოები და ინტერვალები გამოიყენება. მაგალითად, მოყვანილი ორი გამოსახულებიდან მეორე მათგანის წაკითხვა უფრო ადვილია ვიდრე პირველის:

```
y=x/5+z*(w-2);
y = x / 5 + z * ( w - 2 );
```

მრგვალი ფრჩხილები ზრდიან მასში მოთავსებული ოპერანდების პრიორიტეტს. ისინი ისევე გამოიყენება, როგორც ალგებრაში. მრგვალი ფრჩხილები გამოიყენება, აგრეთვე, პროგრამის კოთხვადობის გასაუმჯობესებლად. მაგალითად, მოყვანილი ორი სტრიქონიდან მეორე უკეთესად იკითხება:

```
y = x * 5 + z / 7.2 - w;
y = ( x * 5)  + ( z / 7.2 ) - w;
```

მათემატიკური მეთოდები აღწერილია System.Math სტანდარტულ კლასში. მათი უმრავლესობა double ტიპს იყენებს. მათ გამოსაძახებლად ჯერ ეთითება კლასის სახელი Math, შემდეგ წერტილი და მეთოდის სახელი. ისინი მოყვანილია ცხრილში 2.7.

მოკლედ განვიხილოთ ზოგიერთი მეთოდი. Pow(x, y) მეთოდს x რიცხვი აჰყავს y ხარისხში. მაგალითად, Pow(5, 2) მეთოდის შესრულების შედეგად გაიცემა 25. Exp(x) მეთოდს e რიცხვი აჰყავს x ხარისხში. მაგალითად, Exp(1) მეთოდის შესრულების შედეგად გაიცემა 2.71 . Max(x, y) მეთოდი გასცემს x და y რიცხვებს შორის უდიდესს, Min(x, y) მეთოდი კი - უმცირესს. Floor(x) მეთოდი x რიცხვს ამრგვალებს ნაკლებობით. მაგალითად, Floor(5.998) მეთოდის შესრულების შედეგად გაიცემა 5. Ceiling(x) მეთოდი x რიცხვს ამრგვალებს მეტობით. მაგალითად, Ceiling(5.123) მეთოდის შესრულების შედეგად გაიცემა 6. Round(x, y) მეთოდი გამოიყენება x რიცხვის ფორმატირებული გამოტანისათვის. y მიუთითებს ათწილადში თანრიგების რაოდენობას. მაგალითად, Round(5.1234, 2) მეთოდის შესრულების შედეგად გაიცემა ათწილადი 5.12 . Sign(x) მეთოდი გასცემს x რიცხვის ნიშანს. მაგალითად, Sign(-5) მეთოდის შესრულების შედეგად გაიცემა -1, ხოლო Sign(5) მეთოდის შესრულების შედეგად კი გაიცემა 1.

შევადგინოთ პროგრამა, რომელიც ახდენს კვადრატული ფესვის ამოღებას რიცხვიდან, რომელიც textBox1 კომპონენტში შეგვაქვს.

```
{
//      პროგრამა 2.20
//      კვადრატული ფესვის ამოღება
double ricxvi, shedegi;
ricxvi  = Convert.ToDouble(textBox1.Text);
shedegi = Math.Sqrt(ricxvi);
label1.Text = shedegi.ToString();
label2.Text = Math.Round(shedegi,2).ToString();
}
```

ცხრილი 2.6. ოპერატორების პრიორიტეტები

კატეგორია	ოპერატორები
პირველადი	დაჯგუფება: (x)
	ველთან მიმართვა: x.y
	მეთოდის გამოძახება: f(x)
	ინდექსირებული მიმართვა: a[x]
	პოსტფიქსური ინკრემენტი: x++
	პოსტფიქსური დეკრემენტი: x--
	ახალი ობიექტის შექმნა: new
	ტიპის მიღება: typeof
	ზომის მიღება: sizeof
	გადავსების შემოწმება: checked
	გადავსების არშემოწმება: unchecked
უნარული	დადებითი მნიშვნელობა: +
	უარყოფითი მნიშვნელობა: -
	ლოგიკური “არა”: !
	ბიტობრივი “არა”: ~
	პრეფიქსური ინკრემენტი: ++x
	პრეფიქსური დეკრემენტი: --x
	ტიპის დაყვანა: (type)
მულტიპლიკატიური	გამრავლება: *
	გაყოფა: /
	ნაშთის გამოყოფა: %
ადიტიური	შეკრება: +
	გამოკლება: -
ბიტობრივი ძვრა	ძვრა მარცხნივ: <<
	ძვრა მარჯვნივ: >>
მიმართებები	ნაკლებია: <
	მეტია: >
	ნაკლებია ან ტოლი: <=
	მეტია ან ტოლი: >=
	ტიპების თავსებადობა: is
	ტიპების თავსებადობა გარდაქმნით: as
ტოლობა	ტოლობა: =
	უტოლობა: !=
ბიტობრივი “და”	&
ბიტობრივი გამომრიცხავი “ან”	^
ბიტობრივი “ან”	|
ლოგიკური “და”	&&
ლოგიკური “ან”	||
ტერნარული ოპერატორი	?:
მინიჭებები	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=

ცხრილი 2.7. მათემატიკური მეთოდები

მეთოდი	დანიშნულება
Abs(x)	აბსოლუტური მნიშვნელობა
Sin(x)	სინუსი
Cos(x)	კოსინუსი
Tan(x)	ტანგენსი
Sqrt(x)	კვადრატული ფესვი
Exp(x)	ექსპონენტა
Log(x)	ლოგარითმი
Log10(x)	ათობითი ლოგარითმი
Max(x, y)	ორ რიცხვს შორის უდიდესი
Min(x, y)	ორ რიცხვს შორის უმცირესი
Pow(x, y)	ახარისხება
Floor(x)	დამრგვალება ნაკლებობით
Ceiling(x)	დამრგვალება მეტობით
Round(x, y)	ფორმატირებული გამოტანა
E	2,71
PI	3,14
Sign(x)	არგუმენტის ნიშანი

შევადგინოთ კიდევ ერთი პროგრამა, რომელიც გამოთვლის მოცემული გამოსახულების მნიშვნელობას:

$$y = \frac{\sqrt{x^5 + \sin x}}{1 - e^x}$$

```
{
//      პროგრამა 2.21
//      ფორმულის გამოთვლა
double ricxvi, shedegi;
ricxvi  = Convert.ToDouble(textBox1.Text);
shedegi =  Math.Sqrt( Math.Pow(ricxvi,5) + Math.Sin(ricxvi) ) /
                        ( 1 - Math.Exp(ricxvi) );
label1.Text = Math.Round(shedegi,2).ToString();
}
```

დანართში 1 მოყვანილია ამოცანები თავების მიხედვით.

ტიპების გარდაქმნა

დაპროგრამების დროს, როგორც წესი, ერთი ტიპის მქონე ცვლადს შეგვიძლია მივანიჭოთ მხოლოდ ამავე ტიპის მნიშვნელობა, ე.ი. მინიჭების ოპერატორის მარცხნივ მოთავსებულ ცვლადსა და მარჯვნივ მოთავსებულ გამოსახულებას ერთნაირი ტიპები უნდა ჰქონდეს. მაგრამ, რიგ შემთხვევებში, საჭირო ხდება ერთი ტიპის ცვლადისათვის სხვა ტიპის მნიშვნელობის მინიჭება. მაგალითად, double ტიპის ცვლადს შეგვიძლია

მივანიჭოთ int ტიპის ცვლადის მნიშვნელობა:

```
int mteli;
double wiladi;
mteli = 25;
wiladi = mteli;
```

თუ მინიჭების ოპერატორში გამოიყენება მონაცემთა თავსებადი ტიპები, მაშინ მინიჭების ოპერატორის მარჯვნივ მოთავსებული გამოსახულების ტიპი ავტომატურად გარდაიქმნება მარცხნივ მოთავსებული ცვლადის ტიპად. აქედან გამომდინარე, ჩვენს მაგალითში mteli ცვლადის ტიპი ჯერ გარდაიქმნება double ტიპად და შემდგომ მისი მნიშვნელობა მიენიჭება wiladi ცვლადს. რადგან C# ენაში ტიპების კონტროლი მკაცრად ხორციელდება, ამიტომ ყველა ტიპის გარდაქმნა არ არის დაშვებული. მაგალითად, არათავსებადია int და bool ტიპები.

ერთი ტიპის მქონე ცვლადისთვის სხვა ტიპის მქონე ცვლადის მნიშვნელობის მინიჭებისას ტიპის ავტომატურად გარდაქმნა შესრულდება იმ შემთხვევაში თუ:

- ორივე ტიპი თავსებადია;
- საბოლოო ტიპი (მარცხენა) მეტია საწყის ტიპზე (მარჯვენა). ეს იმას ნიშნავს, რომ საბოლოო ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობა მეტია საწყისი ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობაზე.

ამ პირობების შესრულებისას ხორციელდება გაფართოებადი გარდაქმნა. მაგალითად, int ტიპისთვის გამოყოფილი ბიტების (თანრიგების) რაოდენობა, ყოველთვის საკმარისია byte ტიპის მქონე მნიშვნელობების შესანახად, და რადგან ორივე მთელრიცხვა ტიპია, ამიტომ მათ მიმართ შესრულდება ტიპების ავტომატური გარდაქმნა.

გაფართოებადი გარდაქმნის შესასრულებლად ყველა რიცხვითი ტიპი მთელრიცხვა და მცურავწერტილიანი ტიპების ჩათვლით თავსებადი უნდა იყოს. მაგალითად, ქვემოთ მოყვანილ პროგრამაში სრულდება long ტიპის ცვლადის გარდაქმნა double ტიპის ცვლადად.

```
{
//      პროგრამა 2.22
//      პროგრამაში ხდება long ტიპის გარდაქმნა double ტიპად
long L1;
double D1;

L1 = 100321410L;
D1 = L1;
label1.Text = D1.ToString();
}
```

გარდაქმნა double ტიპიდან long ტიპად დაუშვებელია, რადგან ის არ არის გაფართოებადი. გარდა ამისა, ტიპების ავტომატური გარდაქმნა არ სრულდება decimal და float ცვლადებს შორის, აგრეთვე, decimal და double, char და bool, მთელი ტიპის ცვლადებსა და char, მთელი ტიპის ცვლადებსა და bool ტიპის ცვლადებს შორის.

ტიპების დაყვანის ოპერატორი (cast operator)

ტიპის დაყვანა - ესაა ერთი ტიპის მეორე ტიპად გარდაქმნის ოპერაცია. ის გამოიყენება მონაცემების შეუთავსები ტიპების მიმართ. ტიპების ასეთ გარდაქმნას

ცხადი (აშკარა) ეწოდება. ტიპების დაყვანის ოპერატორის სინტაქსია:
(საბოლოო_ტიპი) გამოსახულება;
აქ საბოლოო_ტიპი მიუთითებს თუ რომელ ტიპად უნდა გარდაიქმნას მითითებული გამოსახულების შედეგი. მოყვანილ პროგრამაში სრულდება ტიპების გარდაქმნა.

```
{
//      პროგრამა 2.23
//      პროგრამაში ხდება ტიპების გარდაქმნა შეუთავსებ ტიპებს შორის
double wiladi1, wiladi2;
byte baiti;
int  mteli;
char simbolo;

wiladi1 = Convert.ToDouble(textBox1.Text);
wiladi2 = Convert.ToDouble(textBox2.Text);
//      double ტიპის გამოსახულება გარდაიქმნება int ტიპად
mteli = ( int ) ( wiladi1 / wiladi2 );          //      აქ ხდება ინფორმაციის დაკარგვა
label1.Text = mteli.ToString();
// აქ ინფორმაცია არ იკარგება, რადგან 50 იმყოფება byte ტიპის დასაშვებ  დიაპაზონში
mteli = 50;
baiti = ( byte ) mteli;
label2.Text = baiti.ToString();
//      აქ ინფორმაცია იკარგება, რადგან 300 სცილდება byte ტიპის დასაშვებ დიაპაზონს
mteli = 300;
baiti = ( byte ) mteli;
label3.Text = baiti.ToString();

baiti = Convert.ToByte(textBox3.Text);
simbolo = ( char ) baiti;                   // ტიპების დაყვანის ოპერაცია შეუთავსებ ტიპებს შორის
label4.Text = baiti.ToString();
}
```

mteli = (int) (wiladi1 / wiladi2); სტრიქონში double ტიპის გამოსახულების შედეგი დაიყვანება int ტიპზე. მრგვალი ფრჩხილების მითითება, რომლებშიც მოთავსებულია wiladi1/wiladi2, აუცილებელია. წინააღმდეგ შემთხვევაში, int ტიპზე დაიყვანება მხოლოდ wiladi1 ცვლადი. ამ შემთხვევაში სრულდება ტიპების დაყვანის ოპერაცია, რადგან double ტიპის მნიშვნელობა int ტიპის მნიშვნელობად ავტომატურად არ გარდაიქმნება.

როცა ტიპების გარდაქმნის დროს სრულდება კუმშვადი გარდაქმნა, მაშინ ინფორმაცია შეიძლება დაიკარგოს. მაგალითად, თუ long ტიპის დაყვანისას int ტიპზე, long ტიპის მქონე ცვლადის მნიშვნელობა არ ეტევა int ტიპის დიაპაზონის ფარგლებში, მაშინ უფროსი თანრიგის ბიტები დაიკარგება, შედეგად, იკარგება ინფორმაციაც. როცა ხდება მცურავწერტილიანი მნიშვნელობის დაყვანა მთელ მნიშვნელობაზე, მაშინ წილადი ნაწილი იკარგება. მაგალითად, თუ მნიშვნელობა 7.85 გარდაიქმნება მთელ მნიშვნელობად, მაშინ შედეგი იქნება 7, ხოლო წილადი 0.85 იკარგება.

ზემოთ მოყვანილ სტრიქონში (wiladi1 / wiladi2) გამოსახულების შედეგი დაიყვანება int ტიპზე, რაც იწვევს წილადი ნაწილის დაკარგვას. შემდეგ, როცა baiti ცვლადს ენიჭება მნიშვნელობა 50, ინფორმაცია არ იკარგება, რადგან იგი მოთავსებულია byte

ტიპის დასაშვები მნიშვნელობების დიაპაზონში. მაგრამ, თუ baiti ცვლადს მივანიჭებთ მნიშვნელობას 300 (მისი ორობითი წარმოდგენაა 00000001 00101100), მაშინ ადგილი ექნება ინფორმაციის დაკარგვას, რადგან 300 აღემატება byte ტიპის მაქსიმალურ მნიშვნელობას. ამ შემთხვევაში baiti ცვლადს მიენიჭება 44 (მისი ორობითი წარმოდგენაა 00101100), რადგან დარჩება 300-ის უმცროსი ბაიტი, უფროსი კი ჩამოიჭრება. დაბოლოს, byte ტიპის გარდაქმნისას char ტიპად ინფორმაცია არ იკარგება.

ტიპების გარდაქმნა გამოსახულებებში

გამოსახულების გამოთვლის დროს შესაძლებელია ტიპების გარდაქმნა იმ პირობით, რომ ისინი თავსებადია. თუ გამოსახულებაში ხდება არათავსებადი ტიპის მონაცემების გამოყენება, მაშინ ისინი გარდაიქმნება ერთსა და იმავე ტიპად ბიჯობრივი გარდაქმნის ალგორითმის გამოყენებით.

გარდაქმნა სრულდება გამოსახულებებში ტიპების ავტომატური გარდაქმნის წესების მიხედვით. განვიხილოთ ეს ალგორითმი არითმეტიკის ოპერაციებისთვის:

- თუ ერთ ოპერანდს აქვს decimal ტიპი, მაშინ მეორე ოპერანდი ავტომატურად გარდაიქმნება ამ ტიპად. გამონაკლისია შემთხვევა, როცა მეორე ოპერანდს აქვს float ან double ტიპი. ამ შემთხვევაში ადგილი ექნება შეცდომას.
- თუ ერთ ოპერანდს აქვს double ტიპი, მეორე ოპერანდიც ავტომატურად გარდაიქმნება ამ ტიპად.
- თუ ერთ ოპერანდს აქვს float ტიპი, მაშინ მეორე ოპერანდიც ავტომატურად გარდაიქმნება ამ ტიპად.
- თუ ერთ ოპერანდს აქვს ulong ტიპი, მაშინ მეორე ოპერანდიც ავტომატურად გარდაიქმნება ამ ტიპად. გამონაკლისია შემთხვევა, როცა ოპერანდს აქვს sbyte, short, int ან long ტიპი. ამ შემთხვევაში ადგილი ექნება შეცდომას.
- თუ ერთ ოპერანდს აქვს long ტიპი, მაშინ მეორე ოპერანდი ავტომატურად გარდაიქმნება ამ ტიპად.
- თუ ერთ ოპერანდს აქვს uint ტიპი, ხოლო მეორეს sbyte, short ან int, მაშინ ორივე ოპერანდი ავტომატურად გარდაიქმნება long ტიპად.
- თუ არც ერთი ზემოთ მოყვანილი წესი არ იყო გამოყენებული, მაშინ ორივე ოპერანდი int ტიპად გარდაიქმნება.

უკანასკნელი წესის თანახმად გამოსახულებაში char, sbyte, byte, ushort და short ტიპები გარდაიქმნება int ტიპად. გარდაქმნის ასეთ პროცედურას მთელრიცხვა ტიპად ავტომატური გარდაქმნა ეწოდება. ეს, აგრეთვე, იმას ნიშნავს, რომ ყველა მათემატიკური ოპერაციის შედეგს ექნება ტიპი, რომელსაც მნიშვნელობის შესანახად გამოყოფილი აქვს თანრიგების არანაკლები რაოდენობა, ვიდრე int ტიპს.

გამოსახულებაში შეუძლებელია ყველა ტიპის ავტომატურად შეთავსება. კერძოდ, შეუძლებელია float ან double ტიპის ავტომატურად გარდაქმნა decimal ტიპად. ასევე შეუძლებელია ulong ტიპის შეთავსება რომელიმე სხვა ნიშნიან მთელრიცხვა ტიპთან. მათი გარდაქმნისათვის საჭიროა ტიპის აშკარა გარდაქმნის ოპერაციის გამოყენება.

ოპერანდების მიმართ ტიპების ავტომატური გარდაქმნა სრულდება მხოლოდ მაშინ, როცა ხდება მათი გამოყენება გამოსახულების გამოთვლისას. მაგალითად, თუ გამოსახულებაში byte ტიპის მქონე ცვლადის მნიშვნელობა გარდაიქმნება int ტიპად, მაშინ გამოსახულების გარეთ ეს ცვლადი ინარჩუნებს byte ტიპს.

ტიპის გარდაქმნამ შეიძლება მოგვცეს არასწორი შედეგი. მაგალითად, როცა არითმეტიკულ ოპერაციას ვასრულებთ byte ტიპის ორ ცვლადზე, მაშინ ჯერ სრულდება

ამ ცვლადების გარდაქმნა int ტიპის ცვლადებად, შემდეგ კი გამოითვლება გამოსახულება, რომლის შედეგს ექნება int ტიპი. ამიტომ, ჩვენ უნდა ვაკონტროლოთ იმ ცვლადების ტიპები, რომლებსაც ენიჭებათ გამოთვლების შედეგები.

განვიხილოთ პროგრამა.

```
{
//      პროგრამა 2.24
//      პროგრამაში ხდება ტიპების გარდაქმნის დემონსტრირება
int mteli;
byte baiti1, baiti2;

baiti1 = Convert.ToByte(textBox1.Text);baiti2 = Convert.ToByte(textBox2.Text);
mteli = baiti1 * baiti2;                    // აქ არ არის საჭირო ტიპის აშკარა გარდაქმნა
label1.Text = mteli.ToString();

baiti1 = Convert.ToByte(textBox3.Text);
// შედეგის ტიპი დაიყვანება baiti1 ცვლადის ტიპზე
baiti1 = ( byte ) ( baiti1 * baiti2 );
label2.Text = baiti1.ToString();
}
```

ტიპების გარდაქმნა საჭირო არ არის როცა baiti1 * baiti2 გამოსახულების შედეგი ენიჭება mteli ცვლადს, რადგან ამ გამოსახულების შედეგი ავტომატურად გარდაიქმნება int ტიპად. თუ baiti1 ცვლადს მივანიჭებთ baiti1 * baiti2 გამოსახულების შედეგს, მაშინ აუცილებელი იქნება byte ტიპზე დაყვანის ოპერაციის შესრულება baiti1 ცვლადის მიმართ.

ასეთივე სიტუაციას ადგილი აქვს char ტიპის მონაცემებზე ოპერაციების ჩატარებისას. მაგალითად, პროგრამის კოდის ფრაგმენტში:

```
char simbolo1 = 'რ', simbolo2 = 'ს';
simbolo1 = ( char ) ( simbolo1 + simbolo2 );
```

შედეგი უნდა გარდაიქმნას char ტიპად, რადგან გამოსახულებაში ch1 და ch2 ცვლადები გარდაიქმნება int ტიპად და შედეგსაც ექნება int ტიპი.

თავი 3. მმართველი ოპერატორები

არსებობს სამი კატეგორიის მმართველი ოპერატორი - ამორჩევის (if და switch), იტერაციისა (for, while, do-while და foreach) და გადასვლის (break, continue, goto და return).

ამორჩევის ოპერატორები

if ოპერატორი

if ოპერატორი ამოწმებს ლოგიკურ პირობას და თუ ის ჭეშმარიტია, მაშინ ასრულებს კოდის მითითებულ ფრაგმენტს. მისი სინტაქსია:

if (*პირობა*) { *ოპერატორები1*; }

[else { *ოპერატორები2*; }]

აქ *პირობა* არის ლოგიკური გამოსახულება, რომელიც იღებს true ან false მნიშვნელობას. თუ *პირობა* ჭეშმარიტია, მაშინ შესრულდება *ოპერატორები1*, წინააღმდეგ შემთხვევაში - *ოპერატორები2*. *ოპერატორები1* და *ოპერატორები2* შეიძლება იყოს ერთი ან მეტი ოპერატორი. კვადრატული ფრჩხილები მიუთითებენ იმაზე, რომ else სიტყვის მითითება აუცილებელი არ არის. ასეთ შემთხვევაში, if ოპერატორის სინტაქსი იქნება:

if (*პირობა*) { *ოპერატორები1*; }

თუ პირობა იღებს true მნიშვნელობას, მაშინ შესრულდება ოპერატორები*1*, წინააღმდეგ შემთხვევაში კი - პროგრამის მომდევნო ოპერატორი.

if ოპერატორის მუშაობის საჩვენებლად შევადგინოთ პროგრამა, რომელიც განსაზღვრავს რიცხვი კენტია თუ ლუწი.

```
{
//      პროგრამა 3.1
//      პროგრამა განსაზღვრავს რიცხვი კენტია თუ ლუწი
int ricxvi;

label1.Text = " ";
ricxvi = Convert.ToInt32(textBox1.Text);
if ( ricxvi % 2 == 1 ) label1.Text = "რიცხვი კენტია";
        else label1.Text = "რიცხვი ლუწია";
}
```

ჩადგმული if ოპერატორი

ჩადგმულია if ოპერატორი, რომელიც მოთავსებულია სხვა if ოპერატორის შიგნით. ჩადგმული ოპერატორი იმ შემთხვევაში გამოიყენება, როცა მოქმედების შესასრულებლად საჭიროა რამდენიმე პირობის შემოწმება, რომელთა მითითება ერთ if ოპერატორში შეუძლებელია. ჩადგმულ if ოპერატორში else ნაწილი ყოველთვის ეკუთვნის უახლოეს if ოპერატორს. შევადგინოთ პროგრამა, რომელიც დაადგენს ორ რიცხვს შორის არის თუ არა დადებითი:

```
{
```

```
//      პროგრამა 3.2
//      პროგრამა დაადგენს ორ რიცხვს შორის არის თუ არა დადებითი
int ricxvi1, ricxvi2;

ricxvi1 = Convert.ToInt32(textBox1.Text);
ricxvi2 = Convert.ToInt32(textBox2.Text);
if ( ricxvi1 >= 0 ) label1.Text = "დადებითია პირველი რიცხვი";
        else    if ( ricxvi2 >= 0 ) label1.Text = "დადებითია მეორე რიცხვი";
                        else label1.Text = "არც ერთი რიცხვი არ არის დადებითი";
}
```

თუ პირველი if ოპერატორის პირობა ჭეშმარიტია, მაშინ label1 კომპონენტში გამოიცემა შესაბამისი შეტყობინება და პროგრამა მუშაობას დაამთავრებს. წინააღმდეგ შემთხვევაში შემოწმდება მეორე if ოპერატორის პირობა. თუ ის ჭეშმარიტია მაშინ label1 კომპონენტში გამოიცემა შესაბამისი შეტყობინება და პროგრამა მუშაობას დაამთავრებს. თუ არც ერთი პირობა არ არის ჭეშმარიტი, მაშინ label1 კომპონენტში გამოიცემა შესაბამისი შეტყობინება და პროგრამა მუშაობას დაამთავრებს.

პროგრამული კოდის ბლოკები. ხილვადობის უბანი

აქამდე, ცვლადების გამოცხადებას ვახდენდით პროგრამის (მეთოდის) დასაწყისში. ისინი წარმოადგენენ ლოკალურ ცვლადებს, რადგან გამოცხადებული არიან მეთოდის ხილვადობის უბანში (მეთოდის შიგნით). ლოკალური ცვლადების გამოცხადება შეიძლება, აგრეთვე, მეთოდში ბლოკის შიგნით. ბლოკი არის ოპერატორების ერთობლიობა, რომელიც მოთავსებულია { და } ფრჩხილებს შორის მეთოდის შიგნით. ბლოკი განსაზღვრავს ხილვადობის უბანს. ხილვადობის ახალი უბანი იქმნება ბლოკის შექმნისას. ხილვადობის უბანი განსაზღვრავს გამოცხადებული ცვლადების სიცოცხლის (არსებობის) დროს.

ხილვადობის უბნები შეიძლება განისაზღვროს კლასით და მეთოდით. ამ ეტაპზე განვიხილავთ მეთოდის მიერ განსაზღვრულ ხილვადობის უბნებს. თუ მეთოდს აქვს პარამეტრები, მაშინ ისინიც ჩართული იქნებიან მეთოდის ხილვადობის უბანში. ხილვადობის უბანში გამოცხადებული ცვლადი უხილავია (მიუწვდომელი) ამ უბნის გარეთ განსაზღვრული კოდისათვის. შედეგად, ხდება ცვლადის დაცვა არასანქცირებული მიმართვისაგან.

მოყვანილ პროგრამაში ხდება ბლოკისა და ხილვადობის უბნის დემონსტრირება.

```
{
//      პროგრამა 3.3
//      პროგრამაში ხდება ბლოკისა და ხილვადობის უბნის დემონსტრირება
int ricxvi1 = Convert.ToInt32(textBox1.Text);
if ( ricxvi1 > 25 )
        {                                       //      ბლოკის დასაწყისი
                int ricxvi2 = 30;
                ricxvi1 = ricxvi2 * ricxvi2;
        }                                       //      ბლოკის დასასრული
label1.Text = ricxvi1.ToString();
// ricxvi2 = 50;
```

}

ricxvi1 ცვლადი ხილულია მთელი კოდისათვის, რადგან გამოცხადებულია პროგრამის დასაწყისში. ricxvi2 ცვლადი გამოცხადებულია if ბლოკის შიგნით. რადგან ბლოკი განსაზღვრავს ხილვადობის უბანს, ამიტომ ricxvi2 ცვლადი ხილულია მხოლოდ ამ ბლოკის შიგნით. სწორედ ამიტომ არის შეცდომა ricxvi2 ცვლადის გამოყენება ამ ბლოკის გარეთ სტრიქონში ricxvi2 = 50;. თუ წავშლით კომენტარის სიმბოლოებს, მაშინ პროგრამის კოდის კომპილირებისას გაიცემა შესაბამისი შეტყობინება შეცდომის შესახებ. if ბლოკის შიგნით შეიძლება ricxvi1 ცვლადის გამოყენება, რადგან ამ ბლოკის კოდს შეუძლია მიმართოს გარე ბლოკში გამოცხადებულ ცვლადს.

ცვლადთან მიმართვა შესაძლებელია მხოლოდ მისი გამოცხადების შემდეგ. ბლოკის შიგნით ცვლადის გამოცხადება ნებისმიერ ადგილში შეიძლება, რის შემდეგ ის ხდება მოქმედი. შედეგად, თუ ცვლადს გამოვაცხადებთ მეთოდის დასაწყისში, მაშინ მასთან მიმართვა შესაძლებელი იქნება მთელი კოდის შიგნით. თუ ცვლადს გამოვაცხადებთ ბლოკის სხვა ადგილში, მაშინ მასთან მიმართვას შევძლებთ მხოლოდ მისი გამოცხადების შემდეგ.

ცვლადები ხილვადობის რომელიმე უბანში იქმნება მათი გამოცხადებისას (მოცემულ უბანში შესვლისას) და იშლება ამ უბნიდან გამოსვლისას. ეს იმას ნიშნავს, რომ ცვლადი კარგავს თავის მნიშვნელობას ხილვადობის უბნიდან გამოსვლისას, ე.ი. მეთოდის შიგნით გამოცხადებული ცვლადი არ შეინახავს თავის მნიშვნელობას ამ მეთოდის გამოძახებებს შორის. ანალოგიურად, ბლოკის შიგნით გამოცხადებული ცვლადიც კარგავს თავის მნიშვნელობას ბლოკიდან გამოსვლისას. შედეგად, ცლადის სიცოცხლის დრო შეზღუდულია მისი ხილვადობის უბნით.

თუ ცვლადის გამოცხადებისას ხდება მისი ინიციალიზაცია, მაშინ მისი ინიციალიზაცია შესრულდება ყოველთვის ამ ბლოკში შესვლისას.

ხილვადობის უბანი შეიძლება იყოს ჩადგმული. მაგალითად, როცა ვქმნით პროგ-რამული კოდის ბლოკს, ჩვენ ვქმნით ახალ ჩადგმულ ხილვადობის უბანს. ამასთან, გარე უბანი ხდება ხილული ჩადგმული უბნისათვის. ეს იმას ნიშნავს, რომ გარე ხილვადობის უბანში გამოცხადებული ობიექტები და ცვლადები ხილული იქნება შიდა ხილვადობის უბნის კოდისთვის. შიდა ხილვადობის უბნში გამოცხადებული ობიექტები და ცვლადები არ იქნება ხილული გარე ხილვადობის უბნის კოდისთვის. ბოლოს, შევნიშნოთ რომ გარე და ჩადგმულ ბლოკებში გამოცხადებულ ცვლადებს არ შეიძლება ერთნაირი სახელები ჰქონდეს.

გლობალურია ცვლადი, რომელიც გამოცხადებულია ყველა მეთოდის გარეთ. მისი გამოცხადება შეიძლება Main() მეთოდის შემდეგ. გლობალური ცვლადები მოქმედებენ ყველა მეთოდის შიგნით და არ კარგავენ თავიანთ მნიშვნელობებს მეთოდიდან გამოსვლის დროს. მოყვანილ პროგრამაში ხდება გლობალურ ცვლადთან მუშაობის დემონსტრირება.

```
//      პროგრამა 3.4
//      პროგრამაში ხდება გლობალურ ცვლადთან მუშობის დემონსტრირება
static void Main()
{
Application.Run(new Form1());
}
//      გლობალური ცვლადის გამოცხადება
int globaluri_cvladi;
```

```
private void button1_Click(object sender, System.EventArgs e)
{
//      ლოკალური ცვლადის გამოცხადება
int lokaluri_cvladi = 10;
globaluri_cvladi = Convert.ToInt32(textBox1.Text);
label1.Text = globaluri_cvladi.ToString();
if ( globaluri_cvladi > 3 )
{
        globaluri_cvladi = 50;
        lokaluri_cvladi = 20;
}
label2.Text = globaluri_cvladi.ToString();
}
```

switch ოპერატორი

იგი საშუალებას გვაძლევს ვარიანტების სიიდან ამოვარჩიოთ საჭირო მოქმედება. იგი შემდეგნაირად მუშაობს: გამოსახულების მნიშვნელობა მიმდევრობით შედარდება სიაში მითითებულ თითოეულ მუდმივას (კონსტანტას). ერთ-ერთ მუდმივასთან მნიშვნელობის დამთხვევისას შესრულდება მასთან ასოცირებული (დაკავშირებული) ოპერატორების მიმდევრობა. switch ოპერატორის სინტაქსია:

```
switch ( გამოსახულება )
 {
        case მუდმივა1 :     ოპერატორების მიმდევრობა;
                            break;
        case მუდმივა2 :     ოპერატორების მიმდევრობა;
                            break;
        . . .
        [ default :         ოპერატორების მიმდევრობა;
                            break; ]
 }
```

აქ *გამოსახულების* მნიშვნელობას უნდა ჰქონდეს მთელრიცხვა ტიპი - char, byte, short ან int, ან სტრიქონული ტიპი - string (მას მოგვიანებით განვიხილავთ). გამოსახულების მნიშვნელობას არ უნდა ჰქონდეს მცურავწერტილიანი ტიპი (double, float). switch ოპერატორის case განშტოებების მუდმივები უნდა იყოს ლიტერალები და ჰქონდეთ გამოსახულებების ტიპთან თავსებადი ტიპი. ამასთან, switch ოპერატორის ერთ ბლოკში მათ არ შეიძლება ერთნაირი მნიშვნელობები ჰქონდეთ.

ერთ case განშტოებასთან დაკავშირებულმა ოპერატორების მიმდევრობამ მართვა არ უნდა გადასცეს მომდევნო case განშტოების ოპერატორებს. ამის თავიდან ასაცილებლად თითოეული case განშტოების ოპერატორების მიმდევრობა break ოპერატორით უნდა დავამთავროთ.

default განშტოების შესაბამისი ოპერატორების მიმდევრობა სრულდება მაშინ, როცა case განშტოებების არც ერთი მუდმივა არ შეესაბამება გამოსახულების მნიშვნელობას. default განშტოების მითითება არ არის აუცილებელი. თუ ის არ არის მითითებული და switch ოპერატორში არც ერთი მუდმივა არ შეესაბამება გამოსახულების მნიშვნელობას, მაშინ არავითარი მოქმედება არ შესრულდება და

მართვა გადაეცემა switch ოპერატორის შემდეგ მოთავსებულ ოპერატორს. თუ გამოსახულების მნიშვნელობა დაემთხვა მუდმივას მნიშვნელობას, მაშინ შესრულდება ამ case განშტოებასთან ასოცირებული ოპერატორების მიმდევრობა break ოპერატორამდე. break ოპერატორი მართვას გადასცემს switch ოპერატორის შემდეგ მოთავსებულ ოპერატორს. მოვიყვანოთ პროგრამა, რომელიც ახდენს ციფრების ფიქსირებას დიაპაზონში 0 - 2.

```
{
//      პროგრამა 3.5
//      პროგრამა აფიქსირებს ციფრებს დიაპაზონში 0 - 2
int ricxvi;

ricxvi = Convert.ToInt32(textBox1.Text);
switch ( ricxvi )
  {
        case 0 : label1.Text = "თქვენ შეიტანეთ ციფრი -  " + ricxvi.ToString();
                 break;
        case 1 : label1.Text = "თქვენ შეიტანეთ ციფრი -  " + ricxvi.ToString();
                 break;
        case 2 : label1.Text = "თქვენ შეიტანეთ ციფრი -  " + ricxvi.ToString();
                 break;
        default : label1.Text = "შეტანილი რიცხვი არ არის დიაპაზონში 0 - 2";
                  break;
  }
}
```

ამ პროგრამაში switch ოპერატორის მართვა ხდება int ტიპის ricxvi ცვლადის მეშვეობით.

ახლა შევადგინოთ პროგრამა, სადაც switch ოპერატორის მართვა ხორციელდება char ტიპის მქონე გამოსახულებით. ამ შემთხვევაში case განშტოების მუდმივებსაც უნდა ჰქონდეთ char ტიპი. პროგრამა გამოთვლებს ასრულებს იმის მიხედვით თუ არითმეტიკის რომელ ოპერატორს შევიტანთ.

```
{
//      პროგრამა 3.6
//      კალკულატორი
int shedegi, ricxvi1, ricxvi2;
char operacia;

ricxvi1  = Convert.ToInt32(textBox1.Text);
ricxvi2  = Convert.ToInt32(textBox3.Text);
operacia = Convert.ToChar(textBox2.Text);
switch ( operacia )
{
case '+' :      shedegi = ricxvi1 + ricxvi2;
                label1.Text = shedegi.ToString();
                break;
case '-' :      shedegi = ricxvi1 - ricxvi2;
```

```
                label1.Text = shedegi.ToString();
                break;
case '*' :      shedegi = ricxvi1 * ricxvi2;
                label1.Text = shedegi.ToString();
                break;
case '/' :      shedegi = ricxvi1 / ricxvi2;
                label1.Text = shedegi.ToString();
                break;
case '%' :      shedegi = ricxvi1 % ricxvi2;
                label1.Text = shedegi.ToString();
                break;
}
}
```

switch ოპერატორის ორი ან მეტი განშტოება შეიძლება დაკავშირებული იყოს ოპერატორების ერთსა და იმავე მიმდევრობასთან. ამას სხვანაირად ჩავარდნას უწოდებენ. ქვემოთ მოყვანილი პროგრამა ახდენს ჩავარდნის დემონსტრირებას. პროგრამა განასხვავებს ანბანის ხმოვან და თანხმოვან ასოებს.

```
{
//      პროგრამა 3.7
//      პროგრამაში ხდება ჩავარდნის დემონსტრირება
char simbolo;
label1.Text = " ";
simbolo = textBox1.Text[0];
switch ( simbolo )
  {
        case 'ა' :
        case 'ე' :
        case 'ი' :
        case 'ო' :
        case 'უ' : label1.Text = "ეს არის ხმოვანი ასო";
                   break;
        default  : label1.Text = "ეს არ არის ხმოვანი ასო";
                     break;
  }
}
```

თუ simbolo ცვლადი იღებს მნიშვნელობას 'ა', 'ე', 'ი', 'ო' ან 'უ', მაშინ შესრულდება label3.Text = "ეს არის ხმოვანი ასო"; ოპერატორი, წინააღმდეგ შემთხვევაში - label3.Text = "ეს არ არის ხმოვანი ასო"; ოპერატორი.

ჩადგმული switch ოპერატორი

ერთი switch ოპერატორი შეიძლება მოთავსებული იყოს გარე switch ოპერატორში. შიდა switch ოპერატორს ეწოდება ჩადგმული. შიდა და გარე switch ოპერატორების case განშტოებების მუდმივებს შეიძლება ერთნაირი მნიშვნელობები ჰქონდეთ. ეს არ გამოიწვევს კონფლიქტს პროგრამის შესრულების დროს. განვიხილოთ მაგალითი.

```
{
//      პროგრამა 3.8
//      პროგრამაში ხდება ჩადგმული switch ოპერატორის მუშობის დემონსტრირება
char simbolo1, simbolo2;

simbolo1 = textBox1.Text[0];
simbolo2 = textBox2.Text[0];
switch ( simbolo1 )
 {
        case 'a' :
        label3.Text = "a მუდმივა ეკუთვნის გარე switch ოპერატორს";
        switch (simbolo2 )
        {
                case 'a' :
                label5.Text = "a მუდმივა ეკუთვნის შიდა switch ოპერატორს";
                break;

                case 'b' :
                label5.Text = "b მუდმივა ეკუთვნის შიდა switch ოპერატორს";
                break;
        }
        break;

        case 'b' :
        label3.Text = "b მუდმივა ეკუთვნის გარე switch ოპერატორს";
        break;
 }
}
```

იტერაციის ოპერატორები

for ოპერატორი

იგი გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად. მისი სინტაქსია:

for ([*ინიციალიზატორი*]; [*პირობა*]; [*იტერატორი*]) [*ციკლის კოდი*];

ინიციალიზატორი არის გამოსახულება, რომელიც ციკლის დაწყების წინ გამოითვლება. აქ ხდება ციკლის მმართველი ცვლადის ინიციალიზაცია. *პირობა* არის ლოგიკური გამოსახულება, რომელიც იღებს true ან false მნიშვნელობას. იგი გამოითვლება ციკლის ყოველი იტერაციის წინ. for ციკლის გამეორება ხდება მანამ, სანამ პირობა იღებს true მნიშვნელობას. თუ მან მიიღო false მნიშვნელობა, მაშინ ციკლი მთავრდება და მართვა გადაეცემა for ოპერატორის შემდეგ მოთავსებულ ოპერატორს. *იტერატორი* არის გამოსახულება, რომელიც ზრდის ან ამცირებს მმართველი ცვლადის მნიშვნელობას. იგი გამოითვლება ციკლის ყოველი იტერაციის შემდეგ. *ციკლის კოდი* (ციკლის ტანი) შეიძლება შეიცავდეს ერთ ან მეტ ოპერატორს.

for ციკლი შემდეგნაირად მუშაობს. ჯერ შესრულდება ციკლის მმართველი ცვლადის ინიციალიზაცია. შემდეგ მოწმდება პირობა. თუ ის ჭეშმარიტია, მაშინ სრულდება ციკლის კოდი. შემდეგ იცვლება ციკლის ცვლადის მნიშვნელობა. კვლავ მოწმდება პირობა და ა.შ.

შევადგინოთ პროგრამა, რომელიც ანგარიშობს 1-დან 10-მდე რიცხვების კვადრატს. პროგრამაში ციკლის მმართველი ცვლადი ერთით იზრდება.

```
{
// პროგრამა 3.9
// პროგრამა ანგარიშობს 1-დან 10-მდე რიცხვების კვადრატს
double ricxvi, kvadrati;

label1.Text = " ";
for ( ricxvi = 1; ricxvi <= 10; ricxvi++ )
  {
        kvadrati = Math.Pow(ricxvi, 2);
        label1.Text +=  ricxvi.ToString() + " - " + kvadrati.ToString() + '\n';
  }
}
```

label2.Text = " "; ოპერატორის შესრულების შედეგად label2 კომპონენტში გამოტანილი სტრიქონები წაიშლება. შემდეგ, ციკლის ტანში სრულდება ამ კომპონენტში რიცხვისა და მისი კვადრატის გამოტანა. თუ label2.Text = " "; ოპერატორს არ შევასრულებთ, მაშინ Ok კლავიშზე განმეორებით დაჭერის შემთხვევაში, მასში ადრე გამოტანილ ინფორმაციას დაემატება ასეთივე ინფორმაცია. '\n' სიმბოლო ახდენს label კომპონენტის შიგნით მომდევნო სტრიქონზე გადასვლას. პროგრამის შესრულებამდე label2 კომპონენტის სიმაღლე უნდა გავზარდოთ.

ციკლის ცვლადი შეიძლება ზრდიდეს ან ამცირებდეს თავის მნიშვნელობას ნებისმიერი სიდიდით. ქვემოთ მოყვანილ პროგრამაში ციკლის ცვლადი მცირდება 10 ერთეულით.

```
{
// პროგრამა 3.10
// ციკლის მმართველი ცვლადი იცვლება უარყოფითი ბიჯით
double ricxvi, kvadrati;

label1.Text = " ";
for ( ricxvi = 100; ricxvi >= 1; ricxvi -= 10 )
  {
        kvadrati = Math.Pow(ricxvi, 2);
        label1.Text += ricxvi.ToString() + " - " + kvadrati.ToString() + '\n';
  }
}
```

როგორც ვიცით, for ციკლის პირობა ციკლის დასაწყისში მოწმდება. ეს იმას ნიშნავს, რომ თუ პირობა თავიდანვე იღებს false მნიშვნელობას, მაშინ ციკლის ტანი არასოდეს არ შესრულდება, მაგალითად,

```
for ( i1 = 5; i1 < 3; i1++ )
        i2 = i1 * i1;
```

ამ ციკლის ტანი არასოდეს არ შესრულდება, რადგან პირობის გამოსახულება თავიდანვე იღებს false მნიშვნელობას.

for ციკლში ერთის ნაცვლად შეგვიძლია ორი მმართველი ცვლადის გამოყენება, მაგალითად,

```
{
//      პროგრამა 3.11
//      პროგრამაში for ციკლისათვის გამოიყენება i1 და i2 მმართველი ცვლადები
int i1, i2;
label1.Text = "";

for ( i1 = 0, i2 = 10; i1 < i2; i1++, i2-- )
        label1.Text = label1.Text + "   " + i1.ToString() + "   " +  i2.ToString() + '\n';
}
```

აქ ინიციალიზაციის ორი ოპერატორი გამოყოფილია მძიმით. ასევე, ორი იტერაციული გამოსახულება გამოყოფილია მძიმით. როცა ციკლი იწყებს მუშაობას ორივე ცვლადს ენიჭება საწყისი მნიშვნელობები. ყოველი იტერაციის შემდეგ i1 ცვლადი მნიშვნელობა ერთით იზრდება, i2 ცვლადის მნიშვნელობა კი - ერთით მცირდება. for ციკლის მართვა რამდენიმე მმართველი ცვლადის გამოყენებით ზოგჯერ საკმაოდ მოხერხებულია და ამარტივებს ალგორითმებს. ციკლში დასაშვებია ნებისმიერი რაოდენობის ინიციალიზაციისა და იტერაციული ოპერატორების გამოყენება, მაგრამ პრაქტიკულად, ორ ცვლადზე მეტი იშვიათად გამოიყენება.

for ციკლის ჩაწერისას შეგვიძლია არ მივუთითოთ ინიციალიზაციის, პირობის ან იტერაციის ნაწილი, მაგალითად,

```
{
//      პროგრამა 3.12
//      პროგრამაში გამოყენებულია for ოპერატორი იტერაციის ნაწილის გარეშე
int i1;
label1.Text = "";

//      ამ ციკლში მითითებული არ არის იტერაციის ნაწილი
for ( i1 = 0; i1 < 10;  )
  {
        label1.Text += i1.ToString() + '\n';
        i1++;
  }
}
```

აქ არ არის მითითებული იტერაციის ნაწილი, სამაგიეროდ, მმართველი ცვლადი ერთით იზრდება ციკლის ტანში.

ქვემოთ მოყვანილ პროგრამაში გამოყენებულია for ციკლი, რომელშიც არ არის მითითებული ინიციალიზაციისა და იტერაციის ნაწილები.

```
{
//      პროგრამა 3.13
//      პროგრამაში გამოყენებულია for ოპერატორი ინიციალიზაციისა და იტერაციის
ნაწილების გარეშე
int i1 = 0;
```

```
label1.Text = "";
//        ციკლი ინიციალიზატორისა და იტერატორის გარეშე
for ( ; i1 < 10;  )
  {
        label1.Text += i1.ToString() + '\n';
        i1++;
  }
}
```

აქ i1 ცვლადის ინიციალიზაცია ხდება ციკლამდე. ასე ძირითადად ვიქცევით მაშინ, როცა მმართველი ცვლადი საწყის მნიშვნელობას იღებს რაიმე რთული გამოსახულების გამოთვლის შედეგად.

თუ for ციკლის არც ერთი ნაწილი არ არის მითითებული, მაშინ მივიღებთ უსასრულო ციკლს, მაგალითად

```
for ( ; ; )
{
// ციკლის კოდი
}
```

უსასრულო ციკლის შესაწყვეტად შეგვიძლია break ოპერატორი გამოვიყენოთ.

for ციკლს შეიძლება არ ჰქონდეს, აგრეთვე, კოდი, ანუ ის იყოს ცარიელი. ეს შესაძლებელია, რადგან ნულოვანი ოპერატორი სინტაქსურად დასაშვებია. ქვემოთ მოყვანილია პროგრამა, რომელშიც ასეთი ციკლი გამოიყენება 1-დან 5-მდე რიცხვების შესაკრებად.

```
{
//        პროგრამა 3.14
//        პროგრამაში გამოყენებულია for ოპერატორი კოდის გარეშე
int i1, jami = 0;

label1.Text = "";
for ( i1 = 1; i1 <= 5; jami += i1++ );          //        ამ ციკლში კოდი არ არის
label1.Text = jami.ToString();
}
```

პროგრამიდან ჩანს, რომ ჯამის გამოთვლა ხდება for ოპერატორის შიგნით. შედეგად, ციკლის ტანი საჭირო არ არის. რაც შეეხება გამოსახულებას jami += i1++; იგი შემდეგნაირად მუშაობს. jami ცვლადს ენიჭება მისსავე მნიშვნელობას დამატებული i1 ცვლადის მნიშვნელობა. შემდეგ, i1 მნიშვნელობა ერთით იზრდება. ეს ოპერატორი შემდეგი ორი ოპერატორის ანალოგიურია:

```
jami = jami + i1;
i1++;
```

ხშირად მმართველი ცვლადი გამოიყენება მხოლოდ for ციკლის შიგნით. ასეთ შემთხვევაში მისი გამოცხადება ხდება ციკლის საინიციალიზაციო ნაწილში. განვიხილოთ პროგრამა, რომელიც ახდენს პირველი ხუთი რიცხვის შეკრებას.

```
{
//        პროგრამა 3.15
//        პროგრამაში გამოყენებულია for ოპერატორი, რომელშიც გამოცხადებულია //
```

მმართველი ცვლადი

```
int jami = 0;

label1.Text = "";
for ( int i1 = 1; i1 <= 5; i1++ )
        jami += i1;
label1.Text = jami.ToString();
}
```

i1 მმართველი ცვლადი გამოცხადებულია for ციკლის საინიციალიზაციო ნაწილში, ამიტომ, იგი ხილულია მხოლოდ ამ ციკლის ფარგლებში. მის გარეთ იგი უხილავია. თუ ვაპირებთ მმართველი ცვლადის გამოყენებას პროგრამის სხვა ნაწილში, მაშინ იგი უნდა გამოვაცხადოთ for ციკლის გარეთ.

მაგალითები

შევადგინოთ პროგრამა, რომელიც შეკრებს რიცხვებს მითითებულ დიაპაზონში.

```
{
//      პროგრამა 3.16
//      მითითებულ დიაპაზონში რიცხვების შეკრების პროგრამა
int index, min, max, jami = 0;

min = Convert.ToInt32(textBox1.Text);                   //      საწყისი რიცხვი
max = Convert.ToInt32(textBox2.Text);                   //      საბოლოო რიცხვი
for ( index = min; index <= max; index++ )
        jami += index;
label1.Text = jami.ToString();
}
```

შევადგინოთ ფაქტორიალის გამოთვლის პროგრამა.

```
{
//      პროგრამა 3.17
//      ფაქტორიალის გამოთვლის პროგრამა
int index, ricxvi, shedegi = 1;

ricxvi = Convert.ToInt32(textBox1.Text);
for ( index = 1; index <= ricxvi; index++ )
        shedegi *= index;
label1.Text = shedegi.ToString();
}
```

შევადგინოთ პროგრამა, რომელიც გამოთვლის მოცემული მწკრივის მნიშვნელობას

$$y = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9}$$

.

```
{
//      პროგრამა 3.18
//      მწკრივის გამოთვლის პროგრამა
```

```
double shedegi = 0, x;
int minusi = -1;

x  = Convert.ToDouble(textBox1.Text);
for ( int indexi = 1; indexi <= 9; indexi += 2 )
        {
                minusi *= -1;
                shedegi += minusi * Math.Pow(x, indexi) / indexi;
        }
label1.Text = shedegi.ToString();
}
```

while ციკლი

მისი სინტაქსია:

```
while ( პირობა )
{
ციკლის კოდი
}
```

აქ *პირობა* არის ლოგიკური გამოსახულება, რომელიც გასცემს true ან false მნიშვნელობას. თუ იგი იღებს false მნიშვნელობას, მაშინ ხდება ციკლიდან გამოსვლა და სრულდება მომდევნო ოპერატორი. თუ იგი იღებს true მნიშვნელობას, მაშინ შესრულდება ციკლის ტანი.

განვიხილოთ პროგრამა, რომელსაც ეკრანზე გამოაქვს ინგლისური ანბანის ასოები.

```
{
//      პროგრამა 3.19
//      პროგრამას label1 კომპონენტში გამოაქვს ინგლისური ანბანის ასოები
char simbolo = 'ა';

while ( simbolo <= 'ჰ' )
  {
        label1.Text += simbolo.ToString() + '\n';
        simbolo++;
  }
}
```

simbolo ცვლადს ენიჭება საწყისი მნიშვნელობა - ‘ა’. ციკლის ყოველი შესრულებისას label1 კომპონენტში გამოიცემა simbolo ცვლადის მნიშვნელობა, შემდეგ კი მისი მნიშვნელობა ერთით იზრდება.

do-while ციკლი

for და while ციკლებისაგან განსხვავებით, რომლებშიც ჯერ მოწმდება პირობა და შემდეგ სრულდება ციკლის კოდი, do-while ციკლში პირობა მოწმდება ციკლის ბოლოს, ე.ი. ჯერ შესრულდება ციკლის კოდი, შემდეგ კი შემოწმდება პირობა. მისი სინტაქსია:

```
do
{
```

```
ციკლის კოდი
}
while ( პირობა )
```

თუ ციკლში ერთი ოპერატორი სრულდება, მაშინ ფიგურული ფრჩხილები აუცილებელი არ არის, მაგრამ მათ ხშირად იყენებენ do-while ციკლის კითხვადობის გაუმჯობესების მიზნით, რადგან იგი ადვილად შეიძლება აგვერიოს while ციკლში.

ქვემოთ მოყვანილია პროგრამა, რომელსაც label კომპონენტში გამოაქვს ქართული ანბანის ასოები 'ა'-დან 'კ'-მდე.

```
{
// პროგრამა 3.20
// პროგრამას label1 კომპონენტში გამოაქვს ინგლისური ანბანის ასოები
char simbolo = 'ა';

do
  {
        label1.Text += simbolo.ToString() + '\n';
        simbolo++;
  }
while ( simbolo != 'კ' );
}
```

გადასვლის ოპერატორები

break ოპერატორი

იგი გვაძლევს ციკლის შესრულების იძულებით შეწყვეტისა და ციკლიდან გამოსვლის საშუალებას. ამ დროს, ციკლის დანარჩენი ოპერატორები არ შესრულდება. მართვა გადაეცემა ციკლის შემდეგ მოთავსებულ ოპერატორს. ქვემოთ მოყვანილი პროგრამა ახდენს რიცხვების გამრავლებას მანამ, სანამ ნამრავლი არ გახდება 30-ზე მეტი.

```
{
// პროგრამა 3.21
// პროგრამა რიცხვებს ამრავლებს მანამ, სანამ ნამრავლი არ გახდება 30-ზე მეტი
int indexi, namravli = 1;

label1.Text = "";
for ( indexi = 1; indexi < 50; indexi++ )
  {
        namravli *= indexi;
        if ( namravli > 30 ) break;
        label1.Text += '\n' + namravli.ToString();
  }
indexi--;
label1.Text = "ციკლი შესრულდა " + indexi.ToString() + "-ჯერ";
}
```

ციკლი უნდა შესრულდეს 50-ჯერ, მაგრამ break ოპერატორის შესრულება იწვევს მის ვადამდე შეწყვეტას. ციკლი შესრულდება მხოლოდ ოთხჯერ.

break ოპერატორის გამოყენება შეიძლება ნებისმიერი ციკლის შიგნით. თუ break ოპერატორი გამოიყენება ჩადგმული ციკლის შიგნით, მაშინ იგი შეწყვეტს მხოლოდ შიდა ციკლის მუშაობას. მაგალითად,

```
{
//      პროგრამა 3.22
//      პროგრამა თვლის თუ რამდენჯერ შესრულდა თითოეული ციკლი
int gare = 0, shida = 1;

for ( int i1 = 0; i1 < 5; i1++ )                    //      გარე ციკლი
  {
        gare++;
        label1.Text = "გარე ციკლში იტერაციების რაოდენობა = " + gare.ToString();

        while ( shida < 30 )                        //      შიდა ციკლი
        {
        //      break ოპერატორი შეწყვეტს შიდა ციკლის შესრულებას
        //      თუ shida ცვლადი იღებს მნიშვნელობას 5
        if ( shida == 5 ) break;
                label2.Text += shida.ToString() + ' ';
                shida++;
        }
        label1.Text += '\n';
  }
}
```

continue ოპერატორი

იგი საშუალებას გვაძლევს იძულებით გადავიდეთ მომდევნო იტერაციაზე. continue ოპერატორის შემდეგ მოთავსებული ოპერატორები არ შესრულდება. განვიხილოთ პროგრამა, რომელსაც label კომპონენტში გამოაქვს კენტი რიცხვები 1-დან 21-მდე.

```
{
//      პროგრამა 3.23
//      პროგრამას label1 კომპონენტში გამოაქვს კენტი რიცხვები 1-დან 21-მდე
int indexi;

for ( indexi = 1; indexi <= 21; indexi++ )
  {
        if ( ( indexi % 2 ) != 1 ) continue;        //      მომდევნო იტერაციაზე გადასვლა
        label1.Text += indexi.ToString() + '\n';
  }
}
```

while და do-while ციკლებში continue ოპერატორის გამოყენება იწვევს მართვის გადაცემას უშუალოდ პირობის გამოსახულებისათვის, რის შემდეგ ციკლის შესრულება

გრძელდება.

goto ოპერატორი

goto არის უპირობო გადასვლის ოპერატორი. მისი სინტაქსია:

goto *ჭდე;*

სადაც, ჭდე არის იდენტიფიკატორი, რომელსაც ორი წერტილი (:) მოსდევს. იგი მიუთითებს იმ ადგილს, საიდანაც უნდა გაგრძელდეს პროგრამის შესრულება. თანამედროვე პროგრამირებაში იგი ნაკლებად გამოიყენება, რადგან მისი გამოყენების დროს პროგრამა ხდება ნაკლებად მოდულური და სტრუქტურირებული. თუმცა მისი გამოყენება ზოგჯერ საკმაოდ მოხერხებულია. ჭდე მოქმედებს მხოლოდ მოცემული მეთოდის შეგნით. მოყვანილ პროგრამაში ნაჩვენებია თუ როგორ შეიძლება ციკლის ორგანიზება goto ოპერატორის გამოყენებით:

```
{
//      პროგრამა 3.24
//      პროგრამაში ხდება goto ოპერატორის მუშაობის დემონსტრირება
int ricxvi = 1;
cikli1:
        ricxvi++;
        if ( ricxvi < 8 ) goto cikli1;
label1.Text = ricxvi.ToString();
}
```

ტერნარული ოპერატორი

? ოპერატორს ეწოდება ტერნარული, რადგან მას სამი ოპერანდი აქვს. მისი სინტაქსია:

გამოსახულება1 ? *გამოსახულება2* : *გამოსახულება3*;

აქ *გამოსახულება1* არის ლოგიკური გამოსახულება, ხოლო *გამოსახულება2* და *გამოსახულება3* გამოსახულებებია, რომლებსაც ერთნაირი ტიპები უნდა ჰქონდეთ. თავდაპირველად შეფასდება *გამოსახულება1*. თუ მისი მნიშვნელობაა true, მაშინ გაიცემა გამოსახულება*2*-ის მნიშვნელობა, წინააღმდეგ შემთხვევაში კი - გამოსახულება*3*-ის მნიშვნელობა. მაგალითად,

```
max = ricxvi1 > ricxvi2 ? ricxvi1 : ricxvi2;
min = ricxvi1 < ricxvi2 ? ricxvi1 : ricxvi2;
```

პირველი გამოსახულება ricxvi1 და ricxvi2 რიცხვებიდან ამოირჩევს მაქსიმალურს, მეორე კი - მინიმალურს.

ქვემოთ მოყვანილ პროგრამაში ხდება ორი რიცხვის გაყოფა ისე, რომ თავიდან ავიცილოთ ნულზე გაყოფა.

```
{
//      პროგრამა 3.25
// 50 იყოფა -3-დან 5-მდე მოთავსებულ რიცხვებზე ისე, რომ ხდება 0-ზე გაყოფის
თავიდან აცილება
int shedegi;
label1.Text = "";
for ( int indexi = -3; indexi < 5; indexi++ )
```

```
  {
        shedegi = indexi != 0 ? 50 / indexi : 0;
        if ( indexi != 0 ) label1.Text += indexi.ToString() + "    " + shedegi.ToString() + '\n';
  }
}
```

თავი 4. მასივები, ბიტობრივი ოპერაციები და ჩამოთვლები

მასივები

მასივი არის ერთი ტიპის მქონე ობიექტების კოლექცია (ერთობლიობა). მასივი გამოიყენება ერთნაირი ტიპის მქონე რამდენიმე ცვლადის ან ობიექტის შესანახად. მასივის ელემენტთან მიმართვისათვის უნდა მივუთითოთ მასივის სახელი და ელემენტის ინდექსი. მასივი შეიძლება იყოს როგორც ერთ, ისე მრავალგანზომილებიანი. ისინი მრავალი ამოცანის გადასაწყვეტად გამოიყენებიან, რადგან წარმოადგენენ ერთი ტიპის მქონე მონაცემების დაჯგუფების მოხერხებულ საშუალებას. მასივში შეგვიძლია შევინახოთ თანამშრომლების გვარები, მათი ხელფასები, ასაკი და ა.შ.

მასივები წარმოადგენენ მიმართვითი ტიპის მქონე ობიექტებს. მონაცემები მასივში ისეა ორგანიზებული, რომ ადვილია მათით მანიპულირება. სწორედ ამაშია მასივის მთავარი დანიშნულება.

ერთგანზომილებიანი მასივები

ერთგანზომილებიან მასივში კონკრეტული ელემენტის მდებარეობა განისაზღვრება ერთი ინდექსით. მისი სინტაქსია:

ტიპი [ ] მასივის_სახელი = new ტიპი[ზომა];

აქ *ტიპი* განსაზღვრავს მასივის ტიპს, ანუ მისი თითოეული ელემენტის ტიპს. ტიპის მარჯვნივ კვადრატული ფრჩხილები მიუთითებენ ერთგანზომილებიან მასივზე. *ზომა* არის მასივში ელემენტების რაოდენობა. რადგან მასივები რეალიზებულია ობიექტების სახით, ამიტომ მასივის შექმნის პროცესი ორი ეტაპისაგან შედგება. პირველ ეტაპზე ხდება მასივის სახელის გამოცხადება, რომელიც წარმოადგენს მიმართვითი ტიპის ცვლადის სახელს. მეორე ეტაპზე მასივისთვის მეხსიერება გამოიყოფა და მასივის ცვლადს მიენიჭება მეხსიერების ამ უბანზე მიმართვა (ანუ ამ უბნის მისამართი). C# ენაში new ოპერატორის გამოყენებით მასივს მეხსიერება დინამიურად (პროგრამის შესრულების დროს) გამოეყოფა. ქვემოთ მოყვანილ სტრიქონში იქმნება int ტიპის მასივი, რომელიც 10 ელემენტისაგან შედგება და მიმართვითი ტიპის masivi ცვლადს ენიჭება მასზე მიმართვა:

```
int [ ] masivi = new int[10];
```

masivi ცვლადი ინახავს მიმართვას მეხსიერების უბანზე (ანუ ამ უბნის მისამართს), რომელიც new ოპერატორის მიერ გამოიყო მისთვის. გამოყოფილი მეხსიერების ზომა საკმარისია მასში 10 ცალი int ტიპის ელემენტის მოსათავსებლად. რადგან int ტიპის ცვლადი 4 ბაიტს (32 ბიტს) იკავებს, ამიტომ masivi მასივისათვის მეხსიერებაში სულ გამოიყოფა 4x10=40 ბაიტი.

მასივის ცალკეულ ელემენტთან მიმართვა ხორციელდება ინდექსის გამოყენებით. მისი საშუალებით ეთითება ელემენტის პოზიცია მასივის ფარგლებში. ნებისმიერი მასივის პირველ ელემენტს აქვს ნულოვანი ინდექსი. რადგან masivi მასივი 10 ელემენტისაგან შედგება, ამიტომ ამ მასივის ინდექსის მნიშვნელობები მოთავსებული იქნება დიაპაზონში 0□9. შედეგად, masivi მასივის პირველი ელემენტი აღინიშნება ასე - masivi[0], უკანასკნელი კი - masivi[9].

ქვემოთ მოყვანილ პროგრამაში სრულდება masivi მასივის შევსება ციფრებით 0-დან 9-მდე და შემდეგ მათი გამოტანა label2 კომპონენტში.

```
{
//      პროგრამა 4.1
//      პროგრამაში ხდება  მასივის შევსება რიცხვებით და მათი label1 კომპონენტში გამოტანა
int[] masivi = new int[10];                     //      masivi მასივის გამოცხადება
int indexi;

label1.Text = "";
//      masivi მასივის შევსება რიცხვებით 0-დან 9-მდე
for ( indexi = 0; indexi < 10; indexi++ )
        masivi[indexi] = indexi;
//      masivi მასივის ელემენტების გამოტანა label1 კომპონენტში
for ( indexi = 0; indexi < 10; indexi++ )
        label1.Text += masivi[indexi].ToString() + " ";
}
```

მასივის ინიციალიზაცია

მასივის ინიციალიზაცია შესაძლებელია მისი შექმნისთანავე. ერთგანზომილებიანი მასივის ინიციალიზაციის სინტაქსია:

ტიპი [ ] მასივის_სახელი = new ტიპი [ ] { სიდიდე1, სიდიდე2, ..., სიდიდეN };

აქ საწყისი მნიშვნელობები მოიცემა *სიდიდე1*-დან *სიდიდეN*-მდე სიდიდეებით. მასივის ელემენტებს მნიშვნელობები ენიჭებათ რიგრიგობით მარცხნიდან მარჯვნივ დაწყებული ნულოვანი ინდექსის მქონე ელემენტიდან. ქვემოთ მოყვანილია პროგრამა, რომელშიც სრულდება მასივის ინიციალიზაცია. პროგრამა პოულობს masivi მასივის მაქსიმალურ და მინიმალურ ელემენტებს.

```
{
//      პროგრამა 4.2
//      პროგრამაში ხდება მასივის ინიციალიზება და
//      მისი მინიმალური და მაქსიმალური ელემენტების პოვნა
//      masivi მასივის ინიციალიზაცია
int[] masivi = new int[]{ 2, -5, 50, 15, -20, 25, 12, -1, 120, 10 };
int indexi, max, min;

max = min = masivi[0];
for ( indexi = 1; indexi < 10; indexi++ )
{
        if ( masivi[indexi] > max ) max = masivi[indexi];
        if ( masivi[indexi] < min ) min = masivi[indexi];
}
label1.Text = " მაქსიმალური მნიშვნელობა = "  + max.ToString() +
                "\n  მინიმალური მნიშვნელობა = " + min.ToString();
}
```

ჩვენ განვიხილეთ მთელრიცხვა მასივის მაგალითი. ბუნებრივია, შეგვიძლია შევქმნათ წილადი, ლოგიკური და ა.შ. ტიპის მასივიც. მაგალითად,

```
double[] wiladi_masivi = new double[5] { 1.2, 3.9, 4.7, 6.0, 8.6 };
bool[] logikuri_masivi = new bool[3] { true, false, true };
```

მასივის ინდექსის მნიშვნელობა არ უნდა გასცდეს ამ მასივის ინდექსის ზღვრულ მნიშვნელობებს, წინააღმდეგ შემთხვევაში აღიძვრება შესაბამისი შეცდომა.

მასივზე მიმართვის მინიჭება.

როცა მიმართვითი ტიპის ერთი ცვლადის მნიშვნელობა, რომელიც მასივს მიმართავს, ენიჭება ასეთივე ტიპის მეორე ცვლადს, მაშინ ამ მეორე ცვლადს ენიჭება მიმართვა იმავე მასივზე, რომელსაც პირველი ცვლადი მიმართავს. ამ დროს არ იქმნება მასივის ასლი და არ მოხდება ერთი მასივის გადაწერა მეორეში. მოყვანილ პროგრამაში ხდება მასივზე მიმართვის მინიჭების დემონსტრირება.

```
{
//      პროგრამა 4.3
//      პროგრამაში ხდება მასივზე მიმართვის მინიჭების დემონსტრირება
//      მასივების ინიციალიზება
int[] masivi1 = new int[]{0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
int[] masivi2 = new int[]{1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
int indexi;

for ( indexi = 0; indexi < 10; indexi++ )
        label1.Text += masivi1[indexi].ToString() + ”   ”;
for ( indexi = 0; indexi < 10; indexi++ )
        label2.Text += masivi2[indexi].ToString() + ”   ”;
//      masivi2 ცვლადს ენიჭება masivi1 მასივზე მიმართვა, რის შედეგად ორივე ცვლადი
//      ერთსა და იმავე მასივს მიმართავს
masivi2 = masivi1;
//      ეკრანზე გამოგვაქვს  masivi2
for ( indexi = 0; indexi < 10; indexi++ )
        label3.Text += masivi2[indexi].ToString() + ”   ”;
//      ამ მინიჭების შედეგად  masivi1[5] ელემენტშიც 55 ჩაიწერება
masivi2[5] = 55;
//      ეკრანზე გამოგვაქვს  masivi1
for ( indexi = 0; indexi < 10; indexi++ )
        label4.Text += masivi1[indexi].ToString() + ”   ”;
}
```

ორგანზომილებიანი მასივები

ორგანზომილებიან მასივებში კონკრეტული ელემენტის განლაგება განისაზღვრება ორი ინდექსით. გამოცხადების დროს პირველი ინდექსი მიუთითებს სტრიქონების რაოდენობას, მეორე კი - სვეტების რაოდენობას. მაგალითად, ორგანზომილებიანი მთელრიცხვა cxrili მასივი ზომით 5x4 შეგვიძლია შემდეგნაირად გამოვაცხადოთ:

```
int[,] cxrili = new int[5,4];
```

მასივის გამოცხადებისას ორივე განზომილება ერთმანეთისაგან მძიმით გამოიყოფა. ოპერატორის პირველ ნაწილში ფრჩხილებში მითითებული მძიმე

მიუთითებს, რომ იქმნება მიმართვითი ტიპის ცვლადი ორგანზომილებიანი მასივისათვის. new ოპერატორი დინამიურად გამოყოფს მეხსიერებას ორგანზომილებიანი მასივისათვის. ოპერატორის მეორე ნაწილში რიცხვი 5 განსაზღვრავს სტრიქონების რაოდენობას, რიცხვი 4 - კი სვეტების რაოდენობას. მეხსიერება გამოიყოფა 5x4=20 რიცხვისათვის. რადგან მთელი რიცხვი იკავებს 4 ბაიტს, ამიტომ სულ მეხსიერებაში გამოიყოფა 4x5x4=80 ბაიტი.

ორგანზომილებიანი მასივის კონკრეტულ ელემენტთან მიმართვისათვის აუცილებელია ორი ინდექსის მითითება, მაგალითად,

```
cxrili[3,2] = 15;
```

აქ რიცხვი 3 არის სტრიქონის ნომერი, რიცხვი 2 - კი სვეტის ნომერი.

ქვემოთ მოყვანილია პროგრამა, რომელიც ჯერ ავსებს ორგანზომილებიან მასივს მთელი რიცხვებით, შემდეგ კი ისინი label2 კომპონენტში გამოაქვს:

```
{
//      პროგრამა 4.4
//      პროგრამაში ხდება ორგანზომილებიანი მასივის შევსება რიცხვებით და
//      მათი label1 კომპონენტში გამოტანა
int[,] cxrili = new int[5,4];
int striqoni, sveti;

label1.Text = "";
for ( striqoni = 0; striqoni < 5; striqoni++ )
        for ( sveti = 0; sveti < 4; sveti++ )
                cxrili[striqoni,sveti] = ( striqoni * 4 ) + ( sveti + 1 );
for ( striqoni = 0; striqoni < 5; striqoni++ )
        {
                for ( sveti = 0; sveti < 4; sveti++ )
                        label1.Text += cxrili[striqoni,sveti].ToString() + " ";
                label1.Text += '\n';
        }
}
```

ორგანზომილებიანი მასივების ინიციალიზაცია

ორგანზომილებიანი მასივების ინიციალიზაციისათვის თითოეული განზომილების მნიშვნელობების სია უნდა მოვათავსოთ ცალკე ბლოკში, რომელიც, თავის მხრივ, ფიგურულ ფრჩხილებშია მოთავსებული. მაგალითად,

```
int[,] cxrili = new int[,]      {
                                        {1,1},
                                        {2,4},
                                        {3,9},
                                        {4,16},
                                        {5,25}
                                };
```

აქ ყოველი შიდა ბლოკი მნიშვნელობას ანიჭებს შესაბამისი სტრიქონის ელემენტებს. სტრიქონის ფარგლებში პირველი მნიშვნელობა მოთავსებული იქნება სტრიქონის ნულოვან ელემენტში, მეორე - პირველ ელემენტში და ა.შ. ბლოკები

ერთმანეთისაგან მძიმეებით გამოიყოფა, ხოლო დამხურავი ფიგურული ფრჩხილის შემდეგ მოთავსებულია წერტილ-მძიმე. ქვემოთ მოყვანილ პროგრამაში ხდება ორგანზომილებიანი მასივის შევსება რიცხვებით და მისი ელემენტების ჯამის გამოთვლა.

```
{
//      პროგრამა 4.5
//      პროგრამა გამოთვლის ორგანზომილებიანი მასივის ელემენტების ჯამს
int striqoni, sveti, jami = 0;

label1.Text = "";
// masivi მასივის ინიციალიზაცია
int[,] masivi = new int[,]      {
                                        {1,1},
                                        {2,4},
                                        {3,9},
                                        {4,16},
                                        {5,25}
                                };
// მასივის ელემენტებისა და მათი ჯამის ეკრანზე გამოტანა
for ( striqoni = 0; striqoni < 5; striqoni++ )
{
        for ( sveti = 0; sveti < 2; sveti++ )
                {
                        jami += masivi[striqoni,sveti];
                        label1.Text += masivi[striqoni,sveti].ToString() + " ";
                }
                label1.Text += '\n';
}
label2.Text = jami.ToString();
}
```

გაუსწორებელი (“დაკბილული”) მასივები

აქამდე ჩვენ განვიხილავდით ორგანზომილებიან მასივებს, რომლებსაც სხვანაირად სწორკუთხა მასივები ეწოდება. სწორკუთხა მასივი არის ორგანზომილებიანი მასივი, რომელშიც სტრიქონების სიგრძე ერთნაირია. C# ენაში შეგვიძლია შევქმნათ ორგანზომილებიანი მასივის სპეციალური ტიპი, რომელსაც გაუსწორებელი მასივი (არასწორკუთხა ან “დაკბილული” მასივი) ეწოდება. ესაა გარე მასივი, რომელიც შედგება სხვადასხვა სიგრძის შიდა მასივებისაგან. შედეგად, გაუსწორებელი მასივი შეგვიძლია გამოვიყენოთ ისეთი ცხრილის შესაქმნელად, რომელიც სხვადასხვა სიგრძის სტრიქონებს შეიცავს.

ორგანზომილებიანი გაუსწორებელი მასივის გამოცხადების სინტაქსია:

ტიპი[ ][ ] მასივის_სახელი = new ტიპი[ზომა][ ];

აქ *ზომა* არის მასივში სტრიქონების რაოდენობა. სვეტების რაოდენობა არ

ეთითება და მეხსიერება მათთვის არ გამოიყოფა. თითოეული სტრიქონის შესაქმნელად აუცილებელია ცალკე new ოპერატორის შესრულება. ასეთი ტექნოლოგია იძლევა სხვადასხვა სიგრძის სტრიქონების განსაზღვრის საშუალებას. მაგალითად, პროგრამის ქვემოთ მოყვანილ ფრაგმენტში masivi_g მასივის გამოცხადებისას მეხსიერება მხოლოდ გარე მასივისათვის გამოიყოფა, შემდეგ კი მომდევნო სამი ოპერატორის შესრულების შედეგად მეხსიერება ცალცალკე გამოეყოფა თითოეულ შიდა მასივს:

```
int[ ][ ] masivi_g = new int[3][ ];
masivi_g[0] = new int[2];
masivi_g[1] = new int[3];
masivi_g[2] = new int[4];
```

შედეგად, masivi_g შემდეგ სახეს მიიღებს:

```
masivi_g[0][0]    masivi_g[0][1]
masivi_g[1][0]    masivi_g[1][1]    masivi_g[1][2]
masivi_g[2][0]    masivi_g[2][1]    masivi_g[2][2]    masivi_g[2][3]
```

გაუსწორებელი მასივის შექმნის შემდეგ მის ელემენტებთან მიმართვა შესაძლებელია თითოეული ინდექსის ცალკე კვადრატულ ფრჩხილებში მითითებით. მაგალითად,

```
masivi_g[1][2] = 15;
```

ქვემოთ მოყვანილ პროგრამაში ხდება გაუსწორებელი ორგანზომილებიანი მასივის შევსება და label2 კომპონენტში გამოტანა.

```
{
//      პროგრამა 4.6
//      პროგრამაში ხდება გაუსწორებელ ორგანზომილებიან მასივთან მუშაობის
დემონსტრირება
int [][] g_masivi = new int[5][];
int striqoni, sveti;

label1.Text = "";
g_masivi[0] = new int[5];
g_masivi[1] = new int[5];
g_masivi[2] = new int[5];
g_masivi[3] = new int[2];
g_masivi[4] = new int[2];
//      g_masivi მასივის შევსება
for ( striqoni = 0; striqoni < 3; striqoni++ )
        for ( sveti = 0; sveti < 5; sveti++ )
                g_masivi[striqoni][sveti] = striqoni + sveti;
for ( striqoni = 3; striqoni < 5; striqoni++ )
        for ( sveti = 0; sveti < 2; sveti++ )
                g_masivi[striqoni][sveti] = striqoni + sveti;
//      g_masivi მასივის label კომპონენტში გამოტანა
for ( striqoni = 0; striqoni < 3; striqoni++ )
{
        for ( sveti = 0; sveti < 5; sveti++ )
          label1.Text += g_masivi[striqoni][sveti].ToString() + " ";
```

```
        label1.Text += '\n';
}
for ( striqoni = 3; striqoni < 5; striqoni++ )
{
        for ( sveti = 0; sveti < 2; sveti++ )
          label1.Text += g_masivi[striqoni][sveti].ToString() + " ";
        label1.Text += '\n';
}
}
```

გაუსწორებელი მასივები იშვიათად გამოიყენება. მაგრამ, რიგ შემთხვევებში მათი გამოყენება საკმაოდ ეფექტურია. მაგალითად, როცა გვაქვს დიდი ზომის ორგანზომილებიანი მასივი, რომელიც ნაწილობრივ იქნება შევსებული.

Length თვისება

C# ენაში მასივები რეალიზებულია როგორც ობიექტები, რაც იძლევა გარკვეულ უპირატესობებს. ერთ-ერთია Length თვისების არსებობა. ყოველ მასივს აქვს Length თვისება. მასში ავტომატურად იწერება მასივის ელემენტების მაქსიმალური რაოდენობა. იგი არ ინახავს ფაქტიურად არსებული ელემენტების რაოდენობას. კერძოდ, თუ გვაქვს 10 ელემენტიანი მასივი და შევსებულია მისი პირველი 4 ელემენტი, მაშინ Length თვისების მნიშვნელობა იქნება 10 და არა 4. Length თვისება წარმატებით გამოიყენება მასივის ინდექსების მნიშვნელობების საზღვრებს გარეთ გასვლის თავიდან ასაცილებლად.

მოყვანილ პროგრამაში Length თვისება გამოიყენება ერთგანზომილებიან მასივთან სამუშაოდ.

```
{
// პროგრამა 4.7
// პროგრამაში ხდება Length თვისების გამოიყენება ერთგანზომილებიან მასივთან სამუშაოდ
// ერთგანზომილებიანი მასივის გამოცხადება
int[] list = new int[5];
// Length თვისება გამოიყენება list მასივის ინიციალიზაციისათვის
for ( int indexi = 0; indexi < list.Length; indexi++ )
        list[indexi] = indexi * indexi;
for ( int indexi = 0; indexi < list.Length; indexi++ )
        label1.Text += list[indexi].ToString() + "   ";
label2.Text = "მასივის  სიგრძე     =  " + list.Length.ToString();
}
```

ქვემოთ მოყვანილ პროგრამაში Length თვისება გამოიყენება გაუსწორებელ მასივთან სამუშაოდ.

```
{
// პროგრამა 4.8
// პროგრამაში ხდება Length თვისების გამოიყენება გაუსწორებელ მასივთან სამუშაოდ
// გაუსწორებელი მასივის გამოცხადება
```

```
int[][] cxrili = new int[3][];
//        შიდა მასივებისათვის მეხსიერების გამოყოფა
cxrili[0] = new int[] {1, 2, 3};
cxrili[1] = new int[] {4, 5};
cxrili[2] = new int[] {6, 7, 8, 9};
label2.Text = "მასივის  სიგრძე      =  " + cxrili.Length.ToString();
label3.Text = "ელემენტის სიგრძე   =  " + cxrili[0].Length.ToString();
label4.Text = "ელემენტის სიგრძე   =  " + cxrili[1].Length.ToString();
label5.Text = "ელემენტის სიგრძე   =  " + cxrili[2].Length.ToString();
}
```

ყურადღება მივაქციოთ Length თვისების გამოყენებას გაუსწორებელ ორგანზომილებიან cxrili მასივში. როგორც აღვნიშნეთ, გაუსწორებელი ორგანზომილებიანი მასივი - ესაა მასივების მასივი. შედეგად, cxrili.Length გამოსახულების გამოყენების შემთხვევაში ჩვენ მივიღებთ cxrili მასივში შემავალი შიდა მასივების რაოდენობას. შიდა მასივის სიგრძის მისაღებად უნდა გამოვიყენოთ შემდეგი გამოსახულება:

```
cxrili[0].Length;
```

ამ შემთხვევაში მივიღებთ იმ მასივის სიგრძეს (ელემენტების რაოდენობას), რომლის ინდექსია 0.

foreach ციკლი

foreach ციკლი გამოიყენება კოლექციის ელემენტების დასამუშავებლად. კოლექცია არის ობიექტების ჯგუფი. C# ენაში განსაზღვრულია კოლექციების რამდენიმე ტიპი, ერთ-ერთი მათგანია მასივი. foreach ციკლის სინტაქსია:

```
foreach ( ტიპი ცვლადის_სახელი  in კოლექცია )
{
ციკლის ტანი
}
```

აქ ტიპი არის იტერაციული ცვლადის ტიპი და უნდა ემთხვეოდეს კოლექციის ტიპს. ცვლადის_სახელი არის იტერაციული ცვლადის სახელი. იტერაციული ცვლადი რიგ-რიგობით იღებს კოლექციის ელემენტების მნიშვნელობებს. კოლექცია შეიძლება იყოს მასივი, სტრიქონი და ა.შ. იტერაციული ცვლადი მისაწვდომია მხოლოდ წაკითხვის რეჟიმში, ამიტომ, ვერ შევძლებთ მასივის შეცვლას იტერაციული ცვლადისთვის ახალი მნიშვნელობის მინიჭების გზით.

ქვემოთ მოყვანილია foreach ციკლის მაგალითი. თავიდან იქმნება მთელრიცხვა მასივი და სრულდება მისი ინიციალიზაცია. შემდეგ, ეს მნიშვნელობები გამოიცემა label კომპონენტში და გამოითვლება მათი ჯამი.

```
{
//        პროგრამა 4.9
//        პროგრამაში გამოითვლება მასივის ელემენტების ჯამი foreach ციკლის
გამოყენებით
int jami = 0;
int[] masivi = new int[]{ 15, 10, -2, 3, -8 };
```

```
foreach ( int indexi in masivi )
{
	label1.Text += indexi.ToString() + '\n';
	jami += indexi;
}
label2.Text = jami.ToString();
}
```

როგორც ვხედავთ მასივის ელემენტების დამუშავება ხდება მიმდევრობით, დაწყებული პირველი ელემენტიდან, ინდექსის გამოყენების გარეშე.

foreach ციკლის საშუალებით მასივი შეიძლება დავამუშავოთ მხოლოდ დასაწყისიდან ბოლოსკენ. ქვემოთ მოყვანილია მასივის მაქსიმალური და მინიმალური ელემენტების პოვნის პროგრამა foreach ციკლის გამოყენებით.

```
{
// პროგრამა 4.10
// პროგრამაში ხდება მასივის მაქსიმალური და მინიმალური ელემენტების პოვნა
// foreach ციკლის გამოყენებით
int max, min;
int[] masivi = new int[]{9, -1, 101, 8, -9, 56, 3, -2, 87, 81};

label1.Text = "";
max = min = masivi[0];
// მასივის ელემენტების ეკრანზე გამოტანა
foreach ( int ricxvi in masivi )
	label1.Text += ricxvi.ToString() + " ";

foreach ( int ricxvi in masivi )
{
	if ( ricxvi > max ) max = ricxvi;
	if ( ricxvi < min ) min = ricxvi;
}
label2.Text = max.ToString();
label3.Text = min.ToString();
}
```

ქვემოთ მოყვანილ პროგრამაში foreach ციკლი გამოიყენება ორგანზომილებიანი მასივის ელემენტების ჯამის გამოსათვლელად.

```
{
// პროგრამა 4.11
// პროგრამაში ხდება ორგანზომილებიანი მასივის ელემენტების ჯამის გამოთვლა
int jami = 0;
int[,] cxrili = new int[,]{
				{ 1, 2, -3 },
				{ -4, 5, 6 },
				{ 7, -8, 9 },
			};
```

```
label1.Text = "";
foreach ( int ricxvi in cxrili )
{
        label1.Text += ricxvi.ToString() + "   ";
        jami += ricxvi;
}
label2.Text = jami.ToString();
}
```

ბიტობრივი ოპერატორები

ბიტობრივი (თანრიგობრივი) ოპერატორების ოპერანდები მთელი რიცხვებია. ეს ოპერატორები არ შეიძლება გამოვიყენოთ bool, float ან double ტიპის მნიშვნელობების, აგრეთვე, კლასის ობიექტების მიმართ. ასეთ ოპერატორებს ბიტობრივი ეწოდება იმიტომ, რომ გამოიყენებიან მთელი რიცხვების ბიტების შესამოწმებლად, დასაყენებლად (1 მდგომარეობაში გადაყვანა) და ძვრისათვის. ბიტობრივი ოპერაციები ხშირად გამოიყენება სისტემურ დონეზე პროგრამირებისათვის, მაგალითად, როცა საჭიროა ინფორმაციის მიღება მოწყობილობის მდგომარეობის შესახებ და ა.შ. ბიტობრივი ოპერაციები მოყვანილია ცხრილში 4.1.

ცხრილი 4.1. ბიტობრივი ოპერაციები

ოპერატორი	აღწერა
&	ბიტობრივი ოპერატორი AND
|	ბიტობრივი ოპერატორი OR
^	ბიტობრივი ოპერატორი XOR
>>	მარჯვნივ ძვრის ოპერატორი
<<	მარცხნივ ძვრის ოპერატორი
~	უნარული ოპერატორი NOT

&, |, ^ და ~ ბიტობრივი ოპერატორები

&, |, ^ და ~ ბიტობრივი ოპერატორები ისეთივე ოპერაციებს ასრულებენ, როგორსაც მათი ბულის ეკვივალენტები. განსხვავება იმაშია, რომ ბიტობრივი ოპერატორები ოპერანდებზე ზემოქმედებენ ბიტების დონეზე. ცხრილში 4.2 მოყვანილია თითოეული ოპერაციის შესრულების შედეგები.

ცხრილი 4.2. ბიტობრივი ოპერაციები.

b1	b2	b1&b2	b1 | b2	b1 ^ b2	~b1
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

ბიტობრივი & ოპერაცია შეგვიძლია გამოვიყენოთ როგორც ბიტების განულების (ჩამოგდების) საშუალება. ეს იმას ნიშნავს, რომ თუ რომელიმე ოპერანდის ერთ-ერთ ბიტს აქვს მნიშვნელობა 0, მაშინ შედეგის შესაბამის ბიტსაც ექნება მნიშვნელობა 0. მაგალითად,

```
 11010011
        &
 10101010
 ---------
 10000010
```

მოყვანილ პროგრამაში ნაჩვენებია & ოპერატორის გამოყენების მაგალითი. პროგრამა ქვედა რეგისტრის სიმბოლოებს გარდაქმნის ზედა რეგისტრის სიმბოლოებად მეექვსე ბიტის განულების გზით. Unicode/ASCII სიმბოლოების სტანდარტულ ნაკრებში ქვედა რეგისტრის სიმბოლოების მნიშვნელობა მეტია ზედა რეგისტრის სიმბოლოების შესაბამის მნიშვნელობაზე 32 ერთეულით ($32_{10}=00100000_2$). შედეგად, ქვედა რეგისტრის სიმბოლოების გადასაყვანად ზედა რეგისტრში საკმარისია მეექვსე ბიტის განულება.

```
{
//      პროგრამა 4.12
//      პროგრამაში ხდება ქვედა რეგისტრის სიმბოლოების გარდაქმნა ზედა რეგისტრის სიმბოლოებად
char simbolo;

for ( int indexi = 0; indexi < 10; indexi++ )
  {
        simbolo = (char) ( 'a' + indexi );
        label1.Text += simbolo + " - ";
        simbolo = (char) ( simbolo & 65503 );
        label1.Text = label1.Text + simbolo + '\n';
  }
}
```

რიცხვს 65503 აქვს შემდეგი ორბაიტიანი ორობითი წარმოდგენა 1111 1111 1101 1111. შედეგად, & ოპერაცია უცვლელად ტოვებს simbolo ცვლადის ყველა ბიტს მეექვსე ბიტის გარდა, რომელიც განულდება. & ოპერაცია შეგვიძლია გამოვიყენოთ, როცა საჭიროა განისაზღვროს ბიტი დაყენებულია (1) თუ ჩამოგდებული (0). მაგალითად, ქვემოთ მოყვანილ ოპერატორში მოწმდება status ცვლადის მეოთხე ბიტის მნიშვნელობა:

```
if ( status & 8 == true ) label2.Text = "მეოთხე ბიტი დაყენებულია";
```

რიცხვი 8 გამოიყენება იმიტომ, რომ მის ორობით წარმოდგენაში დაყენებულია მხოლოდ მეოთხე ბიტი (0000 1000). შედეგად, if ოპერატორი true შედეგს მხოლოდ მაშინ გასცემს, როცა status ცვლადის მნიშვნელობაში მეოთხე ბიტი იქნება დაყენებული. ქვემოთ მოყვანილ პროგრამაში ასეთი მიდგომა გამოიყენება ეკრანზე byte ტიპის მნიშვნელობის ორობითი წარმოდგენის გამოსატანად.

```
{
//      პროგრამა 4.13
//      ათობითი რიცხვის გარდაქმნა ორობით რიცხვად
int indeqsi;
byte ricxvi = Convert.ToByte(textBox1.Text);
label1.Text = " ";
```

```
for ( indeqsi = 128; indeqsi > 0; indeqsi /= 2 )
  {
        if ( ( ricxvi & indeqsi ) != 0 ) label1.Text += "1";
              else label1.Text += "0";
  }
}
```

for ციკლში & ოპერატორის გამოყენებით მოწმდება ricxvi ცვლადის თითოეული ბიტი. თუ იგი დაყენებულია, მაშინ label2 კომპონენტში გამოჩნდება 1, თუ ჩამოგდებულია - 0.

& ოპერატორის საწინააღმდეგოდ | ოპერატორი შეგვიძლია გამოვიყენოთ ბიტების დასაყენებლად. თუ ერთ-ერთ ოპერანდში ბიტი დაყენებულია, მაშინ შედეგში შესაბამისი ბიტი იქნება დაყენებული. მაგალითად,

```
11010011
      |
10101010
--------
11111011
```

ზედა რეგისტრის სიმბოლოების გარდასაქმნელად ქვედა რეგისტრის სიმბოლოებად გამოვიყენოთ | ოპერატორი, როგორც ეს ქვემოთაა ნაჩვენები.

```
{
//      პროგრამა 4.14
//      პროგრამაში ხდება ზედა რეგისტრის სიმბოლოების გარდაქმნა ქვედა რეგისტრის სიმბოლოებად
char simbolo;

for ( int indeqsi = 0; indeqsi < 10; indeqsi++ )
  {
        simbolo = (char) ( 'A' + indeqsi );
        label1.Text += simbolo + " - ";
//      ამ ოპერატორის შესრულება აყენებს მეექვსე ბიტს
//      შედეგად simbolo ცვლადში ჩაიწერება ქვედა რეგისტრის სიმბოლო
        simbolo = (char) ( simbolo | 32 );
        label1.Text = label1.Text + simbolo + '\n';
  }
}
```

| ოპერაცია გამოიყენება თითოეული სიმბოლოს მნიშვნელობის მიმართ და 32 მნიშვნელობის მიმართ, რომლის ორობითი წარმოდგენაა 0000 0000 0010 0000. ამ ორობით წარმოდგენაში დაყენებულია მხოლოდ მეექვსე ბიტი. თუ | ოპერაციის ერთი ოპერანდია რიცხვი 32 და მეორე ოპერანდი - ნებისმიერი რიცხვი, მაშინ შედეგად მივიღებთ მნიშვნელობას, რომელშიც დაყენებული იქნება მეექვსე ბიტი. დანარჩენი ბიტები დარჩება უცვლელი.

^ ოპერაციის შესრულების შედეგად ბიტის დაყენება მოხდება მხოლოდ მაშინ, როცა ორივე ბიტი განსხვავებულია. მაგალითად,

11010011
^
10101010
11111011

^ ოპერატორს საკმაოდ საინტერესო თვისება აქვს, რომელიც საშუალებას გვაძლევს იგი გამოვიყენოთ შეტყობინების ან რაიმე ინფორმაციის დასაშიფრად. თუ ამ ოპერატორს გამოვიყენებთ x და y ცვლადების მიმართ და შემდეგ მიღებული შედეგისა და y ცვლადის მიმართ, მაშინ მივიღებთ x მნიშვნელობას, ე.ი.

R1 = x ^ y;
R2 = R1 ^ y;

ოპერატორების შესრულების შედეგად R2 ცვლადს მიენიჭება x ცვლადის მნიშვნელობა. ქვემოთ მოყვანილ პროგრამაში ეს პრინციპია გამოყენებული შეტყობინების შიფრაციისა და დეშიფრაციისათვის. პირველად ^ ოპერატორი გამოიყენება შიფრაციისათვის, მეორედ კი - დეშიფრაციისათვის. შედეგად, მივიღებთ საწყის ტექსტს. შიფრაციისას რაიმე მთელი რიცხვი შეგვიძლია გამოვიყენოთ როგორც გასაღები. ქვემოთ მოყვანილ პროგრამაში ხდება შიფრაციისა და დეშიფრაციის დემონსტრირება.

```
{
//      პროგრამა 4.15
//      პროგრამაში ხდება შიფრაციისა და დეშიფრაციის დემონსტრირება
string msg = "შეტყობინების შიფრაცია";string encmsg = "";
string decmsg = "";
int key = 55;

label1.Text = "დასაშიფრი შეტყობინება:   " + msg;
//      შეტყობინების შიფრაცია
for ( int indeqsi = 0; indeqsi < msg.Length; indeqsi++ )
        encmsg += (char) ( msg[indeqsi] ^ key );
label2.Text = " დაშიფრული შეტყობინება:   " + encmsg;
//      შეტყობინების დეშიფრაცია
for ( int indeqsi = 0; indeqsi < msg.Length; indeqsi++ )
        decmsg += (char) ( encmsg[indeqsi] ^ key );
label3.Text = "გაშიფრული შეტყობინება:   " + decmsg;
}
```

უნარული ~ ოპერაცია ცვლის ოპერანდის ყველა ბიტს. მაგალითად, თუ A = 1001 0011, მაშინ ~A ოპერაციის შედეგი იქნება 0110 1100.

ქვემოთ მოყვანილ პროგრამაში ხდება ~ ოპერატორის გამოყენების დემონსტრირება. ეკრანზე ჯერ ხდება რიცხვი 35-ის (0010 0011) ორობითი წარმოდგენის გამოტანა, შემდეგ კი ამ რიცხვის მიმართ ~ ოპერაციის გამოყენების შედეგის გამოტანა (1101 1100).

```
{
//      პროგრამა 4.16
//      პროგრამაში ხდება ~ ოპერატორის მუშაობის დემონსტრირება
sbyte ricxvi = 35;
//      ricxvi ცვლადის ორობითი წარმოდგენის label კომპონენტში გამოტანა
```

```
for ( int indeqsi = 128; indeqsi > 0; indeqsi /= 2 )
        if ( ( ricxvi & indeqsi ) != 0 ) label1.Text += " 1";
                else  label1.Text += " 0";
//      ricxvi ცვლადის ორობითი წარმოდგენის ინვერსია და label კომპონენტში გამოტანა
ricxvi = (sbyte) ~ricxvi;
for ( int indeqsi = 128; indeqsi > 0; indeqsi /= 2 )
        if ( ( ricxvi & indeqsi ) != 0 ) label2.Text += " 1";
                else  label2.Text += " 0";
}
```

ყველა ორობითი ბიტობრივი ოპერატორი შეგვიძლია ჩავწეროთ შედგენილი ბიტობრივი ოპერატორების სახით. მაგალითად,

```
x ^= 127;
y &= 63;
z |= 31;
```

ძვრის ოპერატორები

არსებობს ოპერანდების მარცხნივ და მარჯვნივ ძვრის ოპერატორები. მათი სინტაქსია:

ცვლადის_სახელი << *ბიტების_რაოდენობა*
ცვლადის_სახელი >> *ბიტების_რაოდენობა*

მარცხნივ ძვრა იწვევს ყველა ბიტის გადაადგილებას ერთი ბიტით მარცხნივ, ამასთან, თავისუფლდება უმცროსი ბიტები, რომლებიც ნულებით ივსება, ხოლო უფროსი ბიტების შესაბამისი რაოდენობა იკარგება. მარჯვნივ ძვრა იწვევს ყველა ბიტის ერთი ბიტით მარჯვნივ გადაადგილებას, ამასთან, თუ მარჯვნივ იძვრის უნიშნო მნიშვნელობის ბიტები, მაშინ უფროსი თანრიგები ნულებით შეივსება და უმცროსი თანრიგების შესაბამისი რაოდენობა იკარგება. ნიშნიანი მნიშვნელობის მარჯვნივ ძვრისას ხდება ნიშნის ბიტის შენახვა. ნიშნიანი დადებითი რიცხვებისთვის უფროსი ბიტი არის 0, უარყოფითი რიცხვებისათვის - 1.

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია ძვრის ოპერაციების გამოყენების მაგალითი.

```
{
//      პროგრამა 4.17
//      პროგრამაში ხდება ძვრის ოპერატორებთან მუშაობის დემონსტრირება
 int ricxvi = 1;

label1.Text = "";
label2.Text = "";
//      ricxvi ცვლადის ორობითი წარმოდგენის label კომპონენტში გამოტანა
for ( int i = 0; i < 8; i++ )
  {
        for ( int j = 128; j > 0; j /= 2 )
                if ( ( ricxvi & j ) != 0 ) label1.Text += "1";
                        else label1.Text += "0";
        label1.Text += '\n';
        ricxvi = ricxvi << 1;           //      ძვრა მარცხნივ ერთი ბიტით
```

```
  }
ricxvi = 128;
//      ricxvi ცვლადის ორობითი წარმოდგენის label კომპონენტში გამოტანა
for ( int i = 0; i < 8; i++ )
  {
        for ( int j = 128; j > 0; j /= 2 )
              if ( ( ricxvi & j ) != 0 ) label2.Text += "1";
                    else label2.Text += "0";
        label2.Text += '\n';
        ricxvi = ricxvi >> 1;   // ძვრა მარჯვნივ ერთი ბიტით
  }
}
```

ჩამოთვლები

ჩამოთვლები წარმოადგენენ სახელდებული მთელრიცხვა მუდმივების ნაკრებს, რომლებიც განსაზღვრავენ ცვლადის ყველა დასაშვებ მნიშვნელობას მოცემული ტიპისთვის.

ჩამოთვლადი ტიპის განსაზღვრისათვის გამოიყენება enum საკვანძო სიტყვა. მისი სინტაქსია:

enum *ჩამოთვლის_სახელი* { *მუდმივების_სია* };

ქვემოთ მოყვანილია xili ჩამოთვლის განსაზღვრის მაგალითი:

```
enum xili { vaSli, msxali, leRvi, nesvi, yurZeni };
```

ჩამოთვლასთან მუშაობისას მთავარია ის ფაქტი, რომ თითოეულ მუდმივას შეესაბამება მთელრიცხვა მნიშვნელობა. შედეგად, მუდმივების გამოყენების სფერო ემთხვევა მთელრიცხვა მნიშვნელობების (int ტიპი) გამოყენების სფეროს. მთელრიცხვა მნიშვნელობა თითოეული მუდმივასთვის მეტია წინა მუდმივას მნიშვნელობაზე. პირველი მუდმივას მნიშვნელობა ნულის ტოლია.

ჩამოთვლის ელემენტთან მიმართვისათვის ჯერ უნდა მივუთითოთ ჩამოთვლის სახელი, შემდეგ წერტილი (.) და ელემენტის სახელი.

ქვემოთ მოყვანილია პროგრამის მაგალითი, რომელიც ახდენს xili ჩამოთვლასთან მუშაობის დემონსტრირებას.

```
//      პროგრამა 4.18
//      პროგრამაში ხდება ჩამოთვლასთან მუშაობის დემონსტრირება
//      xili ჩამოთვლის გამოცხადება
enum xili { vaSli, msxali, leRvi, nesvi, yurZeni };
private void button1_Click(object sender, System.EventArgs e)
{
//      saxelebi სტრიქონების მასივის გამოცხადება
string[ ] saxelebi = { "ვაშლი", "მსხალი", "ლეღვი", "ნესვი", "ყურძენი" };
xili elementi;                  //      ცვლადის გამოცხადება ჩამოთვლის დამუშავებისათვის
label1.Text = " ";
```

```
//      elementi ცვლადის გამოყენება ჩამოთვლის დამუშავების ციკლისათვის
for ( elementi = xili.vaSli; elementi <= xili.yurZeni; elementi++ )
        label1.Text += saxelebi[(int)elementi] + "-is  mniSvnelobaa  -    " +
                        (int)elementi + '\n';
}
```

პროგრამაში for ციკლის მართვა ხორციელდება elementi ცვლადის საშუალებით, რომელსაც აქვს xili ტიპი. რადგან ჩამოთვლას აქვს მთელრიცხვა ტიპი, ამიტომ, ჩამოთვლის მნიშვნელობების მინიჭება შეიძლება იქ, სადაც შეიძლება მთელრიცხვა მნიშვნელობების გამოყენება. მაგრამ, ჩამოთვლის მნიშვნელობების გამოყენებისას saxelebi მასივის ინდექსაციის მიზნით საჭირო ხდება ტიპის გარდაქმნა.

ჩამოთვლების ინიციალიზება

ინიციალიზატორის საშუალებით შეგვიძლია მივუთითოთ ერთი ან მეტი მუდმივას მნიშვნელობა. ამისათვის, თითოეული მუდმივას შემდეგ უნდა მოვათავსოთ მინიჭების ოპერატორი და მთელრიცხვა მნიშვნელობა. ინიციალიზატორის მიერ განსაზღვრული მნიშვნელობები უნდა იყოს მეტი ინიციალიზაციის წინა მნიშვნელობაზე. მაგალითად, მოყვანილ გამოცხადებაში nesvi მუდმივას ენიჭება მნიშვნელობა 55:

```
enum xili { vaSli, msxali, leRvi, nesvi = 55, yurZeni } ;
```

ამის შემდეგ მუდმივებს ექნებათ შემდეგი მნიშვნელობები:

vaSli	0
msxali	1
leRvi	2
nesvi	55
yurZeni	56

თავი 5. კლასები, ინკაფსულაცია, მეთოდები

კლასის გამოცხადება

კლასი არის ობიექტზე ორიენტირებული პროგრამირების ძირითადი სტრუქტურა. მის საფუძველზე იქმნება ობიექტები. ობიექტი შეიძლება იყოს ავტომობილი, თვითმფრინავი, მაღაზია, გეომეტრიული ფიგურა და ა.შ. ობიექტები წარმოადგენენ კლასის ეგზემპლარებს. განსხვავება კლასსა და ობიექტს შორის იმაში მდგომარეობს, რომ კლასი არის ლოგიკური აბსტრაქცია, ობიექტი კი - ამ კლასის კონკრეტული რეალიზაცია კომპიუტერის მეხსიერებაში.

კლასში შემავალ მეთოდებსა და ცვლადებს (ველებს) კლასის წევრები (ელემენტები) ეწოდებათ. C# ენაში არსებობს კლასის წევრების რამდენიმე სახესხვაობა: ეგზემპლარის ცვლადები, სტატიკური ცვლადები, მუდმივები, მეთოდები, კონსტრუქტორები, დესტრუქტორები, ინდექსატორები, მოვლენები, ოპერატორები და თვისებები. ამ ეტაპზე განვიხილავთ კლასის ძირითად ელემენტებს - ცვლადებს და მეთოდებს.

კლასის სინტაქსი, რომელიც შეიცავს მხოლოდ ცვლადებსა და მეთოდებს ასეთია:

```
მიმართვის_მოდიფიკატორი class კლასის_სახელი
{
// ეგზემპლარის ცვლადების გამოცხადება
მიმართვის_მოდიფიკატორი ტიპი ცვლადის_სახელი1;
// . . .
მიმართვის_მოდიფიკატორი ტიპი ცვლადის_სახელიN;
// მეთოდების გამოცხადება
მიმართვის_მოდიფიკატორი შედეგის_ტიპი მეთოდის_სახელი1(პარამეტრები)
{
მეთოდის კოდი
}
// . . .
მიმართვის_მოდიფიკატორი შედეგის_ტიპი მეთოდის_სახელიN(პარამეტრები)
{
მეთოდის კოდი
}
}
```

class საკვანძო სიტყვა იწყებს კლასის გამოცხადებას. მას კლასის სახელი მოსდევს. *მიმართვის_მოდიფიკატორი* იღებს private (პრივატული) ან public (საერთოწვდომის) მნიშვნელობებს. თითოეული ცვლადისა და მეთოდის გამოცხადება იწყება ამ სიტყვებიდან ერთ-ერთით. ისინი განსაზღვრავენ თუ როგორ უნდა მოხდეს ცვლადთან ან მეთოდთან მიმართვა. private მოდიფიკატორი მიუთითებს, რომ კლასის წევრთან მიმართვა შეუძლიათ მხოლოდ ამ კლასის წევრებს. public მოდიფიკატორი მიუთითებს, რომ კლასის წევრთან მიმართვა შესაძლებელია კლასის გარეთ აღწერილი კოდიდან. თუ მოდიფიკატორი მითითებული არ არის, მაშინ იგულისხმება private. მოდიფიკატორის შემდეგ ეთითება ცვლადის ტიპი და სახელი. შედეგის_ტიპი არის მეთოდის მიერ გაცემული შედეგის ტიპი. მას მოსდევს მეთოდის სახელი და პარამეტრების სია.

კორექტულად დაწერილ პროგრამაში კლასი განსაზღვრავს მხოლოდ ერთ ლოგიკურ ერთეულს. მაგალითად, კლასი, რომელიც შეიცავს ინფორმაციას ავტომანქანების შესახებ, არ უნდა შეიცავდეს სხვა ინფორმაციას, მაგალითად, შენობების ან ცხოველების შესახებ.

ინკაფსულაცია

კლასი ორ ძირითად ფუნქციას ასრულებს, რომლებიც უზრუნველყოფენ მონაცემების ინკაფსულაციას. ჯერ ერთი, იგი მონაცემებს აკავშირებს კოდთან, რომელიც მათზე მანიპულირებს. მეორეც, კლასი უზრუნველყოფს კლასის წევრებთან მიმართვის საშუალებებს. ინკაფსულაცია არის კლასის წევრებთან მიმართვის უფლებამოსილების გამიჯვნა.

როგორც აღვნიშნეთ, არსებობს კლასის წევრების ორი ძირითადი ტიპი - ღია (public) და დახურული (private). public ტიპის წევრებთან მიმართვა შეიძლება შესრულდეს ამ კლასის გარეთ განსაზღვრული კოდიდან. private ტიპის მქონე წევრებთან მიმართვა შეუძლიათ მხოლოდ ამავე კლასის მეთოდებს. ამ კლასის გარეთ განსაზღვრულ კოდს მხოლოდ ამავე კლასის ღია მეთოდების გამოყენებით შეუძლია მიმართოს ამ კლასის დახურულ წევრებს.

კლასის წევრებთან მიმართვის შეზღუდვა არის ობიექტზე ორიენტირებული პროგრამირების ძირითადი პრინციპი. იგი ობიექტებს იცავს არასწორი გამოყენებისაგან. ვაძლევთ რა კლასის დახურულ წევრებთან მიმართვის უფლებას მხოლოდ კლასის შიგნით განსაზღვრულ მეთოდებს, ამით თავიდან ვიცილებთ მონაცემებისათვის არაკორექტული მნიშვნელობების მინიჭებას კლასის გარეთ განსაზღვრული კოდის მხრიდან.

კლასის წევრებთან მიმართვა ხორციელდება ოთხი მოდიფიკატორის გამოყენებით: public, private, protected და internal. თუ მოდიფიკატორი არ არის მითითებული, მაშინ იგულისხმება private.

შევქმნათ Samkutxedi კლასი და მასში მოვათავსოთ სამკუთხედის გვერდების ზომები. ქვემოთ მოყვანილია Samkutxedi კლასის პირველი ვერსია, რომელშიც განსაზღვრულია მხოლოდ სამი ცვლადი gverdi1, gverdi2 და gverdi3. კლასი არ შეიცავს არც ერთ მეთოდს.

```
class Samkutxedi
{
public int gverdi1;          //    სამკუთხედის პირველი გვერდი
public int gverdi2;          //    სამკუთხედის მეორე გვერდი
public int gverdi3;          //    სამკუთხედის მესამე გვერდი
}
```

class სიტყვის გამოყენებით იქმნება მონაცემების ახალი ტიპი, რომლის სახელია Samkutxedi. მას გამოვიყენებთ Samkutxedi ტიპის მქონე ობიექტის გამოსაცხადებლად. უნდა გვახსოვდეს, რომ class სიტყვის გამოყენებით კლასის გამოცხადებისას ხდება მხოლოდ მისი აღწერა, მაგრამ ფიზიკურად ობიექტი ამ დროს არ იქმნება. Samkutxedi ტიპის ობიექტის შესაქმნელად უნდა გამოვიყენოთ new ოპერატორი:

```
Samkutxedi Sam1 = new Samkutxedi();
```

მისი შესრულების შედეგად მეხსიერებაში შეიქმნება Samkutxedi კლასის ეგზემპლარი -

Sam1 ობიექტი.

ობიექტი შეიცავს კლასის თითოეული ცვლადის ასლს. ობიექტის ცვლადებთან მიმართვისათვის ვიყენებთ წერტილი (.) ოპერატორს. იგი ობიექტის სახელს კლასის წევრის სახელს უკავშირებს. მისი სინტაქსია:

ობიექტის_სახელი.წევრის_სახელი

იმისათვის, რომ Sam1 ობიექტის gverdi2 ცვლადს მივანიჭოთ მნიშვნელობა 20, მინიჭების ოპერატორი უნდა ჩავწეროთ შემდეგნაირად:

```
Sam1.gverdi2 = 20;
```

წერტილი (.) ოპერატორი შეგვიძლია, აგრეთვე, გამოვიყენოთ მეთოდებთან მიმართვისთვისაც.

ქვემოთ მოყვანილია პროგრამა, რომელშიც ნაჩვენებია Samkutxedi კლასის გამოყენება.

```
//      პროგრამა 5.1
//      პროგრამაში ხდება ობიექტის ცვლადებთან მუშაობის დემონსტრირება
class Samkutxedi
{
        public int gverdi1;             //      სამკუთხედის პირველი გვერდი
        public int gverdi2;             //      სამკუთხედის მეორე გვერდი
        public int gverdi3;             //      სამკუთხედის მესამე გვერდი
}
private void button1_Click(object sender, System.EventArgs e)
{
Samkutxedi Sam1 = new Samkutxedi();
int perimetri;
//      Sam1 ობიექტის ცვლადებს მნიშვნელობები ენიჭებათ
Sam1.gverdi1 = Convert.ToInt32(textBox1.Text);
Sam1.gverdi2 = Convert.ToInt32(textBox2.Text);
Sam1.gverdi3 = Convert.ToInt32(textBox3.Text);
//      სამკუთხედის პერიმეტრის გამოთვლა
perimetri = Sam1.gverdi1 + Sam1.gverdi2 + Sam1.gverdi3;
label1.Text = "სამკუთხედის პერიმეტრია  " + perimetri.ToString();
}
```

კლასის აღწერა შეგვიძლია მოვათავსოთ Main() მეთოდის შემდეგ. ამისათვის, ჯერ button კომპონენტი მოვათავსოთ ფორმაზე, შემდეგ კი ორჯერ სწრაფად დავაწკაპუნოთ მასზე. შემდეგ კურსორი მოვათავსოთ

```
private void button1_Click(object sender, System.EventArgs e)
```

სტრიქონის ზედა ცარიელ სტრიქონში და დავიწყოთ კლასის შეტანა.

როგორც აღვნიშნეთ ობიექტების შექმნის ძირითადი პრინციპია ის, რომ თითოეულ ობიექტს აქვს კლასის ცვლადების საკუთარი ასლი. შედეგად, ერთი ობიექტის ცვლადები დამოუკიდებელია მეორე ობიექტის ცვლადებისაგან, ამიტომ მათი მნიშვნელობები შეიძლება ერთმანეთისაგან განსხვავდებოდეს. მაგალითად, თუ არსებობს Samkutxedi კლასის ორი ობიექტი, მაშინ თითოეულ მათგანს ექნება gverdi1, gverdi2 და gverdi3 ცვლადების საკუთარი ასლები, რომელთა მნიშვნელობები შეიძლება ერთმანეთისაგან განსხვავდებოდეს. ქვემოთ მოყვანილი პროგრამა ახდენს ამ პრინციპის დემონსტრირებას.

```
//      პროგრამა 5.2
//      პროგრამაში ხდება ორი ობიექტის ცვლადებთან მუშაობის დემონსტრირება
class Samkutxedi
{
public int gverdi1;          //      სამკუთხედის პირველი გვერდი
public int gverdi2;          //      სამკუთხედის მეორე გვერდი
public int gverdi3;          //      სამკუთხედის მესამე გვერდი
}
private void button1_Click(object sender, System.EventArgs e)
{
Samkutxedi Sam1  = new Samkutxedi();          //      იქმნება Sam1 ობიექტი
Samkutxedi Sam2 = new Samkutxedi();           //      იქმნება Sam2 ობიექტი
int Sam1_perimetri, Sam2_perimetri;

// Sam1 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
Sam1.gverdi1 = Convert.ToInt32(textBox1.Text);
Sam1.gverdi2 = Convert.ToInt32(textBox2.Text);
Sam1.gverdi3 = Convert.ToInt32(textBox3.Text);

// Sam2 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
Sam2.gverdi1 = Convert.ToInt32(textBox4.Text);
Sam2.gverdi2 = Convert.ToInt32(textBox5.Text);
Sam2.gverdi3 = Convert.ToInt32(textBox6.Text);

// თითოეული სამკუთხედის პერიმეტრის გამოთვლა
Sam1_perimetri = Sam1.gverdi1 + Sam1.gverdi2 + Sam1.gverdi3;
Sam2_perimetri = Sam2.gverdi1 + Sam2.gverdi2 + Sam2.gverdi3;
label1.Text = Sam1_perimetri.ToString();
label2.Text = Sam2_perimetri.ToString();
}
```

ობიექტების შექმნა

წინა პროგრამებში Samkutxedi ტიპის ობიექტის გამოსაცხადებლად გამოიყენებოდა შემდეგი ოპერატორი

```
Samkutxedi Sam1 = new Samkutxedi();
```

მასში ორი მოქმედება სრულდება:

- ხდება Samkutxedi ტიპის ცვლადის გამოცხადება, რომლის სახელია Sam1. იგი ობიექტს არ განსაზღვრავს (ე.ი. არ არის უშუალოდ ობიექტი), არამედ, მხოლოდ მიმართავს მას.
- new ოპერატორის შესრულებისას იქმნება ფიზიკური ობიექტი, ხოლო Sam1 ცვლადს ენიჭება მასზე მიმართვა (ანუ ამ ობიექტის მისამართი). შედეგად, Sam1 ცვლადი შეიცავს Samkutxedi ტიპის ობიექტზე მიმართვას.

new ოპერატორი დინამიურად (პროგრამის შესრულების დროს) გამოყოფს

მეხსიერებას ობიექტისათვის და გასცემს მიმართვას მეხსიერების ამ უბანზე. მიმართვა არის მეხსიერების იმ უბნის მისამართი, რომელიც ობიექტს გამოეყო new ოპერატორის მიერ. შემდგომში ეს მიმართვა ინახება ცვლადში. ზემოთ მოყვანილი ოპერატორი უფრო მეტი თვალსაჩინოებისათვის შეგვიძლია ჩავწეროთ ორი ოპერატორის სახით:

```
Samkutxedi Sam1;
Sam1 = new Samkutxedi();
```

პირველ სტრიქონში Sam1 ცვლადი გამოცხადებულია როგორც მიმართვა Samkutxedi ტიპის ობიექტზე. Sam1 ცვლადი მიმართავს ობიექტს, მაგრამ არ არის თვით ობიექტი. ამ ეტაპზე ამ ცვლადის მნიშვნელობაა null, რაც იმას ნიშნავს, რომ კავშირი ამ ცვლადსა და ობიექტს შორის არ არსებობს. მომდევნო სტრიქონში იქმნება Samkutxedi ტიპის ახალი ობიექტი, ხოლო Sam1 ცვლადს ენიჭება მიმართვა ამ ობიექტზე. ამის შემდეგ, Sam1 ცვლადი დაკავშირებულია ობიექტთან.

რადგან კლასის ობიექტებთან მუშაობა ხორციელდება მიმართვის საშუალებით, ამიტომ კლასებს სხვანაირად მიმართვით ტიპებს უწოდებენ. როგორც ვიცით, მთავარი განსხვავება ჩვეულებრივ და მიმართვით ტიპებს შორის მდგომარეობს იმ მნიშვნელობებში, რომლებსაც ისინი ინახავენ. ჩვეულებრივი ტიპის ცვლადი ინახავს მნიშვნელობას, მიმართვითი ტიპის ვცლადი კი - ობიექტის მისამართს. განვიხილოთ მაგალითები.

```
int x;
x = 20;
```

ამ ოპერატორების შესრულების შემდეგ x ცვლადში მოთავსებული იქნება მნიშვნელობა 20.

```
Samkutxedi Sam1 = new Samkutxedi();
```

ამ ოპერატორის შესრულების შემდეგ Sam1 ცვლადში მოთავსებული იქნება ობიექტზე მიმართვა ანუ ამ ობიექტის მისამართი.

მიმართვითი ტიპის მნიშვნელობის მინიჭება

მინიჭების ოპერატორის შესრულებისას მიმართვითი ტიპის ცვლადები ჩვეულებრივი ტიპის ცვლადებისაგან განსხვავებით სხვანაირად მოქმედებენ. როცა ჩვეულებრივი ტიპის ერთი ცვლადის მნიშვნელობას ვანიჭებთ ჩვეულებრივი ტიპის მეორე ცვლადის მნიშვნელობას, მაშინ მინიჭების ოპერატორის მარცხნივ მყოფ ცვლადს ენიჭება მარჯვნივ მყოფი ცვლადის მნიშვნელობის ასლი. როცა ერთი მიმართვითი ტიპის ცვლადის მნიშვნელობას ვანიჭებთ მიმართვითი ტიპის მეორე ცვლადის მნიშვნელობას, მაშინ მინიჭების ოპერატორის მარცხნივ მყოფ ცვლადს ენიჭება შესაბამის ობიექტზე მიმართვა. განვიხილოთ კოდის ფრაგმენტი:

```
Samkutxedi Sam1 = new Samkutxedi();
Samkutxedi Sam2 = Sam1;
```

Sam2 ცვლადისათვის Sam1 ცვლადის მნიშვნელობის მინიჭებისას, Sam2 ცვლადს ენიჭება მიმართვა იმავე ობიექტზე, რომელსაც Sam1 ცვლადი მიმართავს. შედეგად, ერთსა და იმავე ობიექტთან მიმართვა შეიძლება შესრულდეს როგორც Sam1 ცვლადის, ისე Sam2 ცვლადის საშუალებით. სხვა სიტყვებით რომ ვთქვათ, Sam1 და Sam2 ცვლადები ერთსა და იმავე ობიექტს მიმართავენ. მაგალითად,

```
Sam1.gverdi1 = 30;
```

მინიჭების ოპერატორის შესრულების შედეგად Sam1.gverdi1 და Sam2.gverdi1

ცვლადებში აღმოჩნდება ერთნაირი მნიშვნელობა - 30. ეს ორი ცვლადი ერთმანეთთან არ არის დაკავშირებული, თუმცა ერთსა და იმავე ობიექტს მიმართავენ.

ქვემოთ მოყვანილი კოდის ფრაგმენტის შესრულების შედეგად Sam2 ცვლადი მიიღებს იმ ობიექტზე მიმართვას, რომელსაც Sam3 ცვლადი მიმართავს.

```
{
Samkutxedi Sam1 = new Samkutxedi();
Samkutxedi Sam2 = Sam1;
Samkutxedi Sam3 = new Samkutxedi();
Sam2 = Sam3;
}
```

მეთოდები

მეთოდები - ესაა პროგრამები, რომლებიც ასრულებენ გარკვეულ ფუნქციას, მუშაობენ კლასში განსაზღვრულ მონაცემებთან და უზრუნველყოფენ ამ მონაცემებთან მიმართვას. მეთოდის გამოძახება ბევრჯერ შეიძლება.

მეთოდი შეიძლება შეიცავდეს ერთ ან მეტ ოპერატორს და უნდა წყვეტდეს მხოლოდ ერთ კონკრეტულ ამოცანას. მეთოდის სახელად შეგვიძლია გამოვიყენოთ ნებისმიერი იდენტიფიკატორი. საკვანძო სიტყვების გამოყენება მეთოდების სახელებად არ შეიძლება. Main() სახელი დარეზერვებულია იმ მეთოდისათვის, საიდანაც იწყება პროგრამის შესრულება.

მეთოდის სინტაქსია:

მიმართვის _მოდიფიკატორი ტიპი მეთოდის_სახელი(პარამეტრების_სია)

{

მეთოდის ტანი

}

აქ მიმართვის_მოდიფიკატორი იღებს private ან public მნიშვნელობებს. თუ იგი არ არის მითითებული, მაშინ იგულისხმება private, ე.ი. მეთოდი მისაწვდომი იქნება მხოლოდ იმ კლასის შიგნით, რომელშიც ის არის აღწერილი. ტიპი არის იმ მნიშვნელობის ტიპი, რომელსაც მეთოდი გასცემს. თუ მეთოდი არ გასცემს მნიშვნელობას, მაშინ უნდა მივუთითოთ void ტიპი. მეთოდის_სახელი არის ნებისმიერი იდენტიფიკატორი, რომელიც არ ემთხვევა კლასის სხვა წევრების სახელებს ხილვადობის იმავე უბანში. პარამეტრების_სია შედგება პარამეტრების სახელებისაგან, რომლებიც ერთმანეთისაგან მძიმეებით გამოიყოფა. თითოეული პარამეტრის სახელის წინ ეთითება შესაბამისი ტიპი. პარამეტრები ესაა ცვლადები, რომლებიც იღებენ არგუმენტების მნიშვნელობებს, რომლებიც მეთოდს გადაეცემა მისი გამოძახებისას. თუ მეთოდს პარამეტრები არ აქვს, მაშინ პარამეტრების_სია ცარიელი იქნება.

ზემოთ მოყვანილ მაგალითებში სამკუთხედის პერიმეტრის გამოთვლა სრულდებოდა ძირითად პროგრამაში. პერიმეტრი უმჯობესია გამოითვალოს უშუალოდ კლასის შიგნით, რადგან მისი მნიშვნელობა დამოკიდებულია სამკუთხედის გვერდების ზომებზე. სამივე ეს სიდიდე კი ინკაფსულირებულია Samkutxedi კლასში. გარდა ამისა, Samkutxedi კლასისთვის იმ მეთოდის დამატება, რომელშიც სრულდება პერიმეტრის გამოთვლა აუმჯობესებს ამ კლასის ობიექტზე ორიენტირებულ სტრუქტურას.

ქვემოთ მოყვანილია Samkutxedi კლასის მომდევნო ვერსია. მის perimetri() მეთოდს ფანჯარაში გამოაქვს სამკუთხედის პერიმეტრის მნიშვნელობა.

```
//      პროგრამა 5.3
//      პროგრამაში ხდება სამკუთხედის პერიმეტრის გამოთვლა კლასის შიგნით
გამოცხადებულ მეთოდში
class Samkutxedi
{
public  int gverdi1;
public  int gverdi2;
public  int gverdi3;
private int perim;
//      perimetri მეთოდის განსაზღვრა
public void perimetri()
{
perim = gverdi1 + gverdi2 + gverdi3;
MessageBox.Show("სამკუთხედის პერიმეტრია - " + perim.ToString());
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Samkutxedi Sam1 = new Samkutxedi();
Samkutxedi Sam2 = new Samkutxedi();
//      Sam1 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
Sam1.gverdi1  = Convert.ToInt32(textBox1.Text);
Sam1.gverdi2  = Convert.ToInt32(textBox2.Text);
Sam1.gverdi3  = Convert.ToInt32(textBox3.Text);
//      პერიმეტრის გამოთვლა პირველი სამკუთხედისათვის
Sam1.perimetri ();
//      Sam2 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
Sam2.gverdi1  = Convert.ToInt32(textBox4.Text);
Sam2.gverdi2  = Convert.ToInt32(textBox5.Text);
Sam2.gverdi3  = Convert.ToInt32(textBox6.Text);
//      პერიმეტრის გამოთვლა მეორე სამკუთხედისათვის
Sam2.perimetri ();
}
```

პროგრამაში perim ცვლადი იმიტომ არის პრივატული, რომ დაცული იყოს ობიექტის მთლიანობა. თუ მას გამოვაცხადებთ როგორც public, მაშინ იარსებებს იმის საფრთხე, რომ ამ ცვლადს არასწორი მნიშვნელობა მიენიჭოს და შედეგად, სამკუთხედის გვერდების ჯამი შეიძლება არ დაემთხვეს პერიმეტრის მნიშვნელობას.

```
public void perimetri()
```

სტრიქონში ხდება perimetri() მეთოდის გამოცხადება, რომელსაც პარამეტრები არ აქვს. იგი განსაზღვრულია როგორც public, რაც იძლევა მისი გამოძახების შესაძლებლობას პროგრამის ნებისმიერ ადგილიდან. void სიტყვა მიუთითებს იმას, რომ perimetri() მეთოდი არ აბრუნებს (არ გასცემს) მნიშვნელობას. შემდეგ, ფიგურულ ფრჩხილებში მოთავსებულია მეთოდის ტანი, რომელშიც გამოითვლება სამკუთხედის პერიმეტრი.

მისი მნიშვნელობა ეკრანზე გამოიცემა MessageBox.Show() მეთოდის გამოყენებით. რადგან Samkutxedi ტიპის თითოეულ ობიექტს აქვს gverdi1, gverdi2 და gverdi3 ცვლადების საკუთარი ასლები, ამიტომ perimetri() მეთოდის გამოძახებისას პერიმეტრის გამოსათვლელად გამოყენებული იქნება გამომძახებელი ობიექტის ცვლადები.

Sam1.perimetri();

სტრიქონში გამომძახებელია Sam1 ობიექტი. აქ ხდება Sam1 ობიექტის perimetri() მეთოდის გამოძახება, რის შემდეგ მართვა ამ მეთოდს გადაეცემა. იგი Sam1 ობიექტის ცვლადებთან იმუშავებს. როცა მეთოდი მუშაობას დაამთავრებს, მართვა გადაეცემა პროგრამის იმ ადგილს, საიდანაც მისი გამოძახება მოხდა. პროგრამის შესრულება გაგრძელდება კოდის იმ სტრიქონიდან, რომელიც მოსდევს მეთოდის გამომძახებელ სტრიქონს. ჩვენს შემთხვევაში Sam1.perimetri() მეთოდის გამოძახება იწვევს პერიმეტრის მნიშვნელობის გამოტანას Sam1 სამკუთხედისათვის, ხოლო Sam2.perimetri() მეთოდის გამოძახება კი იწვევს პერიმეტრის მნიშვნელობის გამოტანას Sam2 სამკუთხედისათვის.

მეთოდიდან მართვის დაბრუნება

მეთოდიდან მართვის დაბრუნება, ანუ მეთოდის შესრულების შეწყვეტა და გამომძახებელი პროგრამისათვის მართვის დაბრუნება, ორ შემთხვევაში ხდება. პირველი, როცა გვხვდება მეთოდის დამხურავი ფიგურული ფრჩხილი და მეორე, როცა სრულდება return ოპერატორი. არსებობს return ოპერატორის ორი ფორმა. პირველი ფორმა გამოიყენება მეთოდებში, რომლებსაც void ტიპი აქვთ, ანუ მნიშვნელობას არ აბრუნებენ, მეორე კი - მეთოდებში, რომლებიც მნიშვნელობას აბრუნებენ.

მეთოდებში, რომლებიც მნიშვნელობას არ აბრუნებენ, გამოიყენება შემდეგი სინტაქსის მქონე return ოპერატორი:

return;

ასეთ შემთხვევაში return ოპერატორი შეგვიძლია არც მივუთითოთ.

return ოპერატორის შესრულების დროს მართვა გადაეცემა გამომძახებელი პროგრამის იმ ადგილს, საიდანაც მეთოდი იყო გამოძახებული, ამასთან მეთოდის კოდის დარჩენილი ნაწილი არ შესრულდება.

მეთოდის ერთ-ერთი მნიშვნელოვანი თვისებაა - მის მიერ მნიშვნელობის დაბრუნება (გაცემა). მეთოდები მნიშვნელობას უბრუნებენ გამომძახებელ პროგრამას შემდეგი სინტაქსის მქონე return ოპერატორის გამოყენებით:

return *მნიშვნელობა;*

აქ *მნიშვნელობა* არის მეთოდის მიერ დაბრუნებული მნიშვნელობა.

ახლა perimetri() მეთოდი გადავაკეთოთ ისე, რომ მან გამოთვალოს პერიმეტრი და დაუბრუნოს იგი პროგრამას. ასეთი მიდგომის ერთ-ერთი უპირატესობაა ის, რომ დაბრუნებული მნიშვნელობა შეგვიძლია გამოვიყენოთ სხვა გამოთვლებში. ქვემოთ მოყვანილ პროგრამაში perimetri() მეთოდს ეკრანზე აღარ გამოაქვს პერიმეტრის მნიშვნელობა, არამედ ახდენს მის გამოთვლასა და შედეგის დაბრუნებას.

```
//      პროგრამა 5.4
//      პროგრამაში ხდება return ოპერატორის გამოყენების დემონსტრირება
class Samkutxedi
{
        public int gverdi1;
```

```
        public int gverdi2;
        public int gverdi3;             //      მეთოდის განსაზღვრა
        public int perimetri()
        {
                return gverdi1 + gverdi2 + gverdi3;         //      მნიშვნელობის დაბრუნება
        }
}
private void button1_Click(object sender, System.EventArgs e)
{
//      tolgverda და tolferda ობიექტების შექმნა
Samkutxedi tolgverda = new Samkutxedi();
Samkutxedi tolferda  = new Samkutxedi();
int tolgverda_perimetri, tolferda_perimetri;

//      tolgverda ობიექტის ცვლადებს მნიშვნელობები ენიჭებათ
tolgverda.gverdi1 = Convert.ToInt32(textBox1.Text);
tolgverda.gverdi2 = Convert.ToInt32(textBox2.Text);
tolgverda.gverdi3 = Convert.ToInt32(textBox3.Text);

//      tolferda ობიექტის ცვლადებს მნიშვნელობები ენიჭებათ
tolferda.gverdi1 = Convert.ToInt32(textBox4.Text);
tolferda.gverdi2 = Convert.ToInt32(textBox5.Text);
tolferda.gverdi3 = Convert.ToInt32(textBox6.Text);

//      tolgverda და tolferda სამკუთხედებისათვის პერიმეტრი გამოითვლება
tolgverda_perimetri = tolgverda.perimetri ();
tolferda_perimetri  = tolferda.perimetri ();
label1.Text = tolgverda_perimetri.ToString();
label2.Text = tolferda_perimetri.ToString();
}
```

პროგრამაში perimetri() მეთოდის მიერ გაცემული მნიშვნელობები შესაბამისად მიენიჭება tolgverda_perimetri და tolferda_perimetri ცვლადებს.

პარამეტრების გამოყენება

გამოძახებისას მეთოდს შეგვიძლია გადავცეთ ერთი ან მეტი პარამეტრი. მეთოდისთვის გადასაცემ მნიშვნელობას არგუმენტი ეწოდება. ცვლადს მეთოდის შიგნით, რომელიც იღებს არგუმენტის მნიშვნელობას, პარამეტრი ეწოდება. პარამეტრების გამოცხადება ხდება მრგვალი ფრჩხილების შიგნით, რომლებიც მეთოდის სახელს მოსდევს. პარამეტრის გამოცხადების სინტაქსი ცვლადების გამოცხადების სინტაქსის ანალიგიურია. პარამეტრი იმყოფება თავისი მეთოდის ხილვადობის უბანში, ანუ ხილულია მხოლოდ ამ მეთოდის შიგნით, და მოქმედებს როგორც ლოკალური ცვლადი.

ქვემოთ მოყვანილია პროგრამა, რომელშიც მეთოდისათვის მნიშვნელობის

გადასაცემად გამოიყენება პარამეტრი. ArisLuwi კლასის შიგნით განსაზღვრული luwi_kenti(int x) მეთოდი აბრუნებს bool ტიპის შედეგს. თუ მისთვის გადაცემული მნიშვნელობა არის ლუწი მაშინ იგი გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.

```
//      პროგრამა 5.5
 //     პროგრამა განსაზღვრავს რიცხვი კენტია თუ ლუწი
class ArisLuwi
{
// მეთოდის განსაზღვრა, რომელიც ამოწმებს რიცხვი კენტია თუ ლუწი
public bool luwi_kenti( int x )
{
if ( ( x % 2 ) == 0 ) return true;
        else return false;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//
int ricxvi;
ArisLuwi obieqti = new ArisLuwi();

label1.Text = "";
ricxvi = Convert.ToInt32(textBox1.Text);
if ( obieqti.luwi_kenti(ricxvi) )
                label1.Text = " რიცხვი " + ricxvi.ToString() + " ლუწია";
        else  label1.Text = " რიცხვი " + ricxvi.ToString() + " კენტია";
}
```

მეთოდს შეიძლება რამდენიმე პარამეტრი ჰქონდეს. ისინი ერთმანეთისაგან მძიმეებით გამოიყოფა. მაგალითად, Gamyofi კლასი შეიცავს arisGamyofi() მეთოდს, რომელიც განსაზღვრავს არის თუ არა პირველი პარამეტრი მეორე პარამეტრის გამყოფი.

```
//      პროგრამა 5.6
//      პროგრამა განსაზღვრავს ერთი რიცხვი არის თუ არა მეორის გამყოფი
class Gamyofi
{
//      მეთოდი ამოწმებს მეორე პარამეტრი უნაშთოდ იყოფა თუ არა პირველ პარამეტრზე
public bool arisGamyofi(int par1, int par2)
{
if ( ( par1 % par1 ) == 0 ) return true;
        else return false;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int ricxvi1, ricxvi2;
Gamyofi obieqti = new Gamyofi();
```

```
label2.Text = "";
ricxvi1 = Convert.ToInt32(textBox1.Text);
ricxvi2 = Convert.ToInt32(textBox2.Text);
if ( obieqti.arisGamyofi(ricxvi1, ricxvi2) )
        label2.Text = ricxvi1.ToString() + " არის " + ricxvi2.ToString() +
                        "-ის გამყოფი";
                else label2.Text = ricxvi1.ToString() + " არ არის " +
                                        ricxvi2.ToString() + "-ის გამყოფი";
}
```

ახლა განვიხილოთ მეთოდისათვის მასივის გადაცემის მაგალითი. ქვემოთ მოყვანილ პროგრამაში Minimumi() მეთოდს გადაეცემა მასივი. მეთოდი პოულობს და აბრუნებს მასივის მინიმალური ელემენტის მნიშვნელობას.

```
//      პროგრამა 5.7
//      პროგრამაში ხდება მეთოდისათვის მასივის გადაცემის დემონსტრირება
class MinSidide
{
//      მეთოდი გასცემს მასივის მინიმალური ელემენტის მნიშვნელობას
public int Minimumi(int[] mas)
{
int min_ricxvi = mas[0];
for ( int indexi =1; indexi < mas.Length; indexi++ )
        if  (min_ricxvi > mas[indexi] ) min_ricxvi = mas[indexi];
return min_ricxvi;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int[] masivi = new int[] { 5, -7, 4, -6, 9 };
MinSidide obieqti = new MinSidide();
int shedegi;

label1.Text = "";
shedegi = obieqti.Minimumi(masivi);        // Minimumi მეთოდს masivi მასივი გადაეცემა
label1.Text = shedegi.ToString();
}
```

კონსტრუქტორები

კონსტრუქტორი არის მეთოდი, რომლის დანიშნულებაა ობიექტის ცვლადების ინიციალიზება. ობიექტის ინიციალიზება სრულდება მისი შექმნისას. კონსტრუქტორს ისეთივე სახელი აქვს, როგორიც კლასს და მისი სინტაქსი მეთოდის სინტაქსის მსგავსია. მეთოდისგან განსხვავებით კონსტრუქტორში დასაბრუნებელი მნიშვნელობის ტიპი არ ეთითება. მისი სინტაქსია:

კლასის_სახელი()
{
კონსტრუქტორის კოდი
}

როგორც წესი კონსტრუქტორები გამოიყენება ობიექტის ცვლადებისათვის საწყისი მნიშვნელობების მისანიჭებლად, ან ინიციალიზაციის ნებისმიერი სხვა პროცედურის შესასრულებლად, რომლებიც აუცილებელია მთლიანად ფორმირებული ობიექტის შესაქმნელად.

ყველა კლასს აქვს კონსტრუქტორი მიუხედავად იმისა იგი განსაზღვრულია თუ არა. C# ენაში გათვალისწინებულია კონსტრუქტორის არსებობა, რომელიც ობიექტის ყველა ცვლადს ანიჭებს ნულოვან (ეს ეხება ჩვეულებრივი ტიპის ცვლადებს) ან null მნიშვნელობას (ეს ეხება მიმართვითი ტიპის ცვლადებს). ასეთ კონსტრუქტორს ავტომატური კონსტრუქტორი (კონსტრუქტორი გაჩუმებით) ეწოდება. იგი იმ შემთხვევაში გამოიძახება, როცა კლასში კონსტრუქტორი განსაზღვრული არ არის. თუ კონსტრუქტორი აშკარადაა განსაზღვრული კლასში, მაშინ სწორედ ის იქნება გამოძახებული.

პროფესიულ დონეზე შედგენილ პროგრამაში ობიექტის ცვლადების ინიციალიზება ყოველთვის კონსტრუქტორის მიერ სრულდება. მაგალითად, 5.1, 5.2, 5.3, 5.5 პროგრამებში ობიექტის ცვლადებს მნიშვნელობებს ჩვენ ვანიჭებდით. უმჯობესია, ეს კონსტრუქტორმა გააკეთოს. განვიხილოთ 5.4 პროგრამის ახალი ვერსია, რომელშიც ობიექტის ცვლადებს მნიშვნელობებს კონსტრუქტორი ანიჭებს.

```
//      პროგრამა 5.8
//      პროგრამაში ხდება კონსტრუქტორთან მუშაობის დემონსტრირება
class Samkutxedi
{
public int gverdi1;
public int gverdi2;
public int gverdi3;
private int perimetri;
//      კონსტრუქტორი აქვს პარამეტრები
public Samkutxedi(int par1, int par2, int par3)
{
gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
}
public int perim()
{
perimetri = gverdi1 + gverdi2 + gverdi3;
return perimetri;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int g1 = Convert.ToInt32(textBox1.Text);
```

```
int g2 = Convert.ToInt32(textBox2.Text);
int g3 = Convert.ToInt32(textBox3.Text);
int g4 = Convert.ToInt32(textBox4.Text);
int g5 = Convert.ToInt32(textBox5.Text);
int g6 = Convert.ToInt32(textBox6.Text);
Samkutxedi Sam1 = new Samkutxedi(g1, g2, g3);
Samkutxedi Sam2 = new Samkutxedi(g4, g5, g6);

int SamkutxedisPerimetri1 = Sam1.perim();
int SamkutxedisPerimetri2 = Sam2.perim();
label1.Text = "პირველი სამკუთხედის პერიმეტრი = " + SamkutxedisPerimetri1.ToString();
label1.Text = " მეორე სამკუთხედის პერიმეტრი = " + SamkutxedisPerimetri2.ToString();
}
```

პროგრამაში new ოპერატორის შესრულებისას გამოიძახება Samkutxedi() კონსტრუქტორი, რადგან იგი განსაზღვრულია კლასში. მას სამი პარამეტრი აქვს. ისინი გამოიყენება სამკუთხედის გვერდებისათვის მნიშვნელობების მისანიჭებლად. პირველი new ოპერატორის შესრულების დროს Samkutxedi() კონსტრუქტორს g1, g2 და g3 არგუმენტები გადაეცემა. ისინი შესაბამისად par1, par2 და par3 პარამეტრებს მიენიჭება. ისინი თავის მხრივ gverdi1, gverdi2 და gverdi3 ცვლადებს ენიჭება.

დესტრუქტორები და "ნაგვის შეგროვება"

როცა C# ენაში არ ხდება ობიექტთან მიმართვა, მაშინ იგი განიხილება როგორც შემდგომში არაგამოყენებადი და სისტემა ავტომატურად, პროგრამისტის მონაწილეობის გარეშე, ათავისუფლებს ობიექტის მიერ დაკავებულ მეხსიერებას. ამ ოპერაციას “ნაგვის შეგროვება” ეწოდება. გამოთავისუფლებული მეხსიერება შემდგომში შეიძლება გამოყენებული იყოს სხვა ობიექტებისათვის. თუ ეს ოპერაცია არ შესრულდა, მაშინ მეხსიერების შეზღუდული ზომის გამო new ოპერატორმა შეიძლება ვერ გამოყოს მეხსიერება სხვა ობიექტებისათვის.

"ნაგვის შეგროვება" პერიოდულად სრულდება პროგრამის მუშაობის განმავლობაში. ამ ოპერაციის დასაწყებად ორი პირობა უნდა შესრულდეს. ჯერ ერთი, უნდა არსებობდეს ობიექტები, რომლებიც შეიძლება წავშალოთ მეხსიერებიდან, და მეორე, ასეთი ოპერაციის აუცილებლობა. რადგან "ნაგვის შეგროვება" პროცესორის დროს იკავებს, ამიტომ იგი მხოლოდ აუცილებლობის შემთხვევაში შესრულდება, ამასთან პროგრამისტი ზუსტად ვერ განსაზღვრავს ამ მომენტს.

დესტრუქტორი არის მეთოდი, რომელიც გამოიყენება მეხსიერებიდან ობიექტის კორექტულად წასაშლელად. მისი სინტაქსია:

```
~კლასის_სახელი()
{
დესტრუქტორის კოდი
}
```

როგორც ვხედავთ დესტრუქტორის სახელის წინ მითითებულია სიმბოლო ტილდა (~). დესტრუქტორის განსაზღვრისას დასაბრუნებელი მნიშვნელობის ტიპი არ ეთითება.

დესტრუქტორის გამოძახება ხდება უშუალოდ "ნაგვის შეგროვების" ოპერაციის წინ მაშინ, როცა სრულდება მისი კლასის ობიექტის წაშლა მეხსიერებიდან. რადგან, წინასწარ არ ვიცით, თუ როდის დაიწყება "ნაგვის შეგროვების" პროცესი, ამიტომ არ გვეცოდინება თუ როდის შესრულდება დესტრუქტორის გამოძახება. თუ პროგრამამ მუშაობა დაამთავრა "ნაგვის შეგროვების" ოპერაციის დაწყებამდე, მაშინ დესტრუქტორი არასოდეს არ გამოიძახება.

this საკვანძო სიტყვა

მეთოდს გამოძახებისას ავტომატურად გადაეცემა არაცხადი არგუმენტი, რომელიც წარმოადგენს გამომძახებელ ობიექტზე მიმართვას. ამ მიმართვას ეწოდება this. იმის გასაგებად თუ როგორ მუშაობს this მიმართვა, განვიხილოთ პროგრამა, რომელშიც იქმნება Axarisxeba კლასი. იგი განკუთვნილია რიცხვის ხარისხის გამოსათვლელად.

```
//      პროგრამა 5.9
//      პროგრამაში ხარისხის გამოთვლა სრულდება კონსტრუქტორის კოდში
class Axarisxeba
{
public double d;
public int i;
public double shedegi;
public Axarisxeba(double par_ricxvi, int par_xarisxi)
 {
        d = par_ricxvi;
        i = par_xarisxi;
        shedegi = 1;

        if ( par_xarisxi == 0 ) return;
        for ( ; par_xarisxi > 0; par_xarisxi-- ) shedegi *= d;
 }
public double xarisxis_dabruneba()
{
        return shedegi;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
double ricxvi  = Convert.ToDouble(textBox1.Text);
int    xarisxi = Convert.ToInt32(textBox2.Text);
Axarisxeba obieqti = new Axarisxeba(ricxvi, xarisxi);

label1.Text = obieqti.d.ToString() + " ხარისხად " + obieqti.i.ToString() +
                " არის " + obieqti.xarisxis_dabruneba().ToString();
}
```

როგორც ვიცით მეთოდის ფარგლებში კლასის სხვა წევრებთან მიმართვა შეძლება განხორციელდეს ობიექტის ან კლასის სახელის მითითების გარეშე, ე.ი. უშუალოდ. შედეგად, xarisxis_dabruneba() მეთოდის შიგნით return shedegi; ოპერატორის შესრულების შედეგად გაიცემა shedegi ცვლადის მნიშვნელობა, რომელიც ასოცირებულია გამომძახებელ ობიექტთან. მაგრამ ეს ოპერატორი შეგვიძლია ასეც ჩავწეროთ:

return this.shedegi;

აქ this მიმართვა მიუთითებს იმ ობიექტზე, რომელსაც ეკუთვნის xarisxis_dabruneba() მეთოდი. this.shedegi ცვლადი არის მოცემული ობიექტის shedegi ცვლადის ასლი. მაგალითად, თუ xarisxis_dabruneba() მეთოდი გამოძახებულია x ობიექტისთვის, მაშინ this მიმართვა მიუთითებს x ობიექტზე. ოპერატორის ჩაწერა this სიტყვის გარეშე არის ჩაწერის შემოკლებული ფორმა.

C# ენის სინტაქსი იძლევა ერთნაირი სახელების გამოყენების საშუალებას პარამეტრებისა და ლოკალური ცვლადებისათვის. ასეთ შემთხვევაში, პარამეტრი მალავს ლოკალურ ცვლადს და მასთან მიმართვა შესაძლებელია მხოლოდ this მიმართვის გამოყენებით. ქვემოთ მოყვანილ პროგრამაში ხდება დამალულ ცვლადთან მიმართვა this სიტყვის გამოყენებით.

```
//      პროგრამა 5.10
//      პროგრამაში ხდება დამალულ ცვლადთან მიმართვის დემონსტრირება
class Axarisxeba
{
public double ricxvi;
public int xarisxi;
public double shedegi;
public Axarisxeba(double ricxvi, int xarisxi)
{
this.ricxvi = ricxvi;          // ობიექტის this.ricxvi ცვლადს ენიჭება ricxvi პარამეტრი
this.xarisxi = xarisxi;        // ობიექტის this.xarisxi ცვლადს ენიჭება
                               // xarisxi პარამეტრი
shedegi = 1;

if ( xarisxi == 0 ) return;
for ( ; xarisxi > 0; xarisxi-- ) shedegi *= ricxvi;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
double ricxvi1 = Convert.ToDouble(textBox1.Text);
int   xarisxi1 = Convert.ToInt32(textBox2.Text);
Axarisxeba obieqti = new Axarisxeba(ricxvi1, xarisxi1);
label1.Text = obieqti.shedegi.ToString();
}
```

Axarisxeba() კონსტრუქტორის ამ ვერსიაში პარამეტრების სახელები და ობიექტის ცვლადების სახელები ერთნაირია, ამიტომ ობიექტის ცვლადები იქნება დამალული. this მიმართვის გამოყენებით შესაძლებელი ხდება მათთან მიმართვა.

სტრუქტურები

სტრუქტურა წარმოადგენს კლასის გამარტივებულ ვარიანტს. სტრუქტურებთან მუშაობა სრულდება ისე, როგორც მნიშვნელობის მქონე ტიპებთან. მათი ძირითადი განსხვავება კლასებისაგან შემდეგში მდგომარეობს. კლასის ობიექტი ინახება გროვაში და მეხსერებიდან იშლება მხოლოდ "ნაგვის შეგროვების" დროს. სტრუქტურის ეგზემპლარი ინახება სტეკში და მეხსიერებიდან იშლება როგორც კი ეს ეგზემპლარი გამოდის ხილვადობის უბნიდან. კლასების მსგავსად სტრუქტურები შეიძლება შეიცავდეს ველებს, მეთოდებს, ინდექსატორებს, თვისებებს, მოვლენებსა და კონსტრუქტორებს. სტრუქტურები უნდა იყოს მცირე ზომის და მათი გამოყენება უმჯობესია მაშინ, როცა გვაქვს მცირე რაოდენობის ველები და მეთოდები. კლასებისგან განსხვავებით მათ აქვთ შემდეგი შეზღუდვები:

- სტრუქტურები არ უზრუნველყოფენ მემკვიდრეობითობას.
- სტრუქტურისთვის არ შეიძლება ავტომატური კონსტრუქტორის განსაზღვრა.
- სტრუქტურისთვის არ შეიძლება დესტრუქტორის განსაზღვრა.
- არ შეიძლება ინიციალიზატორების გამოყენება ველებისათვის მნიშვნელობების მისანიჭებლად.

სტრუქტურის გამოცხადებისათვის გამოიყენება struct საკვანძო სიტყვა. მოყვანილ პროგრამაში ხდება სტრუქტურასთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 5.11
//      პროგრამაში ხდება სტრუქტურასთან მუშაობის დემონსტრირება
public struct Samkutxedi
{
//  ველების გამოცხადება
public int gverdi1;
public int gverdi2;
public int gverdi3;
//  კონსტრუქტორის განსაზღვრა
public Samkutxedi(int par1, int par2, int par3)
{
gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
}
//  Perimetri() მეთოდი
public int Perimetri()
{
return gverdi1 + gverdi2 + gverdi3;
}
}
private void button1_Click(object sender, EventArgs e)
{
```

```
int g1 = Convert.ToInt32(textBox1.Text);
int g2 = Convert.ToInt32(textBox2.Text);
int g3 = Convert.ToInt32(textBox3.Text);
Samkutxedi struqtura = new Samkutxedi(g1, g2, g3);
int Shedegi = struqtura.Perimetri();
label1.Text = Shedegi.ToString();
}
}
```

შეიძლება ერთი სტრუქტურის მეორეში გადაწერა. მაგალითად,

```
Samkutxedi struqtura1 = new Samkutxedi(g1, g2, g3);
Samkutxedi struqtura2 = struqtura1;
```

შემთხვევითი რიცხვების გენერატორი

C# ენაში განსაზღვრულია System.Random კლასი შემთხვევით რიცხვებთან სამუშაოდ. ცხრილში 5.1 მოყვანილია ამ კლასის მეთოდები, რომლებიც ახდენენ შემთხვევითი რიცხვების გენერირებას.

ცხრილი 5.1.

მეთოდი	აღწერა
int Random.Next(int ცვლადი)	გასცემს არაუარყოფით შემთხვევით მთელ რიცხვს, რომელიც ნაკლებია მითითებული ცვლადის მნიშვნელობაზე
double Random.NextDouble()	გასცემს შემთხვევით წილად რიცხვს, რომლის მნიშვნელობა მოთავსებულია 0.0-სა და 1.0-ს შორის
void Random.NextBytes(byte[ ] მასივი)	მითითებულ მასივს შეავსებს შემთხვევითი რიცხვებით, რომელთა ტიპია byte

მოვიყვანოთ ორი პროგრამა, რომლებშიც შემთხვევითი რიცხვების გენერატორი გამოიყენება მასივების შესავსებად. ქვემოთ მოყვანილ პროგრამაში ორგანზომილებიანი მასივის შესავსებად გამოიყენება Next() მეთოდი.

```
{
// პროგრამა 5.12
// პროგრამაში შემთხვევითი რიცხვების გენერატორი გამოიყენება ორგანზომილებიანი მასივის შესავსებად
label1.Text="";
int max = Convert.ToInt32(textBox1.Text);
int striqoni = Convert.ToInt32(textBox2.Text);
int sveti    = Convert.ToInt32(textBox3.Text);
int[,] mas = new int[striqoni, sveti];
System.Random obieqti = new System.Random();
for ( int i = 0; i < striqoni; i++ )
        for ( int j = 0; j < sveti; j++ )
                mas[i, j] = obieqti.Next(max);
for ( int i = 0; i < striqoni; i++ )
```

```
{
        for ( int j = 0; j < sveti; j++ )
                label1.Text += mas[i, j].ToString() + "   ";
        label1.Text += '\n';
}
}
```

ამ პროგრამაში ერთგანზომილებიანი მასივის შესავსებად გამოიყენება NextBytes() მეთოდი.

```
{
//      პროგრამა 5.13
//      პროგრამაში ხდება შემთხვევითი რიცხვების გენერატორის გამოყენება
//      ერთგანზომილებიანი მასივის შესავსებად
label1.Text="";
byte[] mas = new byte[5];
System.Random obieqti = new System.Random();
obieqti.NextBytes(mas);
for ( int i = 0; i < mas.Length; i++ )
        label1.Text += mas[i].ToString() + "   ";
}
```

თავი 6. მეთოდები უფრო დაწვრილებით. პოლიმორფიზმი

მეთოდისათვის ობიექტის გადაცემა

მეთოდს პარამეტრებად ჩვეულებრივი ტიპების მნიშვნელობების გარდა შეგვიძლია გადავცეთ ობიექტებიც. განვიხილოთ პროგრამა, რომელშიც ხდება სამკუთხედების შედარება. მათი ზომები ინახება Samkutxedi კლასის ობიექტებში.

```
//      პროგრამა 6.1
//      პროგრამაში ხდება მეთოდისათვის ობიექტის გადაცემის დემონსტრირება
class Samkutxedi
{
int gverdi1, gverdi2, gverdi3;
int perimetri;
public Samkutxedi(int par1, int par2, int par3)
{
        gverdi1  = par1;
        gverdi2  = par2;
        gverdi3  = par3;
        perimetri = gverdi1 + gverdi2 + gverdi3;
}
/*
        IgiveSamkutxedi() მეთოდი გასცემს true მნიშვნელობას, თუ Sam1 ობიექტში არის
        სამკუთხედის ზომების იგივე  მნიშვნელობები, რაც გამომძახებელ ობიექტში
*/
public bool IgiveSamkutxedi(Samkutxedi Sam1)
{
if (  ( Sam1.gverdi1 == gverdi1 ) && (Sam1.gverdi2 == gverdi2 ) &&
     ( Sam1.gverdi3 == gverdi3 ) ) return true;
        else return false;
}
/*
        IgivePerimetri() მეთოდი გასცემს true მნიშვნელობას, თუ Sam1 ობიექტში არის
        სამკუთხედის პერიმეტრის იგივე  მნიშვნელობა, რაც გამომძახებელ ობიექტში
*/
public bool IgivePerimetri(Samkutxedi Sam1)
{
if ( Sam1.perimetri == perimetri ) return true;
        else return false;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
```

```
int arg1, arg2, arg3;

arg1 = Convert.ToInt32(textBox1.Text);
arg2 = Convert.ToInt32(textBox2.Text);
arg3 = Convert.ToInt32(textBox3.Text);

Samkutxedi obieqti1 = new Samkutxedi(arg1, arg2, arg3);
Samkutxedi obieqti2 = new Samkutxedi(10, 2, 5);
Samkutxedi obieqti3 = new Samkutxedi(Convert.ToInt32(textBox1.Text),
                Convert.ToInt32(textBox2.Text), Convert.ToInt32(textBox3.Text));

label1.Text = "პირველი და მეორე სამკუთხედების ზომები ერთნაირია --- "    +
                        obieqti1.IgiveSamkutxedi(obieqti2).ToString();
label2.Text = "პირველი და მესამე სამკუთხედების ზომები ერთნაირია --- "   +
                        obieqti1.IgiveSamkutxedi(obieqti3).ToString();
label3.Text = "მეორე და მესამე სამკუთხედების პერიმეტრები ერთნაირია --- " +
                        obieqti2.IgivePerimetri(obieqti3).ToString();
}
```

დაწვრილებით განვიხილოთ ამ პროგრამის Samkutxedi კლასში გამოცხადებული IgiveSamkutxedi() მეთოდი. მას აქვს Sam1 პარამეტრი, რომლის ტიპია IgiveSamkutxedi, ე.ი. Sam1 არის Samkutxedi ტიპის ობიექტზე მიმართვა. შედეგად, ამ მეთოდის გამოძახებისას Sam1 ცვლადს შეგვიძლია გადავცეთ კონკრეტულ ობიექტზე მიმართვა. მეთოდის (Sam1.gverdi1 == gverdi1) გამოსახულებაში ხდება Sam1 ობიექტის gverdi1 ცვლადის შედარება ტოლობაზე გამომძახებელი ობიექტის gverdi1 ცვლადთან. ძირითად პროგრამაში ამ მეთოდის გამოძახება ხდება obieqti1.IgiveSamkutxedi(obieqti2) სტრიქონ-ში. obieqti1 არის გამომძახებელი ობიექტი. obieqti2-ის მიმართვა მიენიჭება Sam1 პარა-მეტრს, ამიტომ, Sam1.gverdi1 ჩანაწერში gverdi1 ეკუთვნის obieqti2-ს, ხოლო შედარების (==) მარჯვნივ მითითებული gverdi1 ცვლადი კი - obieqti1-ს, ანუ გამომძახებელ ობიექტს.

მეთოდისათვის არგუმენტების გადაცემის ხერხები

მეთოდს არგუმენტები შეგვიძლია ორი გზით გადავცეთ. პირველია არგუმენტის გადაცემა მნიშვნელობის მიხედვით. ამ შემთხვევაში არგუმენტის მნიშვნელობის ასლი ენიჭება მეთოდის პარამეტრს. ამ შემთხვევაში, პარამეტრისა და არგუმენტის მნიშვნელობები მეხსიერების სხვადასხვა უბანშია მოთავსებული. შედეგად, პარამეტრის მნიშვნელობის ცვლილება არ შეცვლის არგუმენტის მნიშვნელობას. მეორეა არგუმენტის გადაცემა მიმართვის მიხედვით. ამ შემთხვევაში მეთოდის პარამეტრს გადაეცემა არგუმენტზე მიმართვა (არგუმენტის მისამართი). შედეგად, პარამეტრის მნიშვნელობის შეცვლისას შესაბამისად შეიცვლება არგუმენტის მნიშვნელობაც. ეს იმიტომ ხდება, რომ პარამეტრიც და არგუმენტიც მეხსიერების ერთსა და იმავე უბანს მიმართავენ. განვიხილოთ არგუმენტის გადაცემის თითოეული გზის რეალიზების მაგალითები.

მეთოდისთვის ჩვეულებრივი ტიპის მქონე მნიშვნელობების გადაცემისას, როგორიცაა int ან double, გამოიყენება გადაცემა მნიშვნელობის მიხედვით. შედეგად,

პარამეტრის ცვლილება არ გამოიწვევს არგუმენტის ცვლილებას. განვიხილოთ პროგრამა.

```
//      პროგრამა 6.2
//      პროგრამაში ხდება მეთოდისათვის არგუმენტის გადაცემა მნიშვნელობის მიხედვით
class Chveulebrivi_Tipi
{
// ამ მეთოდის შესრულება არ გამოიწვევს არგუმენტის შეცვლას
public void metodi(int par1, int par2)
{
        par1 = par1 + par2;
        par2 = - par2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Chveulebrivi_Tipi obieqti = new Chveulebrivi_Tipi();
int ricxvi1, ricxvi2;

ricxvi1 = Convert.ToInt32(textBox1.Text);
ricxvi2 = Convert.ToInt32(textBox2.Text);
label1.Text = "არგუმენტების მნიშვნელობები მეთოდის გამოძახებამდე:  " +
                ricxvi1.ToString() + "   " + ricxvi2.ToString();
obieqti.metodi(ricxvi1, ricxvi2);               //      metodi მეთოდის გამოძახება
label2.Text = "არგუმენტების მნიშვნელობები მეთოდის გამოძახების შემდეგ: " +
                ricxvi1.ToString() + "   " + ricxvi2.ToString();
}
```

როგორც ვიცით, კლასის ტიპის (მიმართვითი ტიპის) ცვლადის შექმნის დროს ჩვენ სინამდვილეში ვქმნით ობიექტზე მიმართვას და არა თვით ობიექტს. ობიექტი მეხსიერებაში იქმნება new ოპერატორით, ხოლო მეხსიერების ამ უბანზე მიმართვა კი ენიჭება მიმართვითი ტიპის ცვლადს. როცა მიმართვითი ტიპის ცვლადს ვიყენებთ როგორც არგუმენტს, მაშინ მეთოდის პარამეტრი იღებს მიმართვას იმავე ობიექტზე, რომელსაც არგუმენტი მიმართავს. შედეგად, არგუმენტიც და პარამეტრიც ერთსა და იმავე ობიექტს მიმართავენ. მოყვანილ პროგრამაში პარამეტრის მნიშვნელობის ცვლილება იწვევს არგუმენტის მნიშვნელობის ცვლილებას.

```
//      პროგრამა 6.3
//      პროგრამაში ხდება მეთოდისათვის არგუმენტის გადაცემა მიმართვის მიხედვით
class Mimartviti_tipi
{
public int cvladi1, cvladi2;
public Mimartviti_tipi(int par1, int par2)
{
        cvladi1 = par1;
        cvladi2 = par2;
}
```

```
//
public void shecvla(Mimartviti_tipi obieqti1)
{
        obieqti1.cvladi1 = obieqti1.cvladi1 + obieqti1.cvladi2;
        obieqti1.cvladi2 = -obieqti1.cvladi2;
}
}
private void button2_Click(object sender, System.EventArgs e)
{
int ricxvi1, ricxvi2;

ricxvi1 = Convert.ToInt32(textBox1.Text);
ricxvi2 = Convert.ToInt32(textBox2.Text);

Mimartviti_tipi obieqti2 = new Mimartviti_tipi(ricxvi1, ricxvi2);

label1.Text = "არგუმენტების მნიშვნელობები მეთოდის გამოძახებამდე: " +
                        obieqti2.cvladi1.ToString() + "   " + obieqti2.cvladi2.ToString();
obieqti2.shecvla(obieqti2);
label2.Text = "არგუმენტების მნიშვნელობები მეთოდის გამოძახების შემდეგ: " +
                        obieqti2.cvladi1.ToString() + "   " + obieqti2.cvladi2.ToString();
}
```

პროგრამის obieqti2.shecvla(obieqti2) სტრიქონში ხდება obieqti2 ობიექტის shecvla მეთოდის გამოძახება. მისი არგუმენტია obieqti2. ამიტომ, obieqti2-ის მიმართვა ენიჭება obieqti1 ცვლადს. შედეგად, obieqti1 და obieqti2 მიმართვითი ცვლადები მეხსიერების ერთსა და იმავე უბანს მიმართავენ. ამიტომ, obieqti1 ობიექტის ცვლადების ცვლილება ავტომატურად გამოიწვევს obieqti2 ობიექტის ცვლადების შეცვლას.

ref და out მოდიფიკატორები

როგორც ვიცით, ჩვეულებრივი ტიპის მნიშვნელობა (მაგალითად, int ან char) მეთოდს მნიშვნელობის მიხედვით გადაეცემა. ამ შემთხვევაში პარამეტრის მნიშვნელობის ცვლილება არ შეცვლის არგუმენტის მნიშვნელობას. მაგრამ, თუ გამოვიყენებთ ref და out საკვანძო სიტყვებს, მაშინ შეგვეძლება ჩვეულებრივი ტიპის ნებისმიერი მნიშვნელობა გადავცეთ მიმართვის მიხედვით, რაც მეთოდს მისცემს არგუმენტის შეცვლის შესაძლებლობას.

ჩვეულებრივი ტიპის მქონე მნიშვნელობების გადაცემა მიმართვის მიხედვით საჭიროა მაშინ, როცა მეთოდმა უნდა შეცვალოს თავისი არგუმენტის მნიშვნელობა და როცა მეთოდმა უნდა დააბრუნოს ერთზე მეტი მნიშვნელობა.

ref მოდიფიკატორი

ref მოდიფიკატორის გამოყენება იწვევს პარამეტრისათვის არგუმენტის გადაცემას მიმართვის მიხედვით. ref მოდიფიკატორი ეთითება მეთოდის გამოცხადებისას და მისი გამოძახებისას. ქვემოთ მოყვანილ პროგრამაში მოყვანილია

gacvla() მეთოდი, რომელიც თავისი პარამეტრების მნიშვნელობებს ადგილს უცვლის. შედეგად, იცვლება არგუმენტების მნიშვნელობებიც. ყურადღება მივაქციოთ იმას, რომ მეთოდის გამოცხადებისას ref მოდიფიკატორი ეთითება პარამეტრის ტიპის წინ, ხოლო მეთოდის გამოძახებისას კი არგუმენტის წინ.

ref მოდიფიკატორი შეგვიძლია წარმატებით გამოვიყენოთ მეთოდის შესაქმნელად, რომელიც ადგილებს უცვლის თავის არგუმენტებს. სწორედ ამას აკეთებს მოყვანილი პროგრამა.

```
//      პროგრამა 6.4
//      პროგრამაში ხდება ref მოდიფიკატორის გამოყენების დემონსტრირება
class RicxvebisGacvla
{
//      მეთოდი ადგილებს უცვლის ორ პარამეტრს
public void gacvla(ref int par1, ref int par2)
{
int temp;

temp = par1;
par1   = par2;
par2   = temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//
label1.Text = ""; label2.Text = "";
RicxvebisGacvla obieqti = new RicxvebisGacvla();
int ricxvi1 = Convert.ToInt32(textBox1.Text);
int ricxvi2 = Convert.ToInt32(textBox2.Text);

//      არგუმენტების მნიშვნელობები გაცვლამდე
label1.Text = ricxvi1.ToString() + "   " + ricxvi2.ToString();
obieqti.gacvla(ref ricxvi1, ref ricxvi2);
//      არგუმენტების მნიშვნელობები გაცვლის შემდეგ
label2.Text = ricxvi1.ToString() + "   " + ricxvi2.ToString();
}
```

პროგრამაში gacvla() მეთოდს გამოძახების დროს გადაეცემა ricxvi1 და ricxvi2 არგუმენტების მისამართები, რომლებიც შესაბამისად მიენიჭება par1 და par2 პარამეტრებს. შედეგად, ricxvi1 და par1 მიმართავენ მეხსიერების ერთსა და იმავე უბანს. იგივე ეხება ricxvi2 და par2 ცვლადებს. ამიტომ, par1 და par2 პარამეტრების ცვლილება ავტომატურად გამოიწვევს ricxvi1 და ricxvi2 არგუმენტების მნიშვნელობების შეცვლას.

out მოდიფიკატორი

მეთოდს შეუძლია გამომძახებელ პროგრამას დაუბრუნოს მხოლოდ ერთი მნიშვნელობა. იმისათვის, რომ მეთოდმა დაგვიბრუნოს რამდენიმე მნიშვნელობა, უნდა გამოვიყენოთ out მოდიფიკატორი. მეთოდის გამოცხადებისას out მოდიფიკატორი

უნდა მივუთითოთ იმ პარამეტრის წინ, რომლის დაბრუნებაც გვინდა. out მოდიფიკატორიანი პარამეტრი გამოიყენება მეთოდიდან მხოლოდ ინფორმაციის გასაცემად. მეთოდმა ასეთ პარამეტრს აუცილებლად უნდა მიანიჭოს მნიშვნელობა მანამ, სანამ მართვას დაუბრუნებს გამომძახებელ პროგრამას. მოყვანილ პროგრამაში ნაჩვენებია out მოდიფიკატორის გამოყენების მაგალითი.

```
//      პროგრამა 6.5
//      პროგრამაში ხდება out მოდიფიკატორის გამოყენების დემონსტრირება
class Samkutxedi
{
int gverdi1;
int gverdi2;
int gverdi3;
public Samkutxedi(int par1, int par2, int par3)
{
gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
}
public int SamkutxediInfo(out bool TolGverda)
{
if ( ( gverdi1 == gverdi2 ) && ( gverdi1 == gverdi3 )
  && ( gverdi2 == gverdi3 ) ) TolGverda = true;
        else TolGverda = false;
return gverdi1 + gverdi2 + gverdi3;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int gv1 = Convert.ToInt32(textBox1.Text);
int gv2 = Convert.ToInt32(textBox2.Text);
int gv3 = Convert.ToInt32(textBox3.Text);
Samkutxedi obieqti = new Samkutxedi(gv1, gv2, gv3);
int perimetri;
bool ArisTolgverda;

perimetri = obieqti.SamkutxediInfo(out ArisTolgverda);
if ( ArisTolgverda ) label2.Text = "სამკუთხედი არის ტოლგვერდა";
        else label2.Text = "სამკუთხედი არ არის ტოლგვერდა";
label1.Text = " სამკუთხედის პერიმეტრი =   " + perimetri.ToString();
}
```

პროგრამაში SamkutxediInfo() მეთოდი ორ მნიშვნელობას აბრუნებს. ერთი მნიშვნელობაა სამკუთხედის პერიმეტრი, მეორე კი - ArisTolgverda ლოგიკური ტიპის ცვლადი. მას ენიჭება true მნიშვნელობა იმ შემთხვევაში, თუ სამკუთხედს ტოლი გვერდები აქვს, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობა.

params მოდიფიკატორი

მეთოდის შექმნისას წინასწარ ვაფიქსირებთ მისთვის გადასაცემი არგუმენტების რაოდენობას. მაგრამ, რიგ შემთხვევებში საჭირო ხდება მეთოდისათვის ცვლადი რაოდენობის არგუმენტების გადაცემა. ასეთ შემთხვევაში უნდა გამოვიყენოთ params მოდიფიკატორი. იგი საშუალებას გვაძლევს მეთოდს გადავცეთ არგუმენტების ნებისმიერი რაოდენობა. params მოდიფიკატორი მასივს გამოაცხადებს როგორც პარამეტრს, რომელსაც შეუძლია მიიღოს 0, 1 და ა.შ. რაოდენობის არგუმენტი. მოყვანილ პროგრამაში maxSidide მეთოდს ცვლადი რაოდენობის არგუმენტები გადაეცემა. იგი გასცემს არგუმენტებს შორის მაქსიმალურს.

```
//      პროგრამა 6.6
//      პროგრამაში ხდება params მოდიფიკატორის გამოყენების დემონსტრირება
class Maximumi
{
public int maxSidide(params int[ ] ricxvebi)
{
int max;
if ( ricxvebi.Length == 0 )
{
//  რადგან არგუმენტი არ არის მითითებული, ამიტომ მეთოდი აბრუნებს ნულს
        return 0;
}
max = ricxvebi[0];
for ( int indexi = 1; indexi < ricxvebi.Length; indexi++ )
        if ( ricxvebi[indexi] > max ) max = ricxvebi[indexi];
return max;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Maximumi obieqti = new Maximumi();
int maximaluri;
int i1 = 15, i2 = 25;
int i3 = Convert.ToInt32(textBox1.Text);
int[ ] masivi = new int[] { 11, 22, -33, -44, 5 };

//      მეთოდისთვის ორი არგუმენტის გადაცემა
maximaluri = obieqti.maxSidide(i1, i2);
label1.Text = maximaluri.ToString();
//      მეთოდისთვის ოთხი არგუმენტის გადაცემა
maximaluri = obieqti.maxSidide(i1, i2, -30, i3);
label2.Text = maximaluri.ToString();
//      მეთოდისთვის მასივის გადაცემა
maximaluri = obieqti.maxSidide(masivi);
```

```
label3.Text = maximaluri.ToString();
}
```

maxSidide() მეთოდს ყოველი გამოძახებისას გადაეცემა სხვადასხვა რაოდენობის არგუმენტები ricxvebi მასივის საშუალებით. არგუმენტების ტიპები უნდა იყოს თავსებადი ricxvebi მასივის ტიპთან. მაგალითად, ორივე უნდა იყოს მთელრიცხვა, ან წილადი და ა.შ.

თუ maxSidide() მეთოდს არ გადავცემთ არგუმენტებს, მაშინ იგი ჩაიწერება არგუმენტების გარეშე:

```
maximumi = obieqti.maxSidide();
```

ამ შემთხვევაში საჭიროა ვაკონტროლოთ params მოდიფიკატორით გამოცხადებული პარამეტრის მნიშვნელობა. ამიტომ, maxSidide() მეთოდში მასივთან მიმართვამდე უნდა შევამოწმოთ შეიცავს თუ არა იგი ერთ ელემენტს მაინც (ამისათვის ვიყენებთ if(ricxvebi.Length==0) ოპერატორს).

მეთოდში შეგვიძლია, აგრეთვე, მივუთითოთ როგორც ჩვეულებრივი, ისე ცვლადი სიგრძის პარამეტრები. როცა მეთოდში მითითებულია ჩვეულებრივი და ცვლადი სიგრძის პარამეტრები, მაშინ ცვლადი სიგრძის პარამეტრი პარამეტრების სიაში უნდა იყოს უკანასკნელი. ამასთან, მეთოდს უნდა ჰქონდეს მხოლოდ ერთი ცვლადი სიგრძის პარამეტრი. ქვემოთ მოყვანილ პროგრამაში ShowArgs() მეთოდს ორი პარამეტრი აქვს - ერთი string ტიპის, მეორე კი - int ტიპის ცვლადი სიგრძის.

```
//      პროგრამა 6.7
//      პროგრამაში ხდება ჩვეულებრივი და ცვლადი სიგრძის პარამეტრების
//      ერთობლივად გამოყენების დემონსტრირება
class ChemiKlasi
{
string str1;
public string Metodi(string striqoni, params int[ ] ricxvebi)
{
int jami = 0;

foreach ( int ricxvi in ricxvebi )
        jami += ricxvi;
str1 = striqoni + ”   ”;
str1 += jami.ToString();
return str1;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi();

label1.Text=obieqti.Metodi("მეთოდს 5 არგუმენტი გადაეცა. მათი ჯამია: ", 1, 2, 3, 4, 5);
label2.Text = obieqti.Metodi("მეთოდს 2 არგუმენტი გადაეცა. მათი ჯამია: ", 6, 7);
}
```

მეთოდის მიერ ობიექტის დაბრუნება

მეთოდს შეუძლია დააბრუნოს როგორც ნებისმიერი ტიპის მონაცემი, ისე კლასის ობიექტიც. მაგალითისათვის განვიხილოთ ორი პროგრამა. პირველ პროგრამაში მეთოდი გასცემს სტრიქონების მასივის ელემენტს, მეორე კი - ჩვენს მიერ შექმნილი კლასის ობიექტს.

```
//      პროგრამა 6.8
//      პროგრამაში ხდება მეთოდის მიერ სტრიქონის დაბრუნების დემონსტრირება
class ChemiKlasi
{
string[ ] xili = { "ვაშლი", "მსხალი", "ლეღვი", "ყურძენი" };
//
public string StriqonisDabruneba(int indexi)
{
if ( ( indexi >= 0 ) & ( indexi < xili.Length ) ) return xili[indexi];
        else return "შეტანილია არასწორი ინდექსი";
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება ობიექტის დაბრუნების დემონსტრირება
ChemiKlasi obieqti = new ChemiKlasi();
label1.Text = obieqti.StriqonisDabruneba(3);
label2.Text = obieqti.StriqonisDabruneba(15);
}

//      პროგრამა 6.9
//      პროგრამაში ხდება მეთოდის მიერ ობიექტის დაბრუნების დემონსტრირება
class ChemiKlasi_1
{
public string xilis_saxeli;
public int kodi;
public ChemiKlasi_1(string par1, int par2)
{
xilis_saxeli = par1;
kodi = par2;
}
}
class ChemiKlasi_2
{
string[ ] xili = { "ვაშლი  ", "მსხალი  ", "ლეღვი  ", "ყურძენი  " };
int[ ] kodebi = { 1, 0, 3, 7};
//
public ChemiKlasi_1 ObieqtisDabruneba(int indexi)
{
```

```
if ( ( indexi >= 0 ) && ( indexi < xili.Length ) )
//        ChemiKlasi_1 ობიექტის დაბრუნება
return new ChemiKlasi_1(xili[indexi], kodebi[indexi]);
        else return new ChemiKlasi_1("შეტანილია არასწორი კოდი  ", indexi);
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi_2 obieqti1 = new ChemiKlasi_2();
ChemiKlasi_1 obieqti2;

obieqti2 = obieqti1.ObieqtisDabruneba(3);
label1.Text = obieqti2.xilis_saxeli + obieqti2.kodi.ToString();
obieqti2 = obieqti1.ObieqtisDabruneba(20);
label2.Text = obieqti2.xilis_saxeli + obieqti2.kodi.ToString();
}
```

მეთოდის გადატვირთვა. პოლიმორფიზმი

C# ენაში ერთი კლასის შიგნით ორ ან მეტ მეთოდს შეიძლება ერთნაირი სახელი ჰქონდეს იმ პირობით, რომ ან პარამეტრების რაოდენობა უნდა იყოს სხვადასხვა ან პარამეტრებს სხვადასხვა ტიპი უნდა ჰქონდეთ. ასეთ მეთოდებს გადატვირთვადი ეწოდებათ, ხოლო ერთნაირი სახელის მქონე მეთოდების განსაზღვრის პროცესს კი მეთოდის გადატვირთვა. მეთოდების გადატვირთვა არის პოლიმორფიზმის რეალიზების ერთ-ერთი საშუალება. გადატვირთვადი მეთოდის გამოძახებისას სრულდება ის მეთოდი, რომლის პარამეტრებიც ემთხვევა არგუმენტებს. გადატვირთვადი მეთოდები შეიძლება აბრუნებდნენ, სხვადასხვა ტიპის მონაცემებს. განვიხილოთ მაგალითი.

```
//        პროგრამა 6.10
//        პროგრამაში ხდება მეთოდის გადატვირთვის დემონსტრირება
class Gadatvirtva
{
//        მეთოდს პარამეტრები არ აქვს და აბრუნებს სტრიქონს
public string Metodi()
{
return "გამოცხადებული იქნა უპარამეტრებო მეთოდი";
}
//        მეთოდს double ტიპის ერთი პარამეტრი აქვს და აბრუნებს მის კვადრატს
public double Metodi(double a)
{
return Math.Pow(a,2);
}
//        მეთოდს int ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს
public int Metodi(int a, int b)
```

```
{
return a + b;
}
//      მეთოდს double ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს
public double Metodi(double a, double b)
{
return a + b;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Gadatvirtva obieqti = new Gadatvirtva();
int i2;
double d1, d2;
string str1;
//      Metodi მეთოდის ყველა ვერსიის გამოძახება
str1 = obieqti.Metodi();
label1.Text = str1;
d2 = obieqti.Metodi(5.7);
label2.Text = "გამოძახებული იქნა ერთპარამეტრიანი მეთოდი. შედეგი = " + d2.ToString();
i2 = obieqti.Metodi(5, 10);
label3.Text = "გამოძახებული იქნა ორპარამეტრიანი მეთოდი. შედეგი = " + i2.ToString();
d1 = obieqti.Metodi(5.5, 10.10);
label4.Text = " გამოძახებული იქნა ორპარამეტრიანი მეთოდი. შედეგი = " + d1.ToString();
}
```

პროგრამაში ოთხჯერ ხდება Metodi() მეთოდის გადატვირთვა. პირველ ვერსიას არ აქვს პარამეტრი. მეორე ვერსიას აქვს ერთ მთელი ტიპის პარამეტრი, მესამეს - ორი მთელი ტიპის პარამეტრი და მეოთხეს კი - ორი წილადი პარამეტრი.

თუ პარამეტრების რაოდენობა თანაბარია და არგუმენტებსა და პარამეტრებს განსხვავებული ტიპები აქვს, მაშინ სრულდება ტიპების ავტომატური გარდაქმნა. განვიხილოთ მაგალითი.

```
//      პროგრამა 6.11
//      პროგრამაში სრულდება ტიპების ავტომატური გარდაქმნა მეთოდის
გადატვირთვის დროს
class Gadatvirtva
{
public int Metodi( int par1 )
{
        return par1;
}
public double Metodi( double par1 )
{
        return par1;
}
}
```

```
private void button1_Click(object sender, System.EventArgs e)
{
Gadatvirtva obieqti = new Gadatvirtva();
int    i1 = 15;
double d1 = 15.15;
byte   b1 = 33;
short  s1 = 20;
float  f1 = 5.5F;
//        აქ სრულდება Metodi(int par1) მეთოდი
label1.Text = obieqti.Metodi(i1).ToString();
//        აქ სრულდება Metodi(double par1) მეთოდი
label2.Text = obieqti.Metodi(d1).ToString();
//        აქ სრულდება Metodi(int par1) მეთოდი
label3.Text = obieqti.Metodi(b1).ToString();
//        აქ სრულდება Metodi(int par1) მეთოდი
label4.Text = obieqti.Metodi(s1).ToString();
//        აქ სრულდება Metodi(double par1) მეთოდი
label5.Text = obieqti.Metodi(f1).ToString();
}
```

პროგრამაში Metodi() მეთოდის ერთ ვერსიას აქვს int ტიპის პარამეტრი, მეორეს კი - double ტიპის. მათი გამოძახება ხდება შესაბამისად int, double, byte, short და float ტიპის მქონე არგუმენტებისათვის. byte და short ტიპის მნიშვნელობების გადაცემისას ისინი ავტომატურად გარდაიქმნება int ტიპად. მეთოდისათვის float ტიპის მნიშვნელობის გადაცემისას იგი გარდაიქმნება double ტიპად.

მეთოდების გადატვირთვისას შეგვიძლია ref და out მოდიფიკატორების გამოყენება. ამის დემონსტრირება ქვემოთ მოყვანილ პროგრამაში ხდება.

```
//        პროგრამა 6.12
//        პროგრამაში ხდება ref და out მოდიფიკატორების გამოყენება მეთოდის
გადატვირთვის დროს
class Gadatvirtva
{
public int Metodi(int par1, out int par2)
{
        par2 = par1 * par1;
        return par1;
}
public int Metodi(ref int par3)
{
        return par3;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Gadatvirtva obieqti = new Gadatvirtva();
int i1, i2, i3, i4, i5;
```

```
i2 = Convert.ToInt32(textBox1.Text);
i4 = Convert.ToInt32(textBox2.Text);
i1 = obieqti.Metodi(i2, out i5);
label1.Text = i1.ToString() + ”   ” + i5.ToString();
i3 = obieqti.Metodi(ref i4);
label2.Text = i3.ToString();
}
```

კონსტრუქტორების გადატვირთვა

მეთოდების მსგავსად შესაძლებელია კონსტრუქტორების გადატვირთვაც. ეს საშუალებას გვაძლევს ობიექტები შევქმნათ სხვადასხვა საშუალებებით. მოყვანილ პროგრამაში ხდება გადატვირთვადი კონსტრუქტორის დემონსტრირება.

```
//      პროგრამა 6.13
//      პროგრამაში ხდება გადატვირთვადი კონსტრუქტორის დემონსტრირება
class ChemiKlasi
{
public int x;
public ChemiKlasi()
{
x = 0;
}
public ChemiKlasi(int par1)
{
x = par1 * par1;
}
public ChemiKlasi(double par2)
{
x = (int) par2 + 10;
}
public ChemiKlasi(int par3, int par4)
{
x = par3 * par4;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int ricxvi = Convert.ToInt32(textBox1.Text);

ChemiKlasi obieqti1 = new ChemiKlasi();
label1.Text = "" + obieqti1.x.ToString();
ChemiKlasi obieqti2 = new ChemiKlasi(ricxvi);
label2.Text = "" + obieqti2.x.ToString();
ChemiKlasi obieqti3 = new ChemiKlasi(28.89);
label3.Text = "" + obieqti3.x.ToString();
```

```
ChemiKlasi obieqti4 = new ChemiKlasi(3, 5);
label4.Text = "" + obieqti4.x.ToString();
}
```

პროგრამაში მოყვანილია ChemiKlasi() მეთოდის ოთხი ვერსია. საჭირო მათგანის არჩევა ხდება არგუმენტების მიხედვით.

მეთოდების გადატვირთვა იძლევა იმის საშუალებას, რომ ერთი ობიექტის ინიციალიზებისათვის კონსტრუქტორმა მეორე ობიექტი გამოიყენოს. მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```
//      პროგრამა 6.14
//      პროგრამაში ერთი ობიექტის ინიციალეზაციისათვის კონსტრუქტორი მეორე
ობიექტს იყენებს
class Shekreba
{
public int jami;
//      ობიექტის შესაქმნელად კონსტრუქტორი იყენებს int ტიპის პარამეტრს
public Shekreba(int raodenoba)
{
jami = 0;
for ( int indexi = 1; indexi <= raodenoba; indexi++ )
        jami += indexi;
}
//      ობიექტის ინიციალიზაციისათვის კონსტრუქტორი იყენებს სხვა ობიექტს,
//      რომელიც მას გადაეცემა, როგორც პარამეტრი
public Shekreba(Shekreba obieqti)
{
jami = obieqti.jami;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int ricxvi1 = Convert.ToInt32(textBox1.Text);
Shekreba obieqti1 = new Shekreba(ricxvi1);
Shekreba obieqti2 = new Shekreba(obieqti1);

label1.Text = "";
label2.Text = "";
label1.Text = "რიცხვების ჯამი 0-დან " + ricxvi1.ToString() + "-მდე =  " +
                        obieqti1.jami.ToString();
label2.Text = "ცვლადის მნიშვნელობა =  " + obieqti2.jami.ToString();
}
```

პროგრამაში პირველი კონსტრუქტორის არგუმენტია მთელი რიცხვი, მეორე კონსტრუქტორის კი - ობიექტი. პირველი კონსტრუქტორი ანგარიშობს ჯამს ნულიდან textBox1 კომპონენტში შეტანილ რიცხვამდე. მეორე კონსტრუქტორი jami ცვლადს ანიჭებს მითითებული ობიექტის შესაბამისი ცვლადის მნიშვნელობას და აღარ ხდება საჭირო ჯამის მეორეჯერ გამოთვლა. ანალოგიურად შეგვიძლია შევადგინოთ

ფაქტორიალის გამოთვლის პროგრამა.

გადატვირთვადი კონსტრუქტორის გამოძახება this სიტყვის გამოყენებით

რიგ შემთხვევებში გადატვირთვადი კონსტრუქტორის გამოძახებისას სასარგებლოა გამოვიყენოთ ერთი კონსტრუქტორის მიერ მეორის გამოძახება. C# ენაში ეს რეალიზდება this სიტყვის გამოყენებით. მისი სინტაქსია:

კონსტრუქტორის_სახელი(პარამეტრების_სია) : this*(არგუმენტების_სია)*
{
კონსტრუქტორის კოდი
}

გამომძახებელი კონსტრუქტორის შესრულებისას ჯერ გამოიძახება ის გადატვირთვადი კონსტრუქტორი, რომელშიც პარამეტრების სია ემთხვევა არგუმენტების სიას, შემდეგ კი შესრულდება გამომძახებელი კონსტრუქტორის დანარჩენი ოპერატორები. განვიხილოთ პროგრამა.

```
//      პროგრამა 6.15
//      პროგრამაში ხდება გადატვირთვადი კონსტრუქტორის გამოძახება this სიტყვის
გამოყენებით
class ChemiKlasi
{
public int cvladi1, cvladi2;
public ChemiKlasi() : this(0,0)
{
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi() კონსტრუქტორში");
}
public ChemiKlasi(ChemiKlasi obieqti) : this(obieqti.cvladi1, obieqti.cvladi2)
{
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi(obieqti.cvladi1, obieqti.cvladi2)
კონსტრუქტორში");
}
public ChemiKlasi(int par1, int par2)
{
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi(int, int) კონსტრუქტორში");
cvladi1 = par1;
cvladi2 = par2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      ხდება this სიტყვის გამოყენების დემონსტრირება ერთი კონსტრუქტორის
//      გამოსაძახებლად მეორის მიერ
int ricxvi1 = Convert.ToInt32(textBox1.Text);
```

```
int ricxvi2 = Convert.ToInt32(textBox2.Text);
ChemiKlasi obieqti1 = new ChemiKlasi();
ChemiKlasi obieqti2 = new ChemiKlasi(ricxvi1, ricxvi2);
ChemiKlasi obieqti3 = new ChemiKlasi(obieqti2);

label1.Text = " ";
label2.Text = " ";
label3.Text = " ";
label1.Text = obieqti1.cvladi1.ToString() + "   " + obieqti1.cvladi2.ToString();
label2.Text = obieqti2.cvladi1.ToString() + "   " + obieqti2.cvladi2.ToString();
label3.Text = obieqti3.cvladi1.ToString() + "   " + obieqti3.cvladi2.ToString();
}
```

ChemiKlasi კლასში ერთადერთი კონსტრუქტორია - ChemiKlasi(int, int). დანარჩენი ორი კონსტრუქტორი იყენებს this სიტყვას და ახდენენ ChemiKlasi(int, int) კონსტრუქტორის გამოძახებას. მაგალითად, obieqti1 ობიექტის შექმნისას გამოიძახება მისი კონსტრუქტორი - ChemiKlasi(), რომელიც თავის მხრივ გამოიძახებს this(0,0) კონსტრუქტორს. ე.ი. this სიტყვა ეთითება ChemiKlasi(0,0) კონსტრუქტორის სრული სახელის ნაცვლად. შემდეგ შესრულდება ChemiKlasi() კონსტრუქტორის კოდი, კერძოდ, MessageBox() მეთოდი. ანალოგიურად იქმნება obieqti2 და obieqti3 ობიექტები.

გადატვირთვადი კონსტრუქტორების გამოძახებისათვის this სიტყვის გამოყენების ერთ-ერთი მიზეზი არის კოდის დუბლირების თავიდან აცილების შესაძლებლობა. უკანასკნელ მაგალითში this მიმართვის გამოყენების გარეშე მოგვიწევდა მაინიციალიზებელი ოპერატორების მიმდევრობის (cvladi1 = par1; cvladi2 = par2) გამეორება სამივე კონსტრუქტორში.

რეკურსია

მეთოდს შეუძლია თავისი თავის გამოძახება. ამ პროცესს რეკურსია ეწოდება, ხოლო მეთოდს, რომელიც თავის თავს იძახებს - რეკურსიული. რეკურსიული მეთოდის საკვანძო კომპონენტი არის ოპერატორი, რომელიც ამავე მეთოდს იძახებს.

რეკურსიის კლასიკური მაგალითია ფაქტორიალის გამოთვლა. N რიცხვის ფაქტორიალი აღინიშნება N!-ით და არის რიცხვების ნამრავლი 1-დან N-მდე. მაგალითად, 4-ის ფაქტორიალი არის 1x2x3x4=24. ქვემოთ მოყვანილ პროგრამაში წარმოდგენილი რეკურსიული მეთოდი განკუთვნილია რიცხვის ფაქტორიალის გამოთვლისათვის.

```
//      პროგრამა 6.16
//      პროგრამაში ხდება რეკურსიის დემონსტრირება
class Factorial
{
//      რეკურსიული მეთოდი
public int factor(int ricxvi)
{
int shedegi;
if ( ricxvi == 1 ) return 1;
shedegi = factor(ricxvi -1) * ricxvi;
```

```
return shedegi;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Factorial obieqti = new Factorial();
int number = Convert.ToInt32(textBox1.Text);
int fact1;

label1.Text = "";
fact1 = obieqti.factor(number);
label1.Text = fact1.ToString();
}
```

რეკურსიული მეთოდის შედგენისას სწორად უნდა განისაზღვროს რეკურსიის დამთავრების პირობა. ამ დროს უნდა შეწყდეს მეთოდის გამოძახება და მან უნდა დააბრუნოს მნიშვნელობა. ჩვენს შემთხვევაში, რეკურსიული გამოძახება მთავრდება მაშინ, როცა არგუმენტი 1-ის ტოლი ხდება. ამ დროს მეთოდი აბრუნებს 1-ს.

რეკურსიული მეთოდების გამოყენებას აქვს როგორც დადებითი, ისე უარყოფითი მხარეები. უარყოფითი მხარეა ის, რომ რეკურსიული პროგრამები შედარებით ნელა სრულდება. გარდა ამისა, თუ რეკურსიული გამოძახებების რაოდენობა ძალიან დიდია, მაშინ შეიძლება აღიძრას განსაკუთრებული სიტუაცია (შეცდომა). ეს ძირითადად მაშინ ხდება, როცა არასწორად არის განსაზღვრული რეკურსიის დამთავრების პირობა: მეთოდის გამოძახების ნაცვლად უნდა მოხდეს მნიშვნელობის დაბრუნება.

დადებითი მხარეა ის, რომ რიგი ალგორითმებისა რეკურსიულად შეიძლება რეალიზებული იყოს უფრო ეფექტურად და მარტივად. მაგალითად, სწრაფი დახარისხების, კატალოგებში ფაილების ძებნისა და ა.შ. რეკურსიული მიდგომა ხშირად გამოიყენება, აგრეთვე, ხელოვნური ინტელექტის სფეროში.

static მოდიფიკატორი

ზოგჯერ საჭიროა კლასის ისეთი წევრის განსაზღვრა, რომელიც გამოყენებული იქნება ამავე კლასის ნებისმიერი ობიექტისაგან დამოუკიდებლად. ჩვეულებრივ, კლასის წევრთან მიმართვა ხორციელდება ამ კლასის ობიექტის გამოყენებით, მაგრამ არსებობს კლასის წევრთან მიმართვის შესაძლებლობა ობიექტთან მიმართვის გარეშეც. კლასის ასეთ წევრებს სტატიკური წევრები ეწოდებათ და მათი გამოცხადებისათვის გამოიყენება static სიტყვა. თუ კლასის წევრი განისაზღვრება static მოდიფიკატორით, მაშინ ის მისაწვდომი ხდება ამ კლასის ობიექტების შექმნამდე. static სიტყვა შეგვიძლია გამოვიყენოთ როგორც მეთოდების, ისე ცვლადების გამოცხადებისათვის.

კლასის სტატიკურ წევრთან მიმართვისათვის უნდა მივუთითოთ კლასის სახელი, ოპერატორი წერტილი (.) და წევრის სახელი. სტატიკურ მეთოდთან მიმართვა ანალოგიურად სრულდება. სტატიკური მეთოდი ჩვეულებრივი მეთოდისაგან იმით განსხვავდება, რომ იგი შეიძლება გამოძახებული იყოს მისი კლასის სახელის გამოყენებით ამ კლასის რაიმე ობიექტის შექმნის გარეშე.

სტატიკური ცვლადი ფაქტიურად გლობალური ცვლადია. მისი კლასის ობიექტების გამოცხადებისას არ იქმნება ასეთი ცვლადის ასლი. სამაგიეროდ კლასის ყველა ობიექტი ერთობლივად იყენებს სტატიკურ ცვლადს, რომლის ინიციალიზება ხდება მისი კლასის ჩატვირთვისას. თუ საწყისი მნიშვნელობა არ არის აშკარად მითითებული, გაჩუმებით ასეთ ცვლადს ენიჭება მნიშვნელობა 0 (რიცხვითი ტიპის ცვლადისთვის), null მნიშვნელობა (მიმართვითი ტიპის ცვლადისთვის) ან false მნიშვნელობა (bool ტიპის ცლადისთვის). ამრიგად, static მოდიფიკატორიან ცვლადს ყოველთვის აქვს მნიშვნელობა.

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია სტატიკურ ცვლადთან და მეთოდთან მუშაობა.

```
//      პროგრამა 6.17
//      პროგრამაში ხდება სტატიკურ ცვლადთან და მეთოდთან მუშაობის
დემონსტრირება
class ChemiKlasi
{
//      სტატიკური ცვლადის გამოცხადება
public static int Statikuri_cvladi = 100;
//      სტატიკური მეთოდის გამოცხადება
public static int Statikuri_Metodi()
{
        return Statikuri_cvladi / 2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
label1.Text = ChemiKlasi.Statikuri_cvladi.ToString();
//      სტატიკურ ცვლადს მნიშვნელობა ენიჭება
ChemiKlasi.Statikuri_cvladi = Convert.ToInt32(textBox1.Text);
label2.Text = ChemiKlasi.Statikuri_cvladi.ToString();
label3.Text = ChemiKlasi.Statikuri_Metodi().ToString();
}
```

როგორც ვხედავთ სტატიკური ცვლადის ინიციალიზაცია ხდება მისი კლასის რაიმე ობიექტის შექმნის გარეშე.

სტატიკურ მეთოდებს შემდეგი შეზღუდვები აქვთ:

- არ აქვთ this მიმართვა;
- შეუძლიათ უშუალოდ გამოიძახონ მხოლოდ სტატიკური მეთოდები და არ შეუძლიათ გამოიძახონ თავიანთი კლასის ობიექტის მეთოდები;
- შეუძლიათ პირდაპირ მიმართონ მხოლოდ სტატიკურ მონაცემებს.

სტატიკურ მეთოდს მხოლოდ თავისი კლასის ობიექტის გამოყენებით შეუძლია მიმართოს თავისი კლასის ობიექტის მეთოდებს და ცვლადებს (ე.ი. ობიექტის ცვლადთან ან მეთოდთან მიმართვისათვის საჭიროა ამ ობიექტის სახელის მითითება). ქვემოთ მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```
//      პროგრამა 6.18
//      პროგრამაში ხდება სტატიკური მეთოდის მიერ ჩვეულებრივი მეთოდის
გამოძახების დემონსტრირება
```

```
class ChemiKlasi
{
int ricxvi1;                        //      ჩვეულებრივი ცვლადის გამოცხადება
static int ricxvi2 = 555;           //      სტატიკური ცვლადის გამოცხადება
//      ჩვეულებრივი მეთოდი
void ChveulebriviMetodi()
{
ricxvi1 = 50;
ricxvi2 = 70;
MessageBox.Show("ჩვეულებრივი ცვლადი ricxvi1 - " + ricxvi1.ToString() +
                    " სტატიკური ცვლადი ricxvi2 - " + ricxvi2.ToString());
}
//      სტატიკური მეთოდი
public static void StatikuriMetodi(ChemiKlasi obieqti)
{
obieqti.ChveulebriviMetodi();           //      მეთოდის ეს გამოძახება ნამდვილია
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obj = new ChemiKlasi();

ChemiKlasi.StatikuriMetodi(obj);
}
```

თავი 7. ოპერატორების გადატვირთვა, თვისებები და ინდექსატორები

ოპერატორების გადატვირთვა

C# ენაში შეგვიძლია განვსაზღვროთ მოქმედებები (ისინი განსხვავდება ოპერატორის მიერ ავტომატურად შესრულებული მოქმედებებისაგან), რომლებსაც ოპერატორი შეასრულებს ჩვენს მიერ შექმნილი კლასის მიმართ. ამას ეწოდება ოპერატორის გადატვირთვა ან ხელახალი განსაზღვრა. ოპერატორის ფუნქციები განსხვავდებიან იმის და მიხედვით, თუ რომელი კლასის მიმართ იქნებიან გამოყენებული. მაგალითად, კლასში, რომელიც განსაზღვრავს ბმულ სიას, + ოპერატორი გამოიყენება სიისათვის ობიექტის დასამატებლად. კლასში, რომელიც ახდენს სტეკის რეალიზებას, ეს ოპერატორი გამოიყენება სტეკში ობიექტის მოსათავსებლად და ა.შ.

უნდა გვახსოვდეს, რომ გადატვირთვის შესრულებისას ოპერატორის საწყისი დანიშნულება არ იკარგება. უბრალოდ მის არსენალს ემატება ახალი ფუნქციები, რომლებიც გამოყენებული იქნება გარკვეული კლასის მიმართ. მაგალითად, + ოპერატორის გადატვირთვის შემთხვევაში მის არსენალს დაემარება ახალი ფუნქცია, აგრეთვე, შენარჩუნებული იქნება მისი საწყისი დანიშნულება - შეასრულოს მთელი რიცხვების შეკრება.

ოპერატორის გადატვირთვის გამოყენების ძირითადი უპირატესობა იმაში მდგომარეობს, რომ ადვილად შეიძლება პროგრამაში კლასის ახალი ტიპის ინტეგრირება. თუ კლასისათვის განსაზღვრულია ოპერატორები, მაშინ ამ კლასის ობიექტებზე ოპერაციები შეიძლება შევასრულოთ ჩვეულებრივი სინტაქსის გამოყენებით, ე.ი. კლასის ცვლადებთან სამუშაოდ საკმარისია შევიტანოთ ასეთი სტრიქონი: ობიექტი + ობიექტი.

ოპერატორის გადატვირთვისთვის გამოიყენება operator სიტყვა. იგი განსაზღვრავს ოპერატორის მეთოდს, რომელიც აღწერს ოპერატორის მიერ შესასრულებელ მოქმედებებს.

ოპერატორის მეთოდი

C# ენაში ძირითადად გამოიყენება ოპერატორების ორი სახე - უნარული და ბინარული. უნარულია ოპერატორი, რომელსაც ერთი ოპერანდი აქვს. ბინარულია ოპერატორი, რომელსაც ორი ოპერანდი აქვს.

უნარული ოპერატორის გადატვირთვისათვის გამოყენებული მეთოდის სინტაქსია:

მიმართვის_მოდიფიკატორი static *შედეგის_ტიპი* operator *ოპერატორი(ტიპი ოპერანდი)*
{
მეთოდის კოდი
}

ბინარული ოპერატორის გადატვირთვისათვის გამოყენებული მეთოდის სინტაქსია:

მიმართვის_მოდიფიკატორი static *შედეგის_ტიპი* operator *ოპერატორი(ტიპი1 ოპერანდი1,*
ტიპი2 ოპერანდი2)

```
{
მეთოდის კოდი
}
```

აქ *ოპერატორი* არის გადატვირთვადი ოპერატორი, მაგალითად, + ან -. *შედეგის_ტიპი* არის იმ მნიშვნელობის ტიპი, რომელიც გაიცემა მოცემული მეთოდის მიერ.

უნარული ოპერატორების გადატვირთვისას ოპერანდი უნდა ეკუთვნოდეს იმ კლასის ტიპს, რომლისთვისაც განისაზღვრება ოპერატორი. ბინარული ოპერატორების გადატვირთვისას ერთ-ერთი ოპერანდის ტიპი უნდა ემთხვეოდეს იმ კლასის ტიპს, რომლისთვისაც ხდება ოპერატორის ხელახალი განსაზღვრა.

C# ენაში არ შეიძლება ოპერატორების გადატვირთვა სტანდარტული ტიპის მქონე ობიექტებისათვის. მაგალითად, არ შეიძლება + ოპერატორის გადატვირთვა უკვე არსებული ტიპებისათვის, როგორიცაა მაგალითად, int ან string ტიპი. ოპერატორების გადატვირთვა შეიძლება მხოლოდ ჩვენს მიერ შექმნილი კლასების ობიექტებისათვის. გარდა ამისა, ოპერატორის პარამეტრები არ უნდა შეიცავდეს ref ან out მოდიფიკატორს.

ბინარული ოპერატორების გადატვირთვა

განვიხილოთ პროგრამა, რომელშიც ხდება + ბინარული ოპერატორის გადატვირთვა. თავდაპირველად იქმნება Sivrce კლასი, რომელიც შეიცავს სამგანზომილებიანი ობიექტის კოორდინატებს. გადატვირთული + ოპერატორი ასრულებს Sivrce კლასის ორი ობიექტის შესაბამისი კოორდინატების შეკრებას.

```
//      პროგრამა 7.1
//      პროგრამაში ხდება + ბინარული ოპერატორის გადატვირთვა
class Sivrce
{
int x, y, z;                            //      ობიექტის კოორდინატები
public Sivrce()
{
x = y = z = 0;
}
public Sivrce(int a, int b, int c)
{
x = a;
y = b;
z = c;
}
//      + ბინარული ოპერატორის გადატვირთვა
public static Sivrce operator + (Sivrce operandi1, Sivrce operandi2)
{
Sivrce shedegi = new Sivrce();
//      კოორდინატების შეკრება და შედეგის დაბრუნება
shedegi.x = operandi1.x + operandi2.x;      //      ამ სამ სტრიქონში სრულდება მთელი
shedegi.y = operandi1.y + operandi2.y;      //      რიცხვების შეკრების ოპერაცია,
shedegi.z = operandi1.z + operandi2.z;      //      რომლებისთვისაც + ოპერატორი ინახავს
                                            //      თავის საწყის მნიშვნელობას
```

```
return shedegi;
}
// x, y და z კოორდინატების მნიშვნელობების გამოტანა
public string show()
{
return "x = " + x.ToString() + " y = " + y.ToString() + " z = " + z.ToString());
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება გადატვირთული ოპერატორების გამოყენების დემონსტრირება
Sivrce obieqti1 = new Sivrce(3, 4, 5);
Sivrce obieqti2 = new Sivrce(5, 5, 5);
Sivrce obieqti3 = new Sivrce();

label1.Text = obieqti1.show();        // obieqti1-ის კოორდინატების გამოტანა
label2.Text = obieqti2.show();        // obieqti2- ის კოორდინატების გამოტანა
// obieqti1-სა და obieqti2-ის შეკრების ოპერაცია
obieqti3 = obieqti1 + obieqti2;
// obieqti1-სა და obieqti2-ის შეკრების ოპერაციის შედეგი
label3.Text = obieqti3.show();
// obieqti1-ის, obieqti2-სა და obieqti3-ის შეკრების ოპერაცია
obieqti3 = obieqti1 + obieqti2 + obieqti3;
label4.Text = obieqti3.show();
}
```

operator +() მეთოდის შიგნით ცალკეული კოორდინატების მნიშვნელობების შეკრება დაიყვანება მთელი რიცხვების შეკრებაზე. x, y და z კოორდინატებს აქვთ მთელი ტიპი, ამიტომ + ოპერატორის გადატვირთვა Sivrce კლასის ობიექტებისათვის გავლენას არ მოახდენს მოქმედებებზე, რომლებიც ამ ოპერატორის მიერ სრულდება მთელ რიცხვებზე.

- ოპერატორი ისევე მოქმედებს, როგორც +, მაგრამ მისი გადატვირთვისას უნდა გავითვალისწინოთ ოპერანდების მიმდევრობა. გავიხსენოთ, რომ შეკრება კომუტაციურია, ხოლო გამოკლება - არა (ე.ი. A - B გამოსახულება არაა B - A გამოსახულების იგივური). ყველა ბინარული ოპერატორისთვის პირველი პარამეტრი არის მარცხენა ოპერანდი, მეორე კი - მარჯვენა. არაკომუტაციური ოპერატორების გადატვირთვისას უნდა გავითვალისწინოთ ოპერანდების ურთიერთგანლაგება, კომუტაციური ოპერატორების გადატვირთვისას კი - არა.

უნარული ოპერატორების გადატვირთვა

უნარული ოპერატორების გადატვირთვა ხორციელდება ბინარული ოპერატორების გადატვირთვის ანალოგიურად, მაგრამ, ამ შემთხვევაში მხოლოდ ერთი ოპერანდი გვაქვს. უნარული ოპერატორის გადატვირთვა განვიხილოთ ინკრემენტისა და დეკრემენტის ოპერატორების მაგალითზე. როგორც ვიცით მათი დანიშნულებაა შესაბამისი ოპერანდების მნიშვნელობების ერთით გაზრდა და შემცირება, ამიტომ ამ ოპერატორების გადატვირთულმა ვერსიებმაც უნდა შეასრულონ იგივე მოქმედებები.

ქვემოთ მოყვანილი პროგრამაში ხდება ++ უნარული ოპერატორის გადატვირთვის დემონსტრირება.

```
//      პროგრამა 7.2
//      პროგრამაში ხდება ++ უნარული ოპერატორის გადატვირთვა
class Sivrce
{
int x, y, z;                    //      ობიექტის კოორდინატები
public Sivrce()
{
x = y = z = 0;
}
public Sivrce(int a, int b, int c)
{
x = a;
y = b;
z = c;
}
//      ++ უნარული ოპერატორის გადატვირთვა
public static Sivrce operator ++(Sivrce operandi)
{
//      სრულდება არგუმენტის ცვლილება
operandi.x++;
operandi.y++;
operandi.z++;
return operandi;
}
// x, y და z კოორდინატების მნიშვნელობების გამოტანა
public void show()
{
DialogResult pasuxi =
MessageBox.Show("x="+x.ToString()+"\ny="+y.ToString()+"\nz="+z.ToString());
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება გადატვირთული ოპერატორების გამოყენების დემონსტრირება
Sivrce obieqti = new Sivrce(3, 4, 5);

obieqti.show();                 //      obieqti-ის კოორდინატების გამოტანა
//      გადატვირთული ++ ოპერატორის შესრულება
obieqti++;
obieqti.show();
}
```

როგორც ვიცით, ++ და -- ოპერატორებს აქვთ პრეფიქსური და პოსტფიქსური ფორმა. მათი გადატვირთვის შედეგად ორივე ფორმისათვის გამოიძახება ერთი და იგივე მეთოდი. გადატვირთვის შესრულებისას შეუძლებელია მივუთითოთ განსხვავება

ამ ოპერატორების პრეფიქსურ და პოსტფიქსურ ფორმებს შორის.

შედარების ოპერატორების გადატვირთვა

შედარების ოპერატორები, როგორიცაა ==, !=, <, >, <= ან >= შეიძლება იყოს გადატვირთული. როგორც წესი, შედარების გადატვირთული ოპერატორები აბრუნებენ true ან false მნიშვნელობას. ამიტომ, რჩება ამ ოპერატორების ჩვეულებრივი გამოყენების შესაძლებლობა, ხოლო შედარების გადატვირთული ოპერატორები შეგვიძლია გამოვიყენოთ პირობის გამოსახულებებში.

ქვევით მოყვანილია Sivrce კლასის ვერსია, რომელშიც სრულდება < და > ოპერატორების გადატვირთვა. ამ ოპერატორების გადატვირთული ვერსიების თანახმად ერთი ობიექტი ითვლება მეორეზე დიდად, თუ ობიექტის ყველა კოორდინატი მეტია მეორე ობიექტის შესაბამის კოორდინატებზე. შესაბამისად, ერთი ობიექტი ითვლება მეორეზე ნაკლებად თუ მისი კოორდინატები ნაკლებია მეორე ობიექტის კოორდინატებზე.

```
//      პროგრამა 7.3
//      პროგრამაში ხდება > და <  შედარების ოპერატორების გადატვირთვა
class Sivrce
{
int x, y, z;                            //      ობიექტის კოორდინატები
public Sivrce()
{
        x = y = z = 0;
}
public Sivrce(int a, int b, int c)
{
x = a;
y = b;
z = c;
}
//      < ოპერატორის გადატვირთვა
public static bool operator < (Sivrce operandi1, Sivrce operandi2)
{
if (  ( operandi1.x < operandi2.x ) && ( operandi1.y < operandi2.y ) &&
        ( operandi1.z < operandi2.z ) ) return true;
                else return false;
}
//      > ოპერატორის გადატვირთვა
public static bool operator > (Sivrce operandi1, Sivrce operandi2)
{
if (  ( operandi1.x > operandi2.x ) && ( operandi1.y > operandi2.y ) &&
        ( operandi1.z > operandi2.z ) ) return true;
                else return false;
}
// x, y და z კოორდინატების მნიშვნელობების გამოტანა
public void show()
```

```
{
DialogResult pasuxi =
MessageBox.Show("x="+x.ToString()+"\ny="+y.ToString()+"\nz="+z.ToString());
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება გადატვირთული ოპერატორების გამოყენების დემონსტრირება
Sivrce obieqti1 = new Sivrce(3, 4, 5);
Sivrce obieqti2 = new Sivrce(5, 5, 5);
obieqti1.show();                    // obieqti1-ის კოორდინატები
obieqti2.show();                    // obieqti2-ის კოორდინატები
if ( obieqti1 > obieqti2 )
        label1.Text = "გამოსახულება obieqti1 > obieqti2 სამართლიანია";
if ( obieqti1 < obieqti2 )
        label1.Text = " გამოსახულებაobieqti1 < obieqti2 სამართლიანია";
}
```

შედარების ოპერატორების გადატვირთვა უნდა შევასრულოთ წყვილობრივად. მაგალითად, თუ გადაიტვირთება < ოპერატორი, მაშინ, უნდა გადაიტვირთოს > ოპერატორიც და პირიქით. C# ენაში ისევე, როგორც დაპროგრამების სხვა ენებში, არსებობს შედარების ოპერატორების შემდეგი წყვილები:

```
==      !=
<       >
<=      >=
```

ოპერატორების გადატვირთვისას არსებული შეზღუდვები

ოპერატორების გადატვირთვა დაკავშირებულია გარკვეულ შეზღუდვებთან. კერძოდ, არ შეიძლება ოპერაციების პრიორიტეტებისა და ოპერატორის ოპერანდების რაოდენობის შეცვლა. ამას გარდა, არ შეიძლება გადატვირთული იყოს მინიჭების ოპერატორი და მინიჭების შედგენილი ოპერატორები. აგრეთვე, ქვემოთ მოყვანილი ოპერატორები:

```
&&              ||          [ ]     ()      new     is
sizeof          Typeof?     ->      .       =
```

თვისებები

თვისება არის კლასის წევრი, რომელიც საშუალებას გვაძლევს მეთოდების საშუალებით მივიღოთ ან შევცვალოთ ცვლადების (ველების) მნიშვნელობები. თვისებების გამოყენება მოხერხებულია მაშინ, როცა საჭიროა ცვლადის მნიშვნელობებზე შესასრულებელი ოპერაციების კონტროლი (იმის კონტროლი თუ რომელი ოპერაციებია დასაშვები ცვლადის მნიშვნელობებზე და რომელი - არა). მაგალითად, შეიძლება საჭირო გახდეს მნიშვნელობების დიაპაზონის შეზღუდვა, რომელიც ამ ცვლადს ენიჭება. ბუნებრივია, ასეთი მიზნის მისაღწევად შეგვიძლია გამოვიყენოთ პრივატული ცვლადი და მეთოდი, რომელიც მასთან მუშაობს. მაგრამ გაცილებით მარტივი და უკეთესია პრივატული ცვლადებისა და თვისებების

ერთობლივი გამოყენება.

თვისება შედგება სახელისა და მიმართვის get და set მეთოდებისაგან. მიმართვის get მეთოდი გამოიყენება ცვლადის მნიშვნელობის მისაღებად, set მეთოდი კი ცვლადისთვის მნიშვნელობის მისანიჭებლად. set მეთოდს ავტომატურად გადაეცემა value არაცხადი პარამეტრი, რომელიც შეიცავს ცვლადისათვის მისანიჭებელ მნიშვნელობას. თვისების ძირითადი ღირსებაა ის, რომ მისი სახელი შეიძლება გამოვიყენოთ გამოსახულებებში და მინიჭების ოპერაციებში, როგორც ჩვეულებრივი ცვლადი. ამ დროს ავტომატურად გამოიძახება მიმართვის get და set მეთოდები. გარდა ამისა, თვისება მართავს ცვლადთან მიმართვას. თვისებაში არ ხდება ცვლადის გამოცხადება. საჭიროების მიხედვით თვისებაში შეიძლება განისაზღვროს მხოლოდ get ან set მეთოდი, ან ორივე ერთად.

თვისების სინტაქსია:

მიმართვის_მოდიფიკატორი ტიპი თვისების_სახელი
{
get
{
get *მეთოდის კოდი*
}
set
{
set *მეთოდის კოდი*
}
}

აქ *ტიპი* თვისების ტიპია (მაგალითად, int ტიპი). თვისების განსაზღვრის შემდეგ მისი სახელის გამოყენება, იწვევს მიმართვის შესაბამისი მეთოდის გამოძახებას. თუ თვისების სახელი მითითებულია მინიჭების ოპერატორის მარცხნივ, მაშინ გამოიძახება set მეთოდი. თუ თვისების სახელი მითითებულია მინიჭების ოპერატორის მარჯვნივ, მაშინ გამოიძახება get მეთოდი.

ქვემოთ მოყვანილ პროგრამაში განისაზღვრება ChemiTviseba თვისება, რომელიც უზრუნველყოფს Tviseba ცვლადთან მიმართვას. მოცემულ შემთხვევაში თვისება უშვებს ცვლადისათვის მხოლოდ დადებითი მნიშვნელობების მინიჭებას.

```
//      პროგრამა 7.4
//      პროგრამაში ხდება თვისებასთან მუშაობის დემონსტრირება
class MartiviTviseba
{
int tviseba;
public MartiviTviseba()
{
tviseba = 0;
}
//      ChemiTviseba თვისების გამოცხადება
public int ChemiTviseba
{
get
{
```

```
        return tviseba;
}
set
{
        //      tviseba ცვლადს ენიჭება value მნიშვნელობა თუ ის დადებითია
        if ( value >= 0 ) tviseba = value;
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//
MartiviTviseba obieqti = new MartiviTviseba();
int ricxvi = Convert.ToInt32(textBox1.Text);

label1.Text = obieqti.ChemiTviseba.ToString();
//      ChemiTviseba თვისებას ენიჭება ricxvi ცვლადის დადებითი მნიშვნელობა
obieqti.ChemiTviseba = ricxvi;                  // ricxvi ცვლადის მნიშვნელობა ავტომატურად
                                                // ენიჭება  value ცვლადს
label2.Text = obieqti.ChemiTviseba.ToString();
//      obieqti.ChemiTviseba თვისებისთვის უარყოფითი მნიშვნელობის მინიჭების მცდელობა
obieqti.ChemiTviseba = -ricxvi;
label3.Text = obieqti.ChemiTviseba.ToString();
}
```

ახლა დაწვრილებით განვიხილოთ პროგრამა. მასში განისაზღვრება ერთი პრივატული tviseba ცვლადი და ChemiTviseba თვისება, რომელიც მართავს ამ ცვლადთან მიმართვას. რადგან tviseba ცვლადი არის დახურული, მასთან მიმართვა შესაძლებელია მხოლოდ ChemiTviseba თვისების გამოყენებით.

ChemiTviseba თვისება განისაზღვრება როგორც public, ამიტომ მასთან მიმართვა შეიძლება შესრულდეს სხვა მეთოდების მხრიდან. მიმართვის get მეთოდი აბრუნებს ცვლადის მნიშვნელობას, ხოლო მიმართვის set მეთოდი კი tviseba ცვლადს ანიჭებს მნიშვნელობას მხოლოდ იმ შემთხვევაში, თუ იგი არის დადებითი. ასეთი გზით, ChemiTviseba თვისება აკონტროლებს tviseba ცვლადისათვის მნიშვნელობების მინიჭებას.

თვისებების გამოყენებისას უნდა დავიცვათ შემდეგი შეზღუდვები:
- თვისება არ შეიძლება გადაეცეს მეთოდს, როგორც ref ან out პარამეტრი.
- თვისება არ შეიძლება იყოს გადატვირთული.
- თვისებამ არ უნდა შეცვალოს ცვლადის მნიშვნელობა get მეთოდის გამოყენებით.

ინდექსატორები

ინდექსატორი საშუალებას გვაძლევს ობიექტში მოთავსებულ ცვლადებს, აგრეთვე, მასივის ელემენტებს მივმართოთ ობიექტის სახელისა და ინდექსის საშუალებით. ინდექსატორების გამოყენების ძირითადი სფეროა სპეციალიზებული

მასივების შექმნა, რომლებსაც ედებათ გარკვეული შეზღუდვები. ინდექსატორების სინტაქსი მასივების სინტაქსის მსგავსია. ინდექსატორებს შეიძლება ჰქონდეთ ერთი ან მეტი განზომილება.

ერთგანზომილებიანი ინდექსატორები

ერთგანზომილებიანი ინდექსატორის სინტაქსია:

მიმართვის_მოდიფიკატორი ელემენტის_ტიპი this*[ინდექსის_ტიპი ინდექსი]*
{
მიმართვის get *მეთოდი*
get
{
get *მეთოდის კოდი*
}
მიმართვის set *მეთოდი*
set
{
set *მეთოდის კოდი*
}
}

აქ *ელემენტის_ტიპი* არის ინდექსატორის მიერ გაცემული ცვლადის ტიპი.

ინდექსატორის შემადგენლობაში განსაზღვრულია მიმართვის ორი მეთოდი get და set. ჩვეულებრივი მეთოდისაგან განსხვავებით მიმართვის მეთოდისათვის არ ეთითება დასაბრუნებელი მნიშვნელობის ტიპი და პარამეტრები. მიმართვის ორივე მეთოდი ინდექსატორის გამოყენებისას ავტომატურად გამოიძახება და პარამეტრებად იღებენ მნიშვნელობებს, რომლებიც მითითებულია ინდექსის ნაცვლად. თუ ინდექსატორი მითითებულია მინიჭების ოპერატორის მარცხენა ნაწილში, გამოიძახება მიმართვის set მეთოდი. თუ ინდექსატორი იმყოფება მინიჭების ოპერატორის მარჯვენა ნაწილში, მაშინ გამოიძახება მიმართვის get მეთოდი. საჭიროების მიხედვით ინდექსატორში შეიძლება განსაზღვრული იყოს მხოლოდ get ან მხოლოდ set მეთოდი, ან ორივე ერთად.

ქვემოთ მოყვანილ პროგრამაში ინდექსატორი გამოიყენება ობიექტის ორ ცვლადთან სამუშაოდ.

// პროგრამა 7.5
// პროგრამაში ინდექსატორი გამოიყენება ობიექტის ორ ცვლადთან სამუშაოდ

```
class Manqana
{
private string marka;
private string modeli;
//      კონსტრუქტორის გამოცხადება
public Manqana(string marka, string modeli)
{
this.marka = marka;
this.modeli = modeli;
}
//      ინდექსატორის გამოცხადება
```

```
public string this[int index]
{
get
{
switch ( index )
{
case 0 : return marka;
case 1 : return modeli;
default : return "არასწორი ინდექსი";
}
}
set
{
switch ( index )
{
case 0 : this.marka  = value; break;
case 1 : this.modeli = value; break;
}
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Manqana ChemiManqana = new Manqana(”Toiota”, ”MR2”);

label1.Text = ChemiManqana[0].ToString();
label2.Text = ChemiManqana[1].ToString();
ChemiManqana[0] = “BMW”;
ChemiManqana[1] = “MR3”;
label1.Text = ChemiManqana[0].ToString();
label2.Text = ChemiManqana[1].ToString();
}
```

ინდექსატორის გამოყენების ერთ-ერთი უპირატესობა არის ის, რომ იგი იძლევა მასივთან მიმართვის ზუსტი კონტროლის შესაძლებლობას. კერძოდ, შეგვიძლია ვაკონტროლოთ გადის თუ არა მითითებული ინდექსი მასივის საზღვრებს გარეთ. ამის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

```
//      პროგრამა 7.6
//      პროგრამაში ხდება ინდექსატორის გამოყენება მასივთან სამუშაოდ
class ChemiKlasi
{
int[ ] masivi;          //      masivi არის მასივზე მითითებელი
public int Sigrdze;
public bool alami;      //      alami ცვლადი შეიცავს მნიშვნელობას, რომელიც მიუთითებს,
                        //      გადის თუ არა ინდექსი დიაპაზონის გარეთ
public ChemiKlasi(int zoma)
```

```
{
masivi = new int[zoma];
Sigrdze = zoma;
}
//      ინდექსატორის გამოცხადება
public int this[int index]
{
//      მიმართვის get მეთოდი
get
{
        if ( Shemowmeba(index) )
        {
                alami = false;
                return masivi[index];
        }
        else
        {
                alami = true;
                return 0;
        }
}
//      მიმართვის set მეთოდი
set
{
        if ( Shemowmeba(index) )
        {
                masivi[index] = value;
                alami = false;
        }
        else alami = true;
}
}
//      მეთოდი აბრუნებს true იმნიშვნელობას, თუ ინდექსი იმყოფება მასივის მოცემულ //
        საზღვრებში
public bool Shemowmeba(int index)
{
if ( ( index >= 0 ) & ( index < Sigrdze ) ) return true;
        else return false;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi(5);
int ricxvi;
//      აქ ხდება masivi მასივის რიცხვებით შევსება
```

```
for ( int i = 0; i < ( obieqti.Sigrdze * 2 ); i++ )
        obieqti[i] = i * 10;                                    // set მეთოდის გამოძახება
for ( int i = 0; i < ( obieqti.Sigrdze * 2 ); i++ )
{
        ricxvi = obieqti[i];                                    // get მეთოდის გამოძახება
        if ( ricxvi != -1 ) label2.Text = ricxvi.ToString();
}
//      მოიცემა პირობა, რომელსაც შეცდომასთან მივყავართ
label3.Text = "შეცდომა";
for ( int i = 0; i < ( obieqti.Sigrdze * 2 ); i++ )
{
obieqti[i] = i * 10;
if ( obieqti.alami )
   label1.Text = "მასივის ელემენტებთან მიმართვისას ინდექსი გავიდა საზღვრებს გარეთ";
}
for ( int i = 0; i < ( obieqti.Sigrdze * 2 ); i++ )
{
ricxvi = obieqti[i];
if ( !obieqti.Shemowmeba ) label1.Text += ricxvi.ToString();
  else label2.Text = " მასივის ელემენტებთან მიმართვისას ინდექსი გავიდა საზღვრებს გარეთ";
}
}
}
```

პროგრამაში გამოიყენება ინდექსატორი, რომელიც ამოწმებს ინდექსის მნიშვნელობას და თავიდან გვაცილებს ზღვრული მნიშვნელობების გარეთ გასვლას. ინდექსატორის გამოცხადება იწყება სტრიქონიდან:

public int this[int index]

იგი ოპერაციებს ასრულებს int ტიპის ელემენტებზე.

ინდექსის სისწორეს Shemowmeba მეთოდი ამოწმებს. თუ ინდექსის მნიშვნელობა დასაშვებ საზღვრებშია, მაშინ მეთოდი გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.

მიმართვის get მეთოდი იძახებს Shemowmeba მეთოდს. თუ ინდექსი იმყოფება საზღვრების შიგნით, გაიცემა ამ ინდექსის შესაბამისი ელემენტი. თუ ინდექსი გადის მასივის საზღვრებს გარეთ, მაშინ შესრულდება return 0; ოპერატორი, ე.ი. მასივის ელემენტის ნაცვლად გაიცემა 0. alami ცვლადში ინახება შემოწმების შედეგი. ეს ცვლადი შემდეგ მოწმდება გამომძახებელ პროგრამაში მასივთან ყოველი მიმართვის შემდეგ მიმართვის წარმატებულობის შესამოწმებლად და შეცდომის შესახებ ინფორმაციის გამოსატანად.

მიმართვის set მეთოდიც იძახებს Shemowmeba მეთოდს ინდექსის შესამოწმებლად. თუ index პარამეტრი არ გადის მასივის საზღვრებს გარეთ, მაშინ მნიშვნელობა, რომელიც გადაიცემა value პარამეტრით, ენიჭება შესაბამის ელემენტს. წინააღმდეგ შემთხვევაში მნიშვნელობის მინიჭება არ ხდება.

ინდექსატორების გამოყენებისას არსებობს ერთი მნიშვნელოვანი შეზღუდვა: მათთვის არ შეიძლება ref ან out მოდიფიკატორების გამოყენება.

მრავალგანზომილებიანი ინდექსატორები

ინდექსატორები შეიძლება შეიქმნას მრავალგანზომილებიანი მასივებისთვისაც. ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია ორგანზომილებიან ინდექსატორთან მუშაობის მაგალითი.

```
//      პროგრამა 7.7
//      პროგრამაში ხდება ორგანზომილებიან ინდექსატორთან მუშაობის
დემონსტრირება
class ChemiKlasi
{
int[,] masivi;              //      masivi არის ორგანზომილებიან მასივზე მიმართვა
int striqoni, sveti;        //      ორგანზომილებიან მასივის სვეტები და სტრიქონები
public int Length;
public bool alami;          //      alami ცვლადში ფიქსირდება მასივის ელემენტთან
                            //      მიმართვის მცდელობის შედეგი
//      მითითებული პარამეტრების მქონე მასივის შექმნა
public ChemiKlasi(int par1, int par2)
{
striqoni = par1;
sveti = par2;
masivi = new int[striqoni, sveti];
Length = striqoni * sveti;
}
//      ორგანზომილებიანი ინდექსატორის გამოცხადება
public int this[int index1, int index2]
{
// მიმართვის get მეთოდი
get
{
        if ( Shemowmeba(index1, index2) )
        {
                alami = false;
                return masivi[index1, index2];
        }
        else
        {
                alami = true;
                return 0;
        }
        }
// მიმართვის set მეთოდი
set
{
        if (Shemowmeba(index1, index2) )
        {
                masivi[index1, index2] = value;
```

```
                alami = false;
        }
        else alami = true;
}
}
//      true მნიშვნელობის დაბრუნება, თუ ინდექსები არ გადიან მასივის ინდექსების
//      ზღვრული მნიშვნელობების გარეთ
private bool Shemowmeba(int index1, int index2)
{
if ( ( ( index1 >= 0 ) && ( index1 < striqoni ) ) &&
    ( ( index2 >= 0 ) && ( index2 < sveti ) ) )
        return true;
return false;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//
ChemiKlasi obieqti = new ChemiKlasi(2,4);
int ricxvi;
/*
*
*
*/
//      მასივის საზღვრებს გარეთ გასვლისას გაიცემა 0
for ( int i = 0; i < 5; i++ )
        obieqti[i, i] = i * 10;
for ( int i = 0; i < 5; i++ )
{
        ricxvi = obieqti[i, i];
        if (ricxvi != -1 ) label2.Text += ricxvi.ToString() + "   ";
}
for ( int i = 0; i < 5; i++ )
{
        obieqti[i, i] = i * 10;
        if ( obieqti.alami ) label3.Text = "ინდექსი გადის საზღვრებს გარეთ";
}
for ( int i = 0; i < 5; i++ )
{
        ricxvi = obieqti[i, i];
        if ( !obieqti.alami ) label3.Text += ricxvi.ToString() + "   ";
                else label3.Text = " ინდექსი გადის საზღვრებს გარეთ";
}
}
}
```

თავი 8. მემკვიდრეობითობა. ვირტუალური მეთოდები

მემკვიდრეობითობის საფუძვლები

როგორც ვიცით, ობიექტზე ორიენტირებული პროგრამირების ერთ-ერთი პრინციპია მემკვიდრეობითობა. მისი გამოყენებით შეგვიძლია ახალი კლასების შექმნა, რომლებიც წარმოადგენენ საბაზო კლასის მემკვიდრე კლასებს. მემკვიდრე კლასს დამატებული აქვს საბაზო კლასისაგან განსხვავებული თვისებები. მაგალითად, შეგვიძლია შევქმნათ კლასი ავტომობილი, რომელსაც აქვს ისეთი საერთო მახასიათებლები, როგორიცაა ძრავის სიმძლავრე, საწვავის ხარჯი 100 კილომეტრზე, მაქსიმალური სიჩქარე. ამ კლასს შეიძლება ჰქონდეს ორი მემკვიდრე კლასი - სატვირთო, რომელშიც ჩნდება მახასიათებელი - ტვირთამწეობა, და ავტობუსი, რომელშიც ჩნდება მახასიათებელი - გადასაყვანი მგზავრების რაოდენობა.

C# ენაში კლასს, რომლის ცვლადები და მეთოდები ავტომატურად ხდება სხვა ახალი შესაქმნელი კლასის წევრები, წინაპარი *(საბაზო)* კლასი ეწოდება. კლასს, რომელიც მემკვიდრეობით იღებს წინაპარი კლასის არსებულ წევრებს და მათ ახალ წევრებს უმატებს, მემკვიდრე კლასი ეწოდება. ამრიგად, მემკვიდრე კლასი მემკვიდრეობით იღებს წინაპარ კლასში განსაზღვრულ ყველა ცვლადს, მეთოდს, თვისებასა და ინდექსატორს და მათ უმატებს თავის საკუთარ წევრებს.

მემკვიდრე კლასის გამოცხადების სინტაქსია:

class *მემკვიდრე_კლასის_სახელი* : *წინაპარი_კლასის_სახელი*
{
კლასის კოდი
}

როგორც ვხედავთ, მემკვიდრე კლასის განსაზღვრისას ორი წერტილის შემდეგ ეთითება წინაპარი კლასის სახელი. თითოეული მემკვიდრე კლასისათვის შეგვიძლია მივუთითოთ მხოლოდ ერთი წინაპარი კლასი. ასეთი გზით შესაძლებელია მემკვიდრეობითობის იერარქიის შექმნა, რომელშიც მემკვიდრე კლასს თვითონ შეუძლია გახდეს წინაპარი სხვა კლასისთვის. არც ერთი კლასი არ შეიძლება იყოს თავისი თავის წინაპარი.

მემკვიდრეობითობის უპირატესობა იმაშია, რომ კლასის შექმნის შემდეგ, რომელიც განსაზღვრავს საერთო მახასიათებლებს, იგი შეგვიძლია გამოვიყენოთ ნებისმიერი რაოდენობის მემკვიდრე კლასების შესაქმნელად, რომლებიც დამატებით შეიცავენ უნიკალურ მახასიათებლებს. შედეგად, მემკვიდრე კლასში აღარ ხდება საჭირო წინაპარ კლასში გამოცხადებული წევრების განმეორებით გამოცხადება.

განვიხილოთ პროგრამა, რომელშიც ხდება მემკვიდრეობითობის დემონსტრირება. მასში იქმნება Sibrtye კლასი, რომელიც ინახავს ორგანზომილებიანი ობიექტის სიგანისა და სიმაღლის მნიშვნელობებს და რომლის მემკვიდრე კლასია Samkutxedi.

```
//      პროგრამა 8.1
//      პროგრამაში ხდება მემკვიდრეობითობის დემონსტრირება
//      საბაზო კლასი შეიცავს ორგანზომილებიანი გეომეტრიული ფიგურის სიგანესა და სიმაღლეს
```

```
class Sibrtye
{
public double gverdi1;
public double gverdi2;
public double gverdi3;
public string Naxva()
{
return "სიგანე და სიმაღლე \n" + gverdi1.ToString()+ "    " + gverdi2.ToString();
}
}
//        Samkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
class Samkutxedi : Sibrtye    //        Samkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
{
public string tipi;
public double Partobi()
{
return gverdi1 * gverdi2 / 2;          //        Samkutxedi კლასის წევრი შეიძლება მიმართავდეს
                                       //        Sibrtye კლასის წევრებს ისე, თითქოს ისინი
                                       //        გამოცხადებული იყოს Samkutxedi კლასში
}
public double Perimetri()
{
return gverdi1 + gverdi2 + gverdi3;
}
public string TipisNaxva()
{
return "samkuTxedis saxe - " + tipi.ToString();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Samkutxedi figura = new Samkutxedi();
figura.gverdi1 = 5.5;
figura.gverdi2 = 5.5;
figura.tipi = "ტოლგვერდა";
//        ინფორმაციის გამოტანა figura ობიექტის შესახებ
label1.Text = figura.TipisNaxva();
label2.Text = figura.Naxva();
label3.Text = figura.Partobi().ToString();
}
```

Sibrtye კლასში განსაზღვრულია ორგანზომილებიანი გეომეტრიული ფიგურის მახასიათებლები, რომელიც შეიძლება იყოს კვადრატი, მართკუთხედი და ა.შ. Samkutxedi კლასში, რომელიც არის Sibrtye კლასის მემკვიდრე, განისაზღვრება გეომეტრიული ფიგურის კონკრეტული ტიპი, ჩვენს შემთხვევაში - სამკუთხედი. Samkutxedi კლასში ჩართულია Sibrtye კლასის ყველა წევრი და დამატებულია tipi ველი,

აგრეთვე, Partobi() და TipisNaxva() მეთოდები. სამკუთხედის სახის დასახელება ინახება tipi ველში, ხოლო TipisNaxva() მეთოდი კი აბრუნებს სამკუთხედის სახის დასახელებას.

რადგან Samkutxedi კლასში ჩართულია Sibrtye წინაპარი კლასის ყველა წევრი, ამიტომ მის Partobi() მეთოდს შეუძლია მიმართოს gverdi1 და gverdi2 ცვლადებს. გარდა ამისა, წერტილი (.) ოპერატორის გამოყენებით ვუთითებთ რა figura ობიექტისა და gverdi1 ან gverdi2 ცვლადის სახელს, შეიძლება უშუალოდ მეთოდის შიგნით მივმართოთ ამ ცვლადების ასლებს.

მიუხედავად იმისა, რომ Sibrtye კლასი არის Samkutxedi კლასის წინაპარი, იგი არის მთლიანად ავტონომიური კლასი, რომელიც შეგვიძლია დამოუკიდებლად გამოვიყენოთ. მაგალითად, ნამდვილია შემდეგი კოდი:

```
{
Sibrtye figura = new Sibrtye();
Figura.gverdi1 = 15;
Figura.gverdi2 = 25;
Figura.Naxva();
}
```

Sibrtye კლასის ობიექტს არ შეუძლია მიმართოს მემკვიდრე კლასის წევრებს.

protected მოდიფიკატორი

მემკვიდრეობითობის დროს ძალაშია კლასის დახურულ წევრებთან მიმართვისას არსებული შეზღუდვები. შედეგად, მემკვიდრე კლასის მეთოდს არ შეუძლია მიმართოს წინაპარი კლასის დახურულ წევრებს. ამ პრობლემის გადაწყვეტის ერთი გზაა protected მოდიფიკატორის გამოყენება. მეორე გზაა დახურულ მონაცემებთან მიმართვისათვის თვისებების გამოყენება.

C# ენაში არსებობს კლასის დაცული წევრების შექმნის შესაძლებლობა. ისინი გახსნილია მხოლოდ მემკვიდრე კლასებისათვის და დახურულია სხვა კლასებისათვის. კლასის დაცული წევრი იქმნება protected მოდიფიკატორის საშუალებით.

ქვემოთ მოყვანილია პროგრამა, რომელშიც გამოყენებულია protected მოდიფიკატორი.

```
//      პროგრამა 8.2
//      პროგრამაში ხდება protected მოდიფიკატორის გამოყენების დემონსტრირება
class Sibrtye
{
//      ეს ცვლადები ღიაა Samkutxedi კლასის წევრებისთვის
protected int gverdi1, gverdi2, gverdi3;
public void Inicializacia(int a, int b, int c)
{
        gverdi1 = a;
        gverdi2 = b;
        gverdi3 = c;
}
public string Naxva()
{
return gverdi1.ToString() + "   " + gverdi2.ToString() +  "   " +
```

```
        gverdi3.ToString();
}
}
class Samkutxedi : Sibrtye
{
int shedegi;
//      Partobi() მეთოდს შეუძლია მიმართოს Sibrtye კლასში
//      განსაზღვრულ gverdi1, gverdi2 და gverdi3 ცვლადებს
public void Partobi()
{
shedegi = gverdi1 * gverdi2 / 2;
}
public string PartobisNaxva()
{
return shedegi.ToString();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება protected მოდიფიკატორის გამოყენება
Samkutxedi obieqti = new Samkutxedi();
int ricxvi1 = Convert.ToInt32(textBox1.Text);
int ricxvi2 = Convert.ToInt32(textBox2.Text);
int ricxvi3 = Convert.ToInt32(textBox3.Text);

// Inicializacia() და Naxva() მეთოდები მისაწვდომია Samkutxedi კლასის ობიექტიდან
obieqti.Inicializacia(ricxvi1, ricxvi2, ricxvi3);
label1.Text = obieqti.Naxva();
obieqti.Partobi();
label2.Text = obieqti.PartobisNaxva();
}
```

კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა

წინაპარ და მემკვიდრე კლასებს შეიძლება ჰქონდეთ საკუთარი კონსტრუქტორები. ამ დროს წამოიჭრება კითხვა, თუ რომელი კონსტრუქტორი ქმნის მემკვიდრე კლასის ობიექტს. თითოეული კონსტრუქტორი ქმნის (ინიციალიზაციას უკეთებს) ობიექტის თავის ნაწილს, ე.ი. ობიექტი ნაწილ-ნაწილ იქმნება. ეს იმიტომ ხდება, რომ წინაპარი კლასის კონსტრუქტორს არ შეუძლია მიმართოს მემკვიდრე კლასის წევრებს და მოახდინოს მათი ინიციალიზაცია. თუ კონსტრუქტორი არ არის განსაზღვრული წინაპარ კლასში და განსაზღვრულია მემკვიდრე კლასში, მაშინ მემკვიდრე კლასის ობიექტის კონსტრუირება სრულდება მისი კონსტრუქტორის მიერ, ხოლო წინაპარ კლასში განსაზღვრული ობიექტის ნაწილი კი იქმნება მისი ავტომატური კონსტრუქტორის მიერ.

თუ კონსტრუქტორები განსაზღვრულია როგორც წინაპარ, ისე მემკვიდრე კლასებში, მაშინ ობიექტის შექმნის პროცესი რამდენადმე რთულდება, რადგან ამ დროს გამოიძახება ორივე კონსტრუქტორი. ასეთ სიტუაციაში უნდა გამოვიყენოთ base საკვანძო სიტყვა, რომელიც გამოიყენება წინაპარი კლასის კონსტრუქტორის გამოსაძახებლად. იგი, აგრეთვე, გამოიყენება წინაპარი კლასის დამალულ წევრებთან მიმართვისათვის.

base საკვანძო სიტყვა ეთითება მემკვიდრე კლასის კონსტრუქტორის გამოცხადების გაფართოებულ ფორმაში. ამ ფორმის სინტაქსია:

მემკვიდრე_კლასის_კონსტრუქტორის_სახელი(პარამეტრების_სია) :
base*(არგუმენტების_სია)*
{
კონსტრუქტორის კოდი
}

არგუმენტების_სია არის წინაპარი კლასის კონსტრუქტორისთვის გადასაცემი მნიშვნელობების სია.

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია base საკვანძო სიტყვის გამოყენება.

```
//      პროგრამა 8.3
//      პროგრამაში ხდება კონსტრუქტორების მემკვიდრეობითობის დემონსტრირება
class Sibrtye
{
double pri_sigane;                  //      დახურული ცვლადების გამოცხადება
double pri_simagle;
//      Sibrtye კლასის კონსტრუქტორი
public Sibrtye(double w, double h)
{
        sigane = w;
        simagle = h;
}
//      sigane და simagle თვისებები
public double sigane
{
get
{
        return pri_sigane;
}
set
{
        pri_sigane = value;
}
}
public double simagle
{
get
{
        return pri_simagle;
```

```
}
set
{
        pri_simagle = value;
}
}
public string Naxva()
{
        return "გეომეტრიული ფიგურის სიგანე და სიმაღლე\n" +
                sigane.ToString() + "   " + simagle.ToString();
}
}
//      მემკვიდრე კლასის გამოცხადება
class Samkutxedi : Sibrtye
{
string tipi;                    //      დახურული ცვლადი
//      წინაპარი კლასის კონსტრუქტორის გამოძახება
public Samkutxedi(string s, double w, double h) : base(w, h)
{
        tipi = s;
}
public double Partobi()
{
        return sigane * simagle / 2;
}
public string TipisNaxva()
{
        return "samkuTxedis tipia - " + tipi;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Samkutxedi sam1 = new Samkutxedi("tolgverda", 5.5, 5.5);

//      ინფორმაცია sam1 ობიექტის შესახებ
label1.Text = sam1.TipisNaxva();
label2.Text = sam1.Naxva();
label3.Text = sam1.Partobi().ToString();
}
```

ამ პროგრამაში Samkutxedi კლასის კონსტრუქტორი base სიტყვის გამოყენებით იძახებს Sibrtye კლასის კონსტრუქტორს და გადასცემს მას w და h პარამეტრებს. შედეგად, sigane და simagle ცვლადებს ენიჭებათ ეს პარამეტრები. ამიტომ, Samkutxedi კლასის კონსტრუქტორი ახდენს მხოლოდ tipi ცვლადის ინიციალიზაციას, რომელიც განსაზღვრულია მემკვიდრე კლასში.

წინაპარ კლასს შეიძლება ჰქონდეს რამდენიმე კონსტრუქტორი, რომელთაგან

თითოეული შეგვიძლია გამოვიძახოთ base საკვანძო სიტყვის გამოყენებით. გამოიძახება ის კონსტრუქტორი, რომლის პარამეტრების სიაც შეესაბამება არგუმენტების სიას.

მემკვიდრე კლასის კონსტრუქტორში base საკვანძო სიტყვის მითითებისას გამოიძახება უახლოესი წინაპარი კლასის კონსტრუქტორი. ე.ი. base საკვანძო სიტყვა ყოველთვის მიმართავს უშუალო წინაპარ კლასს. ეს ძალაშია კლასების მრავალდონიანი იერარქიისთვისაც. თუ მემკვიდრე კლასის კონსტრუქტორში არ არის მითითებული base საკვანძო სიტყვა, მაშინ გამოიძახება წინაპარი კლასის ავტომატური კონსტრუქტორი.

ცვლადების დამალვა მემკვიდრეობითობის დროს

წინაპარი და მემკვიდრე კლასების წევრებს შეიძლება ერთი და იგივე სახელი ჰქონდეთ. ამ შემთხვევაში წინაპარი კლასის წევრი დამალულია მემკვიდრე კლასის სხვა წევრებისათვის და კომპილატორს ეკრანზე გამოაქვს გამაფრთხილებელი შეტყობინება ამის შესახებ. თუ ჩვენ განზრახ ვმალავთ წინაპარი კლასის წევრს და არ გვინდა, რომ კომპილატორმა გამოიტანოს გამაფრთხილებელი შეტყობინება, მაშინ მემკვიდრე კლასის შესაბამისი წევრის წინ უნდა მივუთითოთ new საკვანძო სიტყვა. ამ შემთხვევაში new ოპერატორი არ ახდენს ობიექტის შექმნას.

ქვემოთ მოყვანილ პროგრამაში ხდება მემკვიდრეობითობის დროს ცვლადის დამალვის დემონსტრირება.

```
//      პროგრამა 8.4
//      პროგრამაში ხდება მემკვიდრეობითობის დროს ცვლადის დამალვის დემონსტრირება
class SabazoKlasi
{
        public int i = 0;
}
//      მემკვიდრე კლასი
class MemkvidreKlasi : SabazoKlasi
{
new int i;
//      ეს i ცვლადი მალავს SabazoKlasi კლასში გამოცხადებულ i ცვლადს
public MemkvidreKlasi(int parametri)
{
i = parametri;          //      i ცვლადის ინიციალიზაცია MemkvidreKlasi კლასში
}
public string Naxva()
{
return "i ცვლადის მნიშვნელობა მემკვიდრე კლასში - " + i.ToString();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება ცვლადის დამალვის დემონსტრირება მემკვიდრეობითობის დროს
int ricxvi = Convert.ToInt32(textBox1.Text);
```

```
MemkvidreKlasi obieqti = new MemkvidreKlasi(ricxvi);

label1.Text = obieqti.Naxva();
}
```

MemkvidreKlasi კლასში i ცვლადის განსაზღვრის შემდეგ დაიმალება SabazoKlasi კლასში განსაზღვრული i ცვლადი. ე.ი. Naxva() მეთოდის გამოძახებისას გამოჩნდება MemkvidreKlasi კლასში განსაზღვრული i ცვლადის მნიშვნელობა.

იმისათვის, რომ შევძლოთ წინაპარი კლასის დამალულ წევრებთან მიმართვა უნდა გამოვიყენოთ base საკვანძო სიტყვა. მისი სინტაქსია:

base.წევრის_სახელი

აქ წევრის_სახელი არის ობიექტის მეთოდის ან ცვლადის სახელი. ასეთი გამოყენების დროს base საკვანძო სიტყვა ისეთივე ფუნქციებს ასრულებს, როგორსაც this სიტყვა, იმ განსხვავებით, რომ base სიტყვა ყოველთვის მიმართავს წინაპარ კლასს.

მოყვანილ პროგრამაში ხდება base საკვანძო სიტყვის გამოყენება წინაპარი კლასის დამალულ ცვლადთან მიმართვისათვის.

```
//      პროგრამა 8.5
//      პროგრამაში ხდება base სიტყვის გამოყენება წინაპარი კლასის
//      დამალულ ცვლადთან მიმართვისათვის
class SabazoKlasi
{
public int i = 0;
}
//      მემკვიდრე კლასის შექმნა
class MemkvidreKlasi : SabazoKlasi
{
new int i;              //      ეს i ცვლადი მალავს SabazoKlasi კლასში გამოცხადებულ i ცვლადს
public MemkvidreKlasi(int a, int b)
{
base.i = a;             //  მიმართვა SabazoKlasi კლასში გამოცხადებულ i ცვლადთან
                        //  base საკვანძო სიტყვის საშუალებით
i = b;                  //  MemkvidreKlasi კლასში გამოცხადებული i ცვლადის ინიციალიზება
}
public void Naxva()
{
//      SabazoKlasi კლასში განსაზღვრული i ცვლადის ეკრანზე გამოტანა
MessageBox.Show("i ცვლადის მნიშვნელობა წინაპარ კლასში - " + base.i.ToString());
//      MemkvidreKlasi კლასში განსაზღვრული i ცვლადის ეკრანზე გამოტანა
MessageBox.Show("i ცვლადის მნიშვნელობა მემკვიდრე კლასში - " + i.ToString());
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int ricxvi1 = Convert.ToInt32(textBox1.Text);
int ricxvi2 = Convert.ToInt32(textBox2.Text);
MemkvidreKlasi obieqti = new MemkvidreKlasi(ricxvi1, ricxvi2);
```

```
obieqti.Naxva();
}
```

obieqti ობიექტის i ცვლადი მალავს SabazoKlasi კლასში განსაზღვრულ i ცვლადს. base საკვანძო სიტყვის გამოყენება უზრუნველყოფს მიმართვას წინაპარ კლასში განსაზღვრულ i ცვლადთან. base სიტყვის გამოყენებით შეგვიძლია, აგრეთვე, გამოვიძახოთ დამალული მეთოდებიც. მოყვანილ პროგრამაში ხდება base სიტყვის გამოყენების დემონსტრირება დამალულ მეთოდებთან მიმართვისათვის.

```
//      პროგრამა 8.6
//      პროგრამაში ხდება base სიტყვის გამოყენება წინაპარი კლასის დამალულ
//      მეთოდთან მიმართვისათვის
class SabazoKlasi
{
public int i = 0;
public void Naxva()
{
MessageBox.Show("i ცვლადის მნიშვნელობა წინაპარ კლასში - " + i.ToString());
}
}
//      მემკვიდრე კლასის შექმნა
class MemkvidreKlasi : SabazoKlasi
{
new int i;              //      ეს i ცვლადი მალავს SabazoKlasi კლასში გამოცხადებულ i ცვლადს
public MemkvidreKlasi(int a, int b)
{
base.i = a;             //      მიმართვა SabazoKlasi კლასში განსაზღვრულ i ცვლადთან base
                        //      საკვანძო სიტყვის გამოყენებით
i = b;                  //      i ცვლადის ინიციალიზაცია MemkvidreKlasi კლასში
}
//      ეს Naxva() მეთოდი მალავს SabazoKlasi კლასში გამოცხადებულ Naxva() მეთოდს
new public void Naxva()
{
base.Naxva ();//      SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდის გამოძახება
MessageBox.Show("i cvladis mniSvneloba memkvidre klasSi - " + i.ToString());
}
}
private void button1_Click(object sender, System.EventArgs e)
{
MemkvidreKlasi obieqti = new MemkvidreKlasi(5, 10);

obieqti.Naxva();
}
```

კლასების მრავალდონიანი იერარქიის შექმნა

აქამდე ჩვენ ვიყენებდით კლასების ორდონიან იერარქიას, რომელიც შედგებოდა წინაპარი და მემკვიდრე კლასებისაგან. C# ენაში შეგვიძლია, შევქმნათ კლასების მრავალდონიანი იერარქია. ამ შემთხვევაში მემკვიდრე კლასი შეგვიძლია გამოვიყენოთ როგორც საბაზო შემდგომი მემკვიდრეობითობისათვის. მაგალითად, Klasi3 კლასი შეიძლება იყოს Klasi2 კლასის მემკვიდრე, რომელიც თავის მხრივ შეიძლება იყოს Klasi1 კლასის მემკვიდრე. ამ შემთხვევაში Klasi3 კლასი მემკვიდრეობით იღებს Klasi1 და Klasi2 კლასების ყველა წევრს.

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია კლასების მრავალდონიანი იერარქიის შექმნა. აქ Sibrtye არის საბაზო კლასი. მისი მემკვიდრეა Samkutxedi კლასი, რომელიც არის საბაზო FeradiSamkutxedi კლასისათვის. FeradiSamkutxedi კლასი მემკვიდრეობით იღებს Sibrtye და Samkutxedi კლასების ყველა წევრს. FeradiSamkutxedi კლასში გამოცხადებულია ახალი peri ცვლადი, რომელიც შეიცავს ინფორმაციას სამკუთხედის ფერის შესახებ. რადგან FeradiSamkutxedi კლასის შექმნისას გამოიყენება მრავალდონიანი მემკვიდრეობითობა, ამ კლასის ობიექტებს შეუძლიათ მიმართონ Sibrtye და Samkutxedi კლასების წევრებს.

```
//      პროგრამა 8.7
//      პროგრამაში ხდება კლასების მრავალდონიანი იერარქიის დემონსტრირება
class Sibrtye
{
double pri_sigane;              //      დახურული ცვლადების გამოცხადება
double pri_simagle;
//      ავტომატური კონსტრუქტორი
public Sibrtye()
{
sigane = simagle = 0.0;
}
//      კონსტრუქტორი ორი პარამეტრით
public Sibrtye(double w, double h)
{
sigane = w;
simagle = h;
}
//      კონსტრუქტორი, განკუთვნილი ობიექტების შესაქმნელად, რომელთა sigane და simagle
//      თვისებებს ენიჭებათ ერთი და იგივე მნიშვნელობები
public Sibrtye(double x)
{
sigane = simagle = x;
}
//      sigane და simagle თვისებების განსაზღვრა
public double sigane
{
get { return pri_sigane; }
set { pri_sigane = value; }
```

```
}
public double simagle
{
get { return pri_simagle; }
set { pri_simagle = value; }
}
public string Naxva()
{
return "გეომეტრიული ფიგურის სიგანე და სიმაღლე \n" + sigane.ToString() +
        "   " + simagle.ToString();
}
}
//      Samkutxedi არის Sibrtye კლასის მემკვიდრე
class Samkutxedi : Sibrtye
{
string tipi;
//      ავტომატური კონსტრუქტორი. იგი იძახებს Sibrtye კლასის ავტომატურ კონსტრუქტორს
public Samkutxedi()
{
tipi = "nuli";
}
//      კონსტრუქტორი სამი პარამეტრით
public Samkutxedi(string s, double w, double h) : base(w, h)
{
tipi = s;
}
//      კონსტრუქტორი ერთი პარამეტრით
public Samkutxedi(double x) : base(x)
{
tipi = "ტოლფერდა";
}
public double Partobi()
{
return sigane * simagle / 2;
}
public string TipisNaxva()
{
return "სამკუთხედის სახე - " + style;
}
}
//      FeradiSamkutxedi კლასი არის Samkutxedi კლასის მემკვიდრე
class FeradiSamkutxedi : Samkutxedi
{
string peri;
public FeradiSamkutxedi(string c, string s, double w, double h) : base(s, w, h)
```

```
{
peri = c;
}
//        ინფორმაციის გამოტანა სამკუთხედის ფერის შესახებ
public string PerisNaxva()
{
return "სამკუთხედის ფერი - " + peri;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
FeradiSamkutxedi sam1 = new FeradiSamkutxedi("მწვანე, ", "სწორკუთხა ", 5.0, 15.0);
label1.Text = sam1.TipiaNaxva();
label2.Text = sam1.Naxva();
label3.Text = sam1.PerisNaxva();
label4.Text = sam1.Partobi().ToString();
}
```

მოცემული პროგრამიდან ჩანს, რომ base საკვანძო სიტყვა ყოველთვის მიმართავს უახლოესი წინაპარი კლასის კონსტრუქტორს. თუ base საკვანძო სიტყვა ეთითება Samkutxedi კლასის კონსტრუქტორთან, მაშინ გამოიძახება Sibrtye კლასის კონსტრუქტორი. თუ base სიტყვა ეთითება FeradiSamkutxedi კლასის კონსტრუქტორთან, მაშინ გამოიძახება Samkutxedi კლასის კონსტრუქტორი. თუ წინაპარი კლასის კონსტრუქტორს სჭირდება რაიმე პარამეტრები, მაშინ კლასების მრავალდონიანი იერარქიის შექმნის წესების შესაბამისად ყველა მემკვიდრე კლასი ამ პარამეტრებს უნდა გადასცემდეს “ზევით” შესაბამის დონეზე, მიუხედავად იმისა ეს პარამეტრები სჭირდება თუ არა წინაპარ კლასს.

კონსტრუქტორების გამოძახების რიგითობა

კლასების იერარქიაში კონსტრუქტორები გამოიძახება იმ რიგითობით, რა რიგითობითაც სრულდება მემკვიდრეობითობა - წინაპარი კლასიდან მემკვიდრე კლასისკენ. უფრო მეტიც, ეს რიგითობა რჩება უცვლელი იმისგან დამოუკიდებლად, გამოიყენება თუ არა base საკვანძო სიტყვა. თუ ეს საკვანძო სიტყვა არ არის მითითებული, მაშინ გამოიძახება თითოეული წინაპარი კლასის ავტომატური კონსტრუქტორი. კონსტრუქტორის გამოძახების რიგითობა ნაჩვენებია მოყვანილ პროგრამაში.

```
//        პროგრამა 8.8
//        პროგრამაში ხდება კონსტრუქტორების გამოძახების რიგითობის დემონსტრირება
class Klasi1
{
public Klasi1()
{
MessageBox.Show("Klasi1 კლასში განსაზღვრული მეთოდი");
}
}
//        Klasi1 კლასის მემკვიდრე კლასის შექმნა
```

```
class Klasi2 : Klasi1
{
public Klasi2()
{
MessageBox.Show("Klasi2 კლასში განსაზღვრული მეთოდი");
}
}
//
class Klasi3 : Klasi2
{
public Klasi3()
{
MessageBox.Show("Klasi3 კლასში განსაზღვრული მეთოდი");
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Klasi3 obieqti = new Klasi3();
}
```

მიმართვები წინაპარი და მემკვიდრე კლასების ობიექტებზე

ერთი კლასის მიმართვით ცვლადს არ შეიძლება მიენიჭოს სხვა კლასის ობიექტზე მიმართვა. მაგრამ წინაპარი კლასის მიმართვით ცვლადს შეიძლება მიენიჭოს ნებისმიერი მემკვიდრე კლასის ობიექტზე მიმართვა. ქვემოთ მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```
//      პროგრამა 8.9
//      პროგრამაში ხდება წინაპარი და მემკვიდრე კლასების ობიექტებზე მიმართვების დემონსტრირება
class SabazoKlasi
{
public int ricxvi1;
public SabazoKlasi(int i)
{
ricxvi1 = i;
}
}
class MemkvidreKlasi : SabazoKlasi
{
public int ricxvi2;
public MemkvidreKlasi(int i, int j) : base(j)
{
ricxvi2 = i;
}
```

```
}
private void button1_Click(object sender, System.EventArgs e)
{
int ricxvi = Convert.ToInt32(textBox1.Text);
SabazoKlasi Sabazo_Obieqti1 = new SabazoKlasi(ricxvi);
SabazoKlasi Sabazo_Obieqti2;
MemkvidreKlasi Memkvidre_Obieqti = new MemkvidreKlasi(2, 5);
// მინიჭების ეს ოპერატორი ნამდვილია, რადგან მასში მონაწილეობს ერთი ტიპის მიმართვითი ცვლადები
Sabazo_Obieqti2 = Sabazo_Obieqti1;
label1.Text = Sabazo_Obieqti2.ricxvi1.ToString();
//      ასევე ნამდვილია მინიჭების ეს ოპერატორი, რადგან MemkvidreKlasi არის
//      SabazoKlasi-ის მემკვიდრე
Sabazo_Obieqti2 = Memkvidre_Obieqti;
Sabazo_Obieqti2.ricxvi1 = 35;                   //      ეს მიმართვა ნამდვილია
label2.Text = Sabazo_Obieqti2.ricxvi1.ToString();
//      Sabazo_Obieqti2.ricxvi2 = 50;           //      შეცდომა! ricxvi2 ცვლადი არ არის
                                                //      გამოცხადებული SabazoKlasi კლასში
}
```

ამ პროგრამაში MemkvidreKlasi კლასი არის SabazoKlasi კლასის მემკვიდრე. ამიტომ, Sabazo_Obieqti2 ცვლადს შეგვიძლია მივანიჭოთ Memkvidre_Obieqti ობიექტზე მიმართვა.

ყურადღება მივაქციოთ იმას, რომ კლასის წევრებთან მიმართვის შესაძლებლობა დამოკიდებულია მიმართვითი ცვლადის ტიპზე, და არა იმ ობიექტის ტიპზე, რომელსაც ის მიმართავს. თუ მიმართვით ცვლადს, რომელსაც აქვს წინაპარი კლასის ტიპი, ენიჭება მემკვიდრე კლასის ობიექტზე მიმართვა, მაშინ ამ ცვლადის საშუალებით შეგვეძლება მხოლოდ იმ წევრებთან მიმართვა, რომლებიც განსაზღვრული იყო წინაპარ კლასში. სწორედ ამიტომ მიმართვითი Sabazo_Obieqti2 ცვლადი ვერ მიმართავს ricxvi2 ცვლადს მას შემდეგაც კი, რაც Sabazo_Obieqti2 ცვლადს მიენიჭა MemkvidreKlasi კლასის ობიექტზე მიმართვა. ასეთ შეზღუდვას აზრი აქვს, რადგან წინაპარმა კლასმა არაფერი არ "იცის" კლასის იმ წევრების შესახებ, რომლებიც დაემატა მემკვიდრე კლასს.

ვირტუალური მეთოდები

ენაში ვირტუალური მეთოდები უზრუნველყოფენ პოლიმორფიზმს პროგრამის შესრულების დროს. პოლიმორფიზმი საშუალებას იძლევა წინაპარ კლასში განვსაზღვროთ მეთოდები, რომლებიც საერთო იქნება ყველა მემკვიდრე კლასისათვის. ამასთან, ამ მეთოდების კოდი მემკვიდრე კლასებში შეიძლება შეიცვალოს ან იგივე დარჩეს. უცვლელი რჩება მეთოდის სახელი და მიმართვითი ცვლადი. მეთოდების ხელახალი განსაზღვრა წარმოადგენს პოლიმორფიზმის "ერთი ინტერფეისი - რამდენიმე მეთოდი" პრინციპის რეალიზების ერთ-ერთ საშუალებას.

მეთოდს, რომლის განსაზღვრისას წინაპარ კლასში მითითებული იყო virtual საკვანძო სიტყვა, და რომელიც ხელახლა იყო განსაზღვრული ერთ ან მეტ მემკვიდრე კლასში, ვირტუალური მეთოდი ეწოდება. თითოეულ მემკვიდრე კლასს შეიძლება

ჰქონდეს ვირტუალური მეთოდის საკუთარი ვერსია. ენაში ვირტუალური მეთოდის ვერსიის არჩევა ხორციელდება იმ ობიექტის ტიპის შესაბამისად, რომელსაც მიმართვითი ცვლადი მიმართავს. ეს არჩევა ხორციელდება პროგრამის შესრულების დროს. მიმართვითი ცვლადი შეიძლება მიმართავდეს სხვადასხვა ტიპის ობიექტებს, ამიტომ, შეიძლება გამოძახებული იყოს ვირტუალური მეთოდების სხვადასხვა ვერსიები. სხვა სიტყვებით რომ ვთქვათ, სწორედ ობიექტის ტიპი, რომელზეც მიუთითებს მიმართვა (და არა მიმართვითი ცვლადის ტიპი), განსაზღვრავს გამოსაძახებელი ვირტუალური მეთოდის ვერსიას. ამრიგად, თუ კლასი შეიცავს ვირტუალურ მეთოდს და ამ კლასიდან მემკვიდრეობით მიღებული იყო სხვა კლასები, რომლებშიც განსაზღვრულია ვირტუალური მეთოდის საკუთარი ვერსიები, მაშინ სხვადასხვა ტიპის ობიექტებზე წინაპარი კლასის ტიპის მქონე ცვლადის მიმართვის დროს შესრულდება ვირტუალური მეთოდის შესაბამისი ვერსია.

წინაპარ კლასში ვირტუალური მეთოდის განსაზღვრისას დასაბრუნებელი მნიშვნელობის ტიპის წინ ეთითება virtual საკვანძო სიტყვა, ხოლო მემკვიდრე კლასში ვირტუალური მეთოდის ხელახალი განსაზღვრისას გამოიყენება override მოდიფიკატორი. მემკვიდრე კლასში ვირტუალური მეთოდის განსაზღვრის პროცესს, როცა ნაწილობრივ ან მთლიანად იცვლება მეთოდის ტანი, ხოლო მეთოდის სახელი, პარამეტრები და მათი ტიპები იგივე რჩება, მეთოდის ხელახალი განსაზღვრა ეწოდება. ვირტუალური მეთოდი არ შეიძლება განსაზღვრული იყოს static ან abstract მოდიფიკატორის გამოყენებით (ისინი მომდევნო თავში განიხილება).

მეთოდის ხელახალი განსაზღვრა საფუძვლად უდევს გამოსაძახებელი მეთოდის დინამიური არჩევის კონცეფციას. ეს არის მექანიზმი, რომლის საშუალებითაც გამოსაძახებელი ხელახლა განსაზღვრული მეთოდის არჩევა ხორციელდება პროგრამის შესრულების და არა პროგრამის კომპილირების დროს.

ქვემოთ მოყვანილია პროგრამა, რომელშიც ნაჩვენებია ვირტუალური მეთოდისა და მისი ხელახლა განსაზღვრული ვერსიების გამოყენება.

```
//      პროგრამა 8.10
//      პროგრამაში ხდება ვირტუალურ მეთოდებთან მუშაობის დემონსტრირება
class SabazoKlasi
{
//      ვირტუალური მეთოდის შექმნა წინაპარ კლასში
public virtual string Naxva()          //      ვირტუალური მეთოდის განსაზღვრა
{
return "Base კლასში განსაზღვრული Naxva() მეთოდი";
}
}
class MemkvidreKlasi1 : SabazoKlasi
{
//
public override string Naxva()         //      ვირტუალური მეთოდის ხელახალი განსაზღვრა
{
return "MemkvidreKlasi1 კლასში განსაზღვრული Naxva() მეთოდი";
}
}
class MemkvidreKlasi2 : SabazoKlasi
```

```
{
public override string Naxva()    // ვირტუალური მეთოდის კიდევ ერთი ხელახალი განსაზღვრა
{
retutn "MemkvidreKlasi2 კლასში განსაზღვრული Naxva() მეთოდი";
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//        პროგრამაში ხდება ვირტუალური მეთოდის გამოყენების დემონსტრირება
SabazoKlasi Sabazo_Obieqti = new SabazoKlasi();
MemkvidreKlasi1 Memkvidre_Obieqti1 = new MemkvidreKlasi1();
MemkvidreKlasi2 Memkvidre_Obieqti2   = new MemkvidreKlasi2();
//        წინაპარი კლასის ტიპის მიმართვითი ცვლადის გამოცხადება
SabazoKlasi Mimartviti_Cvladi;

Mimartviti_Cvladi = Sabazo_Obieqti;
label1.Text = Mimartviti_Cvladi.Naxva();

Mimartviti_Cvladi = Memkvidre_Obieqti1;
label2.Text = Mimartviti_Cvladi.Naxva();

Mimartviti_Cvladi = Memkvidre_Obieqti2;
label3.Text = Mimartviti_Cvladi.Naxva();
}
```

ამ პროგრამაში იქმნება SabazoKlasi წინაპარი კლასი, აგრეთვე, ორი მემკვიდრე კლასი MemkvidreKlasi1 და MemkvidreKlasi2. SabazoKlasi კლასში გამოცხადებულია Naxva() მეთოდი, რომელიც შემდეგ ხელახლა განისაზღვრება მემკვიდრე კლასებში. პროგრამაში გამოცხადებულია SabazoKlasi, MemkvidreKlasi1 და MemkvidreKlasi2 ტიპის ობიექტები, აგრეთვე, გამოცხადებულია SabazoKlasi ტიპის Mimartviti_Cvladi მიმართვითი ცვლადი. ამ ცვლადს რიგრიგობით ენიჭება მიმართვები სამივე ტიპის ობიექტზე. ეს მიმართვები გამოიყენება Naxva() მეთოდის გამოძახებისათვის. როგორც პროგრამის შესრულების შედეგიდან დავინახავთ, გამოსაძახებელი Naxva() მეთოდის ვერსიის არჩევა დამოკიდებულია იმ ობიექტის ტიპზე, რომელსაც მიმართავს Mimartviti_Cvladi ცვლადი.

ვირტუალური მეთოდის გადატვირთვა არ არის აუცილებელი. თუ მემკვიდრე კლასს არ აქვს ვირტუალური მეთოდის საკუთარი ვერსია, მაშინ გამოიძახება წინაპარი კლასის მეთოდი. ქვემოთ მოყვანილ პროგრამაში ხდება ვირტუალურ მეთოდებთან მუშაობის დემონსტრირება.

```
//        პროგრამა 8.11
//        პროგრამაში მემკვიდრე კლასს არ აქვს ვირტუალური მეთოდის საკუთარი ვერსია,
//        ამიტომ გამოიძახება წინაპარი კლასის მეთოდი
class SabazoKlasi
{
//        ვირტუალური მეთოდის შექმნა წინაპარ კლასში
public virtual void Naxva()            //        ვირტუალური მეთოდის გამოცხადება
```

```
{
MessageBox.Show("SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდი");
}
}
class MemkvidreKlasi1 : SabazoKlasi
{
//
public override void Naxva()                    //   ვირტუალური მეთოდის ხელახალი განსაზღვრა
{
MessageBox.Show("MemkvidreKlasi1 კლასში განსაზღვრული Naxva() მეთოდი");
}
}
class MemkvidreKlasi2 : SabazoKlasi
{
//        ამ კლასში არ ხდება ვირტუალური მეთოდის ხელახალი განსაზღვრა
}
private void button1_Click(object sender, System.EventArgs e)
{
//        პროგრამაში ხდება ვირტუალური მეთოდის გამოყენების დემონსტრირება
SabazoKlasi Sabazo_Obieqti = new SabazoKlasi();
MemkvidreKlasi1 Memkvidre_Obieqti1 = new MemkvidreKlasi1();
MemkvidreKlasi2 Memkvidre_Obieqti2 = new MemkvidreKlasi2();
//        წინაპარი კლასის ტიპის მიმართვითი ცვლადის გამოცხადება
SabazoKlasi Mimartviti_Cvladi;

Mimartviti_Cvladi = Sabazo_Obieqti;
Mimartviti_Cvladi.Naxva();

Mimartviti_Cvladi = Memkvidre_Obieqti1;
Mimartviti_Cvladi.Naxva();

Mimartviti_Cvladi = Memkvidre_Obieqti2;
//        SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდის გამოძახება
Mimartviti_Cvladi.Naxva();
}
```

აქ MemkvidreKlasi2 კლასში არ ხდება Naxva() მეთოდის ხელახალი განსაზღვრა, ამიტომ, MemkvidreKlasi2 ტიპის ობიექტის Naxva() მეთოდის გამოძახებისას გამოიძახება SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდი.

განვიხილოთ კიდევ ერთი მაგალითი. Sibrtye კლასში განვსაზღვროთ ვირტუალური Partobi() მეთოდი. იგი შეგვიძლია ხელახლა განვსაზღვროთ თითოეულ მემკვიდრე კლასში. ეს შესაძლებლობას მოგვცემს გამოვთვალოთ ფართობი გეომეტრიული ფიგურის ტიპზე დამოკიდებულებით (სამკუთხედი, კვადრატი და ა.შ.). ქვემოთ მოყვანილია ამ პროგრამის კოდი.

```
//        პროგრამა 8.12
//        პროგრამაში ხდება გეომეტრიული ფიგურის ფართობის გამოთვლა
```

```
//        ფიგურის ტიპზე დამოკიდებულებით
class Sibrtye
{
double pri_sigane;              //        დახურული ცვლადების გამოცხადება
double pri_simagle;
string pri_saxeli;
//        ავტომატური კონსტრუქტორის განსაზღვრა
public Sibrtye()
{
sigane = simagle = 0.0;
saxeli = "ნული";
}
//        კონსტრუქტორი სამი პარამეტრით
public Sibrtye(double w, double h, string n)
{
sigane = w;
simagle = h;
saxeli = n;
}
//        კონსტრუქტორი ორი პარამეტრით
public Sibrtye(double x, string n)
{
sigane = simagle = x;
saxeli = n;
}
//        ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Sibrtye(Sibrtye obieqti)
{
sigane  = obieqti.pri_sigane;
simagle = obieqti.pri_simagle;
saxeli  = obieqti.pri_saxeli;
}
//        sigane, simagle და name თვისებების განსაზღვრა
public double sigane
{
get { return pri_sigane; }
set { pri_sigane = value; }
}
public double simagle
{
get { return pri_simagle; }
set { pri_simagle = value; }
}
public string saxeli
{
```

```
get { return pri_saxeli; }
set { pri_saxeli = value; }
}
public void Naxva()
{
MessageBox.Show("გეომეტრიული ფიგურის სიგანე და სიმაღლე" + sigane.ToString() + "
                    " + simagle.ToString());
}
public virtual double Partobi()
{
MessageBox.Show("ეს მეთოდი ხელახლაა განსაზღვრული მემკვიდრე კლასში");
return 0.0;
}
}
//      Samkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
class Samkutxedi : Sibrtye
{
string tipi;
//      ავტომატური კონსტრუქტორის განსაზღვრა
public Samkutxedi()
{
tipi = "ნული";
}
//      კონსტრუქტორი ორი პარამეტრით, რომელიც იძახებს წინაპარი კლასის კონსტრუქტორს
public Samkutxedi(string s, double w, double h) : base(w, h, "სამკუთხედი")
{
tipi = s;
}
//      კონსტრუქტორი ერთი პარამეტრით, რომელიც იძახებს წინაპარი კლასის
კონსტრუქტორს
public Samkutxedi(double x) : base(x, " სამკუთხედი ")
{
tipi = "tolferda";
}
//      ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Samkutxedi(Samkutxedi obieqti) : base(obieqti)
{
tipi = obieqti.tipi;
}
//      Partobi() მეთოდის ხელახალი განსაზღვრა Samkutxedi კლასში
public override double Partobi()
{
return sigane * simagle / 2;
}
public void TipisNaxva()
```

```
{
MessageBox.Show("სამკუთხედის სახე - " + tipi);
}
}
//        Otxkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
class Otxkutxedi : Sibrtye
{
//        კონსტრუქტორი ორი პარამეტრით, რომელიც იძახებს წინაპარი კლასის კონსტრუქტორს
//        მახასიათებლები, დამახასიათებელი სწორკუთხედისთვის
public Otxkutxedi(double w, double h) : base(w, h, " სწორკუთხედი ")
{
}
//        კონსტრუქტორი ერთი პარამეტრით, რომელიც იძახებს წინაპარი კლასის
კონსტრუქტორს
public Otxkutxedi(double x) : base(x, "კვადრატი") { }
//        ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Otxkutxedi(Otxkutxedi obieqti) : base(obieqti) { }
public bool ArisKvadrati()
{
if ( sigane == simagle ) return true;
return false;
}
//        Partobi() მეთოდის ხელახალი განსაზღვრა Otxkutxedi კლასში
public override double Partobi()
{
return sigane * simagle;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
label1.Text = " ";
//        Sibrtye ტიპის მქონე figurebi ობიექტების მასივის გამოცხადება
Sibrtye[ ] figurebi = new Sibrtye[5];
figurebi[0] = new Samkutxedi("სწორკუთხა", 5.0, 15.0);
figurebi[1] = new Otxkutxedi(15);
figurebi[2] = new Otxkutxedi(15, 5);
figurebi[3] = new Samkutxedi(5.5);
figurebi[4] = new Sibrtye(15, 25, "საბაზო");

for ( int indexi = 0; indexi < figurebi.Length; indexi++ )
{
label1.Text += figurebi[indexi].saxeli + " - ფართობი = " +
                    figurebi[indexi].Partobi().ToString() + '\n';
}
}
```

ამ პროგრამის Sibrtye კლასში Partobi() მეთოდი გამოცხადებულია როგორც ვირტუალური. იგი ხელახლა განისაზღვრება Samkutxedi და Otxkutxedi კლასებში. Sibrtye კლასის Partobi() მეთოდი არ ახდენს გეომეტრიული ფიგურის ფართობის გამოთვლას. მისი შესრულების შედეგად უბრალოდ გაიცემა ინფორმაცია იმის შესახებ, რომ ეს მეთოდი ხელახლა უნდა იყოს განსაზღვრული მემკვიდრე კლასებში. ამიტომ, Partobi() მეთოდის ყოველი ხელახალი განსაზღვრისას, მასში სრულდება შესაბამისი გეომეტრიული ფიგურის ფართობის გამოთვლა.

მთავარ პროგრამაში გამოცხადებულია figurebi მასივი, რომელიც შეიცავს Sibrtye კლასის ობიექტებს. ამ მასივის ელემენტებს ენიჭებათ Samkutxedi, Otxkutxedi და Sibrtye ტიპის ობიექტებზე მიმართვები. ასეთი მინიჭებები სწორია, რადგან წინაპარი კლასის ტიპის მიმართვითი ცვლადები შეიძლება მიმართავდნენ მემკვიდრე კლასის ობიექტებს.

აბსტრაქტული კლასები

ზოგჯერ საჭიროა ისეთი წინაპარი კლასის შექმნა, რომელშიც განსაზღვრულია მეთოდების მხოლოდ ზოგიერთი მახასიათებლები, როგორიცაა დასაბრუნებელი მნიშვნელობის ტიპი, მეთოდის სახელი და პარამეტრების სია. ამ მახასიათებლების ერთობლიობას მეთოდის ხელმოწერა ეწოდება. ასეთი მეთოდების ტანი ცარიელია, რადგან ძნელია წინასწარ განვსაზღვროთ, თუ რა მოქმედებები უნდა შეასრულოს მეთოდმა მემკვიდრე კლასის ობიექტებში. ამიტომ, ამ მეთოდების რეალიზება (მეთოდის კოდის განსაზღვრა) მემკვიდრე კლასებში უნდა შესრულდეს.

ასეთ შემთხვევებში საბაზო კლასში შეგვიძლია განვსაზღვროთ აბსტრაქტული მეთოდები, რომლებსაც ცარიელი ტანი აქვთ. აბსტრაქტული მეთოდის გამოცხადებისათვის გამოიყენება abstract მოდიფიკატორი. თუ კლასის წევრების სიაში გვხვდება მეთოდი ამ მოდიფიკატორით, მაშინ მისი რეალიზება (განსაზღვრა) უნდა მოვახდინოთ მემკვიდრე კლასში. აბსტრაქტული მეთოდი ავტომატურად ხდება ვირტუალური ისე, რომ virtual სიტყვის გამოყენება ასეთი მეთოდის გამოცხადებისას არაა საჭირო. უფრო მეტიც, virtual და abstract მოდიფიკატორების ერთობლივი გამოყენება დაუშვებელია.

აბსტრაქტული მეთოდის გამოცხადების სინტაქსია:

abstract *ტიპი მეთოდის_სახელი (პარამეტრების_სია);*

ამ გამოცხადებაში არ არის მეთოდის კოდი. abstract მოდიფიკატორი შეგვიძლია გამოვიყენოთ მხოლოდ ჩვეულებრივი მეთოდების მიმართ და არ შეიძლება გამოვიყენოთ სტატიკური მეთოდების მიმართ.

კლასი, რომელიც ერთ ან მეტ აბსტრაქტულ მეთოდს შეიცავს, გამოცხადებული უნდა იყოს როგორც აბსტრაქტული, ე.ი. მისი გამოცხადებისას უნდა მივუთითოთ abstract მოდიფიკატორი. რადგან, აბსტრაქტული კლასი არ არის განსაზღვრული ბოლომდე, ამიტომ შეუძლებელია ამ კლასის ობიექტის შექმნა, და აბსტრაქტული კლასის შექმნის მცდელობა new საკვანძო სიტყვის გამოყენებით გამოიწვევს კომპილირების შეცდომას.

როცა კლასი არის აბსტრაქტული კლასის მემკვიდრე, მან უნდა მოახდინოს წინაპარი კლასის ყველა აბსტრაქტული მეთოდის რეალიზება. წინააღმდეგ შემთხვევაში, ასეთი მემკვიდრე კლასი გამოცხადებული უნდა იყოს abstract

მოდიფიკატორით. ამრიგად, abstract ატრიბუტი მემკვიდრეობით გადაიცემა მანამ, სანამ მეთოდები და ე.ი. თვით კლასი, არ იქნება მთლიანად რეალიზებული.

ქვემოთ მოყვანილ პროგრამაში Sibrtye კლასი გამოცხადებულია როგორც აბსტრაქტული. მასში გამოცხადებულია აბსტრაქტული Partobi() მეთოდი, რომელიც განკუთვნილია ორგანზომილებიანი გეომეტრიული ფიგურის ფართობის გამოსათვლელად. ამიტომ, Sibrtye კლასის მემკვიდრე კლასების შექმნისას მათში ხელახლა უნდა იყოს განსაზღვრული Partobi() მეთოდი.

```
//      პროგრამა 8.13
//      პროგრამაში ხდება აბსტრაქტულ კლასებთან მუშაობის დემონსტრირება
abstract class Sibrtye                  //      აბსტრაქტული კლასის გამოცხადება
{
double pri_sigane;                      //      დახურული ცვლადების გამოცხადება
double pri_simagle;
string pri_saxeli;
//      ავტომატური კონსტრუქტორის განსაზღვრა
public Sibrtye()
{
sigane = simagle = 0.0;
saxeli = "ნული";
}
//      კონსტრუქტორი პარამეტრებით
public Sibrtye(double w, double h, string n)
{
sigane  = w;
simagle = h;
saxeli  = n;
}
//      კონსტრუქტორი, განკუთვნილი ობიექტის შესაქმნელად, რომლის sigane და
//      simagle თვისებებს ენიჭებათ ერთნაირი მნიშვნელობები
public Sibrtye(double x, string n)
{
sigane = simagle = x;
saxeli = n;
}
//      ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Sibrtye(Sibrtye obieqti)
{
sigane  = obieqti.pri_sigane;
simagle = obieqti.pri_simagle;
saxeli  = obieqti.pri_saxeli;
}
//      sigane, simagle და name თვისებების განსაზღვრა
public double sigane
{
get { return pri_sigane; }
```

```
set { pri_sigane = value; }
}
public double simagle
{
get { return pri_simagle; }
set { pri_simagle = value; }
}
public string saxeli
{
get { return pri_saxeli; }
set { pri_saxeli = value; }
}
public void Naxva()
{
MessageBox.Show("გეომეტრიული ფიგურის სიგანე და სიმაღლე" +
                    sigane.ToString() + "  " + simagle.ToString());
}
//      აბსტრაქტული Partobi() მეთოდის გამოცხადება
public abstract double Partobi();
}
//      მემკვიდრე კლასის გამოცხადება
class Samkutxedi : Sibrtye
{
string tipi;
//      ავტომატური კონსტრუქტორსი განსაზღვრა
public Samkutxedi()
{
tipi = "nuli";
}
//      Samkutxedi კლასის კონსტრუქტორი
public Samkutxedi(string s, double w, double h) : base(w, h, "სამკუთხედი")
{
tipi = s;
}
//      კონსტრუქტორი, რომელიც იძახებს წინაპარი კლასის კონსტრუქტორს
public Samkutxedi(double x) : base(x, " სამკუთხედი")
{
tipi = "ტოლფერდა";
}
//      ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Samkutxedi(Samkutxedi obieqti) : base(obieqti)
{
tipi = obieqti.tipi;
}
//      Partobi() მეთოდის ხელახალი განსაზღვრა Samkutxedi კლასში
```

```
public override double Partobi()
{
return sigane * simagle / 2;
}
public void TipisNaxva()
{
MessageBox.Show("სამკუთხედის სახე - " + tipi);
}
}
//      Otxkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
class Otxkutxedi : Sibrtye
{
//      კონსტრუქტორი პარამეტრებით, რომელიც იძახებს წინაპარი კლასის კონსტრუქტორს
public Otxkutxedi(double w, double h) : base(w, h, "სწორკუთხედი")
{
//      ამ მეთოდის კოდი არ არის განსაზღვრული
}
//      კონსტრუქტორი, განსაზღვრული ობიექტის შესაქმნელად, რომელშიც
//      განსაზღვრულია მახასიათებლები, დამახასიათებელი კვადრატისათვის
public Otxkutxedi(double x) : base(x, "კვადრატი") { }
//      ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Otxkutxedi(Otxkutxedi obieqti) : base(obieqti) { }
public bool ArisKvadrati()
{
if ( sigane == simagle ) return true;
return false;
}
//      Partobi() მეთოდის ხელახალი განსაზღვრა Otxkutxedi კლასში
public override double Partobi()
{
return sigane * simagle;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
label2.Text = " ";
Sibrtye[ ] figurebi = new Sibrtye[4];
figurebi[0] = new Samkutxedi("სწორკუთხა", 5.0, 15.0);
figurebi[1] = new Otxkutxedi(15);
figurebi[2] = new Otxkutxedi(15, 5);
figurebi[3] = new Samkutxedi(5.5);

for ( int indexi = 0; indexi < figurebi.Length; indexi++ )
{
label2.Text += figurebi[indexi].saxeli + " - ფართობი = " +
```

```
                figurebi[indexi].Partobi().ToString() + '\n';
}
}
```

მართალია, Sibrtye ტიპის ობიექტების შექმნა შეუძლებელია, მაგრამ შესაძლებელია Sibrtye ტიპის მიმართვითი ცვლადის შექმნა. ამიტომ, წინა პროგრამისაგან განსხვავებით, პროგრამის ამ ვერსიაში გამოცხადებულია მასივი, რომელიც შედგება ოთხი და არა ხუთი ელემენტისაგან.

sealed საკვანძო სიტყვა

ზოგჯერ წამოიჭრება ისეთი კლასების შექმნის აუცილებლობა, რომელთა მემკვიდრეობით გადაცემა არ შეიძლება. ასეთ შემთხვევებში გამოიყენება sealed საკვანძო სიტყვა. იგი ეთითება კლასის სახელის წინ. ქვემოთ მოყვანილი კოდის ფრაგმენტი ახდენს ამის დემონსტრირებას:

```
//      პროგრამა 8.14
//      პროგრამაში ხდება მემკვიდრეობითობის აკრძალვის დემონსტრირება
sealed class Klasi1
{
//      Klasi1 კლასის კოდი
}
//      ამ კლასის გამოცხადება არ არის ნამდვილი
class Klasi2 : Klasi1   //      შეცდომა! შეუძლებელია Klasi1 კლასის მემკვიდრე კლასის შექმნა
{
//      Klasi2 კლასის კოდი
}
```

რადგან Klasi1 კლასის გამოცხადებისას გამოყენებულია sealed საკვანძო სიტყვა, ამიტომ მისი მემკვიდრეობით გადაცემა დაუშვებელია. არ შეიძლება, აგრეთვე, abstract და sealed საკვანძო სიტყვების ერთდროულად გამოყენება.

object კლასი

C# ენაში განსაზღვრულია სპეციალური object კლასი, რომელიც არის საბაზო ყველა სხვა კლასისათვის. სხვა სიტყვებით რომ ვთქვათ, ყველა კლასი არის object კლასის მემკვიდრე, ე.ი. object ტიპის მქონე მიმართვითი ცვლადი შეიძლება მიმართავდეს ნებისმიერი სხვა ტიპის ობიექტს. object სახელი - ესაა System.Object სახელის შემოკლებული ფორმა, რომელიც არის .NET Framework კლასების ბიბლიოთეკის შემადგენელი ნაწილი.

თავი 9. ინფორმაციის შეტანა-გამოტანა

შეტანა-გამოტანის ნაკადების კლასები

C#-პროგრამები მონაცემების შეტანა-გამოტანას ნაკადების საშუალებით ახდენენ. ნაკადის ქვეშ იგულისხმება მონაცემების მიმდევრობა. ნაკადი ფიზიკურ მოწყობილობას შეტანა-გამოტანის სისტემის საშუალებით უკავშირდება.

C# ენაში შეტანა-გამოტანის ოპერაციები ყველაზე დაბალ დონეზე ბაიტებთან ოპერირებენ, რადგან შეტანა-გამოტანის მოწყობილობების უმრავლესობა არის ბაიტ-ორიენტირებული. მეორეს მხრივ, კომპიუტერთან ურთიერთობისას ადამიანებისათვის მოხერხებულია სიმბოლოებთან მუშაობა. ამიტომ, შეტანა-გამოტანის ოპერაციების შესრულების დროს საჭირო ხდება ბაიტების გარდაქმნა სიმბოლოებად და პირიქით.

არსებობს სიმბოლოების ASCII და Unicode ნაკრები. ASCII სიმბოლოს ზომაა 1 ბაიტი, ხოლო Unicode სიმბოლოს ზომა კი - 2 ბაიტი. გავიხსენოთ, რომ C# ენაში char ტიპის სიმბოლო იკავებს 2 ბაიტს, byte ტიპის მნიშვნელობა კი - 1 ბაიტს. ამიტომ, ბაიტების ნაკადების გამოყენება არც ისე მოხერხებულია სიმბოლოების შეტანა-გამოტანის შესრულებისას. ამ ამოცანის გადასაწყვეტად C# ენაში განსაზღვრულია რამდენიმე კლასი, რომლებიც ახდენენ ბაიტების ნაკადების გარდაქმნას სიმბოლოების ნაკადებად, ე.ი. ავტომატურად გარდაქმნიან byte ტიპის მონაცემებს char ტიპის მონაცემებად და პირიქით.

C# ენაში არსებობს ბაიტებისა და სიმბოლოების ნაკადების კლასები. ნაკადების კლასები განსაზღვრულია System.IO კლასში. ამიტომ, ნებისმიერი პროგრამის დასაწყისში, რომელიც იყენებს ნაკადების კლასებს, უნდა ჩავრთოთ ინსტრუქცია using System.IO .

ბაიტების ნაკადები

C# ენაში ნაკადები იქმნება System.IO.Stream კლასის საშუალებით. Stream კლასი წარმოადგენს ბაიტების ნაკადებს და არის საბაზო ყველა სხვა კლასებისათვის. ის, აგრეთვე, აბსტრაქტულია, ამიტომ, არ არსებობს Stream ობიექტის რეალიზაცია. Stream კლასში განსაზღვრულია ნაკადებზე სტანდარტული ოპერაციები. ცხრილში 9.1 მოყვანილია Stream კლასში განსაზღვრული რამდენიმე ხშირად გამოყენებადი მეთოდი.

Stream კლასში განსაზღვრულია მეთოდები, განკუთვნილი მონაცემების ჩაწერისა და წაკითხვისათვის. მაგრამ, ყველა ნაკადი არ უზრუნველყოფს მონაცემების ჩაწერისა და წაკითხვის ოპერაციებს, აგრეთვე, პოზიციონირებას Seek() მეთოდის გამოყენებით. კონკრეტული ნაკადის შესაძლებლობების გასაგებად შეგვიძლია გამოვიყენოთ Stream კლასის თვისებები (ცხრილში 9.2).

Stream კლასს აქვს ბაიტების ნაკადების სამი მემკვიდრე კლასი, რომლებიც მოყვანილია ცხრილში 9.3.

ცხრილი 9.1. Stream კლასში განსაზღვრული ზოგიერთი მეთოდი.

მეთოდი	აღწერა
void Close()	ნაკადს ხურავს
void Flush()	ნაკადის შემცველობას ფიზიკურ მოწყობილობაზე წერს
int ReadByte()	შესასვლელი ნაკადიდან კითხულობს 1 ბაიტს და გასცემს მის მთელრიცხვა წარმოდგენას
int Read(byte[ ] მასივი, int პოზიცია, int ბაიტების_რაოდენობა)	შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის ბაიტს და მათ ათავსებს მასივში დაწყებული მითითებული პოზიციიდან. გაიცემა წარმატებით წაკითხული ბაიტების რაოდენობა
long Seek(long პოზიცია, SeekOrigin გადათვლის_დასაწყისი)	გასცემს ნაკადში მიმდინარე ბაიტის პოზიციას, გადათვლილს მითითებული პოზიციიდან
void WriteByte(byte ცვლადი)	გამოსასვლელ ნაკადში წერს ერთ ბაიტს
int Wrtite(byte[ ] მასივი, int პოზიცია, int ბაიტების_რაოდენობა)	გამოსასვლელ ნაკადში მასივიდან წერს მითითებული რაოდენობის ბაიტს, დაწყებული მითითებული პოზიციიდან. გაიცემა ჩაწერილი ბაიტების რაოდენობა

სიმბოლოების ნაკადები

სიმბოლოების ნაკადებთან სამუშაოდ გამოიყენება TextReader და TextWriter აბსტრაქტული კლასები. მათში განსაზღვრული მეთოდები უზრუნველყოფენ შეტანა-გამოტანის ოპერაციებს სიმბოლოების ნაკადებისათვის.

ცხრილი 9.2. Stream კლასის თვისებები.

თვისება	აღწერა
bool CanRead	თვისებას აქვს true მნიშვნელობა, თუ შესაძლებელია ნაკადიდან წაკითხვა. თვისება განკუთვნილია მხოლოდ წასაკითხად.
bool CanSeek	თვისებას აქვს true მნიშვნელობა, თუ ნაკადის ელემენტის მიმდინარე პოზიციის განსაზღვრა შესაძლებელია. თვისება განკუთვნილია მხოლოდ წასაკითხად.
bool CanWrite	თვისებას აქვს true მნიშვნელობა, თუ შესაძლებელია ნაკადში ჩაწერა. თვისება განკუთვნილია მხოლოდ წასაკითხად.
long Length	თვისება განსაზღვრავს ნაკადის სიგრძეს. თვისება განკუთვნილია მხოლოდ წასაკითხად.
long Position	თვისება განსაზღვრავს ნაკადის მიმდინარე ელემენტის პოზიციას. თვისება განსაზღვრულია როგორც წაკითხვის, ისე ჩაწერისათვის.

ცხრილი 9.3. Stream კლასის მემკვიდრე კლასები.

კლასი	აღწერა
BufferedStream	უზრუნველყოფს ბაიტების ნაკადის ბუფერიზაციას
FileStream	ბაიტების ნაკადია, განკუთვნილი ფაილში შეტანა-გამოტანის ოპერაციების შესასრულებლად
MemoryStream	ბაიტების ნაკადია, რომელიც გამოიყენება მეხსიერებასთან სამუშაოდ

ცხრილში 9.4 მოყვანილია TextReader კლასის შეტანის მეთოდები. აღვნიშნოთ, რომ ReadLine() მეთოდის გამოყენება მოხერხებულია შესასვლელი ნაკადიდან იმ მონაცემების წასაკითხად, რომელიც ინტერვალებს შეიცავს. ცხრილში 9.5 მოყვანილია TextWriter კლასში განსაზღვრული Write() და WriteLine() მეთოდების სხვადასხვა ვერსიები.

TextReader და TextWriter კლასების რეალიზება ხდება ცხრილში 9.6 მოყვანილი სიმბოლოების ნაკადების კლასების საშუალებით. ეს ნაკადები უზრუნველყოფენ TextReader და TextWriter კლასებში განსაზღვრულ მეთოდებსა და თვისებებს.

ცხრილი 9.4. TextReader კლასის შეტანის მეთოდები.

მეთოდი	აღწერა
void Close()	ხურავს შესასვლელ ნაკადს
int Peek()	კითხულობს შესასვლელი ნაკადის მომდევნო სიმბოლოს, მაგრამ არ შლის მას. გასცემს -1-ს თუ არ არის მომდევნო სიმბოლო
int Read()	კითხულობს შესასვლელი ნაკადის სიმბოლოს და გასცემს მის მთელრიცხვა წარმოდგენას. გასცემს -1-ს ნაკადის დასასრულის მიღწევისას
int Read(char[ ] მასივი, int პოზიცია, int სიმბოლოების_რაოდენობა)	შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ მასივში დაწყებული მითითებული პოზიციიდან. გაიცემა წარმატებით წაკითხული სიმბოლოების რაოდენობა
int ReadBlock(char[ ] მასივი, int პოზიცია, int სიმბოლოების_რაოდენობა)	შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ მასივში დაწყებული მითითებული პოზიციიდან. ამასთან, გაიცემა წარმატებით წაკითხული სიმბოლოების რაოდენობა
string ReadLine()	კითხულობს ტექსტის სტრიქონს და გასცემს მას C# ენის სტრიქონის სახით. ნულოვანი მნიშვნელობა იმ შემთხვევაში გაიცემა, თუ წაკითხული იქნა ფაილის დასასრულის სიმბოლო
string ReadToEnd()	კითხულობს ნაკადის დარჩენილ სიმბოლოებს და გასცემს მათ C# ენის სტრიქონის სახით

ცხრილი 9.5. TextWriter კლასის გამოტანის მეთოდები.

მეთოდი	აღწერა
void Write(int ცვლადი)	int ტიპის მნიშვნელობის ჩაწერა
void Write(double ცვლადი)	double ტიპის მნიშვნელობის ჩაწერა
void Write(bool ცვლადი)	bool ტიპის მნიშვნელობის ჩაწერა
void WriteLine(string ცვლადი)	სტრიქონის ჩაწერა, რომელიც მთავრდება მომდევნო სტრიქონზე გადასვლის სიმბოლოთი
void WriteLine(uint ცვლადი)	uint ტიპის მნიშვნელობისა და მომდევნო სტრიქონზე გადასვლის სიმბოლოს ჩაწერა
void WriteLine(char ცვლადი)	სიმბოლოს ჩაწერა, რომელსაც მოსდევს მომდევნო სტრიქონზე გადასვლის სიმბოლო
virtual void Close()	ნაკადს ხურავს
virtual void Flush()	ახდენს ბუფერში დარჩენილი მონაცემების ჩაწერას გამოსასვლელ ნაკადში

ცხრილი 9.6. სიმბოლოების ნაკადების კლასები.

ნაკადის კლასი	აღწერა
StreamReader	ბაიტების ნაკადიდან კითხულობს სიმბოლოებს
StreamWriter	სიმბოლოებს წერს ბაიტების ნაკადში
StringReader	სტრიქონიდან კითხულობს სიმბოლოებს
StringWriter	სტრიქონში წერს სიმბოლოებს

ორობითი ნაკადები

C# ენაში განსაზღვრულია აგრეთვე, ორობითი ნაკადების ორი კლასი BinaryReader და BinaryWriter. ისინი შეგვიძლია გამოვიყენოთ ორობითი მონაცემების ჩაწერისა და წაკითხვისათვის.

FileStream კლასი

ეს კლასი გამოიყენება ფაილებში ბაიტ-ორიენტირებული შეტანა-გამოტანის შესასრულებლად, ანუ ფაილში ბაიტების ჩაწერა-წაკითხვის ოპერაციების შესასრულებლად. რადგან ფაილის ყველაზე გავრცელებული ტიპია დისკური ფაილი, ამიტომ შემდგომში ამ ტიპის ფაილებს განვიხილავთ.

ფაილის გახსნა და დახურვა

ფაილთან დაკავშირებული ბაიტების ნაკადის შესაქმნელად საჭიროა FileStream ობიექტის ფორმირება. მის ფორმირებას ახდენს კონსტრუქტორი, რომლის სინტაქსია:
FileStream(string ფაილის_სახელი, FileMode.რეჟიმი);
სადაც, ფაილის_სახელი არის გასახსნელი ფაილის სახელი, რომელიც შეიძლება შეიცავდეს, აგრეთვე, ამ ფაილისაკენ გზას, მაგალითად, C:\My Documents\A1\File1.txt. რეჟიმი პარამეტრი განსაზღვრავს თუ როგორ უნდა გაიხსნას ფაილი. მისი მნიშვნელობები ანუ ფაილის გახსნის რეჟიმები მოყვანილია ცხრილში 9.7.

ქვემოთ მოყვანილ პროგრამაში სრულდება testfile.dat ფაილის გახსნა და

დახურვა. არ უნდა დაგვავიწყდეს პროგრამის დასაწყისში using გამოცხადებების შემდეგ using System.IO ინსტრუქციის ჩართვა. ეს უნდა გავაკეთოთ ამ თავში მოყვანილი ყველა პროგრამისათვის.

```
{
//      პროგრამა 9.1
//      პროგრამაში ხდება ფაილის გახსნა და დახურვა
FileStream file_in;

file_in = new FileStream("filetest.dat", FileMode.Open);
file_in.Close();
}
```

ცხრილი 9.7. mode პარამეტრის მნიშვნელობები.

მნიშვნელობა	აღწერა
FileMode.Append	მონაცემები დაემატება ფაილის ბოლოში
FileMode.Create	იქმნება ახალი გამოსასვლელი ფაილი. ამავე სახელის მქონე ადრე შექმნილი ფაილი იშლება
FileMode.CreateNew	იქმნება ახალი გამოსასვლელი ფაილი
FileMode.Open	იხსნება არსებული ფაილი
FileMode.OpenCreate	თუ ფაილი არსებობს, მაშინ ის გაიხსნება, თუ არ არსებობს, მაშინ ის შეიქმნება
FileMode.Truncate	იხსნება არსებული ფაილი, მაგრამ მისი სიგრძე ჩამოიჭრება ნულამდე

პროგრამის პირველი ორი სტრიქონი შეგვიძლია შევცვალოთ ერთით:

```
FileStream file_in = new FileStream("filetest.dat", FileMode.Open);
```

პროგრამაში მითითებულია ფაილის მხოლოდ სახელი, ამიტომ მისი ძებნა შესრულდება მიმდინარე კატალოგში. თუ ფაილი სხვა კატალოგშია მოთავსებული, მაშინ უნდა მივუთითოთ გზა ამ ფაილისაკენ:

```
file_in = new FileStream("C:\\My Documents\\Visual Studio Projects\\
                          A1\\filetest.dat", FileMode.Open);
```

თუ ფაილთან მიმართვა უნდა შეიზღუდოს მხოლოდ წაკითხვით ან ჩაწერით, მაშინ უნდა მივუთითოთ ფაილთან მიმართვის სახე:

```
FileStream(string ფაილის_სახელი, რეჟიმი, FileAccess მიმართვის_სახე)
```

აქ მიმართვის_სახე იღებს მნიშვნელობებს: FileAccess.Read, FileAccess.Write და FileAccess.ReadWrite. მაგალითად,

```
FileStream file_in = new FileStream("file1.dat",FileMode.Open,FileAccess.Read);
```

აქ file1.dat ფაილი იხსნება წაკითხვის რეჟიმში.

ფაილთან მუშაობის დასამთავრებლად აუცილებელია მისი დახურვა Close() მეთოდის გამოყენებით. ფაილის დახურვისას თავისუფლდება მისთვის გამოყოფილი ოპერაციული სისტემის რესურსები.

ფაილიდან ბაიტების წაკითხვა

FileStream კლასში განსაზღვრულია ორი მეთოდი, რომლებიც ფაილიდან ბაიტებს კითხულობენ: ReadByte() და Read(). ReadByte() მეთოდის სინტაქსია:

int ReadByte()

ამ მეთოდის გამოძახებისას ხდება ერთი ბაიტის წაკითხვა ფაილიდან და გაიცემა ბაიტის კოდი (მთელრიცხვა მნიშვნელობა). თუ წაკითხული იქნა ფაილის დასასრულის სიმბოლო (EOF, End Of File), მაშინ გაიცემა -1.

ბაიტების ბლოკის წასაკითხად უნდა გამოვიყენოთ Read() მეთოდი. მისი სინტაქსია:

int Read(byte[ ] *ბუფერი,* int *პოზიცია,* int *ბაიტების_რაოდენობა)*

Read() მეთოდი ცდილობს ფაილიდან წაიკითხოს მითითებული რაოდენობის ბაიტი და მოათავსოს ისინი ბუფერში დაწყებული მითითებული პოზიციიდან. გაიცემა რეალურად წაკითხული ბაიტების რაოდენობა.

მოყვანილ პროგრამაში სრულდება ბაიტების წაკითხვა ReadByte() მეთოდის გამოიყენებით და მათი ეკრანზე გამოტანა. დავუშვათ filetext.txt ფაილში ჩაწერილია ციფრები: 1234567890 .

```
{
//      პროგრამა 9.2
//      პროგრამაში ხდება ბაიტების წაკითხვის დემონსტრირება ReadByte() მეთოდის გამოყენებით
int i;
FileStream file1 = new FileStream("filetext.txt", FileMode.Open);

// მონაცემების წაკითხვა ფაილიდან მანამ, სანამ არ შეგვხვდება EOF სიმბოლო
do
{
        i = file1.ReadByte();
        if ( i != -1 ) label1.Text += (char)i;
}
while ( i != -1 );          //  თუ i = -1, მაშინ მიღწეულია ფაილის დასასრული
file1.Close();
}
```

მოყვანილ პროგრამაში სრულდება ბაიტების წაკითხვა Read() მეთოდის გამოიყენებით და მათი ეკრანზე გამოტანა. დავუშვათ filetext.txt ფაილში ჩაწერილია ციფრები: 1234567890 .

```
{
//      პროგრამა 9.3
//      პროგრამაში ხდება ბაიტების წაკითხვის დემონსტრირება Read() მეთოდის გამოყენებით
int wakitxuli_baitebis_raodenoba;
int pozicia   = Convert.ToInt32(textBox1.Text),
  raodenoba = Convert.ToInt32(textBox2.Text);
byte[] masivi = new byte[10];
FileStream file1 = new FileStream("filetext.txt", FileMode.Open);
label1.Text = "";

// masivi მასივში ჩაიწერება 4 ბაიტი დაწყებული მე-3 ელემენტიდან
wakitxuli_baitebis_raodenoba = file1.Read(masivi, pozicia, raodenoba);
```

```
file1.Close();
for (int indexi = 0; i < masivi.Length; i++)
        label1.Text += masivi[indexi].ToString() + "   ";
label2.Text = wakitxuli_baitebis_raodenoba.ToString();
}
```

როგორც ვიცით Read() მეთოდს სამი არგუმენტი აქვს: ბუფერი, პოზიცია და ბაიტების რაოდენობა. შესაბამისად, პროგრამაში გამოცხადებულია masivi მასივი, pozicia და raodenoba ცვლადები. დავუშვათ, pozicia ცვლადს მივანიჭეთ 2, raodenoba ცვლადს კი - 4. შედეგად, filetext.txt ფაილიდან შესრულდება 4 ბაიტის წაკითხვა და masivi მასივში მესამე პოზიციიდან (masivi[2]) ჩაწერა. masivi მასივი მიიღებს სახეს:

0 0 49 50 51 52 0 0 0 0

Read() მეთოდი, აგრეთვე, გასცემს რეალურად წაკითხული ბაიტების რაოდენობას, რომელსაც მოვათავსებთ wakitxuli_baitebis_raodenoba ცვლადში.

ფაილში ბაიტების ჩაწერა

FileStream კლასში განსაზღვრულია ორი მეთოდი, რომლებიც ფაილში წერენ ბაიტებს: WriteByte() და Write(). WriteByte() მეთოდი გამოიყენება ფაილში ბაიტების ჩასაწერად. მისი სინტაქსია:

void WriteByte(byte *ცვლადის_სახელი)*

Write() მეთოდი გამოიყენება ფაილში ბაიტების ბლოკის (მასივის) ჩასაწერად. მისი სინტაქსია:

int Write(byte[ ] *ბუფერი,* int *პოზიცია,* int *ბაიტების_რაოდენობა)*

Write() მეთოდი ბუფერიდან ფაილში ჩაწერს მითითებული რაოდენობის ბაიტს, დაწყებული მითითებული პოზიციიდან. გაიცემა რეალურად ჩაწერილი ბაიტების რაოდენობა.

ფაილში ჩაწერის დროს ხშირად არ ხდება მონაცემების მაშინვე ჩაწერა ფიზიკურ მოწყობილობაზე. ამის ნაცვლად, ოპერაციული სისტემა ახდენს გამოტანის ბუფერიზებას, ანუ ჩასაწერი ბაიტების ბუფერში დაგროვებას. ბუფერი არის მეხსიერების გარკვეული უბანი. ბუფერიზება სრულდება მანამ, სანამ არ დაგროვდება მონაცემების საკმარისი რაოდენობა. ბუფერის შევსებისთანავე მონაცემები ჩაიწერება დისკზე. შედეგად, იზრდება ოპერაციული სისტემის მწარმოებლურობა. საქმე ის არის, რომ დისკზე მონაცემების ჩაწერა ხდება სექტორებად, რომელთა ზომები იცვლება დაწყებული 128 ბაიტიდან ზევით. გამოტანის შედეგების ბუფერიზება ხდება მანამ, სანამ დისკზე შესაძლებელი არ გახდება მთელი სექტორის ჩაწერა.

რიგ შემთხვევებში საჭიროა მონაცემების დისკზე ჩაწერა მიუხედავად იმისა ბუფერი შევსებულია თუ არა. ასეთ შემთხვევაში უნდა გამოვიყენოთ Flush() მეთოდი. მისი სინტაქსია:

void Flush()

ფაილთან მუშაობის დამთავრებისას იგი უნდა დაიხუროს Close() მეთოდის გამოყენებით. ამ დროს ბუფერში მოთავსებული მონაცემები, დისკზე აუცილებლად ჩაიწერება. ამიტომ, Close() მეთოდის გამოძახების წინ არაა საჭირო Flush() მეთოდის გამოძახება.

მოყვანილ პროგრამაში სრულდება ერთი ფაილის მეორეში გადაწერა.

```
{
//      პროგრამა 9.4
```

```
//      პროგრამა ასრულებს ერთი ფაილის მეორეში გადაწერას
int ricxvi;
FileStream file_in;
FileStream file_out;
//      საწყისი ფაილის გახსნა
file_in = new FileStream("s_file.txt", FileMode.Open);
//      მიმღები ფაილის გახსნა
file_out = new FileStream("d_file.txt", FileMode.Create);
label1.Text = “”;
//      ბაიტების გადაწერა file_in ფაილიდან file_out ფაილში
do
{
//      ბაიტების წაკითხვა file_in ფაილიდან
ricxvi = file_in.ReadByte();
//      ბაიტების ჩაწერა file_out ფაილში
if ( ricxvi != -1 ) file_out.WriteByte( ( byte ) ricxvi );
label1.Text += (char) ricxvi + “   “;
}
while ( ricxvi != -1 );
//      ფაილების დახურვა
file_in.Close();
file_out.Close();
}
```

მოყვანილ პროგრამაში Write() მეთოდის გამოიყენებით სრულდება filetext.txt ფაილში ბაიტების ჩაწერა და მათი ეკრანზე გამოტანა.

```
{
//      პროგრამა 9.5
//      პროგრამაში ხდება მასივიდან ფაილში ბაიტების ჩაწერა
byte[] masivi = new byte[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
FileStream file_out;
file_out = new FileStream("d_file.txt", FileMode.Create);

//      ბაიტების ჩაწერა d_file.txt ფაილში
file_out.Write(masivi, 2, 4);

file_out.Close();
}
```

პროგრამის შესრულების შედეგად filetext.txt ფაილში ჩაიწერება masivi მასივის 4 ელემენტი დაწყებული მესამე ელემენტიდან (masivi[2]).

ფაილში სიმბოლოების შეტანა-გამოტანა

ფაილში სიმბოლოების ჩასაწერად და წასაკითხად სიმბოლოების ნაკადები გამოიყენება. ისინი Unicode სიმბოლოებზე ოპერირებენ. სიმბოლურ ფაილებთან

მუშაობისას FileStream ობიექტი უნდა ჩავრთოთ StreamReader ან StreamWrtiter კლასის შემადგენლობაში. ეს კლასები უზრუნველყოფენ ბაიტების ნაკადების სიმბოლოების ნაკადებად ავტომატურ გარდაქმნას და პირიქით.

StreamWriter კლასი არის TextWriter კლასის მემკვიდრე. StreamReader კლასი არის TextReader კლასის მემკვიდრე. შედეგად, StreamWriter და StreamReader კლასის წევრებს შეუძლიათ მიმართონ TextWriter და TextReader კლასების მეთოდებსა და თვისებებს.

StreamWriter კლასი

სიმბოლოების გამოსასვლელი ნაკადი გამოიყენება სიმბოლოების ფაილში მონაცემების ჩასაწერად. ამისათვის, FileStream ობიექტი უნდა ჩავრთოთ StreamWriter კლასის შემადგენლობაში. ამ კლასში განსაზღვრულია რამდენიმე კონსტრუქტორი. ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

StreamWriter(Stream *ნაკადის_სახელი)*

ობიექტის შექმნისთანავე StreamWriter კლასი ავტომატურად გარდაქმნის სიმბოლოებს ბაიტებად.

მოყვანილი პროგრამა ახდენს textBox კომპონენტში შეტანილი ტექსტის ჩაწერას ფაილში.

```
{
//      პროგრამა 9.6
//      პროგრამაში ხდება ფაილში ჩაწერის დემონსტრირება StreamWriter კლასის
გამოყენებით
string striqoni;
FileStream file_out;
file_out = new FileStream("text.txt", FileMode.Create);
//
StreamWriter file_stream_out = new StreamWriter(file_out);

//      textBox კომპონენტში შეტანილი ტექსტის მინიჭება striqoni ცვლადისათვის
striqoni = textBox1.Text;
file_stream_out.Write(striqoni);    //      text.txt ფაილში striqoni სტრიქონის ჩაწერა
file_stream_out.Close();
}
```

StreamReader კლასი

სიმბოლოების შესასვლელი ნაკადი გამოიყენება სიმბოლოების ნაკადიდან მონაცემების წასაკითხად. ამისათვის, FileStream ობიექტი უნდა ჩავრთოთ StreamReader კლასის შემადგენლობაში. ამ კლასში განსაზღვრულია რამდენიმე კონსტრუქტორი. ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

StreamReader(Stream *ნაკადის_სახელი)*

StreamReader ობიექტის შექმნისთანავე ავტომატურად სრულდება ბაიტების გარდაქმნა სიმბოლოებად.

მოყვანილი პროგრამა კითხულობს ტექსტურ ფაილს და მისი შემცველობა ეკრანზე გამოაქვს.

```
{
//      პროგრამა 9.7
```

```
//      პროგრამაში ხდება ტექსტური ფაილიდან წაკითხვის დემონსტრირება
//      StreamReader კლასის გამოყენებით
FileStream file_in;
string striqoni;
file_in = new FileStream("file1.txt", FileMode.Open );
StreamReader file_stream_in = new StreamReader(file_in);

while ( ( striqoni = file_stream_in.ReadLine() )  != null )
{
label1.Text += striqoni + '\n';
}
file_stream_in.Close();
}
```

პროგრამაში ფაილიდან სტრიქონების წაკითხვა მთავრდება მაშინ, როცა ReadLine() მეთოდის მიერ გაცემული სტრიქონი მიიღებს ნულოვან მნიშვნელობას. სწორედ ეს მიუთითებს ფაილის დასასრულზე.

ორობითი მონაცემების წაკითხვა და ჩაწერა

ფაილში ბაიტებისა და სიმბოლოების წაკითხვისა და ჩაწერის გარდა შესაძლებელია სხვა ტიპის (ორობითი) მონაცემების წაკითხვა და ჩაწერა, როგორიცაა int, double და ა.შ. მათი წაკითხვისა და ჩაწერისათვის გამოიყენება BinaryReader და BinaryWriter ნაკადები. ასეთი მონაცემების წაკითხვა და ჩაწერა ხდება შიდა ორობით ფორმატში და არა ტექსტური ფორმით.

BinaryWriter კლასი

BinaryWriter ნაკადი გამოიყენება ფაილში ორობითი მონაცემების ჩასაწერად. ამ კლასის ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

BinaryWriter(Stream *გამოსასვლელი_ნაკადი)*

აქ გამოსასვლელი_ნაკადი პარამეტრი განსაზღვრავს იმ ნაკადს, რომელშიც სრულდება მონაცემების ჩაწერა. იგი არის ობიექტი, რომელსაც ქმნის FileStream კონსტრუქტორი.

BynaryWriter კლასში განსაზღვრულია მეთოდები, რომლებიც უზრუნველყოფენ C# ენის ყველა ჩადგმული ტიპის მონაცემების ჩაწერას. ზოგიერთი მეთოდი მოყვანილია ცხრილში 9.8. ამავე კლასში განსაზღვრულია, აგრეთვე, Close() და Flush() მეთოდები.

BinaryReader ნაკადი

BinaryReader ნაკადი გამოიყენება ფაილიდან ორობითი მონაცემების წასაკითხად. ამ კლასის ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

BinaryReader(Stream *შესასვლელი_ნაკადი)*

აქ შესასვლელი_ნაკადი პარამეტრი განსაზღვრავს იმ ნაკადს, საიდანაც სრულდება მონაცემების წაკითხვა. იგი არის ობიექტი, რომელსაც ქმნის FileStream კონსტრუქტორი.

ცხრილი9.8. BynaryWriter ნაკადის ზოგიერთი მეთოდი.

მეთოდი	აღწერა
void Write(sbyte ცვლადი)	ნიშნიანი ბაიტის ჩაწერა
void Write(byte ცვლადი)	უნიშნო ბაიტის ჩაწერა
void Write(byte[ ] მასივი)	ბაიტების მასივის ჩაწერა
void Write(short ცვლადი)	მოკლე მთელი რიცხვის ჩაწერა
void Write(ushort ცვლადი)	უნიშნო მოკლე მთელი რიცხვის ჩაწერა
void Write(int ცვლადი)	მთელი რიცხვის ჩაწერა
void Write(uint ცვლადი)	უნიშნო მთელი რიცხვის ჩაწერა
void Write(long ცვლადი)	გრძელი მთელი რიცხვის ჩაწერა
void Write(ulong ცვლადი)	უნიშნო გრძელი მთელი რიცხვის ჩაწერა
void Write(float ცვლადი)	float ტიპის მნიშვნელობის ჩაწერა
void Write(double ცვლადი)	double ტიპის მნიშვნელობის ჩაწერა
void Write(char ცვლადი)	სიმბოლოს ჩაწერა
void Write(char[ ] მასივი)	სიმბოლოების მასივის ჩაწერა
void Write(string ცვლადი)	სტრიქონის ჩაწერა

ცხრილი 9.9. BinaryReader ნაკადის ხშირად გამოყენებადი მეთოდები.

მეთოდი	აღწერა
bool ReadBoolean()	bool ტიპის მნიშვნელობის წაკითხვა
byte ReadByte()	byte ტიპის მნიშვნელობის წაკითხვა
sbyte ReadSByte()	sbyte ტიპის მნიშვნელობის წაკითხვა
byte[ ] ReadBytes(int ბაიტების_რაოდენობა)	მითითებული რაოდენობის ბაიტის წაკითხვა და მათი გაცემა მასივის სახით
char ReadChar()	char ტიპის მნიშვნელობის წაკითხვა
char[ ] ReadChars(int სიმბოლოების_რაოდენობა)	მითითებული რაოდენობის სიმბოლოს წაკითხვა და მათი გაცემა მასივის სახით
double ReadDouble()	double ტიპის მნიშვნელობის წაკითხვა
float ReadSingle()	float ტიპის მნიშვნელობის წაკითხვა
short ReadInt16()	short ტიპის მნიშვნელობის წაკითხვა
int ReadInt32	int ტიპის მნიშვნელობის წაკითხვა
long ReadInt64()	long ტიპის მნიშვნელობის წაკითხვა
ushort ReadUint16()	ushort ტიპის მნიშვნელობის წაკითხვა
uint ReadUint32()	uint ტიპის მნიშვნელობის წაკითხვა
ulong ReadUint64()	ulong ტიპის მნიშვნელობის წაკითხვა
string ReadString()	სტრიქონის წაკითხვა

BinaryReader კლასში განსაზღვრულია მეთოდები, რომლებიც ახდენენ C# ენის ყველა მარტივი ტიპის მონაცემების წაკითხვას. ხშირად გამოყენებადი მეთოდები მოყვანილია ცხრილში 9.9. ამავე კლასში განსაზღვრულია, აგრეთვე, Read() მეთოდის სამი ვერსია:

int Read() - კითხულობს შესასვლელი ნაკადის მომდევნო სიმბოლოს და გასცემს მის

მთელრიცხვა წარმოდგენას. ფაილის დასასრულის მიღწევისას გაიცემა -1.
int Read(byte[ ] მასივი, int პოზიცია, int ბაიტების_რაოდენობა) - ბუფერიდან კითხულობს მითითებული რაოდენობის ბაიტს დაწყებული მითითებული პოზიციიდან. გაიცემა წარმატებით წაკითხული ბაიტების რაოდენობა.
int Read(char[ ] მასივიი, int პოზიცია, int სიმბოლოების_რაოდენობა) - ბუფერიდან კითხულობს მითითებული რაოდენობის სიმბოლოს დაწყებული მითითებული პოზიციიდან. გაიცემა წარმატებით წაკითხული სიმბოლოების რაოდენობა.

ამავე ნაკადში განსაზღვრულია, აგრეთვე, Close() მეთოდი.

ქვემოთ მოყვანილ პროგრამაში გამოყენებულია BinaryReader და BinaryWriter ნაკადები. პროგრამაში ჯერ სრულდება ფაილში სხვადასხვა ტიპის მონაცემების ჩაწერა, შემდეგ კი წაკითხვა.

```
{
//      პროგრამა 9.8
//      პროგრამაში ხდება ფაილში ორობითი მონაცემების ჩაწერა-წაკითხვის
დემონსტრირება
BinaryReader file_in;
BinaryWriter file_out;
int    i1 = 10, i2;
double d1 = 1001.47, d2, d3;
bool   b1 = true, b2;
file_out = new BinaryWriter(new FileStream("file1.dat", FileMode.Create));

//      ფაილში მთელი რიცხვის ჩაწერა
file_out.Write(i1);
//      ფაილში წილადის ჩაწერა
file_out.Write(d1);
//      ფაილში ლოგიკური მონაცემის ჩაწერა
file_out.Write(b1);
//      ფაილში იწერება 10.2 * 2.3 გამოსახულების გამოთვლის შედეგი
file_out.Write(10.2 * 2.3);
file_out.Close();
//      ფაილიდან წაკითხვა
file_in = new BinaryReader(new FileStream("file1.dat",FileMode.Open));
//      მთელი რიცხვის წაკითხვა ფაილიდან
i2 = file_in.ReadInt32();
label1.Text = i2.ToString();
//      წილადის წაკითხვა ფაილიდან
d2 = file_in.ReadDouble();
label2.Text = d2.ToString();
//      ლოგიკური მონაცემის წაკითხვა ფაილიდან
b2 = file_in.ReadBoolean();
label3.Text = b2.ToString();
//      წილადის წაკითხვა ფაილიდან
d3 = file_in.ReadDouble();
label4.Text = d3.ToString();
```

```
file_in.Close();
}
```

ფაილებთან პირდაპირი მიმართვა

აქამდე ვიყენებდით ფაილებთან მიმდევრობით მიმართვას. ამ დროს მიმდევრობით სრულდება ფაილში მონაცემების ჩაწერა და ფაილიდან მონაცემების წაკითხვა. ფაილთან შესაძლებელია, აგრეთვე, პირდაპირი *(ნებისმიერი)* მიმართვაც. ამისათვის, გამოიყენება საფაილო მიმთითებელი. იგი არის ფაილის მიმდინარე ელემენტის პოზიციის ნომერი. საფაილო მიმთითებლის მნიშვნელობის შესაცვლელად გამოიყენება Seek() მეთოდი. მისი სინტაქსია:

long Seek(long *პოზიციის_ნომერი,* SeekOrigin *გადათვლის_დასაწყისი)*

პოზიციის_ნომერი პარამეტრი მიუთითებს საფაილო მიმთითებლის ახალ პოზიციას (ბაიტებში) გადათვლილს გადათვლის_დასაწყისი პარამეტრით განსაზღვრული ადგილიდან. გადათვლის_დასაწყისი პარამეტრი იღებს SeekOrigin ჩამონათვალში განსაზღვრული მნიშვნელობებიდან ერთ-ერთს (ცხრილი 9.10).

ცხრილი 9.10. გადათვლის_დასაწყისი პარამეტრის მნიშვნელობები.

მნიშვნელობა	აღწერა
Begin	გადათვლა ფაილის დასაწყისიდან
Current	გადათვლა მიმდინარე პოზიციიდან
End	გადათვლა ფაილის დასასრულიდან

Seek() მეთოდის გამოძახების შემდეგ წაკითხვის ან ჩაწერის ოპერაციები სრულდება საფაილო მიმთითებლის მიერ განსაზღვრული ახალი პოზიციიდან.

ქვემოთ მოყვანილ პროგრამაში, ხდება შეტანა-გამოტანის ოპერაციის დემონსტრირება ფაილთან ნებისმიერი მიმართვით. ფაილში ჯერ ჩაიწერება ანბანის ასოები, შემდეგ კი ხდება მათი სხვადასხვა მიმდევრობით წაკითხვა.

```
{
//      პროგრამა 9.9
//      პროგრამაში ხდება ფაილში ჩაწერა-წაკითხვის ოპერაციების შესრულება
პირდაპირი მიმართვით
FileStream file;
file = new FileStream("file.dat",FileMode.Create);
char simbolo;
label1.Text = "";
//      ასოების ჩაწერა ანბანის მიხედვით
for ( int i = 0; i < 26; i++ )
{
        file.WriteByte((byte)('a' + i));
}
//      ასოების წაკითხვა ფაილიდან
file.Seek(0, SeekOrigin.Begin);                         // პირველი ბაიტის არჩევა
```

```
simbolo = (char) file.ReadByte();                    //    პირველი ბაიტის წაკითხვა
label1.Text += "პირველი მნიშვნელობა    " + simbolo + '\n';
file.Seek(4, SeekOrigin.Begin);                      //    მეხუთე ბაიტის არჩევა
simbolo = (char) file.ReadByte();                    //    მეხუთე ბაიტის წაკითხვა
label1.Text += "მეხუთე მნიშვნელობა    " + simbolo + '\n';
file.Close();
}
```

ფაილის შესახებ ინფორმაციის მიღება

System.IO სახელების სივრცეში განსაზღვრულია, აგრეთვე, File და FileInfo კლასები. File კლასში განსაზღვრულია სტატიკური მეთოდები (ცხრილი 9.11), FileInfo კლასში კი - ჩვეულებრივი მეთოდები (ცხრილი 9.12). რადგან File კლასის მეთოდები სტატიკურია, ამიტომ მათთან სამუშაოდ არ არის საჭირო ობიექტის შექმნა, ხოლო FileInfo კლასის მეთოდებთან სამუშაოდ კი საჭიროა ობიექტის შექმნა. ცხრილში 9.13 მოყვანილია FileInfo კლასის თვისებები.

ცხრილი 9.11. File კლასში განსაზღვრული სტატიკური მეთოდები

მეთოდი	აღწერა
AppendText()	არსებულ ფაილს ტექსტს უმატებს
Copy()	ახდენს ფაილის გადაწერას
Create()	ქმნის ფაილს
CreateText()	ქმნის ფაილს და ხსნის მას ტექსტის ჩასაწერად
Delete()	შლის ფაილს
Exists()	ამოწმებს ფაილის არსებობას
GetAttributes()	გასცემს ფაილის ატრიბუტებს
GetCreationTime()	გასცემს ფაილის შექმნის თარიღს
GetLastAccessTime()	გასცემს ფაილთან უკანასკნელი მიმართვის თარიღს
GetLastWriteTime()	გასცემს ფაილში უკანასკნელი ჩაწერის თარიღს
Move()	ახდენს ფაილის გადაადგილებას
Open()	ხსნის ფაილს
OpenRead()	ხსნის ფაილს წაკითხვისათვის
OpenText()	ხსნის ფაილს ტექსტის წაკითხვისათვის
OpenWrite()	ხსნის ფაილს ჩაწერისათვის
SetAttributes()	აყენებს ფაილის ატრიბუტებს
SetCreationTime()	აყენებს ფაილის შექმნის თარიღს
SetLastAccessTime()	აყენებს ფაილთან უკანასკნელი მიმართვის თარიღს
SetLastWriteTime()	აყენებს ფაილში უკანასკნელი ჩაწერის თარიღს

როგორც ცხრილებიდან ჩანს, თითოეული კლასი იძლევა ფაილის შექმნის, წაშლის, გადატანის, გახსნის და ა.შ. საშუალებას. ბუნებრივია, იბადება კითხვა - თუ რომელი კლასი უნდა გამოვიყენოთ ფაილებთან სამუშაოდ. ასეთ დროს უნდა გავითვალისწინოთ რამდენიმე მოსაზრება. ჯერ ერთი, FileInfo კლასში განსაზღვრულია ისეთი ოპერაციები, რომლებიც არ არის განსაზღვრული File კლასში. მეორე, File კლასი

უფრო სწრაფად მუშაობს ფაილებზე ერთი ოპერაციის შესრულებისას, რადგან დრო არ იხარჯება ობიექტის შექმნაზე. FileInfo კლასი უფრო სწრაფად მუშაობს ფაილზე რამდენიმე ოპერაციის შესრულებისას, რადგან ფაილთან მიმართვის უფლებები მოწმდება ერთხელ მისი შექმნის დროს, და არა მასთან ყოველი მიმართვისას.

ცხრილი 9.12. FileInfo კლასში განსაზღვრული მეთოდები

მეთოდი	აღწერა
AppendText()	არსებულ ფაილს ტექსტს უმატებს
CopyTo()	ახდენს ფაილის გადაწერას
Create()	ქმნის ფაილს
CreateText()	ქმნის ტექსტურ ფაილს
Delete()	შლის ფაილს
MoveTo()	ახდენს ფაილის გადატანას
Open()	ხსნის ფაილს
OpenText()	ხსნის ფაილს ტექსტის წასაშლელად
OpenWrite()	ხსნის ფაილს ჩაწერისათვის

ცხრილი 9.13. FileInfo კლასში განსაზღვრული თვისებები

თვისება	ტიპი	აღწერა
Attributes	FileAttributes	გასცემს ან აყენებს ფაილის ატრიბუტებს
CreationTime	DateTime	გასცემს ან აყენებს ფაილის შექმნის თარიღს
Directory	DirectoryInfo	გასცემს კატალოგს, რომელშიც ფაილია მოთავსებული
DirectoryName	string	გასცემს კატალოგის სახელს, რომელშიც ფაილია მოთავსებული
Exists	bool	თუ ფაილი არსებობს, მაშინ გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი false მნიშვნელობას
Extension	string	გასცემს ფაილის გაფართოებას
FullName	string	გასცემს ფაილისკენ გზას და ფაილის სახელს
LastAccessTime	DateTime	გასცემს ან აყენებს ფაილთან უკანასკნელი მიმართვის თარიღს
LastWriteTime	DateTime	გასცემს ან აყენებს ფაილში უკანასკნელი ჩაწერის თარიღს
Length	long	გასცემს ფაილის ზომას
Name	string	გასცემს ფაილის სახელს

განვიხილოთ მაგალითები. მოყვანილ პროგრამაში ხდება File კლასის AppendText მეთოდთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 9.10
//      პროგრამაში ხდება File კლასის AppendText() მეთოდთან მუშაობის
დემონსტრირება
//      ფაილისათვის ხდება სტრიქონების დამატება
```

```
string path = "d_file.txt";
using ( StreamWriter faili = File.AppendText(path) )
{
        faili.WriteLine("რომან");
        faili.WriteLine("სამხარაძე");
        faili.WriteLine("C# დაპროგრამების ენაა");
        faili.Close();
}
}
```

მოყვანილ პროგრამაში ხდება File კლასის OpenText მეთოდთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 9.11
//      პროგრამაში ხდება File კლასის OpenText() მეთოდთან მუშაობის დემონსტრირება
//      ფაილის გახსნა წაკითხვის მიზნით
using ( StreamReader faili = File.OpenText("file1.txt") )
{
        string striqoni = "";
        while ( ( striqoni = faili.ReadLine() ) != null )
        {
                label1.Text += striqoni + “   “;
        }
}
}
```

მოყვანილ პროგრამაში ხდება File კლასის CreateText და Exists მეთოდთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 9.12
//      პროგრამაში ხდება File კლასის CreateText() მეთოდთან მუშაობის
დემონსტრირება
string path = "file1.txt";
//      მოწმდება ფაილის არსებობა
if ( !File.Exists(path) )
{
//      ფაილის შექმნა მასში სტრიქონების ჩაწერის მიზნით
using ( StreamWriter faili = File.CreateText(path) )
{
        faili.WriteLine("ანა ");
        faili.WriteLine("და ");
        faili.WriteLine("საბა ");
        faili.WriteLine("სამხარაძეები");
}
}
}
```

მოყვანილ პროგრამაში ხდება File კლასის Delete და Copy მეთოდებთან მუშაობის

დემონსტრირება.

```
{
//      პროგრამა 9.13
//      პროგრამაში ხდება File კლასის Delete() და Copy() მეთოდთან მუშაობის
დემონსტრირება
string path1 = "file1.txt";
string path2 = "file1.tmp";

//      file1.txt ფაილის შექმნა
using ( FileStream fs = File.Open(path1, FileMode.Open) ) { }

//      file1.tmp  ფაილის წაშლა
File.Delete(path2);

//      file1.txt ფაილის გადაწერა file1.tmp ფაილში
File.Copy(path1, path2);
label1.Text = path1 + “  ფაილი გადაიწერა  ” + path2 + “ფაილში”;
}
```

მოყვანილ პროგრამაში ხდება File კლასის GetAttributes თვისებასთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 9.14
//      პროგრამაში ხდება File კლასის GetAttributes() მეთოდთან მუშაობის
დემონსტრირება
public class F
{
public static FileAttributes Attr(FileAttributes f)
{
        return FileAttributes.Hidden;
}
}
{
System.IO.FileAttributes faili = System.IO.File.GetAttributes("file1.txt");
F obieqti = new F();
label1.Text = F.Attr(faili).ToString();
}
}
```

მოყვანილ პროგრამაში ხდება FileInfo კლასის თვისებებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 9.15
//      პროგრამაში ხდება FileInfo კლასის თვისებებთან მუშაობის დემონსტრირება
System.IO.FileInfo obieqti = new System.IO.FileInfo("file1.txt");

label1.Text += obieqti.Directory.ToString() + '\n';
```

```
label1.Text += obieqti.DirectoryName + '\n';
label1.Text += obieqti.FullName + '\n';
label1.Text += obieqti.Name + '\n';
label1.Text += obieqti.Extension + '\n';
label1.Text += obieqti.Length.ToString() + '\n';
label1.Text += obieqti.Exists.ToString() + '\n';
label1.Text += obieqti.CreationTime.ToString() + '\n';
label1.Text += obieqti.LastAccessTime.ToString() + '\n';
label1.Text += obieqti.LastWriteTime.ToString() + '\n';
}
```

კატალოგებთან მუშაობა

System.IO სახელების სივრცეში განსაზღვრულია, აგრეთვე, Directory და DirectoryInfo კლასები. File კლასის ანალოგიურად Directory კლასში განსაზღვრულია სტატიკური მეთოდები (ცხრილი 9.14), DirectoryInfo კლასში კი - ჩვეულებრივი მეთოდები (ცხრილი 9.15). რადგან Directory კლასის მეთოდები სტატიკურია, ამიტომ მათთან სამუშაოდ არ არის საჭირო ობიექტის შექმნა, ხოლო DirectoryInfo კლასის მეთოდებთან სამუშაოდ კი საჭიროა ობიექტის შექმნა. ცხრილში 9.16 მოყვანილია DirectoryInfo კლასის თვისებები.

ცხრილი 9.14. Directory კლასში განსაზღვრული მეთოდები

მეთოდი	აღწერა
CreateDirectory()	ქმნის ახალ კატალოგს
Delete()	შლის კატალოგს მის შემცველობასთან ერთად
Exicts()	განსაზღვრავს არსებობს თუ არა კატალოგი
GetCreationTime()	გასცემს კატალოგის შექმნის თარიღს
GetCurrentDirectory()	გასცემს მიმდინარე კატალოგს
GetDirectories()	გასცემს მინდინარე კატალოგის ქვეკატალოგების სახელებს
GetDirectoryRoot()	მითითებული კატალოგისათვის გასცემს ძირითადი კატალოგის სახელს
GetFiles()	გასცემს მითითებულ კატალოგში მოთავსებული ყველა ფაილის სახელს
GetFileSystemEntries()	გასცემს მითითებულ კატალოგში მოთავსებული ყველა ქვეკატალოგისა და ფაილის სახელს
GetLastAccessTime()	გასცემს კატალოგთან უკანასკნელი მიმართვის თარიღს
GetLastWriteTime()	გასცემს კატალოგში უკანასკნელი ჩაწერის თარიღს
GetLogicalDrives()	გასცემს ლოგიკური დისკების სიას
GetParent()	გასცემს მშობელი კატალოგის სახელს
Move()	ახდენს კატალოგის გადატანას
SetCreationTime()	აყენებს კატალოგის შექმნის თარიღს
SetCurrentDirectory()	აყენებს მიმდინარე კატალოგს
SetLastAccessTime()	აყენებს კატალოგთან უკანასკნელი მიმართვის თარიღს
SetLastWritetTime()	აყენებს კატალოგში უკანასკნელი ჩაწერის თარიღს

მოვიყვანოთ რამდენიმე პროგრამა, რომლებშიც სრულდება კატალოგებთან მუშაობის დემონსტრირება Directory კლასის მეთოდების გამოყენებით. ქვემოთ მოყვანილ პროგრამაში ხდება ეკრანზე მითითებული კატალოგის სახელის, მასში მოთავსებული ფაილებისა და კატალოგების სახელების გაცემა, კატალოგის გახსნა, ეკრანზე დისკების სახელების გამოტანა. პროგრამაში კატალოგის სახელის შეტანა ხდება textBox1.Text კომპონენტში, მაგალითად, C:\\Windows. ამ დროს ბრჭყალების გამოყენება საჭირო არ არის.

ცხრილი 9.15. DirectoryInfo კლასში განსაზღვრული მეთოდები

მეთოდი	აღწერა
Create()	ქმნის ახალ კატალოგს
CreateSubdirectory()	ქმნის ახალ ქვეკატალოგს
Delete()	შლის კატალოგს
GetDirectories()	გასცემს მიმდინარე კატალოგის ქვეკატალოგების სიას
GetFiles()	გასცემს მიმდინარე კატალოგის ფაილების სიას
MoveTo()	ახდენს კატალოგის გადატანას

ცხრილი 9.15. DirectoryInfo კლასში განსაზღვრული თვისებები

მეთოდი	ტიპი	აღწერა
Attributes	FileAttributes	გასცემს ან აყენებს კატალოგის ატრიბუტებს
CreationTime	DateTime	გასცემს ან აყენებს კატალოგის შექმნის დროს
Exist	bool	ამოწმებს კატალოგის არსებობას
Extension	string	გასცემს კატალოგის გაფართოებას
FullName	string	გასცემს გზას კატალოგისკენ და კატალოგის სახელს
LastAccessTime	DateTime	გასცემს ან აყენებს კატალოგთან უკანასკნელი მიმართვის დროს
LastWriteTime	DateTime	გასცემს ან აყენებს კატალოგში უკანასკნელი ჩაწერის დროს
Name	string	გასცემს კატალოგის სახელს
Parent	string	გასცემს მშობელი კატალოგის სახელს
Root	string	გასცემს ძირითადი კატალოგის სახელს მითითებული კატალოგისათვის

```
{
//      პროგრამა 9.16
//      პროგრამაში ხდება მიმდინარე და მშობელი კატალოგების სახელების, აგრეთვე,
//      მითითებულ კატალოგში მოთავსებული ფაილებისა და კატალოგების სახელების
გამოტანა ეკრანზე
label1.Text = "";
//      მიმდინარე კატალოგის სახელის გაცემა
label1.Text = Directory.GetCurrentDirectory() + '\n';
//      მშობელი კატალოგის სახელის გაცემა
```

```
label1.Text = Directory.GetParent(textBox1.Text).ToString()+ '\n';
//      მითითებული კატალოგის გახსნა
Directory.SetCurrentDirectory(textBox1.Text);
//      ეკრანზე ფაილების გამოტანა
string[] a1 = Directory.GetFiles(textBox1.Text);
foreach ( string s in a )
        label2.Text += s + '\n';
//      ეკრანზე კატალოგების გამოტანა
string[] a2 = Directory.GetDirectories(textBox1.Text);
foreach ( string s in a )
        label1.Text += s + '\n';
//      ეკრანზე დისკების გამოტანა
string[] a3 = Directory.GetLogicalDrives();
foreach ( string s in a )
        label1.Text += s + '\n';
}
```

ქვემოთ მოყვანილ პროგრამაში სრულდება კატალოგის შექმნის, წაშლისა და გადატანის დემონსტრირება. Move() მეთოდში ჯერ ეთითება საწყისი კატალოგის სახელი, შემდეგ კი - საბოლოო კატალოგის სახელი. ორივე კატალოგი ერთ დისკზე უნდა იყოს მოთავსებული.

```
{
// პროგრამა 9.17
// პროგრამაში ხდება კატალოგის შექმნა, წაშლისა და გადატანის დემონსტრირება
label2.Text = "";
//      კატალოგის შექმნა
System.IO.Directory.CreateDirectory(textBox1.Text);
//      კატალოგის წაშლა. წასაშლელი კატალოგი ცარიელი უნდა იყოს
System.IO.Directory.Delete(textBox2.Text);
// კატალოგის გადატანა. Move მეთოდში ჯერ ეთითება საწყისი კატალოგი, შემდეგ კი - მიმღები.
//      მიმღები კატალოგი არ უნდა არსებობდეს
System.IO.Directory.Move(textBox3.Text, "C:\\Katalogi2");
}
```

ქვემოთ მოყვანილ პროგრამაში სრულდება კატალოგებთან მუშაობის დემონსტრირება DirectoryInfo კლასის მეთოდების გამოყენებით. MoveTo() მეთოდი ახდენს Katalogi3 კატალოგში მოთავსებული ფაილებისა და კატალოგების გადატანას Katalogi4 კატალოგში. გადატანის შემდეგ Katalogi3 კატალოგი იშლება.

```
{
// პროგრამა 9.18
// პროგრამაში ხდება ეკრანზე მითითებული კატალოგის სახელისა და
ატრიბუტების გამოტანა,
//      აგრეთვე, ერთი კატალოგის შემცველობის გადატანა მეორე კატალოგში
label1.Text = "";
System.IO.DirectoryInfo obj1 = new DirectoryInfo("C:\\Zprint");
label1.Text += obj1.Name + "\n";
label1.Text += obj1.Attributes.ToString() + "\n";
```

```
//      Katalogi3 კატალოგის შემცველობის გადატანა Katalogi4 კატალოგში
obj1.MoveTo("C:\\Katalogi4");
}
```

ფაილების არჩევა

ამ თავში განხილულ პროგრამებთან მუშაობისას ფაილის სახელის მითითებისათვის საჭიროა პროგრამის კოდის შეცვლა, რაც ბუნებრივია, გარკვეულ წილად ართულებს სხვადასხვა ფაილთან მუშაობას. ამ პრობლემის გადასაწყვეტად გამოიყენება OpenFileDialog კლასი, რომელიც განსაზღვრულია System.Windows.Forms სახელების სივრცეში. მისი ShowDialog() მეთოდი გამოიყენება ფაილის გახსნის სტანდარტული დიალოგური ფანჯრის გამოსატანად ეკრანზე. ამ მეთოდთან მუშაობის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

```
{
//      პროგრამა 9.19
//      პროგრამაში ხდება Open ფანჯრის გახსნის დემონსტრირება
label1.Text = "";
System.Windows.Forms.OpenFileDialog FailisGaxsna = new
                                                System.Windows.Forms.OpenFileDialog();
if ( FailisGaxsna.ShowDialog() == DialogResult.OK )
{
        FileInfo faili = new FileInfo(FailisGaxsna.FileName);
        label1.Text += faili.FullName + '\n';
        label1.Text += faili.Length.ToString() + '\n';
        label1.Text += faili.LastAccessTime.ToString() + '\n';
        label1.Text += faili.DirectoryName + '\n';
}
}
```

ქვემოთ მოყვანილ პროგრამაში ხდება ფაილიდან მონაცემებისა წაკითხვის დემონსტრირება Open დიალოგური ფანჯრის გამოყენებით. FailisGaxsna ობიექტის ShowDialog() მეთოდი ხსნის Open დიალოგურ ფანჯრას. მოვნიშნავთ საჭირო ფაილს და ვაჭერთ Open კლავიშს. მონიშნული ფაილის სახელი მიენიჭება FailisGaxsna ობიექტის FileName თვისებას.

```
{
//      პროგრამა 9.20
//      პროგრამაში ხდება ფაილიდან წაკითხვის დემონსტრირება Open დიალოგური
ფანჯრის საშუალებით
FileStream file_in;
string striqoni;
OpenFileDialog FailisGaxsna = new OpenFileDialog();
if ( FailisGaxsna.ShowDialog() == DialogResult.OK )
{
        file_in = new FileStream(FailisGaxsna.FileName, FileMode.Open );
        StreamReader file_stream_in = new StreamReader(file_in);
```

```
        while ( ( striqoni = file_stream_in.ReadLine() )  != null )
        {
                label1.Text += striqoni + '\n';
        }
        file_stream_in.Close();
}
}
```

ქვემოთ მოყვანილ პროგრამაში ხდება ფაილში მონაცემების ჩაწერის დემონსტრირება Save დიალოგური ფანჯრის გამოყენებით.

```
{
//      პროგრამა 9.21
//      პროგრამაში ხდება ფაილში ჩაწერის დემონსტრირება Save დიალოგური ფანჯრის საშუალებით
string striqoni;
FileStream file_out;
SaveFileDialog FailisGaxsna = new SaveFileDialog();
if ( FailisGaxsna.ShowDialog() == DialogResult.OK )
{
        file_out = new FileStream(FailisGaxsna.FileName, FileMode.Create);
        StreamWriter file_stream_out = new StreamWriter(file_out);
        striqoni = textBox1.Text;
        //      ფაილში striqoni სტრიქონის ჩაწერა
        file_stream_out.Write(striqoni);
        file_stream_out.Close();
}
}
```

თავი 10. განსაკუთრებული სიტუაციები

განსაკუთრებული სიტუაციების დამუშავების საფუძვლები

განსაკუთრებული სიტუაცია არის სიტუაცია, რომელიც გამოწვეულია პროგრამის შესრულების დროს აღძრული შეცდომის მიერ. ასეთი სიტუაციებია: ნულზე გაყოფა, გადავსება, მასივის ინდექსის გასვლა დიაპაზონის გარეთ, არარსებული ფაილის გახსნის მცდელობა და ა.შ. ეს შეცდომები იწვევენ პროგრამის შესრულების ავარიულად დამთავრებას. C# ენაში არსებობს განსაკუთრებული სიტუაციების დამუშავების საშუალებები, რომელთა გამოყენებით შესაძლებელია ასეთი შეცდომების დამუშავება ისე, რომ არ შეფერხდეს პროგრამის შესრულება.

C# ენაში განსაკუთრებული სიტუაციები წარმოდგენილია კლასებით. განსაკუთრებული სიტუაციების ყველა კლასი მემკვიდრეობით არის მიღებული განსაკუთრებული სიტუაციის Exception კლასისაგან, რომელიც წარმოადგენს System სახელების სივრცის ნაწილს. C# ენაში განსაზღვრულია განსაკუთრებული სიტუაციების ორი კატეგორია: გენერირებული C# ენის შესრულების გარემოს (CLR) მიერ და გენერირებული პროგრამა-დანართების (Application) მიერ. პირველი მათგანი აღწერილია SystemException კლაში, მეორე კი - ApplicationException კლასში.

System სახელების სივრცეში განსაზღვრულია ბევრი სტანდარტული ჩადგმული განსაკუთრებული სიტუაცია. ცხრილში 10.1 მოყვანილია ხშირად აღძრული განსაკუთრებული სიტუაციები.

განსაკუთრებული სიტუაციების დამუშავება სრულდება ოთხი საკვანძო სიტყვის გამოყენებით: try, catch, throw და finally. ისინი, ხშირ შემთხვევაში, ერთობლივად გამოიყენება. try ბლოკში მოთავსებულია პროგრამის ის ოპერატორები, რომელთა შესრულებასაც თვალყური უნდა ვადევნოთ. გენერირებულ განსაკუთრებულ სიტუაციას იჭერს და ამუშავებს catch ბლოკი. throw სიტყვა იწვევს განსაკუთრებული სიტუაციის ხელოვნურად გენერირებას. finally ბლოკში მოთავსებულია კოდი, რომელიც ყოველთვის შესრულდება try ბლოკიდან გამოსვლისას.

ცხრილი 10.1. ხშირად აღძრული განსაკუთრებული სიტუაციები.

განსაკუთრებული სიტუაცია	მნიშვნელობა
ArrayTypeMismatchException	მნიშვნელობის ტიპი შეუთავსებია მასივის ტიპთან
Divide ByZeroException	ნულზე გაყოფის შეცდომა
IndexOutOfRangeException	მასივის ინდექსი გადის დიაპაზონის გარეთ
InvalidCastException	არაკორექტული გარდაქმნა პროგრამის შესრულების პროცესში
OutOfMemoryException	new ოპერატორის გამოძახება იყო არაკო-რექტული მეხსიერების უკმარისობის გამო
OverflowException	გადავსება არითმეტიკული ოპერაციის შესრულების დროს
StackOverflowException	სტეკის გადავსება

try და catch ბლოკები

განსაკუთრებული სიტუაციების დამუშავება ეფუძნება try და catch ბლოკების გამოყენებას. ეს ბლოკები ერთობლივად მუშაობენ. მათი სინტაქსია:

```
try
{
//      კოდი, რომლისთვისაც სრულდება შეცდომების მონიტორინგი
}
catch ( ტიპი_1 ობიექტი )
{
//      ExcepType1 განსაკუთრებული სიტუაციის დამუშავება
}
catch ( ტიპი_2 ობიექტი )
{
//      ExcepType2 განსაკუთრებული სიტუაციის დამუშავება
}       . . .
```

როგორც სინტაქსიდან ჩანს, შესამოწმებელი კოდი მოთავსებული უნდა იყოს ფიგურულ ფრჩხილებში. ტიპი პარამეტრები განსაზღვრავენ დასამუშავებელი განსაკუთრებული სიტუაციების ტიპებს. როცა catch ოპერატორი იჭერს განსაკუთრებული სიტუაციას, მაშინ შესაბამისი ობიექტი იღებს მის მნიშვნელობას. თუ განსაკუთრებული სიტუაციის დამამუშავებელი არ მიმართავს ობიექტს, რაც პრაქტიკულად ხშირად ხდება, მაშინ არ არის აუცილებელი მისი მითითება.

განსაკუთრებული სიტუაციის გენერირების შემდეგ მისი დაჭერა ხდება შესაბამისი catch ოპერატორის მიერ, რომელიც ახდენს ამ განსაკუთრებული სიტუაციის დამუშავებას. try ბლოკთან შეიძლება დაკავშირებული იყოს რამდენიმე catch ოპერატორი. განსაკუთრებული სიტუაციის ტიპი განსაზღვრულია catch ოპერატორში. თუ გენერირებული განსაკუთრებული სიტუაცია შეესაბამება catch ოპერატორში მითითებულ ტიპს, მაშინ იგი დაიჭერს ამ განსაკუთრებულ სიტუაციას და შესრულდება ამ catch ოპერატორის კოდი. დანარჩენი catch ბლოკები იგნორირდება.

იმ შემთხვევაში თუ განსაკუთრებული სიტუაციის გენერირება არ ხდება, მაშინ try ბლოკის ოპერატორების მუშაობა ჩვეულებრივი რიგითობით მთავრდება და მისი შესაბამისი catch ოპერატორები არ შესრულდება. პროგრამის შესრულება გაგრძელდება უკანასკნელი catch ოპერატორის შემდეგ მოთავსებული ოპერატორიდან. ამრიგად, catch ოპერატორები გამოიძახება მხოლოდ განსაკუთრებული სიტუაციების გენერირების შემთხვევაში.

როგორც ცნობილია, როცა ხდება ინდექსთან მიმართვა, რომელიც გადის მასივის საზღვრებს გარეთ, აღიძვრება შეცდომა. ამ დროს გენერირდება განსაკუთრებული სიტუაცია. მოყვანილი პროგრამა ახდენს ამ განსაკუთრებული სიტუაციის გენერირებისა და დაჭერის დემონსტრირებას.

```
{
//      პროგრამა 10.1
//      პროგრამაში ხდება მასივის საზღვრებს გარეთ ინდექსის გასვლის შეცდომის
დამუშავება
int[] masivi = new int[5];
```

```
try
{
label2.Text = "ეს სტრიქონი გამოჩნდება განსაკუთრებული სიტუაციის გენერირებამდე";
masivi[6] = 20;                    // ინდექსის მნიშვნელობა გადის დიაპაზონის გარეთ
label3.Text = "ეს სტრიქონი არ გამოჩნდება";
}
//      განსაკუთრებული სიტუაციის დაჭერა
catch (IndexOutOfRangeException)
{
label1.Text = "ინდექსი დიაპაზონის გარეთაა ";
}
label2.Text = "ეს ოპერატორი შესრულდება";
}
```

ამ პროგრამაში განსაკუთრებული სიტუაციის გენერირების შემთხვევაში მისი დაჭერა ხდება catch ოპერატორის მიერ. ამ მომენტიდან დაწყებული მუშაობას იწყებს catch ბლოკი, ხოლო try ბლოკი კი მუშაობას ამთავრებს. შედეგად,

```
label3.Text = "ეს სტრიქონი არ გამოჩნდება";
```

ოპერატორი, რომელიც მოსდევს ინდექსის არასწორად გამოყენების ოპერატორს, არ შესრულდება. catch ბლოკის შესრულების შემდეგ მართვა გადაეცემა ოპერატორს, რომელიც მოსდევს ამ catch ბლოკს. ამრიგად, მიუხედავად აღძრული შეცდომისა, პროგრამა აგრძელებს მუშაობას და ავარიულად არ მთავრდება.

catch ოპერატორში შეიძლება არ იყოს მითითებული არც ერთი პარამეტრი. პარამეტრი მოითხოვება მხოლოდ მაშინ, როცა აუცილებელია განსაკუთრებული სიტუაციის ობიექტთან მიმართვა. რიგ შემთხვევებში განსაკუთრებული სიტუაციის ობიექტის მნიშვნელობა შეიძლება გამოყენებული იქნას განსაკუთრებული სიტუაციების დამამუშავებლის მიერ შეცდომის შესახებ დამატებითი ინფორმაციის მიღების მიზნით.

თუ არ მოხდა განსაკუთრებული სიტუაციის გენერირება, მაშინ catch ოპერატორი არ გამოიძახება და მართვა გადაეცემა catch ოპერატორის შემდეგ მყოფ ოპერატორს.

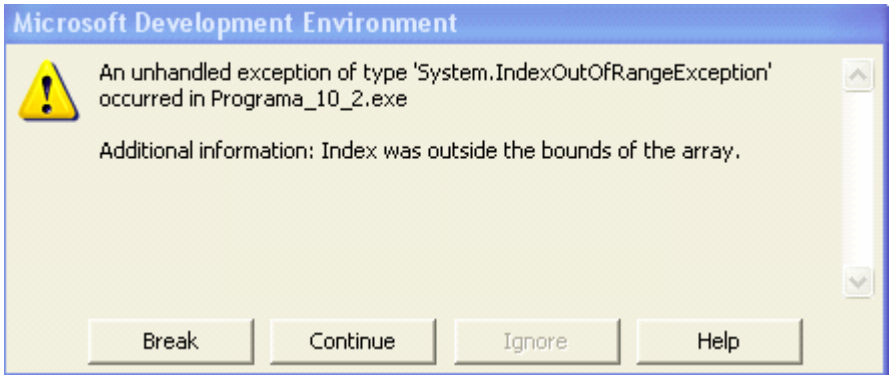
სტანდარტული განსაკუთრებული სიტუაციის დაჭერა და დამუშავება თავიდან აგვაცილებს პროგრამის ავარიულად დამთავრებას. თუ პროგრამა არ ახდენს განსაკუთრებული სიტუაციის დაჭერას, მაშინ განსაკუთრებული სიტუაციას დაიჭერს C#-ის შესრულების სისტემა. პრობლემა, რომელსაც ამ შემთხვევაში აქვს ადგილი, იმაში მდგომარეობს, რომ შესრულების სისტემას ეკრანზე გამოაქვს შეტყობინება შეცდომის შესახებ და ავარიულად ამთავრებს პროგრამის შესრულებას. ქვემოთ მოყვანილ პროგრამაში განსაკუთრებული სიტუაციის დაჭერა პროგრამის მიერ არ ხდება.

```
{
//      პროგრამა 10.2
//      პროგრამაში არ ხდება განსაკუთრებული სიტუაციის დაჭერა
int[] masivi = new int[5];

masivi[0] = 5;   //   ეს ოპერატორი შესრულდება განსაკუთრებული სიტუაციის გენერირებამდე
masivi[6] = 20;          //        ინდექსის მნიშვნელობა გადის დიაპაზონის გარეთ
}
```

ამ შემთხვევაში პროგრამის შესრულება ავარიულად შეწყდება და გამოჩნდება

შეტყობინება შეცდომის შესახებ:



ნახ.10.1

მართალია, ეს შეტყობინება საკმაოდ სასარგებლოა პროგრამის გამართვისას, მაგრამ იგი არ უნდა გამოჩნდეს პროგრამის ექსპლუატაციის დროს. ამიტომ, პროფესიულ დონეზე შედგენილმა პროგრამამ თვითონ უნდა დაამუშაოს განსაკუთრებული სიტუაციები.

როგორც ვიცით, განსაკუთრებული სიტუაციის ტიპი უნდა შეესაბამებოდეს catch ოპერატორში მითითებულ ტიპს. წინააღმდეგ შემთხვევაში, განსაკუთრებული სიტუაცია არ იქნება დაჭერილი. ქვემოთ მოყვანილ პროგრამაში ხდება ნულზე გაყოფით გამოწვეული განსაკუთრებული სიტუაციის (DivideByZeroException) დაჭერა. როგორც კი ინდექსის მნიშვნელობა გავა დიაპაზონის გარეთ, მოხდება IndexOutOfRangeException განსაკუთრებული სიტუაციის გენერირება, რომელსაც ეს catch ოპერატორი ვერ დაიჭერს. შედეგად, პროგრამა ავარიულად დაამთავრებს მუშაობას.

```
{
//      პროგრამა 10.3
//      პროგრამაში ხდება ნულზე გაყოფით გამოწვეული შეცდომის დაჭერა და
//      არ ხდება დიაპაზონის გარეთ ინდექსის გასვლის შეცდომის დაჭერა
int ricxvi;
int[] masivi = new int[5];

try
{
label1.Text = "ეს სტრიქონი გამოჩნდება განსაკუთრებული სიტუაციის გენერირებამდე";
masivi[6] = 20;                 //      ინდექსის მნიშვნელობა გადის დიაპაზონის გარეთ
ricxvi = 10;                    //      ეს ოპერატორი არ შესრულდება
}
//      განსაკუთრებული სიტუაციის დაჭერა
catch ( DivideByZeroException )
{
label2.Text = "ინდექსი დიაპაზონის გარეთაა";
}
}
```

განსაკუთრებული სიტუაციების დამუშავების ერთ-ერთი მთავარი უპირატესობა ის არის, რომ პროგრამას შეუძლია მოახდინოს რეაგირება შეცდომაზე და შემდეგ გააგრძელოს შესრულება. ქვემოთ მოყვანილ პროგრამაში ერთი მასივის ელემენტები

იყოფა მეორე მასივის ელემენტებზე. ნულზე გაყოფის შემთხვევაში გენერირდება DivideByZeroException განსაკუთრებული სიტუაცია. პროგრამა ახდენს მის დამუშავებას გასცემს რა შესაბამის შეტყობინებას. ამრიგად, ნულზე გაყოფის შეცდომა არ გამოიწვევს პროგრამის ავარიულ გაჩერებას. ამის ნაცვლად, label კომპონენტში გამოჩნდება შესაბამისი შეტყობინება.

```
{
//      პროგრამა 10.4
//      პროგრამაში ხდება ნულზე გაყოფის შეცდომის დაჭერა მასივების გაყოფის დროს
int[] masivi1 = { 1, 3, 5, 7, 9 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[5];

label1.Text = "";
for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text = label2.Text + masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
catch ( DivideByZeroException )
{
label2.Text = "ადგილი აქვს ნულზე გაყოფას!";
}
}
```

ამ პროგრამაში, try ბლოკი მოთავსებულია ციკლის ტანში, ამიტომ, შესაძლებელი ხდება გამეორებადი შეცდომების დამუშავება.

try და catch ბლოკების გამოყენების მაგალითები

მოყვანილ პროგრამაში სრულდება ფაილის გახსნისა და ფაილიდან წაკითხვის ოპერაციების შემოწმება.

```
{
//      პროგრამა 10.5
//      პროგრამაში ხდება ფაილის გახსნისა და ფაილიდან წაკითხვის ოპერაციების
მონიტორინგი
int i;
FileStream file1;
//      try  ბლოკი ამოწმებს ფაილის გახსნის ოპერაციას
try
{
file1 = new FileStream("filetext.txt", FileMode.Open);
}
```

```
catch(FileNotFoundException arg1)
{
MessageBox.Show(arg1.Message);
return;
}
//        მონაცემების წაკითხვა ფაილიდან მანამ, სანამ არ შეგვხვდება EOF სიმბოლო
do
{
//        try  ბლოკი ამოწმებს ფაილიდან წაკითხვის ოპერაციას
try
{
i = file1.ReadByte();
}
catch(IOException arg1)
{
MessageBox.Show(arg1.Message);
return;
}
if ( i != -1 ) label1.Text += (char)i;
}
while ( i != -1 );   //  თუ i = -1, მაშინ მიღწეულია ფაილის დასასრული
file1.Close();
}
```

მოყვანილ პროგრამაში სრულდება ფაილის შექმნისა და ფაილში ჩაწერის ოპერაციების შემოწმება.

```
{
//        პროგრამა 10.6
//        პროგრამაში ხდება ფაილის შექმნისა და ფაილში ჩაწერის ოპერაციების
მონიტორინგი
BinaryReader file_in;
BinaryWriter file_out;
int    i1 = 10, i2;
double d1 = 1001.47, d2, d3;
bool   b1 = true, b2;
//        try  ბლოკი ამოწმებს ფაილის შექმნის ოპერაციას
try
{
file_out = new BinaryWriter(new FileStream("file1.dat", FileMode.Create));
}
catch(IOException arg1)
{
MessageBox.Show(arg1.Message + "\n შეუძლებელია ფაილის შექმნა");
return;
}
//        try  ბლოკი ამოწმებს ფაილში ჩაწერის ოპერაციებს
```

```
try
{
//        ფაილში მთელი რიცხვის ჩაწერა
file_out.Write(i1);
//        ფაილში წილადის ჩაწერა
file_out.Write(d1);
//        ფაილში ლოგიკური მონაცემის ჩაწერა
file_out.Write(b1);
//        ფაილში იწერება 10.2 * 2.3 გამოსახულების გამოთვლის შედეგი
file_out.Write(10.2 * 2.3);
}
catch(IOException arg1)
{
MessageBox.Show(arg1.Message + "\n ჩაწერის შეცდომა");
return;
}
file_out.Close();

//        ფაილიდან წაკითხვა
try
{
file_in = new BinaryReader(new FileStream("file1.dat",FileMode.Open));
}
catch(IOException arg1)
{
MessageBox.Show(arg1.Message + "\n შეუძლებელია ფაილის გახსნა");
return;
}
try
{
//        მთელი რიცხვის წაკითხვა ფაილიდან
i2 = file_in.ReadInt32();
label1.Text = i2.ToString();
//        წილადის წაკითხვა ფაილიდან
d2 = file_in.ReadDouble();
label2.Text = d2.ToString();
//        ლოგიკური მონაცემის წაკითხვა ფაილიდან
b2 = file_in.ReadBoolean();
label3.Text = b2.ToString();
//        წილადის წაკითხვა ფაილიდან
d3 = file_in.ReadDouble();
label4.Text = d3.ToString();
}
catch(IOException arg1)
{
```

```
MessageBox.Show(arg1.Message + “\n წაკითხვის შეცდომა”);
return;
}
file_in.Close();
}
```

მოყვანილ პროგრამაში სრულდება ფაილის შექმნისა და ფაილში ჩაწერა-წაკითხვის ოპერაციების შემოწმება.

```
{
//      პროგრამა 10.7
//      პროგრამაში ხდება ფაილის შექმნისა და ფაილში ჩაწერა-წაკითხვის ოპერაციების მონიტორინგი
FileStream file;
//      try  ბლოკი ამოწმებს ფაილის შექმნის ოპერაციას
try
{
file = new FileStream("file.dat",FileMode.Create);
}
catch(FileNotFoundException arg1)
{
MessageBox.Show(arg1.Message);
return;
}
char simbolo;
label1.Text = "";
//      ასოების ჩაწერა ანბანის მიხედვით
for ( int i = 0; i < 26; i++ )
{
//      try  ბლოკი ამოწმებს ფაილში ჩაწერის ოპერაციას
try
{
file.WriteByte((byte)('a' + i));
}
catch(IOException arg1)
{
MessageBox.Show(arg1.Message);
return;
}
}
// try  ბლოკი ამოწმებს ფაილიდან წაკითხვის ოპერაციას
try
{
file.Seek(0, SeekOrigin.Begin);              //      პირველი ბაიტის არჩევა
simbolo = (char) file.ReadByte();            //      პირველი ბაიტის წაკითხვა
label1.Text += "პირველი მნიშვნელობა    " + simbolo + '\n';
file.Seek(4, SeekOrigin.Begin);              //      მეხუთე ბაიტის არჩევა
```

```
simbolo = (char) file.ReadByte();              //      მეხუთე ბაიტის წაკითხვა
label1.Text += "მეხუთე მნიშვნელობა    " + simbolo + '\n';
}
catch(IOException arg1)
{
MessageBox.Show(arg1.Message);
return;
}
file.Close();
}
```

რამდენიმე catch ბლოკის გამოყენება

ერთ try ბლოკთან შეიძლება დაკავშირებული იყოს რამდენიმე catch ბლოკი. მაგრამ, თითოეული catch ბლოკი იჭერს კონკრეტული ტიპის განსაკუთრებული სიტუაციას. მოყვანილ პროგრამაში ხდება ინდექსის მნიშვნელობის დიაპაზონის გარეთ გასვლისა და ნულზე გაყოფის შეცდომების დაჭერა.

```
{
//      პროგრამა 10.8
//      პროგრამაში ხდება რამდენიმე catch ოპერატორის გამოყენების დემონსტრირება
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];

label1.Text = "";
for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
//      ეს catch ოპერატორი იჭერს ნულზე გაყოფის შეცდომას
catch ( DivideByZeroException )
{
label2.Text = "ადგილი აქვს ნულზე გაყოფას!";
}
//      ეს catch ოპერატორი იჭერს მასივის ინდექსის დიაპაზონის გარეთ გასვლის შეცდომას
catch ( IndexOutOfRangeException )
{
label3.Text = "ინდექსი დიაპაზონის გარეთაა!";
}
}
```

catch ოპერატორები მოწმდება იმ მიმდევრობით, როგორც არიან პროგრამაში მითითებული. შესრულდება ის ბლოკი, რომლის catch ოპერატორში მითითებული ტიპი დაემთხვევა განსაკუთრებული სიტუაციის ტიპს. დანარჩენი catch ბლოკები არ შესრულდება.

ყველა განსაკუთრებული სიტუაციის დაჭერა

ზოგჯერ საჭიროა ყველა განსაკუთრებული სიტუაციის დაჭერა მათი ტიპის მიუხედავად. ასეთ შემთხვევაში გამოიყენება catch ოპერატორი პარამეტრების გარეშე. უპარამეტრებო catch ოპერატორი იჭერს ყველა განსაკუთრებული სიტუაციას. მოყვანილ პროგრამაში IndexOutOfRangeException და DivideByZeroException განსაკუთრებული სიტუაციის დაჭერა და დამუშავება სრულდება ერთი catch ოპერატორის მიერ.

```
{
//      პროგრამა 10.9
//      პროგრამაში ხდება ყველა განსაკუთრებული სიტუაციის დაჭერა
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];

label1.Text = "";
label2.Text = "";
for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
//      ეს catch ოპერატორი იჭერს ყველა შეცდომას
catch
{
label2.Text = "ადგილი აქვს განსაკუთრებული სიტუაციის გენერირებას!";
}
}
```

ჩადგმული try ბლოკები

შესაძლებელია try ბლოკების ერთმანეთში მოთავსება. შიდა try ბლოკში გენერირებული განსაკუთრებული სიტუაცია, რომელიც ვერ დაიჭირა ამ შიდა try ბლოკის შესაბამისმა catch ბლოკმა, ვრცელდება გარე try ბლოკზე. მოყვანილ მაგალითში გენერირებული IndexOutOfRangeException განსაკუთრებული სიტუაცია, რომელიც არ იყო დაჭერილი შიდა catch ბლოკის მიერ, დაჭერილი იქნება გარე catch ბლოკის მიერ.

```
{
//      პროგრამა 10.10
//      პროგრამაში ხდება ჩადგმული try ბლოკების გამოყენების დემონსტრირება
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];

label1.Text = ""; label2.Text = "";
try
{
for ( int ind = 0; ind < masivi1.Length; ind++ )
{
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
                    masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
catch ( DivideByZeroException )
{
label2.Text = "ადგილი აქვს ნულზე გაყოფას!";
}
}
}
catch ( IndexOutOfRangeException )
{
label3.Text = "ადგილი აქვს ინდექსის გასვლას დიაპაზონის გარეთ!";
}
}
```

პროგრამაში ნულზე გაყოფის შედეგად გენერირებული განსაკუთრებული სიტუაცია მუშავდება შიდა try ბლოკში. ამასთან, პროგრამა აგრძელებს მუშაობას. დიაპაზონის გარეთ ინდექსის გასვლისას შეცდომას ხელში იგდებს გარე try ბლოკის შესაბამისი catch ოპერატორი.

ხშირად, ჩადგმული try ბლოკები გამოიყენება სხვადასხვა სახის შეცდომების დასამუშავებლად. გარე try ბლოკი შეგვიძლია გამოვიყენოთ სერიოზული შეცდომების დასაჭერად. ხოლო შიდა try ბლოკები ნაკლებად სერიოზული შეცდომების დასამუშავებლად. შეგვიძლია, აგრეთვე, გამოვიყენოთ გარე try ბლოკი უპარამეტრებო catch ოპერატორით იმ შეცდომების დასაჭერად, რომლებიც არ მუშავდებიან შიდა try ბლოკში.

განსაკუთრებული სიტუაციის იძულებით გენერირება

წინა მაგალითებში ხდებოდა C# სისტემის მიერ ავტომატურად გენერირებული განსაკუთრებული სიტუაციის დაჭერა. მაგრამ, განსაკუთრებული სიტუაციის

გენერირება შეიძლება, აგრეთვე, throw ოპერატორის საშუალებით. მისი სინტაქსია:
throw ობიექტი;
აქ ობიექტი არის განსაკუთრებული სიტუაციის კლასის ობიექტი. მოყვანილ მაგალითში ხდება DivideByZeroException განსაკუთრებული სიტუაციის გენერირება throw ოპერატორის მიერ.

```
{
//      პროგრამა 10.11
//      პროგრამაში ხდება throw ოპერატორთან მუშაობის დემონსტრირება
int ricxvi1, ricxvi2, shedegi;
try
{
//      ეს ოპერატორები შესრულდება throw ოპერატორის წინ
ricxvi1 = Convert.ToInt32(textBox1.Text);
ricxvi2 = Convert.ToInt32(textBox2.Text);
//      განსაკუთრებული სიტუაციის გენერირება
if ( ricxvi2 == 0 ) throw new DivideByZeroException();
shedegi = ricxvi1 / ricxvi2;
}
catch ( DivideByZeroException )
{
label1.Text = "განსაკუთრებული სიტუაცია დაჭერილია";
}
//      try-catch ბლოკის დასასრული
}
```

throw ოპერატორი ოპერირებს ობიექტებზე. ამიტომ, მის გამოყენებამდე საჭიროა ობიექტის შექმნა new ოპერატორით. ჩვენს შემთხვევაში, DivideByZeroException ობიექტის შესაქმნელად გამოიყენება ავტომატური კონსტრუქტორი.

finally ბლოკი

როგორც ვიცით, განსაკუთრებული სიტუაციის გენერირება იწვევს მიმდინარე მეთოდის შესრულების ავარიულ შეწყვეტას. მაგრამ, ამ მეთოდმა შეიძლება გახსნას ფაილი ან ქსელური შეერთება, რომლებიც უნდა იყოს დახურული. ასეთ შემთხვევებში უნდა გამოვიყენოთ finally ბლოკი. იგი ეთითება try/catch მიმდევრობის ბოლოში. try/catch/finally კონსტრუქციის სინტაქსია:

```
try
{
//      try ბლოკის კოდი, რომლისთვისაც სრულდება შეცდომის მონიტორინგი
}
catch ( ტიპი_1 ობიექტი1 )
{
//      ტიპი_1 განსაკუთრებული სიტუაციის დამუშავება
}
catch ( ტიპი_2 ობიექტი2 )
```

```
{
//      ტიპი_2 განსაკუთრებული სიტუაციის დამუშავება
}
finally
{
finally ბლოკის კოდი
}
```

finally ბლოკი გამოიძახება იმისგან დამოუკიდებლად გენერირდება თუ არა განსაკუთრებული სიტუაცია. ამრიგად, არა აქვს მნიშვნელობა try ბლოკი როგორ დამთავრდება - ნორმალურად თუ აღიძვრება განსაკუთრებული სიტუაცია. finally ბლოკის კოდი ყოველთვის შესრულდება. მოყვანილ პროგრამაში ხდება finally ბლოკის გამოყენების დემონსტრირება.

// პროგრამა 10.12
// პროგრამაში ხდება finally ბლოკის გამოყენების დემონსტრირება

```
class finally1
{
public static void gen_finally(int par1)
{
int ricxvi;
int[] masivi = new int[5];

MessageBox.Show(par1.ToString() + "-ის მიღება");
try
{
switch ( par1 )
{
case 0 : ricxvi = 10 / par1;              //      ნულზე გაყოფის შეცდომა
          break;
case 1 : masivi[7] = 7;                   //      დიაპაზონის გარეთ ინდექსის გასვლის შეცდომა
          break;
case 2 : return;                          //      try ბლოკიდან გამოსვლა
}
}
catch ( DivideByZeroException )
{
MessageBox.Show("ნულზე გაყოფის შეცდომა!");
return;
}
catch ( IndexOutOfRangeException )
{
MessageBox.Show(" დიაპაზონის გარეთ ინდექსის გასვლის შეცდომა!");
return;
}
finally
{
```

```
MessageBox.Show("try ბლოკიდან გამოსვლა");
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
for ( int indeqsi = 0; indeqsi < 3; indeqsi++ )
{
finally1.gen_finally(indeqsi);
}
}
```

checked და unchecked ოპერატორები

არითმეტიკული ოპერაციების შესრულებისას შეიძლება ადგილი ჰქონდეს გადავსების შეცდომებს. მაგალითად, მოყვანილ პროგრამაში ricxvi1 და ricxvi2 მნიშვნელობების ნამრავლი აჭარბებს byte ტიპის მქონე მნიშვნელობის დასაშვებ დიაპაზონს. შედეგად, ადგილი ექნება გადავსების შეცდომას.

```
{
//      პროგრამა 10.13
byte ricxvi1, ricxvi2, shedegi;
ricxvi1 = 127;
ricxvi2 = 127;
shedegi = ( byte ) ( ricxvi1 * ricxvi2 );
label1.Text = shedegi.ToString();
}
```

C# ენაში გადავსების შემთხვევაში შესაძლებელია განსაკუთრებული სიტუაციის გენერირება checked და unchecked ოპერატორების გამოყენებით. checked ოპერატორი გამოიყენება იმ გამოსახულების მისათითებლად, რომელიც უნდა შემოწმდეს გადავსების არსებობაზე. გადავსების იგნორირებისათვის გამოიყენება unchecked ოპერატორი. ამ შემთხვევაში მოხდება შედეგის ჩამოჭრა იმ მიზნით, რომ მისი ტიპი დაემთხვეს საბოლოო გამოსახულების ტიპს. ეს კი გამოიწვევს შედეგის დამახინჯებას.

checked ოპერატორს ორი ძირითადი ფორმა აქვს. პირველი ფორმა გამოიყენება მითითებული გამოსახულების შემოწმებისათვის. მისი სინტაქსია:

checked (*გამოსახულება)*

აქ სრულდება გამოსახულების შემოწმება. თუ მისი გამოთვლისას ადგილი აქვს გადავსებას, მაშინ გენერირდება OverflowException განსაკუთრებული სიტუაცია.

მეორე ფორმა გამოიყენება ოპერატორების ბლოკის შესამოწმებლად. მისი სინტაქსია:

```
checked
{
```
შესამოწმებელი ოპერატორები
```
}
```

unchecked ოპერატორს აქვს ორი ძირითადი ფორმა. პირველი ფორმის

გამოყენებისას იგნორირდება გადავსება მითითებული გამოსახულებისათვის. მისი სინტაქსია:
unchecked *(გამოსახულება)*

აქ გამოსახულება არ შემოწმდება გადავსების არსებობაზე. თუ მისი გამოთვლისას ადგილი ექნება გადავსებას, მაშინ მოხდება შედეგის ჩამოჭრა.

მეორე ფორმა ახდენს გადავსების იგნორირებას ოპერატორების ბლოკისათვის. მისი სინტაქსია:
unchecked
{
შესამოწმებელი ოპერატორები
}

ქვემოთ მოყვანილ პროგრამაში ხდება checked და unchecked ოპერატორების გამოყენება გამოსახულების მიმართ.

```
{
//      პროგრამა 10.14
byte ricxvi1, ricxvi2, shedegi;

ricxvi1 = 127;
ricxvi2 = 127;
try
{
shedegi = unchecked( (byte) ( ricxvi1 * ricxvi2 ) );
label1.Text = shedegi.ToString();
shedegi = checked( (byte) ( ricxvi1 * ricxvi2 ) );
label2.Text = shedegi.ToString();
}
catch (OverflowException exc)      //      ეს catch ოპერატორი იჭერს გადავსების შეცდომას
{
label3.Text = exc.ToString();
}
}
```

მოყვანილ პროგრამაში სრულდება checked და unchecked ოპერატორების გამოყენება ოპერატორების ბლოკის მიმართ.

```
{
//      პროგრამა 10.15
byte ricxvi1, ricxvi2, shedegi;

ricxvi1 = ricxvi2 = 127;
try
{
unchecked
{
ricxvi1 = ricxvi2 = 127;
shedegi = unchecked( (byte) ( ricxvi1 * ricxvi2 ) );
label1.Text = shedegi.ToString();
```

```
ricxvi1 = 120;
ricxvi2 = 10;
shedegi = unchecked( ( byte ) ( ricxvi1 * ricxvi2 ) );
label2.Text = shedegi.ToString();
}
checked
{
ricxvi1 = 2;
ricxvi2 = 7;
shedegi = checked( ( byte ) ( ricxvi1 * ricxvi2 ) );
label3.Text = shedegi.ToString();
ricxvi1 = 127;
ricxvi2 = 127;
shedegi = checked( ( byte ) ( ricxvi1 * ricxvi2 ) );
label4.Text = shedegi.ToString();
}
}
catch ( OverflowException exc )     //          ეს catch ოპერატორი იჭერს გადავსების შეცდომას
{
label5.Text = exc.ToString();
}
}
```

გამართვა

.NET გარემოს შემადგენლობაში შედის პროგრამა გამმართველი. იგი საშუალებას გვაძლევს პროგრამის კოდი შევასრულოთ ბიჯობრივ რეჟიმში და ვნახოთ პროგრამის ცვლადების მნიშვნელობები. ეს, თავის მხრივ, აადვილებს პროგრამის კოდში შეცდომების პოვნას.

მანამ, სანამ დავიწყებდეთ გამმართველთან მუშაობას, შევქმნათ ახალი პროექტი. ფორმაზე მოვათავსოთ ერთი button და ერთი label კომპონენტი. button კომპონენტს მივაბათ პროგრამა, რომელიც ახდენს მასივის ელემენტების შეკრებას:

```
{
int jami = 0;
int[] masivi = new int[5] { 3, 1, 8, -4, 1 };
for (int indexi = 0; indexi < masivi.Length; indexi++)
   jami += masivi[indexi];
label1.Text = jami.ToString();
}
```

წყვეტის წერტილების შექმნა

წყვეტის წერტილი არის პროგრამის კოდის ის ადგილი (სტრიქონი), სადაც წყდება პროგრამის შესრულება. ამ სტრიქონში მოთავსებული გამოსახულება (ოპერატორი ან მეთოდი) არ შესრულდება და მართვა მომხმარებელს გადაეცემა.

წყვეტის წერტილის შესაქმნელად საჭირო გამოსახულების გასწვრივ მარცხენა ველში უნდა მოვათავსოთ კურსორი და დავაჭიროთ თაგვის მარცხენა კლავიშს. გამოჩნდება მუქი წრე. წყვეტის წერტილის შესაქმნელად შეგვიძლია, აგრეთვე, კურსორი მოვათავსოთ საჭირო გამოსახულების შიგნით და დავაჭიროთ F9 კლავიშს. პროგრამის კოდში შეგვიძლია შევქმნათ წყვეტის რამდენიმე წერტილი. ჩვენი პროგრამის კოდში კურსორი მოვათავსოთ for ოპერატორის შემცველ სტრიქონში და დავაჭიროთ F9 კლავიშს. შეიქმნება წყვეტის წერტილი (ნახ. 10.2).

წყვეტის წერტილის გასაუქმებლად უნდა დავაჭიროთ მუქ წრეს ან F9 კლავიშს. წყვეტის წერტილთან პროგრამის კოდის შესრულება წყდება. ამის შემდეგ, პროგრამის კოდი შეგვიძლია ბიჯობრივ რეჟიმში შევასრულოთ და დავაკვირდეთ ცვლადების მნიშვნელობებს.

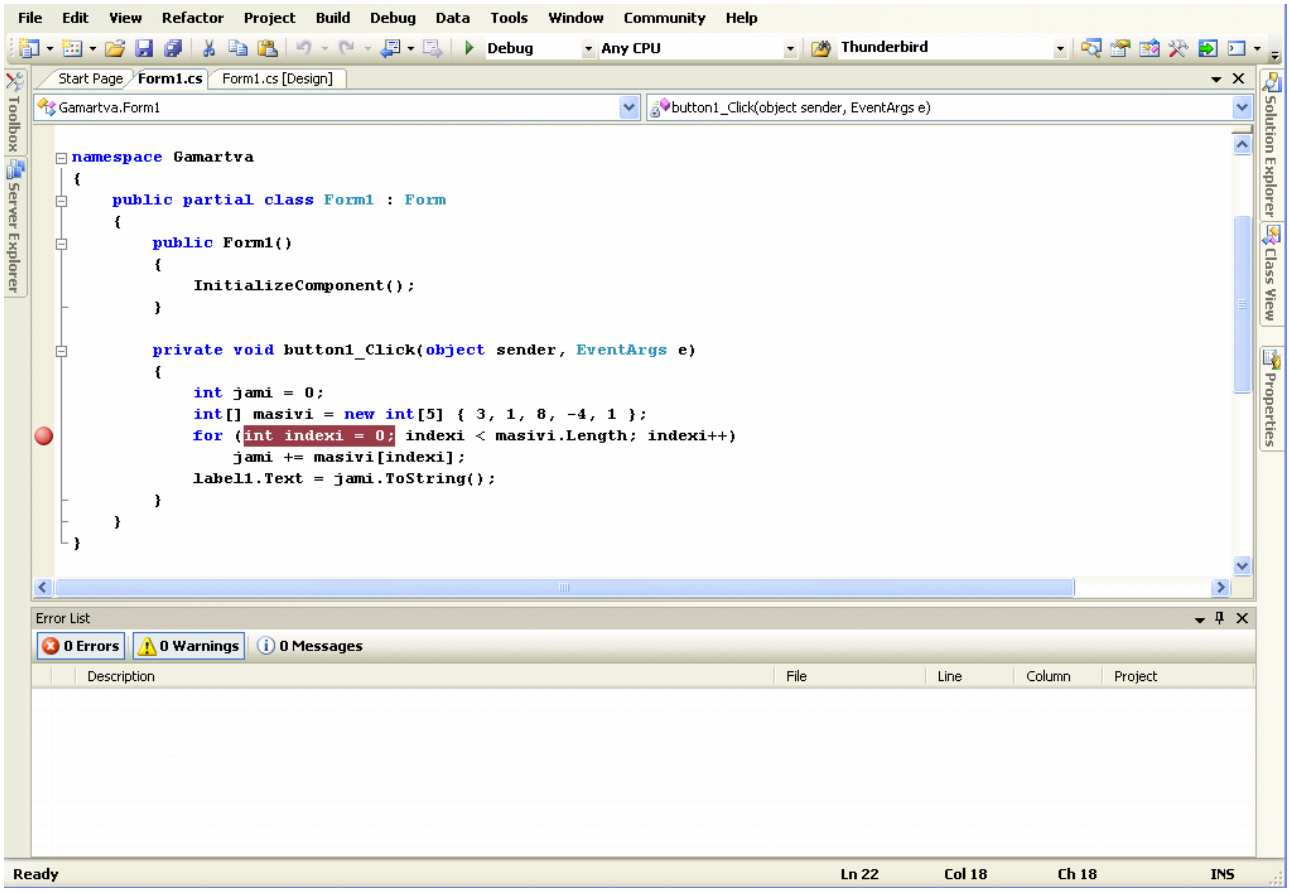
გამმართველის გაშვება ხდება F5 კლავიშით. თუ გვინდა კოდის შესრულება გამმართველის გარეშე, მაშინ უნდა დავაჭიროთ Ctrl+F5 კლავიშებს. დავაჭიროთ F5 კლავიშს. გამოჩნდება ფორმა. შემდეგ დავაჭიროთ button კლავიშს. გამმართველი კოდს წყვეტის წერტილამდე ასრულებს და მართვას მომხმარებელს გადასცემს. შესაბამისი გამოსახულების (int indexi = 0;) გასწვრივ მარცხენა ველში გამოჩნდება ისარი (ნახ. 10.3). ეს გამოსახულება ჯერ არ არის შესრულებული.

ცვლადების მნიშვნელობების ნახვა

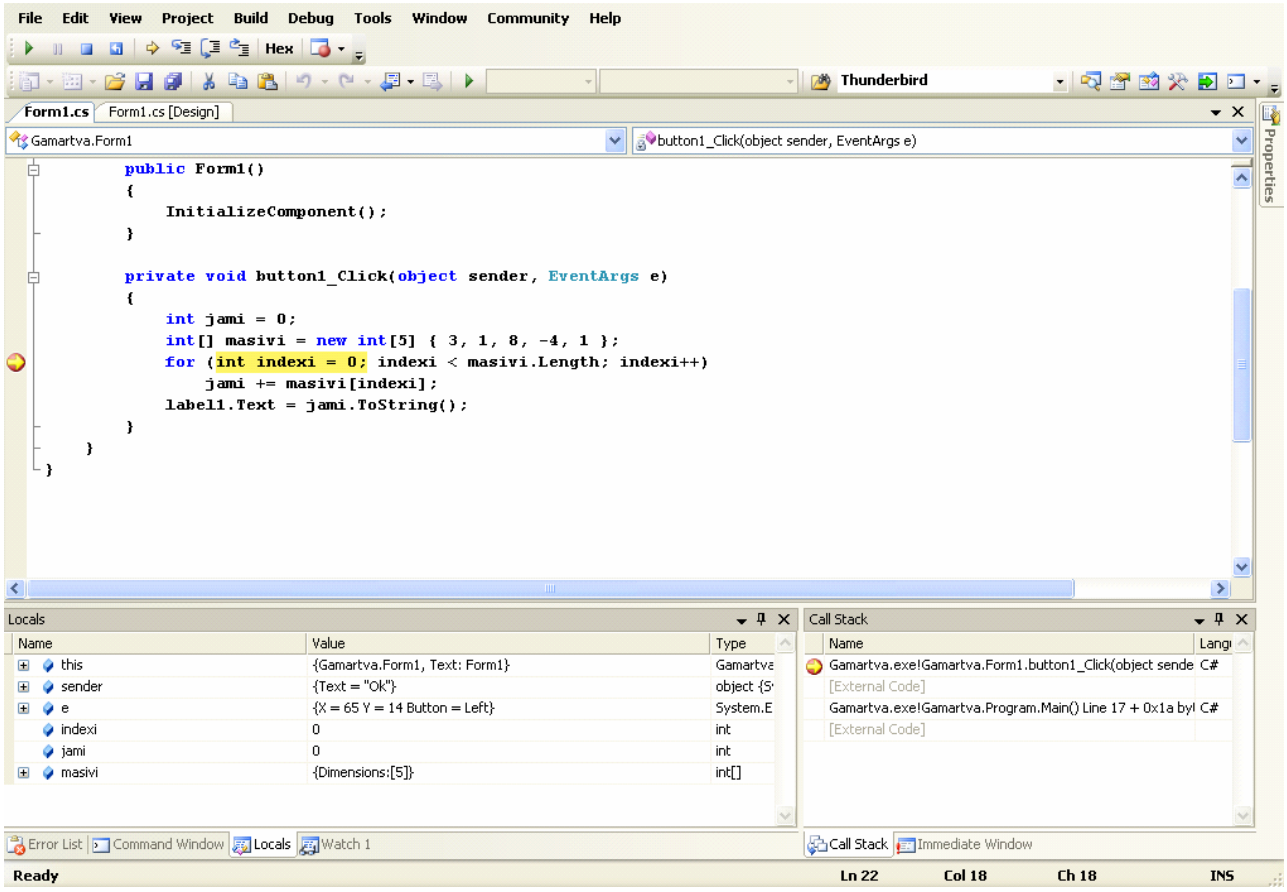
როგორც აღვნიშნეთ, წყვეტის წერტილთან გამმართველი წყვეტს კოდის შესრულებას. ამ დროს შეგვიძლია ვნახოთ ცვლადების მნიშვნელობები.

Autos ფანჯარა. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ (ნახ. 10.4). თუ ის არ ჩანს, მაშინ ვხსნით Debug მენიუს, Windows ქვემენიუს და ვასრულებთ Autos ბრძანებას (Debug→Windows→Autos). Windows ქვემენიუს ბრძანებები გამოჩნდება წყვეტის წერტილთან პროგრამის შეჩერების შემდეგ. ამ ფანჯარაში ჩანს იმ ცვლადების მნიშვნელობები, რომლებიც გამოიყენება მიმდინარე და წინა გამოსახულებებში. ამ ეტაპზე მასში ჩანს indexi, masivi და masivi.Length ცვლადების მნიშვნელობები. ცვლადის მნიშვნელობა შეგვიძლია ვნახოთ, აგრეთვე, თუ პროგრამის კოდში მის სახელზე გავაჩერებთ კურსორს. თუ Autos ფანჯარაში დავაჭერთ masivi სახელის მარცხნივ მოთავსებულ + ნიშანზე, მაშინ გამოჩნდება ამ მასივის ელემენტების მნიშვნელობები.

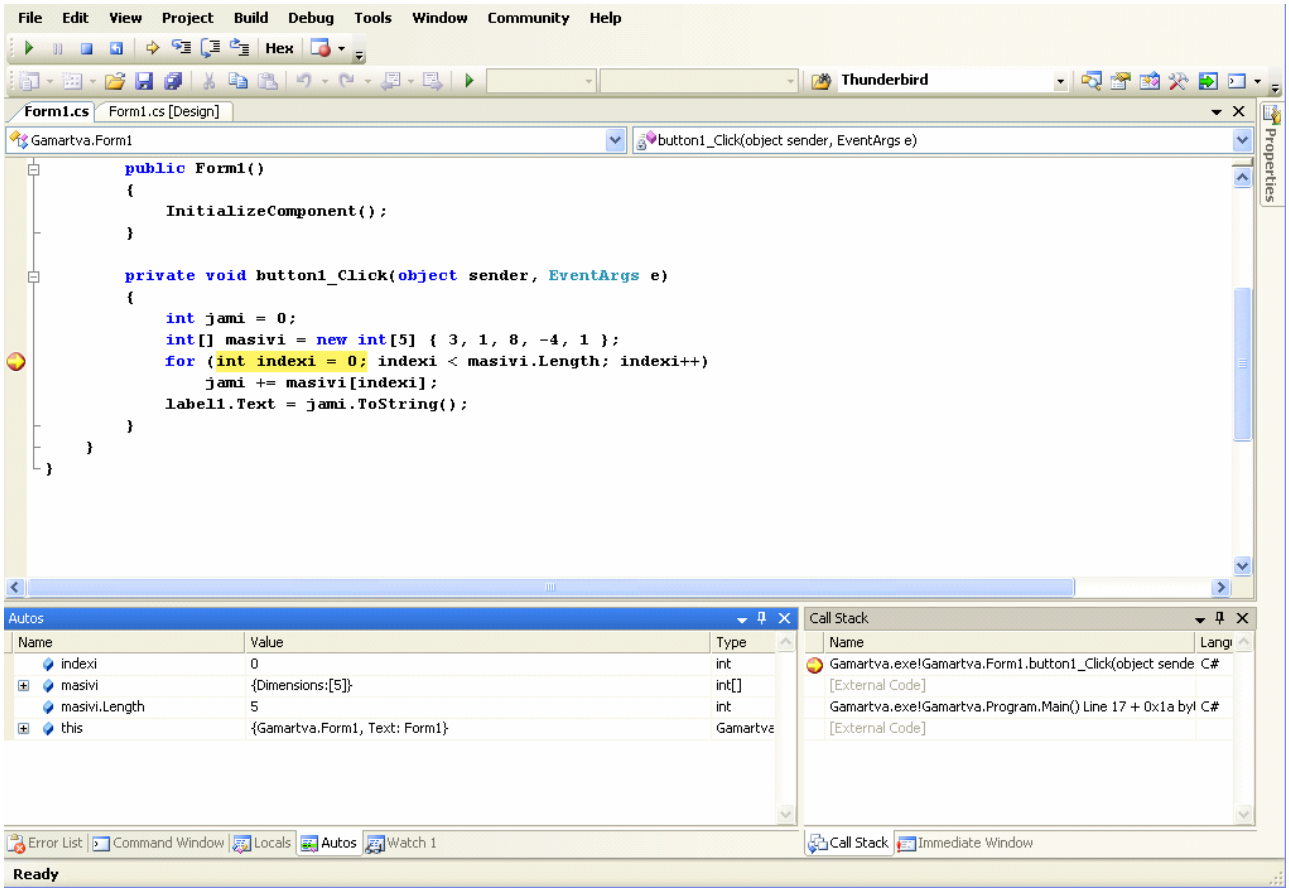
Locals ფანჯარა. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ (ნახ. 10.5). მის გასახსნელად უნდა დავაჭიროთ Locals ჩანართს. თუ ეს ფანჯარა არ ჩანს, მაშინ Debug→Windows→Locals. მასში ჩანს იმ ცვლადების მნიშვნელობები, რომლებიც იმყოფებიან ხილვადობის მოცემულ უბანში. ამ მომენტისათვის ხილვადობის უბანში იმყოფება indexi, jami და masivi ცვლადები.



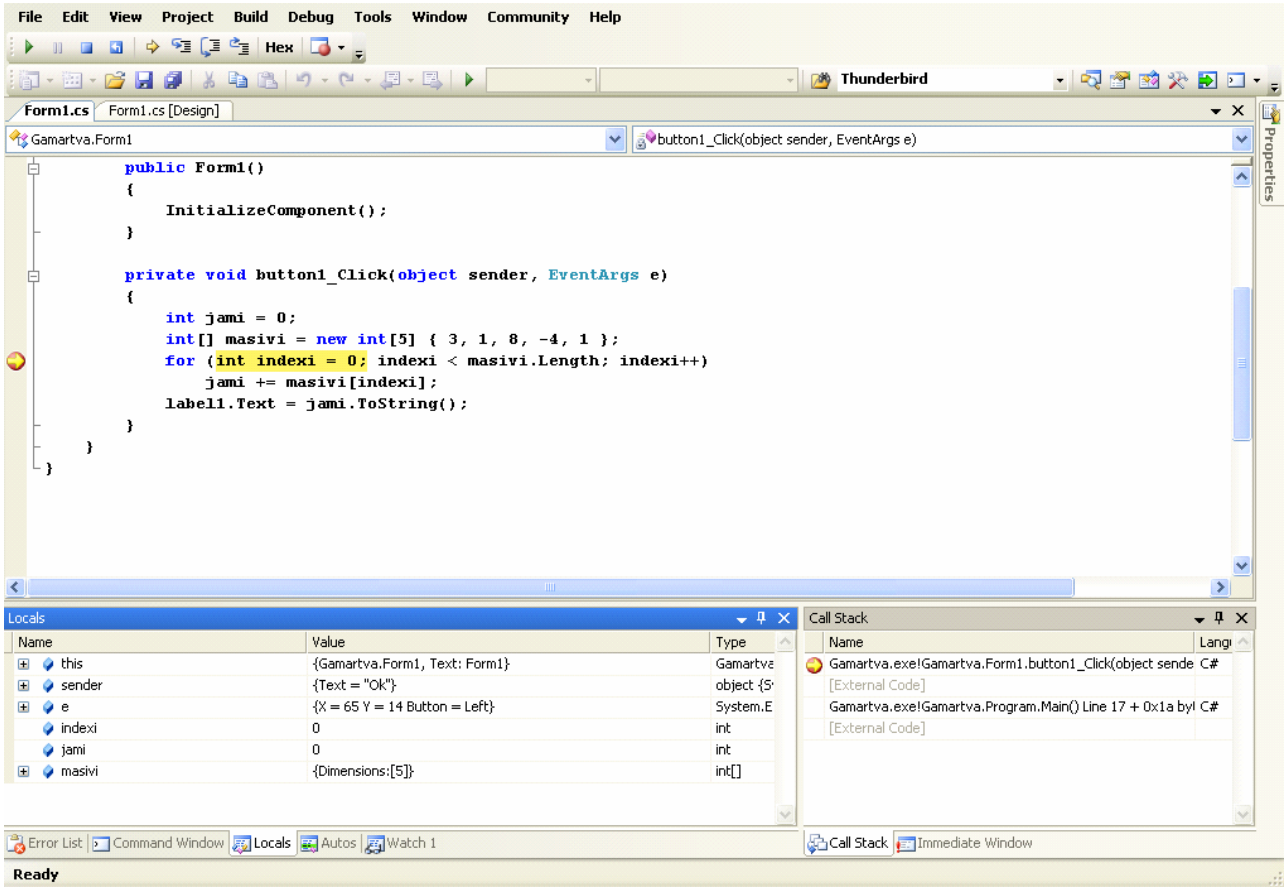
бзб. 10.2.



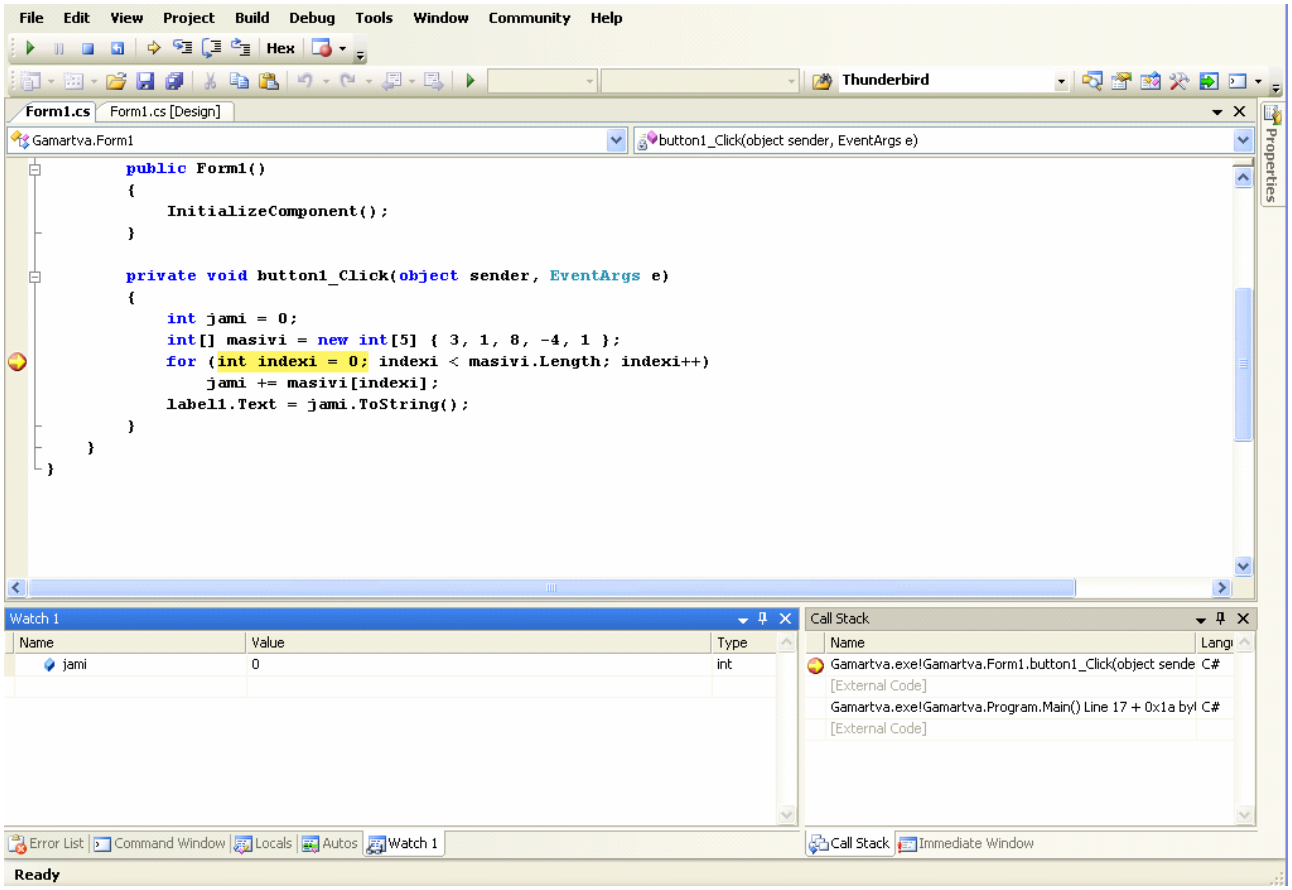
бзб. 10.3.



бзб. 10.4.



бзб. 10.5.



ნახ. 10.6.

Watch ფანჯარა. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ (ნახ. 10.6). მის გასახსნელად უნდა დავაჭიროთ Watch ჩანართს. თუ ეს ფანჯარა არ ჩანს, მაშინ Debug□Windows□Watch□Watch1. მასში ჩანს სპეციალურად მითითებული ცვლადების მნიშვნელობები. ეს საჭიროა მაშინ, როცა ცვლადების დიდი რაოდენობიდან გვაინტერესებს მხოლოდ რამდენიმე. Watch ფანჯარაში ცვლადის დასამატებლად პროგრამის კოდში ცვლადის სახელზე უნდა დავაჭიროთ თაგვის მარჯვენა კლავიშს და კონტექსტური მენიუდან შევასრულოთ Add Watch ბრძანება. ასეთი გზით jami ცვლადი დავუმატოთ Watch ფანჯარას.

პროგრამის კოდის შესრულება ბიჯობრივ რეჟიმში

პროგრამის გამართვის დროს შეგვიძლია ცვლადების მნიშვნელობების ნახვა და ოპერატორების ბიჯობრივ რეჟიმში შესრულება. მიმდინარე გამოსახულების შესასრულებლად შეგვიძლია დავაჭიროთ F10 კლავიშს ან შევასრულოთ Debug□Step Over ბრძანება. თუ გვინდა მეთოდის კოდის ბიჯობრივ რეჟიმში შესრულება, მაშინ უნდა დავაჭიროთ F11 კლავიშს ან შევასრულოთ Debug□Step Into ბრძანება. მეთოდის კოდიდან გამოსასვლელად ვასრულებთ Debug□Step Out ბრძანებას ან ვაჭერთ Shift+F11 კლავიშებს. მეთოდიდან გამოსვლისას მართვა გადაეცემა ამ მეთოდის გამომძახებელ გამოსახულებას.

ახლა ჩვენს პროგრამას დავუმატოთ ChemiKlasi კლასი, რომელიც Gamravleba() მეთოდს შეიცავს. ეს მეთოდი ასრულებს მასივის ელემენტების ერთმანეთზე გამრავლებას და ნამრავლის გაცემას. კლასს შემდეგი სახე აქვს:

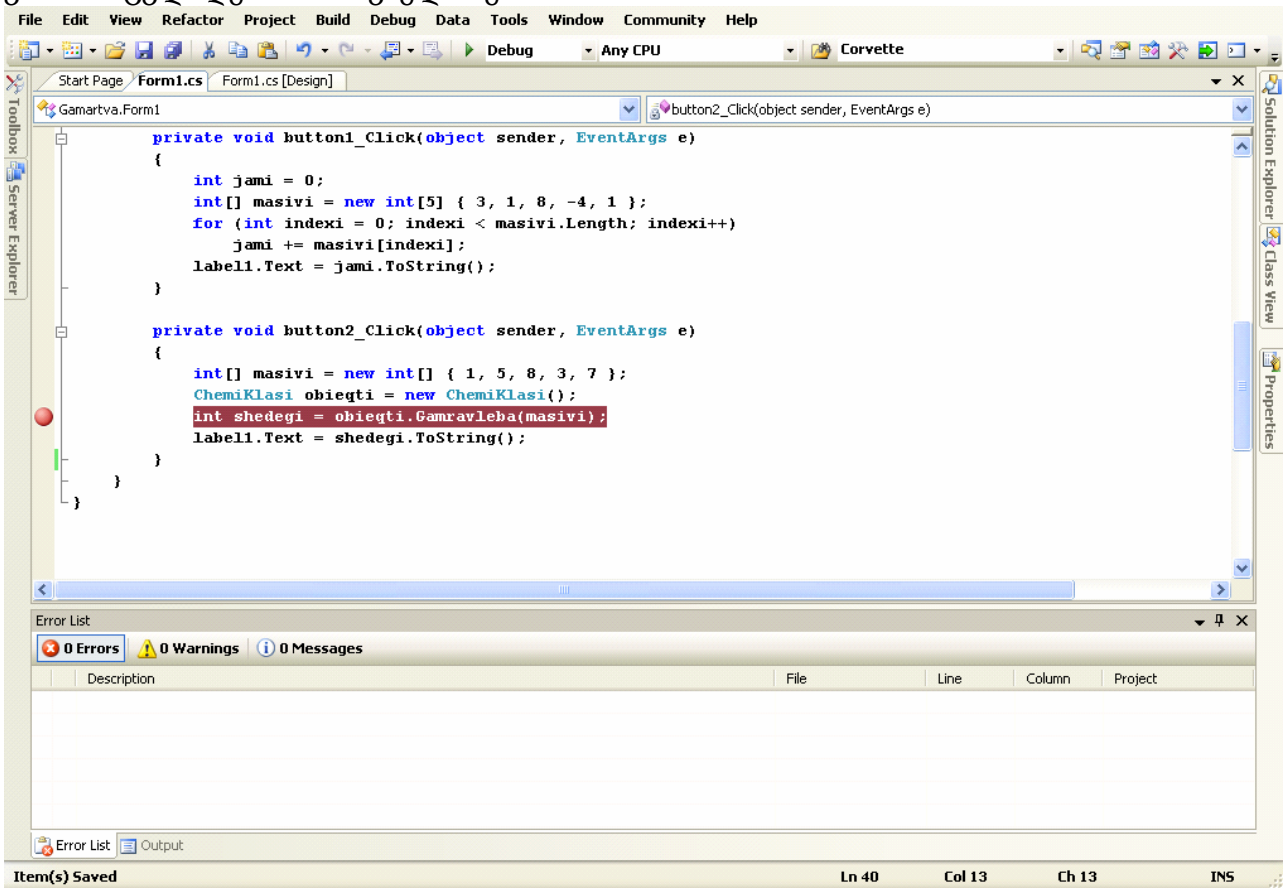
class ChemiKlasi

```
{
public int Gamravleba(int[ ] mas1)
{
int namravli = 1;
for ( int indexi = 0; indexi < mas1.Length; indexi++ )
    namravli *= mas1[indexi];
return namravli;
}
}
```

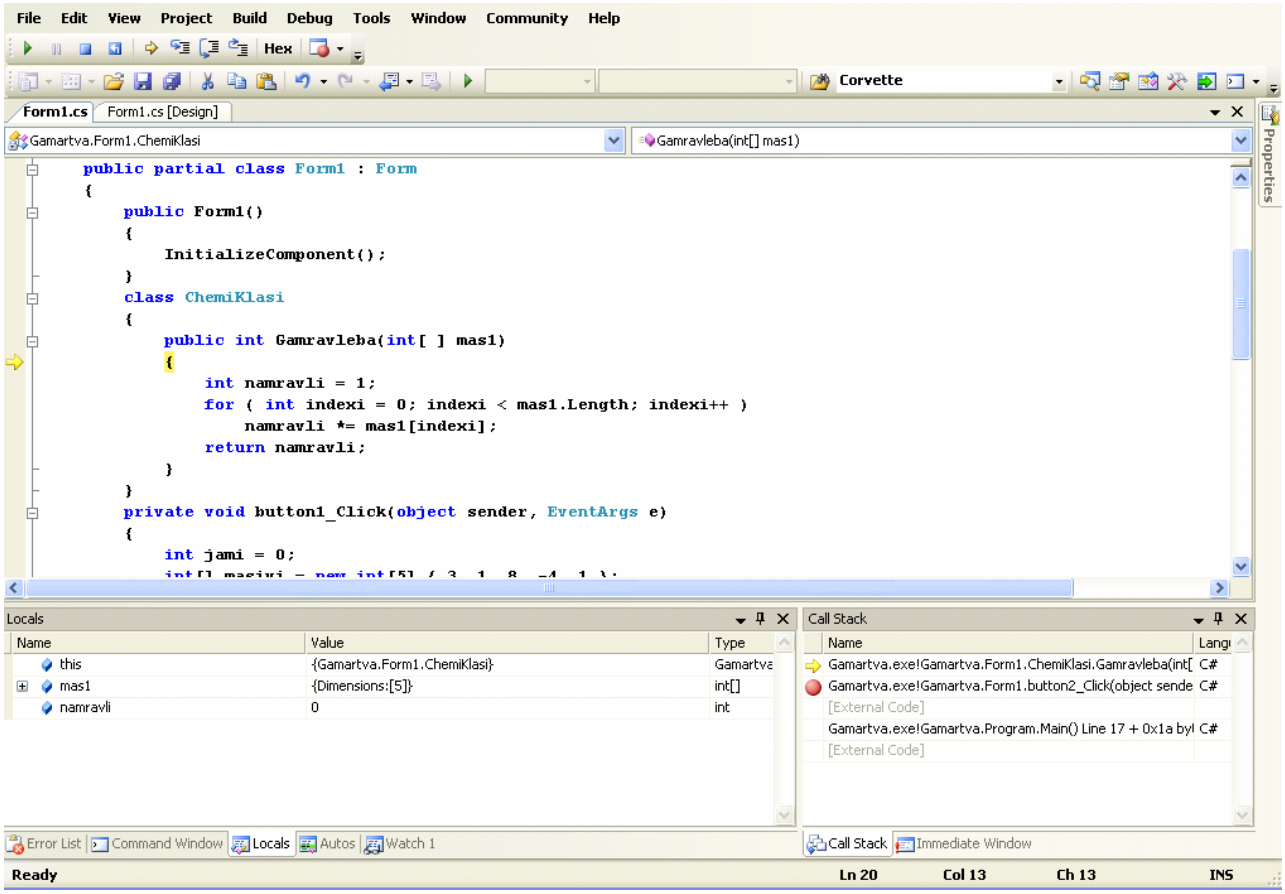
ფორმაზე მოვათავსოთ მეორე button კომპონენტი. მას მივაბათ პროგრამის კოდი, რომელიც Gamravleba() მეთოდს გამოიძახებს:

```
{
int[] masivi = new int[] { 1, 5, 8, 3, 7 };
ChemiKlasi obieqti = new ChemiKlasi();
int shedegi = obieqti.Gamravleba(masivi);
label1.Text = shedegi.ToString();
}
```

ახლა სტრიქონში, რომელშიც ხდება Gamravleba() მეთოდის გამოძახება შევქმნათ წყვეტის წერტილი (ნახ. 10.7). დავაჭიროთ F5 კლავიშს. გაიხსნება ფორმა. დავაჭიროთ მეორე button კლავიშს. პროგრამის შესრულება შეწყდება წყვეტის წერტილში. ამის შემდეგ, თუ გვინდა Gamravleba() მეთოდის შესრულება ბიჯობრივ რეჟიმში უნდა დავაჭიროთ F11 კლავიშს (ნახ. 10.8). ამ კლავიშზე ყოველი დაჭერის შემდეგ შეგვიძლია ვნახოთ ცვლადების მნიშვნელობები.



ნახ. 10.7.


File Edit View Project Build Debug Tools Window Community Help
Hex
Corvette
Form1.cs
Form1.cs [Design]
Gamartva.Form1.ChemiKlasi
Gamravleba(int[] mas1)
Properties
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
class ChemiKlasi
{
public int Gamravleba(int[ ] mas1)
{
int namravli = 1;
for (int indexi = 0; indexi < mas1.Length; indexi++)
namravli *= mas1[indexi];
return namravli;
}
}
private void button1_Click(object sender, EventArgs e)
{
int jami = 0;
Locals
Name
Value
Type
this
{Gamartva.Form1.ChemiKlasi}
Gamartva
mas1
{Dimensions:[5]}
int[]
namravli
0
int
Call Stack
Name
Gamartva.exe!Gamartva.Form1.ChemiKlasi.Gamravleba(int[C#
Gamartva.exe!Gamartva.Form1.button2_Click(object sende C#
[External Code]
Gamartva.exe!Gamartva.Program.Main() Line 17 + 0x1a byt C#
[External Code]
Error List
Command Window
Locals
Autos
Watch 1
Call Stack
Immediate Window
Ready
Ln 20
Col 13
Ch 13
INS

ნახ. 10.8.

თავი 11. დელეგატები, მოვლენები, ინტერფეისები და სახელების სივრცე

დელეგატები

დელეგატი - ესაა ობიექტი, რომელიც მეთოდს მიმართავს. დელეგატები გამოიყენება მეთოდების გამოძახებისათვის. ამასთან, გამოსაძახებელი მეთოდის არჩევა ხდება დინამიურად, პროგრამის შესრულების დროს. ამიტომ, დელეგატები იმ შემთხვევაში გამოიყენება, როცა წინასწარ არ ვიცით, თუ რომელი მეთოდი უნდა გამოვიძახოთ. დელეგატების გამოყენების კიდევ ერთი უპირატესობა იმაში მდგომარეობს, რომ ისინი უზრუნველყოფენ მოვლენებს.

დელეგატი ინახავს გამოსაძახებელი მეთოდის სახელს, მის მიერ დაბრუნებული შედეგის ტიპსა და პარამეტრების სიას, ანუ მეთოდის სიგნატურას (ხელმოწერას). ამრიგად, მეთოდის სიგნატურა - ესაა მეთოდის სახელი, მის მიერ დაბრუნებული მნიშვნელობის ტიპი და პარამეტრების სია.

როგორც ცნობილია, მეთოდისათვის მეხსიერებაში გარკვეული ზომის უბანი გამოიყოფა. ამ უბნის მისამართი - ესაა მეთოდის შესვლის წერტილი. სწორედ ამ მისამართის ინიცირება ხდება მეთოდის გამოძახებისას. მეთოდის მისამართი ენიჭება დელეგატს. თუ დელეგატი მიმართავს მეთოდს, მაშინ მოცემული მეთოდი შეგვიძლია გამოვიძახოთ ამ დელეგატის საშუალებით. გარდა ამისა, ერთი და იგივე დელეგატი შეგვიძლია გამოვიყენოთ სხვადასხვა მეთოდის გამოძახებისათვის. ამისათვის, მას უნდა მივანიჭოთ ამ მეთოდებზე მიმართვა.

დელეგატის გამოცხადებისათვის უნდა გამოვიყენოთ delegate საკვანძო სიტყვა. დელეგატის გამოცხადების სინტაქსია:

delegate *ტიპი დელეგატის_სახელი(პარამეტრების_სია);*

აქ ტიპი არის იმ მნიშვნელობის ტიპი, რომელსაც ის მეთოდი აბრუნებს, რომელიც დელეგატის მიერ იქნება გამოძახებული.

როგორც უკვე აღვნიშნეთ, დელეგატს შეუძლია გამოიძახოს ის მეთოდი, რომლის სიგნატურა შეესაბამება დელეგატის სიგნატურას. ეს საშუალებას იძლევა პროგრამის შესრულების დროს გამოსაძახებელი მეთოდი. ასეთი მეთოდი შეიძლება იყოს ობიექტის მეთოდი ან კლასში განსაზღვრული სტატიკური მეთოდი. მეთოდი მხოლოდ მაშინ შეიძლება გამოვიძახოთ, როცა მისი ხელმოწერა შეესაბამება დელეგატის ხელმოწერას.

ქვემოთ მოყვანილ პროგრამაში ხდება დელეგატთან მუშაობის დემონსტრირება:

```
//      პროგრამა 11.1
//      პროგრამაში ხდება დელეგატთან მუშაობის დემონსტრირება დელეგატის
გამოცხადება
delegate string Delegati(string striqoni1);
class ChemiKlasi
{
//      მეთოდი სტრიქონში ინტერვალებს დეფისებით ცვლის
public static string shecvla(string striqoni2)
{
return striqoni2.Replace(' ', '-');
```

```
}
//        მეთოდი სტრიქონიდან შლის ინტერვალებს
public static string washla(string striqoni2)
{
string temp = "";
int i;

for ( i = 0; i < striqoni2.Length; i++ )
        if ( striqoni2[i] != ' ' ) temp += striqoni2[i];
return temp;
}
//        მეთოდი უკუ მიმდევრობით ალაგებს სტრიქონის სიმბოლოებს
public static string shebruneba(string striqoni2)
{
string temp = "";
int i;

for ( i = striqoni2.Length - 1; i >= 0; i-- )
        temp += striqoni2[i];
return temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//        დელეგატის შექმნა
Delegati obieqti = new Delegati(ChemiKlasi.shecvla);
string striqoni;

//        shecvla() მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = obieqti("მეთოდების მუშაობის შემოწმება.");
label1.Text = striqoni;

obieqti = new Delegati(ChemiKlasi.washla);
//        washla() მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = obieqti("მეთოდების მუშაობის შემოწმება.");
label2.Text = striqoni;

obieqti = new Delegati(ChemiKlasi.shebruneba);
//        shebruneba() მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = obieqti("მეთოდების მუშაობის შემოწმება.");
label3.Text = striqoni;
}
```

პროგრამაში გამოცხადებულია Delegati დელეგატი, რომელსაც აქვს string ტიპის ერთი პარამეტრი და აბრუნებს string ტიპის მნიშვნელობას. ChemiKlasi კლასში გამოცხადებულია სამი სტატიკური მეთოდი დელეგატის შესაბამისი სიგნატურით.

პროგრამაში იქმნება Delegati ტიპის obieqti ობიექტი, რომელსაც ენიჭება shecvla() მეთოდზე მიმართვა.

Delegati obieqti = new Delegati(shecvla);

აქ shecvla() მეთოდი გადაეცემა დელეგატის კონსტრუქტორს პარამეტრის სახით. გამოიყენება მხოლოდ მეთოდის სახელი პარამეტრის მითითების გარეშე, ე.ი. დელეგატის ეგზემპლარის შექმნისას ეთითება იმ მეთოდის სახელი, რომელზეც უნდა მიუთითებდეს დელეგატი. მეთოდის ხელმოწერა უნდა შეესაბამებოდეს დელეგატის გამოცხადებაში განსაზღვრულ ხელმოწერას. თუ ეს პირობა არ სრულდება, მაშინ პროგრამის კომპილირებისას ადგილი ექნება შეცდომას.

მომდევნო სტრიქონში obieqti დელეგატი გამოიყენება shecvla() მეთოდის გამოძახებისათვის. შემდეგ, obieqti დელეგატს ენიჭება მიმართვა washla() მეთოდზე და სრულდება washla() მეთოდის გამოძახება.

პროგრამის ბოლოს obieqti დელეგატს ენიჭება shebruneba() მიმართვა და სრულდება შესაბამისი მეთოდის გამოძახება.

დელეგატებს, სტატიკურ მეთოდებზე მიმართვის გარდა, შეიძლება მიენიჭოს აგრეთვე, ობიექტის მეთოდებზე მიმართვა. ამის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

```
//      პროგრამა 11.2
//      პროგრამაში ხდება დელეგატისათვის ობიექტის მეთოდზე მიმართვის მინიჭება
delegate string Delegati(string striqoni1);
class ChemiKlasi
{
public string shecvla(string striqoni2)
{
return striqoni2.Replace(' ', '-');
}
public string washla(string striqoni2)
{
string temp = "";
int i;

for ( i = 0; i < striqoni2.Length; i++ )
        if (striqoni2[i] != ' ') temp += striqoni2[i];
return temp;
}
public string shebruneba(string striqoni2)
{
string temp = "";
int i;

for ( i = striqoni2.Length - 1; i >= 0; i-- )
        temp += striqoni2[i];
return temp;
}
}
```

```
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi();
//        დელეგატის ეგზემპლარის შექმნა
Delegati delegati_obieqti = new Delegati(obieqti.shecvla);
string striqoni;
//        მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
label1.Text = striqoni;

delegati_obieqti = new Delegati(obieqti.washla);
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
label2.Text = striqoni;

delegati_obieqti = new Delegati(obieqti.shebruneba);
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
label3.Text = striqoni;
}
```

პროგრამის მუშაობის შედეგად ეკრანზე გამოიცემა იგივე სტრიქონები, რომლებიც წინა პროგრამაში. მაგრამ, ამ შემთხვევაში დელეგატი მეთოდებს მიმართავს ChemiKlasi კლასის ობიექტის გამოყენებით.

დელეგატების მრავალმისამართიანობა

დელეგატს შეუძლია შეინახოს რამდენიმე მეთოდის მისამართი. ამ მისამართების მიმდევრობით ინიციალიზების გზით დელეგატს შეუძლია ერთმანეთის მიყოლებით გამოიძახოს შესაბამისი მეთოდები. დელეგატის ამ თვისებას მრავალმისამართიანობა ეწოდება. გამოსაძახებელი მეთოდების მისამართების ასეთი მიმდევრობა, ანუ მეთოდებზე მიმართვების მწკრივი, შემდეგნაირად იქმნება. თავდაპირველად იქმნება დელეგატი (დელეგატი-ობიექტი), შემდეგ კი გამოიყენება += ოპერატორი მწკრივისათვის მეთოდების მისამართების დასამატებლად. შედეგად, დელეგატი მრავალმისამართიანი ხდება. მწკრივიდან მეთოდის მისამართების წასაშლელად გამოიყენება -= ოპერატორი. ერთადერთი შეზღუდვაა ის, რომ დელეგატს, რომელიც ინახავს რამდენიმე მიმართვას, უნდა ჰქონდეს დასაბრუნებელი მნიშვნელობის void ტიპი.

ქვემოთ მოყვანილია პროგრამა, რომელიც ახდენს მრავალმისამართიანი დელეგატის გამოყენების დემონსტრირებას. რადგან, მეთოდის მიერ დაბრუნებული შედეგის ტიპია void, ამიტომ გამომძახებელი პროგრამისათვის შეცვლილი სტრიქონის დასაბრუნებლად გამოიყენება ref მოდიფიკატორი.

```
//        პროგრამა 11.3
//        პროგრამაში ხდება მრავალმისამართიან დელეგატთან მუშაობის დემონსტრირება
delegate void Delegati(ref string striqoni1);
class ChemiKlasi
{
public void shecvla(ref string striqoni2)
{
```

```
striqoni2 = striqoni2.Replace(' ', '-');
}
public void washla(ref string striqoni2)
{
string temp = "";
int i;

for ( i = 0; i < striqoni2.Length; i++ )
        if (striqoni2[i] != ' ' ) temp += striqoni2[i];
striqoni2 = temp;
}
public void shebruneba(ref string striqoni2)
{
string temp = "";
int i;

for ( i = striqoni2.Length - 1; i >= 0; i-- )
        temp += striqoni2[i];
striqoni2 = temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi();
Delegati delegati_obieqti;
Delegati shecvla_mimartva  = new Delegati(obieqti.shecvla);
Delegati washla_mimartva   = new Delegati(obieqti.washla);
Delegati shebruneba_mimartva = new Delegati(obieqti.shebruneba);
string striqoni = " მეთოდების მუშაობის შემოწმება";
label1.Text = striqoni;
//        დელეგატს ორი მიმართვა ენიჭება
delegati_obieqti = shecvla_mimartva;
delegati_obieqti += shebruneba_mimartva;                    //        მიმართვების მწკრივის შექმნა
//        მრავალმისამართიანი დელეგატის გამოძახება
delegati_obieqti(ref striqoni);
label2.Text = striqoni;
//        მიმართვების მწკრივიდან shecvla მიმართვის წაშლა და washla მიმართვის დამატება
delegati_obieqti -= shecvla_mimartva;
delegati_obieqti += washla_mimartva;       //        მიმართვების ახალი მწკრივის შექმნა
striqoni = " მეთოდების მუშაობის შემოწმება";
//        მრავალმისამართიანი დელეგატის გამოძახება
delegati_obieqti(ref striqoni);
label3.Text = striqoni;
}
```

პროგრამაში იქმნება ოთხი დელეგატი (დელეგატი-ობიექტი). delegati_obieqti

დელეგატს აქვს null მნიშვნელობა, რადგან შექმნისას ის არ იყო ინიციალიზებული. დანარჩენი სამი დელეგატი მიმართავს შესაბამის მეთოდებს. შემდეგ, პროგრამაში იქმნება მრავალმისამართიანი დელეგატი, რომელიც იძახებს shecvla() და shebruneba() მეთოდებს. ეს ხორციელდება სამი ოპერატორის საშუალებით:

```
delegati_obieqti  = shecvla_mimartva;
delegati_obieqti += shebruneba_mimartva;
delegati_obieqti(ref striqoni);
```

პირველ ოპერატორში delegati_obieqti დელეგატს ენიჭება shecvla_mimartva მიმართვა. შემდეგ += ოპერატორის საშუალებით მწკრივს ემატება shebruneba_mimartva მიმართვა. delegati_obieqti დელეგატთან მიმართვისას ჯერ შესრულდება shecvla() მეთოდი, შემდეგ კი shebruneba() მეთოდი. delegati_obieqti -= shecvla_mimartva; ოპერატორი მწკრივიდან წაშლის shecvla_mimartva მიმართვას, ხოლო delegati_obieqti += washla_mimartva; ოპერატორი მწკრივს დაუმატებს washla_mimartva მიმართვას. შედეგად, delegati_obieqti დელეგატთან მიმართვისას ჯერ შესრულდება shebruneba() მეთოდი, შემდეგ კი washla() მეთოდი.

მოვლენები

მოვლენა არის ავტომატური უწყება რაიმე მომხდარ მოქმედებაზე. მოვლენის საშუალებით ერთი ობიექტი მეორე ობიექტს ატყობინებს, რომ რაღაც მოხდა. მოვლენის მაგალითებია: კლავიატურის კლავიშის დაჭერა, კლავიატურის კლავიშის აშვება, თაგვის მარცხენა კლავიშის დაჭერა, თაგვის მარჯვენა კლავიშის დაჭერა, თაგვის კლავიშის აშვება და ა.შ.

ობიექტისთვის, რომელიც მისი განსაზღვრის (კოდის) მიხედვით უნდა რეაგირებდეს რაიმე მოვლენაზე, რეგისტრირდება ამ მოვლენის დამამუშავებელი (მეთოდი). მოვლენების დამამუშავებლები იქმნება დელეგატების საფუძველზე.

მოვლენები წარმოადგენენ კლასის წევრებს და მათი გამოცხადება ხდება event საკვანძო სიტყვის გამოყენებით. მოვლენების გამოცხადების სინტაქსია:

event *დელეგატის_სახელი მოვლენის_სახელი;*

აქ დელეგატის_სახელი იმ დელეგატის სახელია, რომელიც გამოიყენება მოვლენის დასამუშავებლად, ხოლო მოვლენის_სახელი არის შესაქმნელი მოვლენის (მოვლენა-ობიექტის) სახელი.

ქვემოთ მოყვანილი პროგრამა ახდენს მოვლენასთან მუშაობის დემონსტრირებას.

```
//      პროგრამა 11.4
//      პროგრამაში ხდება მოვლენასთან მუშაობის დემონსტრირება
//      დელეგატის გამოცხადება, რომლის საფუძველზეც განსაზღვრული იქნება
მოვლენა
delegate void ChemiDelegati();
//      კლასის გამოცხადება, რომელშიც ხდება მოვლენის ინიცირება
class Klasi1
{
public event ChemiDelegati ChemiMovlena;              //      მოვლენის გამოცხადება
//      ამ მეთოდში ინიცირდება მოვლენა
public void metodi1()
```

```
{
//        პირობის განსაზღვრა მოვლენის ინიციალიზაციისათვის
if ( ChemiMovlena != null ) ChemiMovlena();
}
}
class Klasi2
{
public void MovlenisDamamushavebeli()
{
MessageBox.Show("აღიძვრა მოვლენა");
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Klasi2 obieqti = new Klasi2();
Klasi1 obieqti_movlena  = new Klasi1();              //        მოვლენის ეგზემპლარის შექმნა
//
obieqti_movlena.ChemiMovlena += new
                                        ChemiDelegati(obieqti.MovlenisDamamushavebeli);
//        მოვლენის ინიცირება
obieqti_movlena.metodi1();
}
```

პროგრამა იწყება დელეგატის გამოცხადებით, რომელიც ინახავს მიმართვას მოვლენის დამამუშავებელზე:

```
delegate void ChemiDelegati();
```

მოვლენების ყველა დამამუშავებელი აქტიურდება დელეგატის საშუალებით. შედეგად, დელეგატი განსაზღვრავს მოვლენის ხელმოწერას. მოცემულ შემთხვევაში მოვლენას არ აქვს პარამეტრები, თუმცა მათი არსებობა დასაშვებია. რადგან, მოვლენა შეიძლება იყოს მრავალმისამართიანი, ამიტომ მოვლენის დამამუშავებლისათვის უნდა მიეთითოს დასაბრუნებელი მნიშვნელობის void ტიპი.

შემდეგ, პროგრამაში იქმნება Klasi1 კლასი, რომელშიც გამოცხადებულია ChemiMovlena მოვლენა. metodi1() მეთოდი პროგრამის მიერ გამოძახებული იქნება მოვლენის ინიციალიზაციისათვის. პირობა, როცა სრულდება მოვლენის ინიციალიზაცია, განსაზღვრულია if ოპერატორით

```
if ( ChemiMovlena != null ) ChemiMovlena();
```

აქედან გამომდინარე, მოვლენის დამამუშავებელი მეთოდი მხოლოდ იმ შემთხვევაში გამოიძახება თუ ChemiMovlena მოვლენასთან ასოცირებულ დელეგატს აქვს null-საგან განსხვავებული მნიშვნელობა.

Klasi2 კლასის შიგნით იქმნება მოვლენების დამამუშავებელი MovlenisDamamushavebeli() მეთოდი. ამ მაგალითში მოვლენების დამამუშავებელს უბრალოდ გამოაქვს შეტყობინება. შემდეგ, პროგრამაში იქმნება Klasi1 ტიპის obieqti_movlena ობიექტი, ხოლო MovlenisDamamushavebeli() მეთოდი რეგისტრირდება როგორც ამ კლასში გამოცხადებული მოვლენის დამამუშავებელი.

```
Klasi1 evt  = new Klasi1();
obieqti_movlena.ChemiMovlena += new
```

ChemiDelegati(obieqti.MovlenisDamamushavebeli);

მოვლენის დამამუშავებლის დამატება ხდება += ოპერატორის გამოყენებით, წაშლა კი - -= ოპერატორის გამოყენებით.

პროგრამის ბოლოს ხდება მოვლენის ინიცირება

obieqti_movlena.metodi1();

metodi1() მეთოდის გამოძახება იწვევს მოვლენის დამამუშავებლის გამოძახებას. მოცემულ შემთხვევაში გამოიძახება მხოლოდ ერთი დამამუშავებელი, თუმცა ისინი შეიძლება რამდენიმე იყოს.

ფართოსამაუწყებლო მოვლენა

როგორც ადრე აღვნიშნეთ მოვლენა იქმნება დელეგატების საფუძველზე. რადგანაც, დელეგატი შეიძლება იყოს მრავალმისამართიანი, ამიტომ მოვლენებს შეუძლიათ რამდენიმე დამამუშავებლის გააქტიურება, რომელთა ნაწილი შეიძლება განსაზღვრული იყოს სხვა ობიექტებში. ასეთ მოვლენებს ფართოსამაუწყებლო ეწოდებათ. მათი გამოყენება ბევრ ობიექტს საშუალებას აძლევს რეაგირება მოახდინოს მოვლენის შესახებ უწყებაზე. ქვემოთ მოყვანილ პროგრამაში ხდება ფართოსამაუწყებლო მოვლენასთან მუშაობის დემონსტრირება.

```
//      პროგრამა 11.5
//      პროგრამაში ხდება ფართოსამაუწყებლო მოვლენასთან მუშაობის
დემონსტრირება
delegate void ChemiDelegati();
//      კლასის გამოცხადება, რომელშიც ინიცირდება მოვლენა
class Klasi1
{
public event ChemiDelegati ChemiMovlena;
//      მოვლენის ინიციალიზაცია
public void metodi1()
{
if ( ChemiMovlena != null ) ChemiMovlena();
}
}
class Damamushavebeli1
{
public void Movlenis_Damamushavebeli_2()
{
MessageBox.Show("მოვლენა მიღებულია Damamushavebeli1 კლასის ობიექტის მიერ");
}
}
class Damamushavebeli2
{
public void Movlenis_Damamushavebeli_3()
{
MessageBox.Show("მოვლენა მიღებულია Damamushavebeli2 კლასის ობიექტის მიერ");
}
}
```

```
class Klasi2
{
public void Movlenis_Damamushavebeli_1()
{
MessageBox.Show("მოვლენა მიღებულია Klasi2 კლასის ობიექტის მიერ");
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Klasi2 obieqti = new Klasi2();
Klasi1 obieqti_movlena = new Klasi1();
Damamushavebeli1 obieqti1 = new Damamushavebeli1();
Damamushavebeli2 obieqti2 = new Damamushavebeli2();
//      მოვლენების დამამუშავებლების მწკრივი: Movlenis_Damamushavebeli_1(),
//      Movlenis_Damamushavebeli_2() და Movlenis_Damamushavebeli_3()
//      მეთოდების დამატება
obieqti_movlena.ChemiMovlena += new
                              ChemiDelegati(obieqti.Movlenis_Damamushavebeli_1);
obieqti_movlena.ChemiMovlena += new
                              ChemiDelegati(obieqti1.Movlenis_Damamushavebeli_2);
obieqti_movlena.ChemiMovlena += new
                              ChemiDelegati(obieqti2.Movlenis_Damamushavebeli_3);
//      მოვლენის ინიცირება
obieqti_movlena.metodi1();

//      მწკრივიდან ერთი დამამუშავებლის წაშლა
obieqti_movlena.ChemiMovlena -=
                              new ChemiDelegati(obieqti1.Movlenis_Damamushavebeli_2);
obieqti_movlena.metodi1();
}
```

ამ პროგრამაში იქმნება ორი კლასი Damamushavebeli1 და Damamushavebeli2. მათში განსაზღვრულია მოვლენების დამამუშავებლები, რომლებიც თავიანთი ხელმოწერით თავსებადია ChemiDelegati ტიპთან. შედეგად, ეს დამამუშავებლები ხდებიან მწკრივის ნაწილი და გამოიძახებიან ცალ-ცალკე.

გენერირებული მოვლენები ცალ-ცალკე გადაეცემა თითოეულ ობიექტს. შედეგად, მოვლენის შესახებ უწყების მისაღებად წინასწარ უნდა იყოს დარეგისტრირებული (განსაზღვრული) ამ მოვლენის დამამუშავებელი თითოეული ობიექტისათვის. მაგალითად, ქვემოთ მოყვანილ პროგრამაში აღძრული მოვლენა (metodi1() მეთოდის გამოძახება) ახდენს Klasi1 კლასის ორი ობიექტის დამამუშავებლების ინიცირებას.

```
//      პროგრამა 11.6
//      პროგრამაში ხდება ორი ობიექტის დამამუშავებლების ინიცირება
delegate void ChemiDelegati();
//
class ChemiKlasi
```

```
{
public event ChemiDelegati ChemiMovlena;
//
public void metodi1()
{
if ( ChemiMovlena != null ) ChemiMovlena();
}
}
class Klasi1
{
int id;
public Klasi1(int par)
{
id = par;
}
public void Damamushavebeli()
{
MessageBox.Show("მოვლენა მიღებულია ობიექტის მიერ, რომლის # = " + id.ToString());
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti_movlena = new ChemiKlasi();
Klasi1 obieqti1 = new Klasi1(1);
Klasi1 obieqti2 = new Klasi1(2);

obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti1.Damamushavebeli);
obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti2.Damamushavebeli);
//      მოვლენის ინიცირება
obieqti_movlena.metodi1();
}
```

როგორც ვხედავთ, თითოეული ობიექტისათვის ცალცალკე რეგისტრირდება მოვლენების დამამუშავებელი, და შესაბამისად, თითოეული ობიექტი იღებს ცალკე უწყებას მომხდარი მოვლენის შესახებ.

ინტერფეისები

C# ენის ერთ-ერთი მნიშვნელოვანი ელემენტია ინტერფეისი. იგი განსაზღვრავს მეთოდების, თვისებების, ინდექსატორებისა და მოვლენების ნაკრებს, რომელთა რეალიზება შესაძლებელია კლასის საშუალებით. პროგრამირების დროს ზოგჯერ წამოიჭრება კლასის მიერ შესრულებული მოქმედებების განსაზღვრის აუცილებლობა, მათი რეალიზების გზის მითითების გარეშე. ჩვენ ადრე განვიხილეთ მსგავსი მაგალითი - აბსტრაქტული მეთოდი. იგი განსაზღვრავს მეთოდის ინტერფეისს, და არა რეალიზაციას. C# ენაში შეიძლება მთლიანად განვაცალკევოთ კლასის ინტერფეისი ამ

კლასის რეალიზაციისაგან. ინტერფეისის განსაზღვრის შემდეგ მისი რეალიზება შესაძლებელია ერთი ან მეტი კლასის საშუალებით. ამასთან, ასეთი კლასი დამატებით შეიძლება შეიცავდეს საკუთარ ელემენტებს.

ინტერფეისის გამოცხადების სინტაქსია:

interface *ინტერფეისის_სახელი*
{
ინტერფეისის ტანი
}

ინტერფეისის ტანი შეიძლება შეიცავდეს მეთოდებს, თვისებებს, ინდექსატორებსა და მოვლენებს. ინტერფეისი არ შეიძლება შეიცავდეს ცვლადებს (ველებს), სტატიკურ წევრებს, კონსტრუქტორებს, დესტრუქტორებსა და ოპერატორულ მეთოდებს. ინტერფეისში შეიძლება გამოცხადებული იყოს მეთოდების, თვისებების, ინდექსატორებისა და მოვლენების სიგნატურები. მეთოდისათვის დამატებით ეთითება დასაბრუნებელი მნიშვნელობის ტიპი. გარდა ამისა, მეთოდის სიგნატურა ზუსტად უნდა შეესაბამებოდეს interface განსაზღვრაში მითითებულ სიგნატურას.

ინტერფეისების რეალიზაცია

როგორც კი ინტერფეისი განისაზღვრება, იგი შეიძლება რეალიზებული იყოს ერთი ან მეტი კლასის მიერ. ინტერფეისის რეალიზაციისათვის კლასის სახელის შემდეგ უნდა მივუთითოთ ორი წერტილი და ინტერფეისის სახელი.

თუ კლასი ახდენს ინტერფეისის რეალიზებას, ის ვალდებულია მისი რეალიზება მოახდინოს მთლიანად.

კლასებს, რომლებიც ახდენს ინტერფეისების რეალიზებას, შეუძლიათ განსაზღვრონ, აგრეთვე, დამატებითი წევრები. ქვემოთ მოყვანილია მაგალითი, რომელშიც ChemiKlasi კლასის საშუალებით რეალიზებულია ChemiInterfeisi ინტერფეისი. ეს კლასი საშუალებას გვაძლევს შევასრულოთ რიცხვების ნაკრების გენერირება, ამასთან, ყოველი მომდევნო რიცხვი 5-ით მეტია წინაზე.

```
//      პროგრამა 11.7
//      პროგრამაში რეალიზებულია ინტერფეისი, რომელშიც გამოცხადებულია
ცვლადი და მეთოდები
public interface ChemiInterpeisi
{
int Momdevno();
void Ganuleba();
void Minicheba(int x);
}
class ChemiKlasi : ChemiInterpeisi
{
int ricxvi;
int cvladi;
int wina;
public ChemiKlasi()
{
ricxvi = 0;
cvladi = 0;
```

```
wina   = -5;
}
public int Momdevno()
{
wina = cvladi;
cvladi += 5;
return cvladi;
}
public void Ganuleba()
{
ricxvi = 0;
cvladi = 0;
wina   = -5;
}
public void Minicheba(int x)
{
ricxvi = x;
cvladi = x;
wina   = x - 5;
}
int Wina()
{
return wina;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi Obieqti = new ChemiKlasi();

Obieqti.Ganuleba();
Obieqti.Minicheba(10);
label1.Text = Obieqti.Momdevno().ToString();
}
```

კლასების მემკვიდრეობითობა და ინტერფეისის რეალიზება

კლასებს შეუძლიათ ერთზე მეტი ინტერფეისის რეალიზება. ამ შემთხვევაში ინტერფეისების სახელები ერთმანეთისაგან მძიმეებით უნდა გამოიყოს. კლასი შეიძლება იყოს, აგრეთვე, საბაზო კლასის მემკვიდრე და ახდენდეს ერთი ან მეტი ინტერფეისის რეალიზებას. ამ შემთხვევაში სიაში, წინაპარი კლასის სახელი უნდა იყოს პირველი. მოყვანილ პროგრამაში ხდება ყოველივე ამის დემონსტრირება.

```
//      პროგრამა 11.8
//      პროგრამაში ხდება ინტერფეისის რეალიზება მემკვიდრეობითობის შემთხვევაში
//      Interpeisi_A ინტერფეისის გამოცხადება
public interface Interpeisi_A
{
```

```
//        მეთოდების გამოცხადება
void Start();
void Stop();
//        Started თვისების გამოცხადება
bool Started
{
get;
}
}
//        Interpeisi_B ინტერფეისის გამოცხადება
public interface Interpeisi_B
{
//        მეთოდების გამოცხადება
void Marcxniv();
void Marjvniv();
}
public class ChemiKlasi
{
private string model;
public ChemiKlasi(string model)
{
this.model = model;
}
//        Model თვისების გამოცხადება
public string Model
{
get
{
return model;
}
set
{
model = value;
}
}
}
public class Manqana : ChemiKlasi, Interpeisi_A, Interpeisi_B
{
private bool started = false;
//        საბაზო კლასის კონსტრუქტორის გამოძახება
public Manqana(string model) : base(model)
{
//        ცარიელი ტანი
}
//        Interpeisi_A ინტერფეისის Start() მეთოდის რეალიზაცია
```

```
public void Start()
{
MessageBox.Show("მანქანა დაიქოქა");
started = true;
}
//        Interpeisi_A ინტერფეისის Stop() მეთოდის რეალიზაცია
public void Stop()
{
MessageBox.Show("მანქანა დამუხრუჭებულია");
started = false;
}
//        Interpeisi_A ინტერფეისის Started თვისების რეალიზაცია
public bool Started
{
get
{
return started;
}
}
//        Interpeisi_B ინტერფეისის Marcxniv() მეთოდის რეალიზაცია
public void Marcxniv()
{
MessageBox.Show("მანქანა მარცხნივ უხვევს");
}
//        Interpeisi_B ინტერფეისის Marjvniv() მეთოდის რეალიზაცია
public void Marjvniv()
{
MessageBox.Show("მანქანა მარჯვნივ უხვევს");
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Manqana ChemiManqana = new Manqana("BMW");
label1.Text = "ChemiManqana.Model = " + ChemiManqana.Model;
ChemiManqana.Steart();              //        მანქანა დაიქოქა
ChemiManqana.Marcxniv();            //        მარცხნივ მოხვევა
}
```

ინტერფეისული მიმართვები

ჩვენ შეგვიძლია გამოვაცხადოთ მიმართვითი ცვლადი, რომელსაც აქვს ინტერფეისული ტიპი, ანუ შეგვიძლია შევქმნათ ინტერფეისული მიმართვითი ცვლადი. ასეთი ცვლადი შეიძლება მიმართავდეს ნებისმიერ ობიექტს, რომელიც ახდენს მისი ინტერფეისის რეალიზებას. ინტერფეისული მიმართვის საშუალებით ობიექტის მეთოდის გამოძახებისას გამოიძახება ის მეთოდი, რომლის რეალიზებას მოცემული

ობიექტი ახდენს.

მოყვანილ პროგრამაში ნაჩვენებია ინტერფეისული მიმართვის გამოყენება. აქ გამოცხადებულია obieqti ინტერფეისული ცვლადი ChemiKlasi და Klasi1 ობიექტების მეთოდების გამოსაძახებლად.

```
//      პროგრამა 11.9
public interface ChemiInterpeisi
{
        int Momdevno();
        void Ganuleba();
        void Minicheba(int x);
}
class ChemiKlasi : ChemiInterpeisi
{
int ricxvi;
int cvladi;
public ChemiKlasi()
{
        ricxvi = 0;
        cvladi = 0;
}
public int Momdevno()
{
        cvladi += 5;
        return cvladi;
}
public void Ganuleba()
{
        ricxvi = 0;
        cvladi = 0;
}
public void Minicheba(int x)
{
        ricxvi = x;
        cvladi = x;
}
}
class Klasi1 : ChemiInterpeisi
{
int ricxvi;
int cvladi;
public Klasi1()
{
        ricxvi = 0;
        cvladi = 0;
}
```

```
public int Momdevno()
{
        cvladi += 7;
        return cvladi;
}
public void Ganuleba()
{
        ricxvi = 0;
        cvladi = 0;
}
public void Minicheba(int x)
{
        ricxvi = x;
        cvladi = x;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi OriObieqti = new ChemiKlasi();
Klasi1 SamiObieqti = new Klasi1();
ChemiInterpeisi obieqti;

for ( int i = 0; i < 5; i++ )
{
obieqti = OriObieqti;
label1.Text = obieqti.Momdevno().ToString();

obieqti = SamiObieqti;
label2.Text = obieqti.Momdevno().ToString();
}
}
```

პროგრამაში obieqti ობიექტი გამოცხადებულია როგორც მიმართვა ChemiInterpeisi ინტერფეისზე. ეს იმას ნიშნავს, რომ იგი შეიძლება გამოყენებულ იქნას ნებისმიერ იმ ობიექტზე მიმართვების შენახვის მიზნით, რომელიც ახდენს ChemiInterpeisi ინტერფეისის რეალიზებას. მოცემულ შემთხვევაში იგი გამოიყენება OriObieqti და SamiObieqti ობიექტებზე მიმართვის შენახვისათვის, რომლებიც წარმოადგენენ ChemiKlasi და Klasi1 ტიპის ობიექტებს შესაბამისად. ორივე ეს ობიექტი ახდენს ChemiInterpeisi ინტერფეისის რეალიზებას. obieqti ინტერფეისულმა მიმართვამ იცის მხოლოდ იმ მეთოდების შესახებ, რომლებიც გამოცხადებულია ინტერფეისში.

ინტერფეისის თვისებები

ისევე, როგორც მეთოდების შემთხვევაში, ინტერფეისში თვისებების განსაზღვრისას არ ეთითება ტანი. ინტერფეისის თვისების სინტაქსია:
type *თვისების_სახელი*

```
{
get;
set;
}
```

ქვემოთ მოყვანილია ChemiInterpeisi ინტერფეისი, აგრეთვე, ChemiKlasi კლასის კოდი, რომელიც თვისებას იყენებს ნაკრების მომდევნო ელემენტის გაცემისა და შეცვლისათვის.

```
//      პროგრამა 11.10
//      პროგრამაში ხდება ინტერფეისის თვისებასთან მუშაობის დემონსტრირება
public interface ChemiInterpeisi
{
//      ინტერფეისის თვისება
int shemdegi
{
get;                    //      ნაკრების მომდევნო რიცხვის მიღება
set;                    //      მომდევნო რიცხვის დაყენება
}
}
class ChemiKlasi : ChemiInterpeisi
{
int cvladi;
public ChemiKlasi()
{
cvladi = 0;
}
//      მნიშვნელობის მიღება და დაყენება
public int shemdegi
{
get
{
cvladi += 5;
return cvladi;
}
set
{
cvladi = value;
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      ინტერფეისის თვისებების დემონსტრირება
ChemiKlasi obieqti = new ChemiKlasi();

for ( int i = 0; i < 5; i++ )
```

```
        label1.Text += "მომდევნო მნიშვნელობა - " +
                        obieqti.shemdegi.ToString() + '\n';
obieqti .shemdegi = 30;
for ( int i = 0; i < 5; i++ )
        label2.Text += " მომდევნო მნიშვნელობა - " +
                        obieqti.shemdegi.ToString() + '\n';
}
```

ცხადი რეალიზება

ცხადი რეალიზაცია არის შემთხვევა, როცა რეალიზაციის დროს ინტერფეისის ელემენტის წინ ეთითება ინტერფეისის სახელი. კლასს შეუძლია ორი ინტერფეისის რეალიზება, რომლებშიც შეიძლება გამოცხადებული იყოს ერთნაირი სახელის ელემენტები. წამოიჭრება პრობლემა - როგორ განვასხვავოთ ასეთი ელემენტები. ეს პრობლემა წყდება ცხადი რეალიზაციის გამოყენებით.

ქვემოთ მოყვანილ პროგრამაში გამოცხადებულია ორი ინტერფეისი, რომლებიც შეიცავენ metodi() მეთოდს. იმისათვის, რომ განვასხვავოთ ერთნაირი სახელის მქონე ეს მეთოდები უნდა გამოვიყენოთ ცხადი რეალიზაცია:

```
//      პროგრამა 11.11
//      პროგრამაში ხდება ცხადი რეალიზების გამოყენების დემონსტრირება
interface Interpeisi_A
{
int metodi(int x);
}
interface Interpeisi_B
{
int metodi(int x);
}
//      ChemiKlasi კლასში ხდება ორივე ინტერფეისის რეალიზება
class ChemiKlasi : Interpeisi_A, Interpeisi_B
{
Interpeisi_A a_obieqti;
Interpeisi_B b_obieqti;
//      ორივე metodi() მეთოდის აშკარა რეალიზება
int Interpeisi_A.metodi(int x)
{
return x + x;
}
int Interpeisi_B.metodi(int x)
{
return x * x;
}
//      metodi() მეთოდის გამოძახება ინტერფეისული მიმართვის საშუალებით
public int metodi_A(int x)
{
a_obieqti = this;                       //      a_obieqti ობიექტს ენიჭება გამომძახებელ ობიექტზე
```

```
მიმართვა
return a_obieqti.metodi(x);          //      Interpeisi_A მეთოდის გამოძახება
}
public int metodi_B(int x)
{
b_obieqti = this;      //      b_obieqti ობიექტს ენიჭება გამომძახებელ ობიექტზე მიმართვა
return b_obieqti.metodi(x);          //      Interpeisi_B მეთოდის გამოძახება
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      აშკარა რეალიზაციის გამოყენება არაერთგვაროვნობის აღმოსაფხვრელად
ChemiKlasi obieqti = new ChemiKlasi();
int ricxvi = Convert.ToInt32(textBox1.Text);

//      Interpeisi_A.metodi() მეთოდის გამოძახება
label1.Text = obieqti.metodi_A(ricxvi).ToString();
//      Interpeisi_B.metodi() მეთოდის გამოძახება
label2.Text = obieqti.metodi_B(ricxvi).ToString();
}
```

ყურადღება მივაქციოთ იმას, რომ metodi() მეთოდს აქვს ერთი და იგივე სიგნატურა Interpeisi_A და Interpeisi_B ინტერფეისებში. ამიტომ, იმ შემთხვევაში თუ ChemiKlasi ახდენს ორივე ამ ინტერფეისის რეალიზებას, მაშინ მან აშკარად უნდა მოახდინოს თითოეული მათგანის ცალ-ცალკე რეალიზება, მათი სახელების სრული განსაზღვრით. რადგან ერთადერთი საშუალება, რომლის საშუალებითაც შეიძლება გამოვიძახოთ აშკარად რეალიზებული მეთოდი არის ინტერფეისული მიმართვა, ამიტომ ChemiKlasi კლასში გამოცხადებულია ორი მიმართვა, რომელთაგან ერთი განკუთვნილია Interpeisi_A, მეორე კი - Interpeisi_B ინტერფეისისათვის. შემდეგ სრულდება კლასის ორი მეთოდი გამოიძახება ინტერფეისის მეთოდების საშუალებით. რადგან, metodi() მეთოდი ცხადად არის რეალიზებული, ამიტომ იგი მიუწვდომელია ChemiKlasi კლასის გარეთ. ამ კლასის შიგნით metodi() მეთოდთან მიმართვა შესაძლებელია მხოლოდ ინტერფეისული a_obieqti და b_obieqti ობიექტების საშუალებით.

სახელების სივრცე

სახელების სივრცე თავიდან გვაცილებს პროგრამის კლასების სახელებს შორის კონფლიქტს. დავუშვათ, რომ ჩვენი პროგრამა იყენებს სხვა პროგრამისტის მიერ შემუშავებულ კლასს, რომლის სახელია Manqana. დავუშვათ, ასევე, რომ ჩვენც დაგვჭირდა ამავე სახელის მქონე კლასის განსაზღვრა. ასეთ შემთხვევაში მოგვიწევს სხვა სახელის გამოყენება. სწორედ ასეთი პრობლემების გადასაწყვეტად გამოიყენება სახელების სივრცე.

სახელების სივრცე განსაზღვრავს გამოცხადების უბანს, რომელიც საშუალებას გვაძლევს შევინახოთ სახელების თითოეული ნაკრები სხვა ნაკრებებისაგან ცალკე. სახელების ერთ სივრცეში გამოცხადებული სახელები არ არიან კონფლიქტში

სახელების სხვა სივრცეში გამოცხადებულ ასეთივე სახელებთან. System სახელების სივრცეში მრავალი კლასია განსაზღვრული, ამიტომ თითოეული პროგრამა იწყება დირექტივით: using System;

სახელების სივრცის გამოცხადება

სახელების სივრცის გამოცხადება ხდება namespace საკვანძო სიტყვის საშუალებით. მისი სინტაქსია:

namespace *სახელების_სივრცის_სახელი*
{
სახელების სივრცის წევრები
}

სახელების სივრცეში განსაზღვრული ყველა ელემენტი, როგორიცაა კლასები, დელეგატები, ჩამოთვლები, ინტერფეისები ან სხვა სახელების სივრცეები, იმყოფება ამ სახელების სივრცის მხედველობის უბნის საზღვრებში.

ქვემოთ მოყვანილ პროგრამაში ხდება სახელების სივრცესთან მუშაობის დემონსტრირება. Bmw და Opel სახელების სივრცე უნდა მივუთითოთ პროგრამის დასაწყისში using დირექტივების შემდეგ.

```
//      პროგრამა 11.12
//      პროგრამაში ხდება სახელების სივრცესთან მუშაობის დემონსტრირება
//      Bmw სახელების სივრცის გამოცხადება
namespace Bmw
{
public class Manqana
{
public string ManqanisMarka;
}
}
//      Opel სახელების სივრცის გამოცხადება
namespace Opel
{
public class Manqana
{
public string ManqanisMarka;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Bmw.Manqana ChemiManqana1 = new Bmw.Manqana();
ChemiManqana1.ManqanisMarka = ”BMW”;
label1.Text = ChemiManqana1.ManqanisMarka.ToString();

Opel.Manqana ChemiManqana2 = new Opel.Manqana();
ChemiManqana2.ManqanisMarka = ”Opel”;
label2.Text = ChemiManqana2.ManqanisMarka.ToString();
}
```

პროგრამაში განსაზღვრულია ორი კლასი, რომლებსაც ერთნაირი სახელები აქვთ - Manqana. იმისათვის, რომ ეს ორი კლასი ერთმანეთისაგან განვასხვავოთ საჭიროა კლასის სახელის წინ სახელების სივრცის მითითება. მაგალითად, მოყვანილ სტრიქონში ხდება იმ Manqana კლასის ობიექტის შექმნა, რომელიც BMW სახელების სივრცეშია გამოცხადებული:

Bmw.Manqana ChemiManqana1 = new Bmw.Manqana();

ობიექტის შექმნის შემდეგ არ არის აუცილებელი სახელების სივრცის მითითება ობიექტის სახელის ან მისი წევრის წინ, მაგალითად,

ChemiManqana1.ManqanisMarka = ”BMW”;

using დირექტივა

ძალზე დამღლელია ყოველთვის მივუთითოთ სახელების სივრცე, როცა პროგრამაში ხდება ხშირი მიმართვები ამ სახელების სივრცის წევრებთან. ასეთ შემთხვევებში სასურველია using დირექტივის გამოყენება. ჩვენს მიერ განხილული ყველა პროგრამისათვის System სახელების სივრცე ხილული ხდება using დირექტივაში მისი მითითების შემდეგ. using დირექტივა შეგვიძლია, აგრეთვე, გამოვიყენოთ იმისათვის, რომ ხილული გავხადოთ ჩვენს მიერ შექმნილი სახელების სივრცე.

using დირექტივის სინტაქსია:

using *სახელების_სივრცის_სახელი*

ყველა წევრი, განსაზღვრული მითითებული სახელების სივრცის ფარგლებში, ხდება მისაწვდომი, ე.ი. ხდება მიმდინარე სახელების სივრცის ნაწილი, და შეიძლება გამოყენებული იყოს ადგილმდებარეობის სრული მითითების გარეშე. using დირექტივა მითითებული უნდა იყოს პროგრამის საწყისი კოდის შემცველი ფაილის დასაწყისში, ყველა სხვა გამოცხადების წინ.

ქვემოთ მოყვანილია პროგრამა, რომელშიც using დირექტივა გამოიყენება იმისთვის, რომ System და Opel სახელების სივრცე გავხადოთ ხილული.

```
//      პროგრამა 11.13
//      პროგრამაში ხდება using დირექტივასთან მუშაობის დემონსტრირება
using System;
using  Opel;
//      Opel სახელების სივრცის გამოცხადება
namespace Opel
{
public class Manqana
{
public string ManqanisMarka;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      Opel სიტყვის მითითება არ არის აუცილებელი ChemiManqana2 ობიექტის შექმნისას
Manqana ChemiManqana = new Manqana();
ChemiManqana.ManqanisMarka = ”Opel”;
label1.Text = ChemiManqana.ManqanisMarka.ToString();
}
```

ყურადღება მივაქციოთ იმას, რომ პროგრამაში ერთი სახელების სივრცე არ მალავს მეორე სახელების სივრცეს. სახელების სივრცის გამოცხადებისას იგი თავის სახელებს უმატებს მოცემულ მომენტში სხვა მისაწვდომ სახელებს. ამრიგად, ორივე სახელების სივრცე System და Opel, ხდება ხილული ამ პროგრამის კოდისათვის. გარდა ამისა, თუ სახელების ორ სივრცეში გამოცხადებულია ერთი და იგივე სახელის მქონე კლასები, მაშინ ამ კლასებთან მიმართვისას აუცილებლად უნდა მივუთითოთ სახელების სივრცის სახელი. წინააღმდეგ შემთხვევაში, კომპილატორი ვერ გაარკვევს თუ რომელ კლასთან უნდა შესრულდეს მიმართვა და გაიცემა შეტყობინება შეცდომის შესახებ.

ჩადგმული სახელების სივრცე

ერთი სახელების სივრცე შეიძლება მოვათავსოთ მეორის შიგნით. ასეთი გზით შეგვიძლია შევქმნათ სახელების სივრცის იერარქია. სახელების სივრცის იერარქია შეიძლება გადანაწილებული იყოს სხვადასხვა პროგრამაში (სხვადასხვა ფაილში). ასეთი გადანაწილების უპირატესობა იმაშია, რომ პროგრამისტებს შეუძლიათ ცალცალკე იმუშაონ ამ პროგრამებზე და შემდეგ ერთდროულად შეასრულონ მათი კომპილაცია.

ქვემოთ მოყვანილია ორი პროგრამა, რომლებშიც გადანაწილებულია Saxelebis_Sivrce სახელების სივრცე.

```
//      პროგრამა 11.14
//      პროგრამაში ხდება ჩადგმულ სახელების სივრცესთან მუშაობის დემონსტრირება
namespace Saxelebis_Sivrce
{
//      Chadgmuli_Sivrce1 სახელების სივრცე არის ჩადგმული სახელების NS1 სივრცე
namespace Chadgmuli_Sivrce1
{
public class ChemiKlasi
{
public string Metodi()
{
return ” Chadgmuli_Sivrce1 ”;
}
}
}
}
namespace Saxelebis_Sivrce.Chadgmuli_Sivrce2
{
public class ChemiKlasi
{
public string Metodi()
{
return ” Chadgmuli_Sivrce2 ”;
}
}
}
```

```
private void button1_Click(object sender, System.EventArgs e)
{
Saxelebis_Sivrce.Chadgmuli_Sivrce1.ChemiKlasi obieqti1 =
                                new Saxelebis_Sivrce. Chadgmuli_Sivrce1.ChemiKlasi();
Saxelebis_Sivrce.Chadgmuli_Sivrce2.ChemiKlasi obieqti2 =
                                new Saxelebis_Sivrce.Chadgmuli_Sivrce2.ChemiKlasi();
Saxelebis_Sivrce.Chadgmuli_Sivrce3.ChemiKlasi obieqti3 =
                                new Saxelebis_Sivrce.Chadgmuli_Sivrce3.ChemiKlasi();
obieqti1.Metodi();
obieqti2.Metodi();
obieqti3.Metodi();
}
//      პროგრამა 2 (ფაილი 2)
namespace Saxelebis_Sivrce
{
namespace Chadgmuli_Sivrce3
{
public class ChemiKlasi
{
public string Metodi()
{
return " Chadgmuli_Sivrce3 ";
}
}
}
}
```

პირველ პროგრამაში Saxelebis_Sivrce სახელების სივრცეში ჩადგმულია Chadgmuli_Sivrce1 და Chadgmuli_Sivrce2 სახელების სივრცეები. მეორე პროგრამაში Saxelebis_Sivrce სახელების სივრცეში ჩადგმულია Chadgmuli_Sivrce3 სახელების სივრცე. ChemiKlasi კლასთან მიმართვისას სრულად უნდა მივუთითოთ მისი ადგილმდებარეობა. ამისათვის, ჯერ უნდა მივუთითოთ ზედა დონის სახელების სივრცის სახელი, შემდეგ წერტილი და ბოლოს, ქვედა დონის სახელების სივრცის სახელი, მაგალითად, Saxelebis_Sivrce. Chadgmuli_Sivrce1.ChemiKlasi. მეორე პროგრამის შესატანად გავხსნათ Project მენიუ და შევასრულოთ Add New Item ბრძანება. გახსნილი ფანჯრის Templates ზონაში მოვნიშნოთ Code File ელემენტი და დავაჭიროთ Open კლავიშს. გახსნილ ფანჯარაში შევიტანოთ მეორე პროგრამა.

სახელების ჩადგმული სივრცე შეგვიძლია გამოვაცხადოთ ერთ სტრიქონში, ერთი namespace საკვანძო სიტყვის გამოყენებით. მაგალითად, კოდის ფრაგმენტი

```
namespace OuterNS
{
namespace InnerNS
{
//      კოდი
}
}
```

შეგვიძლია ასე გადავწეროთ

```
namespace OuterNS.InnerNs
{
//      კოდი
}
```

ავტომატურად განსაზღვრული სახელების სივრცე

თუ პროგრამაში არ არის გამოცხადებული სახელების სივრცე, მაშინ გამოიყენება ავტომატურად განსაზღვრული სახელების სივრცე. სახელების ასეთი სივრცე მოხერხებულია გამოვიყენოთ მოკლე და მარტივი პროგრამებისთვის. დიდი გამოყენებითი პროგრამების კოდი უნდა შედიოდეს სახელების რომელიმე სივრცეში. კოდი უნდა იყოს ინკაფსულირებული სახელების სივრცეში სახელების კონფლიქტების თავიდან აცილების მიზნით.

თავი12. სტრიქონები

სტრიქონები

სტრიქონები გამოიყენება Unicode სიმბოლოების მიმდევრობის შესანახად. ერთი Unicode სიმბოლო იკავებს 2 ბაიტს ანუ 16 ბიტს. სტრიქონები წარმოადგენენ System.String კლასის მიმართვითი ტიპის მქონე ობიექტებს. ისინი აღიწერება string სიტყვის საშუალებით. სტრიქონის (string ტიპის ობიექტის) შექმნის ყველაზე მარტივი საშუალებაა სტრიქონული ლიტერალის გამოყენება. მაგალითად,

```
string striqoni = "ეს არის სტრიქონული ლიტერალი";
```

აქ striqoni ცვლადი არის string ტიპის მიმართვითი ცვლადი, რომელსაც ენიჭება სტრიქონულ ლიტერალზე (“ეს არის სტრიქონული ლიტერალი”) მიმართვა. ამ შემთხვევაში striqoni ცვლადის ინიციალიზება ხდება სიმბოლოების მიმდევრობით - “ეს არის სტრიქონული ლიტერალი”.

ისევე როგორც მასივებში, აქაც პირველი ელემენტის (სიმბოლოს) ინდექსია 0. ინდექსი გამოიყენება მხოლოდ სიმბოლოს მისაღებად. მაგალითად,

```
string striqoni = "კომპიუტერი";
label1.Text = striqoni[2];
```

ორივე ოპერატორის შესრულების შედეგად label კომპონენტში გამოჩნდება ასო “მ”. რადგან, სტრიქონები ობიექტებს წარმოადგენენ, ამიტომ მათ Length თვისება აქვთ. მასში ინახება სტრიქონის სიგრძე (სტრიქონში სიმბოლოების რაოდენობა).

სტრიქონი შეიძლება შეიცავდეს მმართველ სიმბოლოებს. მაგლითად,

```
string striqoni = "საბა \n ანა";
label1.Text = striqoni;
```

ამ სტრიქონების შესრულების შედეგად label1 კომპონენტის პირველ სტრიქონში გამოჩნდება “საბა”, მეორე სტრიქონში კი - “ანა”.

შეგვიძლია შევქმნათ ისეთი სტრიქონებიც, რომლებშიც მმართველი სიმბოლოები არ იმუშავებენ. ასეთი სტრიქონები @ სიმბოლოთი იწყება. მაგალითად,

```
string striqoni = @“ საბა \n ანა”;
label1.Text = striqoni;
```

ამ სტრიქონების შესრულების შედეგად label1 კომპონენტში გამოჩნდება სტრიქონი - “საბა \n ანა”.

სტრიქონებთან სამუშაო მეთოდები

String კლასის თვისება და მეთოდები მოყვანილია ცხრილებში 4.1 და 4.2.

მოყვანილ პროგრამაში ხდება Length თვისებასთან მუშაობის დემონსტრირება.

```
{
string striqoni1 = @" საბა \n ანა \n";
label1.Text = striqoni1.Length.ToString() + '\n';                    // გაიცემა 15
string striqoni2 = " საბა \n ანა \n";
label2.Text = striqoni2.Length.ToString() + '\n';                    // გაიცემა 13
}
```

ცხრილი 4.1. String კლასის თვისება.

თვისება	ტიპი	აღწერა
Length	int	შეიცავს სტრიქონში სიმბოლოების რაოდენობას

ცხრილი 4.2. String კლასის მეთოდები.

მეთოდი	აღწერა
int Compare(string სტრიქონი*1*, string სტრიქონი*2*)	ადარებს ორ სტრიქონს. შედარების დროს გაითვალისწინება ენისა და კულტურის თავისებურებები
CompareOrdinal(string სტრიქონი*1*, string სტრიქონი*2*)	ადარებს ორ სტრიქონს. შედარების დროს არ გაითვალისწინება ენისა და კულტურის თავისებურებები
string Concat(string სტრიქონი*1*, string სტრიქონი*2*)	გასცემს ორი ან მეტი სტრიქონის გაერთიანებას
string Copy(string სტრიქონი)	გასცემს მითითებული სტრიქონის ასლს
bool Equals(string სტრიქონი*1*, string სტრიქონი*2*)	გასცემს true მნიშვნელობას, თუ სტრიქონები ტოლია, წინააღმდეგ შემთხვევაში false მნიშვნელობას
string Format(string სტრიქონი, object ობიექტი)	გასცემს მითითებული ფორმატის მიხედვით დაფორმატებულ სტრიქონს
string Intern(string სტრიქონი)	გასცემს სისტემურ მიმართვას მითითებულ სტრიქონზე
string IsInterned(string სტრიქონი)	გასცემს მიმართვას მითითებულ სტრიქონზე
string Join(string გამყოფი, string[ ] მასივი)	მეთოდი გამყოფს ათავსებს მასივის ელემენტებს შორის და გასცემს მიღებულ სტრიქონს
int CompareTo(string სტრიქონი)	გამომძახებელ სტრიქონს ადარებს მითითებულ სტრიქონთან
void CopyTo(int საწყისი_ინდექსი, char[] სიმბოლოების_მასივი, int საბოლოო_ინდექსი, int რაოდენობა)	გამომძახებელი სტრიქონის მითითებული პოზიციიდან სიმბოლოების მასივის მითითებულ პოზიციაში გადაწერს მითითებული რაოდენობის Unicode სიმბოლოს
bool EndsWith(string სტრიქონი)	ამოწმებს მთავრდება თუ არა გამომძახებელი სტრიქონი მითითებული სტრიქონით
Type GetType()	გასცემს მიმდინარე ეგზემპლარის ტიპს
int IndexOf(string სტრიქონი)	გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც პირველად იყო ნაპოვნი მითითებული ქვესტრიქონი
int IndexOfAny(char[ ] მასივი)	გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც პირველად იყო ნაპოვნი სიმბოლოების მასივის ნებისმიერი სიმბოლო

ცხრილი 4.2. (გაგრძელება)

string Insert(int ინდექსი, string სტრიქონი)	გამომძახებელ სტრიქონში ინდექსი პოზიციიდან დაწყებული ჩასვამს მითითებულ სტრიქონს
int LastIndexOf(string სტრიქონი)	გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც უკანასკნელად იყო ნაპოვნი მითითებული ქვესტრიქონი
int LastIndexOfAny(char[ ] მასივი)	გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც უკანასკნელად იყო ნაპოვნი სიმბოლოების მასივის ნებისმიერი სიმბოლო
string PadLeft(int სიგანე, char სიმბოლო)	სტრიქონის სიმბოლოებს ასწორებს მარცხენა საზღვარზე, უმატებს რა სტრიქონის დასაწყისს ინტერვალებს ან მითითებულ სიმბოლოს იმდენჯერ, რომ სტრიქონის სიგრძე გახდეს მითითებული სიგანის ტოლი
string PadRight(int სიგანე, char სიმბოლო)	სტრიქონის სიმბოლოებს ასწორებს მარჯვენა საზღვარზე, უმატებს რა სტრიქონის ბოლოს ინტერვალებს ან მითითებულ სიმბოლოს იმდენჯერ, რომ სტრიქონის სიგრძე გახდეს მითითებული სიგანის ტოლი
string Remove(int ინდექსი, int რაოდენობა)	გამომძახებელ სტრიქონში ინდექსი პოზიციიდან დაწყებული წაშლის მითითებული რაოდენობის სიმბოლოს
string Replace(string ძველი_სტრიქონი, string ახალი_სტრიქონი)	გამომძახებელ სტრიქონში ძველ სტრიქონს ახალი სტრიქონით შეცვლის
string[ ] Split(params char[ ] გამყოფი)	სტრიქონს ჰყოფს სტრიქონების მასივად მითითებული გამყოფის გამოყენებით.
bool StartsWith(string სტრიქონი)	ამოწმებს იწყება თუ არა გამომძახებელი სტრიქონი მითითებული სტრიქონით
string Substring (int ინდექსი, int რაოდენობა)	გამომძახებელი სტრიქონის ინდექსი პოზიციიდან გასცემს მითითებული რაოდენობის სიმბოლოს
char[ ] ToCharArray()	სტრიქონის სიმბოლოებს გადაწერს სიმბოლოების მასივში
string ToLower()	გამომძახებელი სტრიქონის ყველა სიმბოლო გადაყავს ქვედა რეგისტრში
string ToString()	გამომძახებელ ობიექტს სტრიქონად გარდაქმნის
string ToUpper()	გამომძახებელი სტრიქონის ყველა სიმბოლო გადაყავს ზედა რეგისტრში

ცხრილი 4.2. (გაგრძელება)

string Trim()	გამომძახებელ სტრიქონში შლის საწყის და ბოლო ინტერვალებს
string TrimEnd(char[ ] სიმბოლოები)	გამომძახებელი სტრიქონის ბოლოში შლის სიმბოლოების მასივში მითითებულ სიმ-ბოლოებს
string TrimStart(char[ ] სიმბოლოები)	გამომძახებელი სტრიქონის დასაწყისში შლის სიმბოლოების მასივში მითითებულ სიმბოლოებს

განვიხილოთ ზოგიერთი მეთოდი.

ქვემოთ მოყვანილ პროგრამაში ხდება **Insert, Remove** და **Replace** მეთოდების გამოყენების დემონსტრირება.

```
{
//      პროგრამა 12.1
//      პროგრამაში ხდება Insert, Remove და Replace მეთოდებთან მუშაობის
დემონსტრირება
string str1, str2 = "C# დაპროგრამების ენაა", str3 = ”თანამედროვე ”;
str1 = str2.Insert(17, str3);      // მიიღება სტრიქონი “C# დაპროგრამების თანამედროვე
ენაა”
label1.Text = str1;
str1 = str2.Remove(3, 14);                    // მიიღება სტრიქონი “C# ენაა”
label2.Text = str1;
str1 = str2.Replace(“დაპროგრამების ”, str3);   // მიიღება სტრიქონი “C# თანამედროვე ენაა”
label3.Text = str1;
}
```

პროგრამის str1 = str2.Insert(17, str3); სტრიქონში str2 არის გამომძახებელი სტრიქონის სახელი. მის მარჯვნივ წერტილის დასმის შემდეგ გაიხსნება სტრიქონებთან სამუშაო მეთოდების სია. ვირჩევთ Insert მეთოდს. მისი შესრულების შედეგად str2 სტრიქონის მე-17 პოზიციიდან მოხდება str3 სტრიქონის ჩასმა, ანუ “თანამედროვე” სტრიქონის ჩასმა. შედეგად, მიიღება სტრიქონი “C# დაპროგრამების თანამედროვე ენაა”, რომელიც მიენიჭება str1 ცვლადს.

+ ოპერატორი გამოიყენება სტრიქონების კონკატენაციის (შეერთების) ოპერაციის შესასრულებლად. განვიხილოთ კოდის ფრაგმენტი:

```
string striqoni1 = "ეს არის ";
string striqoni2 = "კომ";
string striqoni3 = "პიუტერი";
string striqoni   = striqoni1 + striqoni2 + striqoni3;
```

ამ ოპერატორების შესრულების შედეგად striqoni ცვლადს მიენიჭება სტრიქონი - "ეს არის კომპიუტერი".

ორი სტრიქონის შესადარებლად შეგვიძლია == ოპერატორის გამოყენება.

მოყვანილ პროგრამაში ხდება **Copy, CompareTo, IndexOf, LastIndexOf, Substring** მეთოდების მუშაობის დემონსტრირება.

```
{
//      პროგრამა 12.2
```

```
//      პროგრამაში ხდება Copy, CompareTo, IndexOf, LastIndexOf, Substring
//      მეთოდებთან მუშაობის დემონსტრირება
string striqoni1, striqoni2 = "ერთი ორი სამი ერთი ორი სამი",
         striqoni3 = textBox1.Text;
int shedegi, indexi;

// striqoni2  სტრიქონი გადაიწერება striqoni1 სტრიქონში
striqoni1 = string.Copy(striqoni2);
label1.Text = striqoni1;

striqoni1 = striqoni2.Substring(9, 13);          // მიიღება სტრიქონი “ერთი ორი სამი”
label2.Text = striqoni1;

indexi = striqoni2.IndexOf(”ორი”);               // გაიცემა 5
label3.Text = indexi.ToString();
indexi = striqoni2.LastIndexOf(”ორი”);           // გაიცემა 19
label4.Text = indexi.ToString();

// striqoni2  და striqoni4 სტრიქონების შედარება
shedegi = striqoni2.CompareTo(striqoni3);
if ( shedegi == 0 ) label5.Text = "სტრიქონები ერთმანეთის ტოლია";
        else if ( shedegi > 0 ) label5.Text = " striqoni2 > striqoni3";
                        else label5.Text = " striqoni2 < striqoni3";
}
```

Compare() და **Equals()** მეთოდები. Equals() მეთოდი ორ სტრიქონს ადარებს და გასცემს true მნიშვნელობას თუ სტრიქონები ტოლია, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას. მისი სინტაქსია:

Equals*(სტრიქონი1, სტრიქონი2)*

მაგალითი.

```
{
string striqoni1 = "საბა ანა ლიკა რომანი",
     striqoni2 = "ანა რომანი საბა ლიკა";
bool shedegi = striqoni1.Equals(striqoni2);
if ( shedegi == true ) label1.Text = "სტრიქონები ერთნაირია";
        else label1.Text = " სტრიქონები ერთნაირი არ არის";
}
```

Compare() მეთოდი ორ სტრიქონს ადარებს ენის, ეროვნული და კულტურული თავისებურებების გათვალისწინებით. თუ პირველი სტრიქონი მეტია მეორეზე, მაშინ გაცემა 1. თუ პირველი სტრიქონი ნაკლებია მეორეზე, მაშინ გაიცემა -1. თუ სტრიქონები ტოლია, მაშინ გაიცემა 0. ეს მეთოდი გადატვირთულია და ამიტომ აქვს რამდენიმე ვერსია. ჩვენ განვიხილავთ რამდენიმე მათგანს.

Compare() მეთოდის უმარტივესი ვერსიის სინტაქსია:

String.Compare*(სტრიქონი1, სტრიქონი2)*

მაგალითი.

```
{
```

```
string striqoni1 = "საბა ანა ლიკა რომანი",
      striqoni2 = "ანა რომანი საბა ლიკა";
int  shedegi = String.Compare(striqoni1, striqoni2);
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";
        else label1.Text = " სტრიქონები ერთნაირი არ არის";
}
```

Compare() მეთოდის მეორე ვერსია შედარების დროს ითვალისწინებს სიმბოლოების რეგისტრს. მისი სინტაქსია:

String.Compare*(სტრიქონი1, სტრიქონი2, რეგისტრი)*

თუ რეგისტრი იღებს true მნიშვნელობას, მაშინ შედარების დროს რეგისტრი არ იქნება გათვალისწინებული. თუ ის იღებს false მნიშვნელობას, მაშინ შედარების დროს რეგისტრი იქნება გათვალისწინებული.

მაგალითი.

```
{
string striqoni1 = "საბა ანა ლიკა რომანი",
      striqoni2 = "ანა რომანი საბა ლიკა";
int  shedegi = String.Compare(striqoni1, striqoni2, true);
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";
        else label1.Text = " სტრიქონები ერთნაირი არ არის";
}
```

Compare() მეთოდის მესამე ვერსია საშუალებას გვაძლევს შევადაროთ ორი სტრიქონის ნაწილები (ქვესტრიქონები). მისი სინტაქსია:

String.Compare*(სტრიქონი1, ინდექსი1, სტრიქონი2, ინდექსი2, სიმბოლოების_რაოდენობა)*

მოყვანილ პროგრამაში ხდება ამ მეთოდთან მუშაობის დემონსტრირება.

```
{
string striqoni1 = "საბა ანა ლიკა რომანი",
        striqoni2 = "ანა რომანი საბა ლიკა";
int  shedegi = String.Compare(striqoni1, 5, striqoni2, 5, 4);
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";
        else label1.Text = " სტრიქონები ერთნაირი არ არის";
}
```

მოყვანილ პროგრამაში ხდება **Concat()** მეთოდთან მუშაობის დემონსტრირება.

```
{
string striqoni1 = "საბა";
string striqoni2 = string.Concat(striqoni1, " ანა");
string striqoni3 = striqoni2 + " ლიკა";
label1.Text = striqoni3;
}
```

როგორც პროგრამის კოდიდან ჩანს სტრიქონების კონკატენაციისათვის შეიძლება გამოვიყენოთ + ოპერატორიც.

Format() მეთოდი სტრიქონის ფორმატირებისათვის იყენებს ფორმატირების სიმბოლოებს, რომლებიც მოყვანილია ცხრილში 12.3.

ცხრილი 12.3. სტრიქონის დაფორმატების სიმბოლოები.

დაფორმატების სიმბოლო	აღწერა
f ან F	რიცხვს აფორმატებს როგორც წილადს
e ან E	რიცხვს აფორმატებს როგორც რიცხვს ექსპონენციალური წარმოდგენით
p ან P	რიცხვს აფორმატებს როგორც პროცენტს
n ან N	რიცხვს აფორმატებს როგორც რიცხვს თანრიგების გამყოფებით
c ან C	რიცხვს აფორმატებს როგორც თანხას ადგილობრივ ვალუტაში
d ან D	რიცხვს აფორმატებს როგორც ათობით რიცხვს
g ან G	რიცხვს აფორმატებს როგორც წილადს ან როგორც რიცხვს ექსპონენციალური წარმოდგენით
x ან X	მთელი რიცხვი გადაჰყავს თექვსმეტობით წარმოდგენაში

მოყვანილ პროგრამაში ხდება **Format()** მეთოდის გამოყენების დემონსტრირება.

```
{
//      პროგრამა 12.3
//      პროგრამაში ხდება Format() მეთოდის გამოყენების დემონსტრირება
label1.Text = "";
string striqoni;
int mteli = 12;
double wiladi_d = 1234.56789;
float wiladi_f = 1234.56789f;
decimal valuta = 12345.67m;
striqoni = String.Format("{0:D5}\n", mteli);              // 00012
label1.Text += striqoni;
striqoni = String.Format("{0:X}\n", mteli);               // C
label1.Text += striqoni;
striqoni = String.Format("{0:P3}\n", mteli);              // 1 200,000%
label1.Text += striqoni;
striqoni = String.Format("{0:C3}\n", valuta);             // 12 345,670 Lari
label1.Text += striqoni;
striqoni = String.Format("{0:F3}\n", wiladi_d);           // 1234,568
label1.Text += striqoni;
striqoni = String.Format("{0:F3}\n", wiladi_f);           // 1234,568
label1.Text += striqoni;
striqoni = String.Format("{0:P3}\n", wiladi_d);           // 123 456,789%
label1.Text += striqoni;
striqoni = String.Format("{0:E3}\n", wiladi_d);           // 1,235E+003
label1.Text += striqoni;
striqoni = String.Format("{0:N3}\n", wiladi_d);           // 1 234,568
label1.Text += striqoni;
```

```
striqoni = String.Format("{0:G3}\n", wiladi_d);                // 1,23E+03
label1.Text += striqoni;
}
```

{0:F3} ფორმატში პირველი სიმბოლო “0” მიუთითებს, რომ უნდა დაფორმატდეს რიგით პირველი ცვლადი. მეორე სიმბოლო მიუთითებს, რომ ათწილადში წილად ნაწილს უნდა დაეთმოს 3 თანრიგი. მოყვანილ მაგალითში დაფორმატდება რიგით მეორე ცვლადი - wiladi_f, რადგან ფორმატში მითითებულია 1.

```
{
double wiladi_d = 9234.56781;
float wiladi_f = 1234.56789f;
striqoni = String.Format("{1:F3}\n", wiladi_d, wiladi_f);          // 1234,568
label1.Text = striqoni;
}
```

თუ ფორმატში არ მივუთითებთ თანრიგების რაოდენობას წილადი ნაწილისათვის, მაშინ ის აიღება ორის ტოლი. მაგალითად,

```
{
float wiladi_f = 1234.56789f;
string striqoni = String.Format("{0:F}\n", wiladi_f);              // 1234,56
label1.Text = striqoni;
}
```

მოყვანილ პროგრამაში ხდება Join() მეთოდთან მუშაობის დემონსტრირება.

```
{
string[] striqonebis_masivi = { "საბა", "ანა", "ლიკა", "რომანი" };
string striqoni = String.Join("-", striqonebis_masivi);
label17.Text = striqoni;                    //      striqoni =  “საბა-ანა-ლიკა-რომანი”
}
```

მოყვანილ პროგრამაში ხდება Split() მეთოდთან მუშაობის დემონსტრირება.

```
{
string striqoni = "რომანი,ანა:საბა ლიკა.ნატა";
string[] shedegi = striqoni.Split(new Char[] { ' ', ',', '.', ':' });

foreach ( string sityva in shedegi )
{
if ( sityva.Trim() != "" )
        label1.Text += sityva + '\n';          // shedegi = "რომანი ანა საბა ლიკა ნატა"
}
}
```

მოყვანილ პროგრამაში ხდება **StartWith()** და **EndWith()** მეთოდთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 12.4
//      პროგრამაში ხდება StartWith() და EndWith() მეთოდებთან მუშაობის
//      დემონსტრირება
string striqoni1, striqoni2;
striqoni1 = textBox1.Text;
```

```
striqoni2 = textBox2.Text;
if ( striqoni1.StartsWith(striqoni2) )
                label1.Text = "სტრიქონი იწყება " + striqoni2 + " სტრიქონით";
        else  label1.Text = " სტრიქონი არ იწყება " + striqoni2 + " სტრიქონით";
if ( striqoni1.EndsWith(striqoni2) )
                label2.Text = "სტრიქონი მთავრდება " + striqoni2 + " სტრიქონით";
        else  label2.Text = " სტრიქონი არ მთავრდება " + striqoni2 + " სტრიქონით";
}
```

IndexOfAny() და **LastIndexOfAny()** მეთოდები გადატვირთულია და ამიტომ აქვთ რამდენიმე ვერსია. ყველაზე მარტივი ვერსიის სინტაქსია:

int IndexOfAny(char[ ] *სიმბოლოების_მასივი)*

int LastIndexOfAny(char[ ] *სიმბოლოების_მასივი)*

ამ მეთოდების მეორე ვერსია საშუალებას გვაძლევს ძებნა დავიწყოთ მითითებული ინდექსდან. ამ ვერსიის სინტაქსია:

int IndexOfAny*(*char[ ] *სიმბოლოების_მასივი, ინდექსი)*

int LastIndexOfAny*(*char[ ] *სიმბოლოების_მასივი, ინდექსი)*

ამ მეთოდების მესამე ვერსია საშუალებას გვაძლევს მივუთითოთ შესამოწმებელი სიმბოლოების რაოდენობა. ამ ვერსიის სინტაქსია:

int IndexOfAny*(*char[ ] *სიმბოლოების_მასივი, ინდექსი, სიმბოლოების_რაოდენობა)*

int LastIndexOfAny*(*char[ ] *სიმბოლოების_მასივი, ინდექსი, სიმბოლოების_რაოდენობა)*

მოყვანილ პროგრამაში ხდება IndexOfAny() და LastIndexOfAny() მეთოდებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 12.5
//      პროგრამაში ხდება IndexOfAny() და LastIndexOfAny()მეთოდებთან მუშაობის
//      დემონსტრირება
char[] simboloebis_masivi = { 'ნ', 'ა' };
string striqoni1 = "ლიკა, ანა და რომანი";

int indexi1 = striqoni1.IndexOfAny(simboloebis_masivi);                 // indexi1 = 3
int indexi2 = striqoni1.LastIndexOfAny(simboloebis_masivi);             // indexi2 = 17
label1.Text = indexi1.ToString() + "    " + indexi2.ToString() + '\n';

int indexi3 = striqoni1.IndexOfAny(simboloebis_masivi, 5);              // indexi3 = 6
int indexi4 = striqoni1.LastIndexOfAny(simboloebis_masivi, 5);          // indexi4 = 3
label2.Text += indexi3.ToString() + "    " + indexi4.ToString() + '\n';

int indexi5 = striqoni1.IndexOfAny(simboloebis_masivi, 5, 6);           // indexi5 = 6
int indexi6 = striqoni1.LastIndexOfAny(simboloebis_masivi, 5, 6);       // indexi5 = 3
label3.Text += indexi5.ToString() + "    " + indexi6.ToString() + '\n';     // 6 3
}
```

PadLeft() და **PadRight()** მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
{
striqoni2 = "საბა";
```

```
string striqoni1 = striqoni2.PadLeft(10, '*');          //      striqoni1 = “******საბა”
string striqoni3 = striqoni2.PadRight(10, '*');         //      striqoni3 = “საბა******”
label1.Text = striqoni1.ToString() + '\n';
label2.Text = striqoni3.ToString() + '\n';
}
```

Trim(), TrimStart() და **TrimEnd()** მეთოდები გადატვირთულია. Trim() მეთოდის ყველაზე მარტივი ვერსია, რომელშიც არ ეთითება პარამეტრი, ახდენს ინტერვალების წაშლას სტრიქონის დასაწყისში და ბოლოში. TrimStart() მეთოდის ყველაზე მარტივი ვერსია ახდენს ინტერვალების წაშლას სტრიქონის დასაწყისში. TrimEnd() მეთოდის ყველაზე მარტივი ვერსია ახდენს ინტერვალების წაშლას სტრიქონის ბოლოში. ამ მეთოდების სინტაქსია:

```
string Trim()
string TrimStart()
string TrimEnd()
```

ამ მეთოდების მეორე ვერსია საშუალებას გვაძლევს სტრიქონს დასაწყისში და ბოლოში მოვაცილოთ მითითებული სიმბოლოები. ამ ვერსიის სინტაქსია:

```
string Trim(char[ ] სიმბოლოების_მასივი)
string TrimStart(char[ ] სიმბოლოების_მასივი)
string TrimEnd(char[ ] სიმბოლოების_მასივი)
```

მოყვანილ პროგრამაში ხდება Trim(), TrimStart() და TrimEnd() მეთოდებთან მუშაობის დემონსტრირება.

```
{
string striqoni2, striqoni3, striqoni4 = ";,რომანი.,;";
char[] simboloebi = { '.', ';', ',' };

label1.Text = striqoni4.Trim(simboloebi);
striqoni2   = striqoni4.TrimStart(simboloebi);      // striqoni2   = “რომანი.,;”
striqoni3   = striqoni4.TrimEnd(simboloebi);        // striqoni3   = “;,რომანი”
label2.Text = striqoni2.ToString() + '\n';
label3.Text = striqoni3.ToString();
}
```

მოყვანილ პროგრამაში ხდება **ToLower()** და **ToUpper()** მეთოდებთან მუშაობის დემონსტრირება.

```
{
string striqoni1 = "ANA SABA";
string striqoni2 = "lika romani";

label15.Text = striqoni1.ToLower();                 //      “ana saba”
label17.Text = striqoni2.ToUpper();                 //      “LIKA ROMANI”
}
```

სტრიქონების მასივები

სტრიქონების მასივის გამოცხადებისათვის string სიტყვის შემდეგ კვადრატული ფრჩხილები უნდა მივუთითოთ. ქვემოთ მოყვანილ პროგრამაში ხდება str სტრიქონების მასივის გამოცხადება და ინიციალიზაცია. იგი სამ სტრიქონულ ლიტერალს შეიცავს.

```
{
//      პროგრამა 12.6
//      პროგრამაში ხდება სტრიქონების მასივთან მუშაობის დემონსტრირება
string[] striqonebis_masivi = { "სტრიქონული ", "მასივის ", "მაგალითი" };
// მასივის გამოტანა label კომპონენტში
for ( int indexi = 0; indexi < striqonebis_masivi.Length; indexi++ )
        label1.Text += striqonebis_masivi[indexi] + " ";
// სტრიქონული მასივის მნიშვნელობების შეცვლა
striqonebis_masivi[0] = "მეორე ";
striqonebis_masivi[1] = "სტრიქონული ";
striqonebis_masivi[2] = textBox1.Text;
label1.Text += '\n';
// მასივის გამოტანა label კომპონენტში
foreach ( string striqoni in striqonebis_masivi )
        label1.Text += striqoni + " ";
}
```

სტრიქონების უცვლელობა

string ტიპის ობიექტების ერთ-ერთი მნიშვნელოვანი თავისებურებაა ის, რომ სტრიქონში ერთხელ შექმნილი სიმბოლოების მიმდევრობა აღარ შეიცვლება, ე.ი. არ შეიძლება ინდექსის გამოყენება სტრიქონის რომელიმე სიმბოლოსათვის ახალი მნიშვნელობის მისანიჭებლად. მაგალითად, დაუშვებელია ასეთი მინიჭება:

```
striqoni1[1] = 'ს';
```

იმისათვის, რომ სტრიქონში სიმბოლოები შევცვალოთ, უნდა შევქმნათ ახალი სტრიქონი, რომელიც საჭირო ცვლილებებს შეიცავს. ამისათვის, შეგვიძლია Replace ან Substring მეთოდის გამოყენება. ქვემოთ მოყვანილ პროგრამაში ხდება სტრიქონის შეცვლის დემონსტრირება.

```
{
//      პროგრამა 12.7
//      პროგრამაში ხდება სტრიქონის შეცვლის დემონსტრირება
string str1 = "კომპიუტერი", str2, str3;
str1 = "დაპროგრამება";
// str1[0] = 'ბ';                       // ასეთი მინიჭება დაუშვებელია
str2 = str1.Replace('ა', 'ბ');          // str2 სტრიქონში ჩაიწერება შეცვლილი სტრიქონი
str3 = str1.Substring(2, 3);            // str3 სტრიქონში ჩაიწერება ახალი სტრიქონი
str1 = str1.Replace("პროგ", "აბგ"); // ასეთი მინიჭება დასაშვებია. ამ შემთხვევაში
                                        // str1 სტრიქონშივე ჩაიწერება შეცვლილი სტრიქონი
label1.Text = str3;
label2.Text = str1;
}
```

დინამიური სტრიქონები

დინამიურ სტრიქონებთან სამუშაოდ გამოიყენება System.Text.StringBuilder კლასი. StringBuilder კლასის მეთოდები მუშაობენ უფრო სწრაფად, რადგან ისინი არ ქმნიან სტრიქონის ასლს და უშუალოდ სტრიქონთან მუშაობენ. დინამიურ სტრიქონებში, String კლასის სტრიქონებისაგან განსხვავებით, შესაძლებელია სიმბოლოების პირდაპირ შეცვლა. პროგრამის დასაწყისში, რომელიც დინამიურ სტრიქონებთან მუშაობს, უნდა მოვათავსოთ დირექტივა:

```
using System.Text;
```

დინამიური სტრიქონის შექმნა

StringBuilder კლასის კონსტრუქტორი გადატვირთულია, ამიტომ არსებობს ამ კლასის კონსტრუქტორის რამდენიმე ვარიანტი. ჩვენ განვიხილავთ დინამიური სტრიქონის შექმნის რამდენიმე გზას. ერთ-ერთი გზაა ცარიელი დინამიური სტრიქონის შექმნა:

```
StringBuilder DinamiuriStriqoni1 = new StringBuilder();
```

თუ კოდის კომპილაციის დროს გამოჩნდა შეტყობინება იმის შესახებ, რომ StringBuilder კლასი არ არის განსაზღვრული, მაშინ ის უნდა მივუთითოთ შემდეგნაირად:

```
System.Text.StringBuilder
```

შექმნის დროს დინამიური სტრიქონის ტევადობაა 16 სიმბოლო. ახალი სიმბოლოს დამატების შემდეგ დინამიური სტრიქონის ტევადობა ავტომატურად იზრდება. ტევადობა შეგვიძლია მივუთითოთ დინამიური სტრიქონის შექმნის დროს. მაგალითად,

```
{
int Tevadoba = 25;
StringBuilder DinamiuriStriqoni2 = new StringBuilder(Tevadoba);
}
```

დინამიური სტრიქონის შექმნისას კონსტრუქტორს შეგვიძლია გადავცეთ, აგრეთვე, მაქსიმალური ტევადობა. მაგალითად,

```
{
int Tevadoba = 25;
int MaxTevadoba = 100;
StringBuilder DinamiuriStriqoni3 = new StringBuilder(Tevadoba, MaxTevadoba);
}
```

შექმნის დროს დინამიურ სტრიქონს ავტომატურად ენიჭება 2147483647-ის ტოლი მაქსიმალური ტევადობა.

დინამიური სტრიქონის შექმნისას კონსტრუქტორს შეგვიძლია გადავცეთ სტრიქონი, რომელიც დინამიურ სტრიქონში უნდა ჩაიწეროს. მაგალითად,

```
{
string Striqoni = "ანა და საბა";
StringBuilder DinamiuriStriqoni4 = new StringBuilder(Striqoni);
}
```

დინამიური სტრიქონის შექმნის დროს კონსტრუქტორს შეგვიძლია გადავცეთ სტრიქონი; საწყისი ინდექსი, საიდანაც უნდა დაიწყოს სიმბოლოების გადაწერა;

გადასაწერი სიმბოლოების რაოდენობა და დინამიური სტრიქონის ტევადობა. მაგალითად,

```
{
string Striqoni = "ანა და საბა";
int Tevadoba = 50;
int SawyisiIndexi = 4;
int SimboloebisRaodenoba = 7;
StringBuilder DinamiuriStriqoni5 = new
        StringBuilder(Striqoni, SawyisiIndexi, SimboloebisRaodenoba, Tevadoba);
label1.Text = DinamiuriStriqoni6;          //      DinamiuriStriqoni6 = “და საბა”
}
```

StringBuilder კლასის თვისებები და მეთოდები

ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 12.2 და 12.3. განვიხილოთ ზოგიერთი მათგანი.

Append() და **AppendFormat()** მეთოდები. Append() მეთოდი ახდენს მითითებული ობიექტის სტრიქონად გარდაქმნას და დინამიური სტრიქონისათვის მის დამატებას. ობიექტს შეიძლება ჰქონდეს ნებისმიერი ჩადგმული ტიპი, როგორიცაა, bool, byte, int, double და ა.შ. Append() მეთოდის უმარტივეს ვერსიას აქვს შემდეგი სინტაქსი:

```
stringBuilder.Append(მნიშვნელობა)
```

აქ stringBuilder არის StringBuilder კლასის ობიექტი, სიდიდე კი დინამიური სტრიქონისათვის დასამატებელი მნიშვნელობა.

ცხრილი 12.2. StringBuilder კლასის თვისებები.

თვისება	ტიპი	აღწერა
Capacity	int	გასცემს ან აყენებს დინამიური სტრიქონის ტევადობას
Length	int	გასცემს ან აყენებს დინამიურ სტრიქონში სიმბოლოების რაოდენობას
MaxCapacity	int	გასცემს დინამიური სტრიქონის მაქსიმალურ ტევადობას

ცხრილი 12.3. StringBuilder კლასის მეთოდები.

მეთოდი	დაბრუნებული მნიშვნელობის ტიპი	აღწერა
Append()	StringBuilder	ობიექტის სტრიქონულ წარმოდგენას ამატებს დინამიური სტრიქონის ბოლოში
AppendFormat()	StringBuilder	დაფორმატებულ სტრიქონს ამატებს დინამიური სტრიქონის ბოლოში
EnsureCapacity()	int	ამოწმებს, ტოლია თუ მეტი დინამიური სტრიქონის მიმდინარე ტევადობა მითითებულ მნიშვნელობაზე

ცხრილი 12.3. (გაგრძელება)

Equals()	bool	გასცემს true მნიშვნელობას თუ დინამიური სტრიქონი მითითებული ობიექტის ტოლია, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას
GetType()	Type	გასცემს ობიექტის ტიპს
Insert()	StringBuilder	მითითებული ობიექტის სტრიქონულ წარმოდგენას ამატებს დინამიურ სტრიქონში დაწყებული მითითებული პოზიციიდან
Remove()	StringBuilder	დინამიური სტრიქონიდან შლის მითითებული რაოდენობის სიმბოლოს დაწყებული მითითებული პოზიციიდან
Replace()	StringBuilder	მითითებულ სიმბოლოს ან ქვესტრიქონს დინა-მიურ სტრიქონში ყველგან ცვლის მითითებული სიმბოლოთი ან ქვესტრიქონით.
ToString()	string	დინამიურ სტრიქონს გარდაქმნის სტრიქონად

მოყვანილ პროგრამაში ხდება სხვადასხვა ტიპის ობიექტების დინამიური სტრიქონისათვის დამატების დემონსტრირება.

```
{
//      პროგრამა 12.8
//      პროგრამაში ხდება დინამიური სტრიქონისათვის ელემენტების დამატება
string striqoni = "საბა და ანა";
int    ricxvi = 5;
double wiladi = 2.8765;
bool logikuri = true;
StringBuilder DinamiuriStriqoni = new StringBuilder();
DinamiuriStriqoni.Append(striqoni);         // DinamiuriStriqoni = “საბა და ანა”
DinamiuriStriqoni.Append(wiladi);           // DinamiuriStriqoni = “საბა და ანა2.8765”
DinamiuriStriqoni.Append(logikuri);         // DinamiuriStriqoni = “საბა და ანა2.8765true”
DinamiuriStriqoni.Append(ricxvi);           // DinamiuriStriqoni = “საბა და ანა2.8765true5”
label1.Text = DinamiuriStriqoni.ToString();
}
```

Append() მეთოდის მეორე ვერსია საშუალებას გვაძლევს დინამიურ სტრიქონს დავუმატოთ გამეორებადი სიმბოლოს სერია. მეთოდის სინტაქსია:
stringBuilder.Append(სიმბოლო, გამეორების_რაოდენობა)

მაგალითად, DinamiuriStriqoni.Append(‘ა’, 10); მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს დაემატება 10 ცალი ‘ა’ სიმბოლო.

დინამიურ სტრიქონს შეგვიძლია დავუმატოთ მითითებული სტრიქონის მითითებული სიმბოლოები. შესაბამისი მეთოდის სინტაქსია:
stringBuilder.Append(სტრიქონი, საწყისი_ინდექსი, სიმბოლოების_რაოდენობა)

მაგალითად, DinamiuriStriqoni.Append(“ანა და საბა”, 4, 7); ფუნქციის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს დაემატება სტრიქონი “და საბა”.

AppendFormat() მეთოდი ახდენს დინამიური სტრიქონისათვის დაფორმატებული სტრიქონის დამატებას. მისი სინტაქსია:

AppendFormat(string ფორმატი, object ობიექტი)

აქ ობიექტი წარმოდგენილი იქნება ფორმატის მიხედვით და შემდეგ დაემატება დინამიურ სტრიქონს. მაგალითად:

```
{
double wiladi = 2.8765;
DinamiuriStriqoni.AppendFormat("{0, 5:f2}", wiladi);
label1.Text = DinamiuriStriqoni.ToString();
}
```

AppendFormat() მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს დაემატება სტრიქონი “ 2.88”. ფორმატში რიცხვი 5 არის ათწილადში თანრიგების რაოდენობა მძიმის ჩათვლით, რიცხვი 2 კი - თანრიგების რაოდენობა წილად ნაწილში. შედეგად, ათწილადის მთელ ნაწილს დაეთმობა 2 თანრიგი, მძიმეს ერთი, ხოლო წილად ნაწილს კი - 2. 0 მიუთითებს, რომ უნდა დაფორმატდეს პირველი ცვლადი, კერძოდ კი - wiladi.

Insert() მეთოდი ახდენს ობიექტის სტრიქონად გარდაქმნას და დინამიურ სტრიქონში მითითებული პოზიციიდან ჩამატებას. Insert() მეთოდის უმარტივეს ვერსიას აქვს შემდეგი სინტაქსი:

stringBuilder.Insert*(ინდექსი, სტრიქონი)*

მაგალითად, DinamiuriStriqoni.Insert(5, "საბა, "); მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს მე-5 პოზიციიდან ჩაემატება "საბა, " სტრიქონი.

დინამიურ სტრიქონში შეგვიძლია ჩავსვათ, აგრეთვე, ჩასასმელი სტრიქონის რამდენიმე ასლი. Insert() მეთოდის ამ ვერსიის სინტაქსია:

stringBuilder.Insert*(ინდექსი, სტრიქონი, გამეორების_რაოდენობა)*

მაგალითად, DinamiuriStriqoni.Insert(5, "საბა, ", 4); მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს მე-5 პოზიციიდან 4-ჯერ ჩაემატება "საბა, " სტრიქონი.

Remove() მეთოდი. ის დინამიური სტრიქონის მითითებული პოზიციიდან შლის მითითებული რაოდენობის სიმბოლოს. მეთოდის სინტაქსია:

stringBuilder.Remove*(ინდექსი, სიმბოლოების_რაოდენობა)*

მაგალითი.

```
{
System.Text.StringBuilder DinamiuriStriqoni1 =
                                    new System.Text.StringBuilder("ანა და საბა");
DinamiuriStriqoni1.Remove(3, 4);                    // DinamiuriStriqoni1 = "ანასაბა"
label1.Text = "DinamiuriStriqoni1 = " + DinamiuriStriqoni1.ToString();
}
```

Replace() მეთოდი. ის მითითებულ სიმბოლოს ან სტრიქონს დინამიურ სტრიქონში ყველგან ცვლის ახალი სიმბოლოთი ან სტრიქონით. მეთოდის სინტაქსია:

stringBuilder.Replace*(ძველი_მნიშვნელობა, ახალი_მნიშვნელობა)*

```
{
System.Text.StringBuilder DinamiuriStriqoni1 =
                                    new System.Text.StringBuilder("ანა და საბა");
DinamiuriStriqoni1.Replace('ა', 'რ');        // DinamiuriStriqoni1 = "რნრ დრ სრბრ"
label1.Text += "DinamiuriStriqoni1 = " + DinamiuriStriqoni1.ToString();
}
```

ToString() მეთოდი. ის დინამიურ სტრიქონს გარდაქმნის ჩვეულებრივ სტრიქონად. მეთოდის სინტაქსია:
stringBuilder.ToString()
მაგალითად, string striqoni = DinamiuriStriqoni.ToString(); მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიური სტრიქონი გარდაიქმნება ჩვეულებრივ სტრიქონად.

II ნაწილი. C# ენის სხვა შესაძლებლობები

თავი 13. თარიღი, დრო და დროის ინტერვალები

თარიღი და დრო

თარიღთან და დროსთან სამუშაოდ გამოიყენება System.DateTime სტრუქტურა. მისი თვისებები და მეთოდები მოყვანილია ცხრილებში 13.1 და 13.2. რაც შეეხება Ticks თვისებას, მის ქართულ შესატყვისად შევარჩიე სიტყვა "წიკი", გამომდინარე საათის წიკწიკიდან (რუსულად тик). წიკი არის 100-ნანოწამიანი ინტერვალი.

თავდაპირველად ვნახოთ თუ როგორ ხდება DateTime სტრუქტურის ეგზემპლარის შექმნა, შემდეგ კი განვიხილოთ მისი ზოგიერთი თვისება და მეთოდი. არსებობს DateTime სტრუქტურის კონსტრუქტორის რამდენიმე ვერსია. შესაბამისად, DateTime სტრუქტურის ეგზემპლარი შეგვიძლია რამდენიმე საშუალებით შევქმნათ. ერთ-ერთია კონსტრუქტორისთვის წლის, თვის და დღის გადაცემა. მაგალითი.

```
{
int weli = 2005;
int tve  = 3;
int dge  = 19;
//        იქმნება Tarigi სტრუქტურა, რომელიც შეიცავს წელს, თვესა და დღეს
DateTime Tarigi = new DateTime(weli, tve, dge);
label1.Text = Tarigi.Year.ToString() + " " +
                                Tarigi.Month.ToString() + " " + Tarigi.Day.ToString();
}
```

ცხრილი 13.1. DateTime სტრუქტურის თვისებები.

თვისება	ტიპი	აღწერა
Now (სტატიკური)	DateTime	შეიცავს მიმდინარე თარიღსა და დროს
Today (სტატიკური)	DateTime	შეიცავს მიმდინარე თარიღს. დროის ველები განულებულია 00:00:00
UtcNow() (სტატიკური)	DateTime	შეიცავს მიმდინარე თარიღსა და დროს სასაათო სარტყლის გათვალისწინებით
Date	DateTime	შეიცავს თარიღს. დროის ველები განულებულია 00:00:00
Day	int	შეიცავს თვის დღის მნიშვნელობას 1□31 დიაპაზონში
DayOfWeek	DayOfWeek	შეიცავს კვირის დღის მნიშვნელობას 0 (კვირა) □ 6 (შაბათი) დიაპაზონში
DayOfYear	int	შეიცავს წლის დღის ნომერს 1□366 დიაპაზონში
Hour	int	შეიცავს საათის მნიშვნელობას 0□23 დიაპაზონში
Millisecond	int	შეიცავს მილიწამის მნიშვნელობას 0□999 დიაპაზონში

ცხრილი 13.1. (გაგრძელება)

Minute	int	შეცავს წუთის მნიშვნელობას 0–59 დიაპაზონში
Month	int	შეიცავს თვის მნიშვნელობას 1–12 დიაპაზონში
Second	int	შეიცავს წამის მნიშვნელობას 0–59 დიაპაზონში
Ticks	long	შეიცავს წიკების (100-ნანოწამიანი ინტერვალების) რაოდენობას, რომელიც გავიდა 0001 წლის 1 იანვრიდან ეგზემპლარში დაყენებულ დრომდე
TimeOfDay	TimeSpan	შეიცავს შუაღამიდან გასულ დროს
Year	int	შეიცავს წლის მნიშვნელობას 1–9999 დიაპაზონში

ცხრილი 13.2. DateTime სტრუქტურის მეთოდები.

მეთოდი	დასაბრუნებელი ტიპი	აღწერა
Compare() (სტატიკურია)	int	ადარებს DateTime სტრუქტურის ორ ეგზემპლარს.
DaysInMonth() (სტატიკურია)	int	გასცემს დღეების რაოდენობას მითითებული წლის მითითებული თვისათვის
Equals()	bool	ტოლობაზე ადარებს ორ DateTime სტრუქტურას
FromFileTime() (სტატიკურია)	DateTime	გასცემს DateTime სტრუქტურის ეგზემპლარს, რომელიც ფაილის დროითი ჭდის ეკვივალენტურია
FromOADate() (სტატიკურია)	DateTime	გასცემს DateTime სტრუქტურის ეგზემპლარს, რომელიც OLE თარიღის ეკვივალენტურია
IsLeapYear() (სტატიკურია)	bool	გასცემს true-ს, თუ წელი ნაკიანია, წინააღმდეგ შემთხვევაში false-ს
Parse() (სტატიკურია)	DateTime	თარიღისა და დროის სტრიქონულ წარმოდგენას გარდაქმნის DateTime სტრუქტურის ეკვივალენტურ ეგზემპლარად
ParseExact() (სტატიკურია)	DateTime	თარიღისა და დროის სტრიქონულ წარმოდგენას გარდაქმნის DateTime სტრუქტურის ეკვივალენტურ ეგზემპლარად ფორმატის მიხედვით
Add()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს TimeSpan კლასის ეგზემპლარს
AddDays()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს დღეების მითითებულ რაოდენობას
AddHours()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს საათების მითითებულ რაოდენობას
AddMilliseconds()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს მილიწამების მითითებულ რაოდენობას
AddMinutes()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს წუთების მითითებულ რაოდენობას

ცხრილი 13.2. (გაგრძელება)

AddMonths()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს თვეების მითითებულ რაოდენობას
AddSeconds()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს წამების მითითებულ რაოდენობას
AddTicks()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს წიკების მითითებულ რაოდენობას
AddYears()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს წლების მითითებულ რაოდენობას
CompareTo()	int	DateTime სტრუქტურის ეგზემპლარს ადარებს მითითებულ ობიექტთან.
GetDateTimeFormats()	string[ ]	DateTime სტრუქტურის ეგზემპლარს გარდაქმნის სტრიქონების მასივად, რომელიც შეიცავს ამ სტრუქტურის ყველა შესაძლო ფორმატს
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს
Subtract()	TimeSpan	DateTime სტრუქტურის ეგზემპლარს აკლებს ამავე სტრუქტურის ეგზემპლარს ან TimeSpan კლასის ეგზემპლარს
ToFileTime()	long	DateTime სტრუქტურის ეგზემპლარს გარდაქმნის ფაილის დროით ჭდედ
ToLocalTime()	DateTime	გრინვიჩის დროს გარდაქმნის ადგილობრივ დროდ
ToLongDateString()	string	DateTime სტრუქტურის ეგზემპლარის თარიღს გარდაქმნის გრძელი თარიღის შემცველ სტრიქონად
ToLongTimeString()	string	DateTime სტრუქტურის ეგზემპლარის დროს გარდაქმნის გრძელი დროის შემცველ სტრიქონად
ToOADate()	double	DateTime სტრუქტურის ეგზემპლარს გარდაქმნის ეკვივალენტურ OLE თარიღად
ToShortDateString()	string	DateTime სტრუქტურის ეგზემპლარის თარიღს გარდაქმნის მოკლე თარიღის შემცველ სტრიქონად
ToShortTimeString()	string	DateTime სტრუქტურის ეგზემპლარის დროს გარდაქმნის მოკლე დროის შემცველ სტრიქონად
ToString()	string	DateTime სტრუქტურის ეგზემპლარს გარდაქმნის ეკვივალენტურ სტრიქონად
ToUniversalTime()	DateTime	ადგილობრივ დროს გარდაქმნის გრინვიჩის დროდ

კონსტრუქტორს შეიძლება, აგრეთვე, გადავცეთ საათი, წუთი, წამი და მილიწამი. მაგალითი.

```
{
int weli = 2005;
int tve  = 3;
int dge  = 19;
int saati = 20;
int wuti  = 30;
int wami  = 25;
int miliwami = 45;
// იქმნება Tarigi სტრუქტურა, რომელიც შეიცავს წელს, თვეს, დღეს, საათს, წუთს, წამს და მილიწამს
DateTime Tarigi = new DateTime(weli, tve, dge, saati, wuti, wami, miliwami);
label1.Text = Tarigi.Year.ToString() + " " +
                Tarigi.Month.ToString()  + " " + Tarigi.Day.ToString()    + "  " +
                Tarigi.Hour.ToString()   + " " + Tarigi.Minute.ToString() + "  " +
                Tarigi.Second.ToString() + " " + Tarigi.Millisecond.ToString();
}
```

კონსტრუქტორს უკანასკნელ პარამეტრად შეიძლება გადავცეთ კალენდარი - System.Globalization.Calendar კლასის ობიექტი. Calendar კლასი უზრუნველყოფს დროის დაყოფას ინტერვალებად, კერძოდ, წლებად, თვეებად და კვირეებად. არსებობს რამდენიმე კლასი, რომლებიც ახდენენ Calendar კლასის მეთოდების რეალიზებას. ეს კლასებია: EastAsianLunisolarCalendar, GregorianCalendar, HebrewCalendar, HijriCalendar, JapaneseCalendar, JapaneseLunisolarCalendar, JulianCalendar, KoreanCalendar, KoreanLunisolarCalendar, PersianCalendar, TaiwanCalendar, TaiwanLunisolarCalendar, ThaiBuddhistCalendar, UmAlQuraCalendar. მოყვანილ მაგალითში კონსტრუქტორს გადაეცემა JulianCalendar კლასის ობიექტი.

```
{
System.Globalization.JulianCalendar Kalendari = new
                                                System.Globalization.JulianCalendar();
System.DateTime Tarigi = new System.DateTime(2005, 3, 19, Kalendari);
label1.Text = Tarigi.Year.ToString() + "  " + Tarigi.Month.ToString() + "  " +
                Tarigi.Day.ToString();
}
```

ახლა განვიხილოთ ზოგიერთი თვისება და მეთოდი.

Now და **UtcNow** თვისებები. Now თვისება შეიცავს კომპიუტერის სისტემური საათიდან აღებულ თარიღსა და დროს. UtcNow თვისება შეიცავს თარიღსა და დროს UTC (Universal Coordinated Time) ფორმატში, რომელსაც, აგრეთვე, გრინვიჩის დროს (GMT - Greenwich Mean Time) უწოდებენ. იგი შეესაბამება ინგლისის ერთ-ერთი დაბის - გრინვიჩის სასაათო სარტყელს. სტანდარტული წყნარი ოკეანის დრო (PST - Pacific Standard Time) გრინვიჩის დროს რვა საათით ჩამორჩება. მოყვანილ პროგრამაში ხდება ამ თვისებებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 13.1
//      პროგრამაში ხდება Now და UtcNow თვისებებთან მუშაობის დემონსტრირება
DateTime Tarigi_Dro1 = DateTime.Now;
DateTime Tarigi_Dro2 = DateTime.UtcNow;
```

```
label1.Text = Tarigi_Dro1.Day.ToString() + " " + Tarigi_Dro1.Month.ToString() + " " +
Tarigi_Dro1.Year.ToString() + '\n';
label1.Text += Tarigi_Dro2.Day.ToString() + " " + Tarigi_Dro2.Month.ToString() + " " +
Tarigi_Dro2.Year.ToString();
}
```

მოყვანილ პროგრამაში ხდება **Date, Day, DayOfWeek, DayOfYear** და **TimeOfDay** თვისებებთან მუშაობის დემონსტრირება.

```
{
// პროგრამა 13.2
// პროგრამაში ხდება Date, Day, DayOfWeek, DayOfYear, TimeOfDay
// თვისებებთან მუშაობის დემონსტრირება
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text = Tarigi_Dro1.Date.ToString();
label1.Text += Tarigi_Dro1.Day.ToString();
label1.Text += Tarigi_Dro1.DayOfWeek.ToString();
label1.Text += Tarigi_Dro1.DayOfYear.ToString();
label1.Text += Tarigi_Dro1.TimeOfDay.ToString();
}
```

Compare() მეთოდი ადარებს DateTime სტრუქტურის ორ ეგზემპლარს. თუ პირველი ეგზამპლარი მეტია მეორეზე, მაშინ გაიცემა 1. თუ პირველი ეგზემპლარი ნაკლებია მეორეზე, მაშინ გაიცემა -1. თუ ეგზემპლარები ტოლია, მაშინ გაიცემა 0. მეთოდის სინტაქსია:

DateTime.Compare*(თარიღი_დრო1, თარიღი_დრო2)*

აქ *თარიღი_დრო1* და *თარიღი_დრო2* არის DateTime სტრუქტურის ეგზემპლარები, რომელთა შედარებაც ხდება. მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
DateTime Tarigi_Dro2 = new DateTime(2007, 1, 27);
int Shedegi = DateTime.Compare(Tarigi_Dro1, Tarigi_Dro2);
switch ( Shedegi )
{
case -1 : label1.Text = "თარიღები ტოლია"; break;
case  0 : label1.Text = "პირველი თარიღი მეტია მეორეზე"; break;
case  1 : label1.Text = " პირველი თარიღი ნაკლებია მეორეზე "; break;
}
}
```

DateTime სტრუქტურის ეგზემპლარების შედარება შეიძლება, აგრეთვე, შედარების გადატვირთული ოპერატორებით: >, >=, <, <=, ==, != . მაგალითად,

```
if ( Tarigi_Dro1 == Tarigi_Dro2 ) label1.Text = “თარიღები ტოლია”;
        else label1.Text = “თარიღები არ არის ტოლი”;
```

Equals() მეთოდი ადარებს DateTime სტრუქტურის ორ ეგზემპლარს. თუ ისინი ტოლია, მაშინ გაიცემა true, წინააღმდეგ შემთხვევაში - false. ამ მეთოდს აქვს როგორც სტატიკური, ისე ჩვეულებრივი ვერსია. სტატიკური ვერსიის სინტაქსია:

DateTime.Equals*(თარიღი_დრო1, თარიღი_დრო2)*

აქ *თარიღი_დრო1* და *თარიღი_დრო2* არის DateTime სტრუქტურის ეგზემპლარები.

მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
DateTime Tarigi_Dro2 = new DateTime(2007, 1, 27);
bool Shedegi = DateTime.Equals(Tarigi_Dro1, Tarigi_Dro2);
if (Shedegi == true) label1.Text = "თარიღები ტოლია";
  else label1.Text = "თარიღები არ არის ტოლი";
}
```

Equals() მეთოდის ჩვეულებრივი ვერსიის სინტაქსია:

თარიღი_დრო1.Equals*(თარიღი_დრო2)*

მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
DateTime Tarigi_Dro2 = new DateTime(2007, 1, 27);
bool Shedegi = Tarigi_Dro1.Equals(Tarigi_Dro2);
if ( Shedegi == true ) label1.Text = "თარიღები ტოლია";
   else label1.Text = "თარიღები არ არის ტოლი";
}
```

DaysInMonth() მეთოდი გასცემს მითითებულ თვეში დღეების რაოდენობას. მისი სინტაქსია:

DateTime.DaysInMonth*(წელი, თვე)*

მაგალითი.

```
{
int DgeebisRaodenoba = DateTime.DaysInMonth(2007, 2);
label1.Text = DgeebisRaodenoba.ToString();        // DgeebisRaodenoba = 31
}
```

IsLeapYear() მეთოდი გასცემს true-ს თუ თვე ნაკიანია და false-ს წინააღმდეგ შემთხვევაში. მისი სინტაქსია:

DateTime.IsLeapYear*(წელი)*

მაგალითი.

```
{
bool ArisNakiani = DateTime.IsLeapYear(2007);
if ( ArisNakiani == true ) label1.Text = "წელი ნაკიანია";
   else label1.Text = " წელი არ არის ნაკიანი";
}
```

Parse() მეთოდი თარიღისა და დროის შემცველ სტრიქონს გარდაქმნის DateTime სტრუქტურის ეკვივალენტურ ეგზემპლარად. მისი სინტაქსია:

DateTime.Parse*(სტრიქონი)*

მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Parse("10/5/2000");
DateTime Tarigi_Dro2 = DateTime.Parse("10/5/2000 8:30:35");
label1.Text = Tarigi_Dro1.ToString() + "    " + Tarigi_Dro2.ToString();
}
```

Add() და **Subtract()** მეთოდები. Add() მეთოდი DateTime სტრუქტურის

ეგზემპლარს უმატებს TimeSpan კლასის ობიექტს. Subtract() მეთოდი DateTime სტრუქტურის ეგზემპლარს აკლებს TimeSpan კლასის ობიექტს. ამ მეთოდების სინტაქსია:

*თარიღი_დრო.*Add*(დროის_ინტერვალი)*

*თარიღი_დრო.*Subtract*(დროის_ინტერვალი)*

აქ *თარიღი_დრო* არის DateTime სტრუქტურის ეგზემპლარი, *დროის_ინტერვალი* კი - TimeSpan კლასის ობიექტი. მაგალითი.

```
{
TimeSpan Drois_Intervali = new TimeSpan(1, 5, 10, 20);
DateTime Tarigi_Dro1 = new DateTime(2007, 5, 15, 17, 35, 30);
DateTime Tarigi_Dro2 = Tarigi_Dro1.Add(Drois_Intervali);              // 16.05.2007 22:45:50
DateTime Tarigi_Dro3 = Tarigi_Dro1.Subtract(Drois_Intervali);         // 14.05.2007 12:25:10
label1.Text = Tarigi_Dro2.ToString() + "  " + Tarigi_Dro3.ToString();
}
```

თავდაპირველად პროგრამში იქმნება დროის ინტერვალი: 1 დღე, 5 საათი, 10 წუთი და 20 წამი. შემდეგ იქმნება Tarigi_Dro1 სტრუქტურა: 2007 წლის 15 მაისი, 17 საათი, 35 წუთი და 30 წამი. Add() მეთოდი Tarigi_Dro1 სტრუქტურას დაუმატებს Drois_Intervali დროის ინტერვალს. შედეგად, Tarigi_Dro2 სტრუქტურას მიენიჭება მნიშვნელობა - 16.05.2007 22:45:50. Subtract() მეთოდი Tarigi_Dro1 სტრუქტურას გამოაკლებს Drois_Intervali დროის ინტერვალს. შედეგად, Tarigi_Dro3 სტრუქტურას მიენიჭება მნიშვნელობა - 14.05.2007 12:25:10.

იგივე ოპერაციები შეგვიძლია შევასრულოთ გადატვირთული “+” და “-“ ოპერატორების გამოყენებით. მაგალითი.

```
DateTime Tarigi_Dro4 = Tarigi_Dro1 + Drois_Intervali;
DateTime Tarigi_Dro5 = Tarigi_Dro1 - Drois_Intervali;
```

AddYears(), AddMonths(), AddDays(), AddHours() და **AddMinutes()** მეთოდები. ეს მეთოდები DateTime სტრუქტურის ეგზემპლარს შესაბამისად უმატებს წლებს, თვეებს, დღეებს, საათებსა და წუთებს. ამ მეთოდებს აქვთ double ტიპის პარამეტრი. მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text = Tarigi_Dro1.ToString();
Tarigi_Dro1 = Tarigi_Dro1.AddYears(3);
Tarigi_Dro1 = Tarigi_Dro1.AddMonths(2);
Tarigi_Dro1 = Tarigi_Dro1.AddDays(10);
Tarigi_Dro1 = Tarigi_Dro1.AddHours(7);
Tarigi_Dro1 = Tarigi_Dro1.AddMinutes(40);
label1.Text += Tarigi_Dro1.ToString();
}
```

ცხრილი 13.3. საფორმატო სტრიქონის კომპონენტები.

ფორმატის ელემენტი	აღწერა
d	თვის დღე წინა ნულის გარეშე დღის ნომრებისათვის, რომლებიც ერთი ციფრისგან შედგება
Dd	თვის დღე წინა ნულით დღის ნომრებისათვის, რომლებიც ერთი ციფრისგან შედგება
ddd	კვირის დღის შემოკლებული დასახელება
dddd	კვირის დღის სრული დასახელება
f	წამის ნაწილი ერთი ციფრის სიზუსტით. სიზუსტის გასაზრდელად შეგვიძლია გამოვიყენოთ შვიდი “f” სიმბოლო. დანარჩენი ციფრები არ გამოჩნდება.
M	თვის ნომერი წინა ნულის გარეშე თვის ნომრებისათვის, რომლებიც ერთი ციფრისაგან შედგება
MM	თვის ნომერი წინა ნულით თვის ნომრებისათვის, რომლებიც ერთი ციფრისაგან შედგება
MMM	თვის შემოკლებული დასახელება
MMMM	თვის სრული დასახელება
y	წელი ასწლეულის გარეშე. 10-ზე ნაკლები მნიშვნელობა აისახება წინა ნულის გარეშე
Yy	წელი ასწლეულის გარეშე. 10-ზე ნაკლები მნიშვნელობა აისახება წინა ნულით
yyyy	წლის სრული ნომერი შემდგარი ოთხი ციფრისგან
Gg	ერა. ეს ელემენტი იგნორირდება, თუ თარიღი არ შეიცავს ერას
h	საათი თორმეტსაათიანი წარმოდგენით, წინა ნულის გარეშე, ერთი ციფრისგან შემდგარი საათებისთვის
Hh	საათი თორმეტსაათიანი წარმოდგენით, წინა ნულით, ერთი ციფრისგან შემდგარი საათებისთვის
H	საათი, ოცდაოთხსაათიანი წარმოდგენით, წინა ნულის გარეშე, ერთი ციფრისგან შემდგარი საათებისთვის
HH	საათი, ოცდაოთხსაათიანი წარმოდგენით, წინა ნულით, ერთი ციფრისგან შემდგარი საათებისთვის
m	წუთები, წინა ნულის გარეშე, ერთი ციფრისგან შემდგარი წუთებისთვის
mm	წუთები, წინა ნულით, ერთი ციფრისგან შემდგარი წუთებისთვის
s	წამები, წინა ნულის გარეშე, ერთი ციფრისგან შემდგარი წამებისთვის
ss	წამები, წინა ნულით, ერთი ციფრისგან შემდგარი წამებისთვის
t	A.M./P.M მიმთითებლის პირველი სიმბოლო
tt	A.M./P.M სრული მიმთითებელი
zz	სასაათო სარტყლის წანაცვლება, წინა ნულით, ერთი ციფრისგან შემ-დგარი წანაცვლებისთვის
zzz	სასაათო სარტყლის წანაცვლება შემდგარი საათებისა და წუთებისაგან. ერთი ციფრისგან შემდგარი მნიშვნელობები შეივსება ნულით

მოყვანილ პროგრამაში ხდება **ToLongDateString(), ToShortDateString(), ToLongTimeString()** და **ToShortTimeString()** მეთოდების გამოყენების დემონსტრირება.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text  = Tarigi_Dro1.ToLongDateString() + '\n';
label1.Text += Tarigi_Dro1.ToShortDateString()+ '\n';
label1.Text += Tarigi_Dro1.ToLongTimeString() + '\n';
label1.Text += Tarigi_Dro1.ToShortTimeString();
}
```

String() მეთოდი. მისი გადატვირთული ვერსია ახდენს DateTime სტრუქტურის ეგზემპლარის გარდაქმნას ეკვივალენტურ სტრიქონად. მისი სინტაქსია:

*თარიღი_დრო.*ToString()

მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text = Tarigi_Dro1.ToString();
}
```

ToString() მეთოდს შეგვიძლია პარამეტრად გადავცეთ ფორმატი (სტრიქონი), რომლის მიხედვითაც შესრულდება DateTime სტრუქტურის ეგზემპლარის გარდაქმნა. ToString() მეთოდის ამ ვერსიის სინტაქსია:

*თარიღი_დრო.*ToString*(ფორმატი)*

მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text  = Tarigi_Dro1.ToString() + '\n';
label1.Text += Tarigi_Dro1.ToString("MM/dd/yy");
label1.Text += Tarigi_Dro1.ToString("D");
}
```

ცხრილი 13.4. თარიღისა და დროის ჩადგმული ფორმატები.

სიმბოლო	ფორმატი	მაგალითი
d	MM/dd/yyyy	03.19.2007
D	dddd, MMMM dd, yyyy	2007 წლის 04 02, კვირა
f	dddd, MMMM dd, yyyy HH:mm	2007 წლის 04 02, კვირა 9:36
F	dddd, MMMM dd, yyyy HH:mm:ss	2007 წლის 04 02, კვირა 9:36:51
g	MM/dd/yyyy/ HH:mm	04.02.2007 9:36:51
G	MM/dd/yyyy HH:mm:ss	04.02.2007 9:36
m, M	MMMM dd	04 02
r, R	ddd, dd MM yyyy HH':'mm':'ss 'GMT'	Sun, 04 Feb 2007 09:42:17 GMT
s	yyyy'-'MM'-'dd'T'HH':'mm':'ss	2007-02-04T09:42:17
t	HH:mm	9:42
T	HH:mm:ss	9:42:17
u	yyyy'-'MM'-'dd HH':'mm':'ss'Z'	2007-02-04 09:42:17Z
U	dddd, MMMM dd, yyyy HH:mm:ss	2007 წლის 04 02, კვირა 5:42:17
y, Y	yyyy, MMMM	თებერვალი 2007

პროგრამის შესრულების შედეგად მიიღება სტრიქონი, მაგალითად, “02.03.07”. აქ 02 არის თვე, 03 - დღე, 07 კი - წელი. კოდის უკანასკნელ სტრიქონში გამოყენებულია ჩადგმული ფორმატებიდან ერთ-ერთი, კერძოდ კი - “D”.

ცხრილებში 13.3 და 13.4 შესაბამისად მოყვანილია საფორმატო სტრიქონის კომპონენტები და თარიღისა და დროის ჩადგმული ფორმატები.

დროის ინტერვალები

თარიღისა და დროის გარდა შეგვიძლია, აგრეთვე, ვიმუშაოთ დროის ინტერვალებთან. დროის ინტერვალებთან სამუშაოდ გამოიყენება System.TimeSpan კლასი. ამ კლასის კონსტრუქტორს აქვს რამდენიმე გადატვირთული ვერსია. მათი სინტაქსია:

TimeSpan(int *საათი,* int *წუთი,* int *წამი)*
TimeSpan(int *დღე,* int *საათი,* int *წუთი,* int *წამი)*
TimeSpan(int *დღე,* int *საათი,* int წუთი, int *წამი,* int *მილიწამი)*
TimeSpan(int *წიკი)*

მაგალითი.

```
{
//      დროითი ინტერვალის შექმნა საათის, წუთისა და წამის მითითებით
TimeSpan Drois_Intervali1 = new TimeSpan(5, 15, 45);
//      დროითი ინტერვალის შექმნა დღის, საათის, წუთისა და წამის მითითებით
TimeSpan Drois_Intervali2 = new TimeSpan(2, 5, 15, 45);
//      დროითი ინტერვალის შექმნა დღის, საათის, წუთის, წამისა და მილიწამის
მითითებით
TimeSpan Drois_Intervali3 = new TimeSpan(2, 5, 15, 45, 80);
//      დროითი ინტერვალის შექმნა წიკის მითითებით
TimeSpan Drois_Intervali4 = new TimeSpan(1000);
}
```

TimeSpan კლასის ობიექტი შეგვიძლია დავუმატოთ DateTime სტრუქტურის ეგზემპლარს. მაგალითად

```
{
DateTime Tarigi_Dro = new DateTime(2007, 04, 12);
TimeSpan Drois_Intervali1 = new TimeSpan(2, 5, 15, 45);
Tarigi_Dro += Drois_Intervali1;                    //      14.04.2007 5:15:45
label1.Text = Tarigi_Dro.ToString();
}
```

განვიხილოთ ზოგიერთი თვისება და მეთოდი.

Days, Hours, Minutes, Seconds, Milliseconds და **Ticks** თვისებებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
{
//      პროგრამა 13.3
//      პროგრამაში ხდება Days, Hours, Minutes, Seconds, Milliseconds და
//      Ticks თვისებებთან მუშაობის დემონსტრირება
TimeSpan Drois_Intervali1 = new TimeSpan(5, 15, 45);
```

```
label1.Text  = Drois_Intervali1.Days.ToString()  + '\n';
label1.Text += Drois_Intervali1.Hours.ToString() + '\n';
label1.Text += Drois_Intervali1.Minutes.ToString() + '\n';
label1.Text += Drois_Intervali1.Seconds.ToString() + '\n';
label1.Text += Drois_Intervali1.Milliseconds.ToString() + '\n';
label1.Text += Drois_Intervali1.Ticks.ToString() + '\n';
}
```

შედეგში Days და Milliseconds თვისებებს ექნებათ ნულოვანი მნიშვნელობები, რადგან Drois_Intervali1 ობიექტის შექმნის დროს მათი მნიშვნელობები არ იყო მითითებული.

TimeSpan კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 13.5 და 13.6.

ცხრილი 13.5. TimeSpan კლასის თვისებები.

თვისება	ტიპი	აღწერა
Days	int	შეიცავს დღეების რაოდენობას TimeSpan კლასის ობიექტში
Hours	int	შეიცავს საათების რაოდენობას TimeSpan კლასის ობიექტში
Milliseconds	int	შეიცავს მილიწამების რაოდენობას TimeSpan კლასის ობიექტში
Minutes	int	შეიცავს წუთების რაოდენობას TimeSpan კლასის ობიექტში
Seconds	int	შეიცავს წამების რაოდენობას TimeSpan კლასის ობიექტში
Ticks	long	შეიცავს წიკების რაოდენობას TimeSpan კლასის ობიექტში
TotalDays	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას დღეებში
TotalHours	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას საათებში
TotalMilliseconds	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას მილიწამებში
TotalMinutes	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას წუთებში
TotalSeconds	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას წამებში

ცხრილი 13.6. TimeSpan კლასის მეთოდები.

მეთოდი	დაბრუნებული მნიშვნელობის ტიპი	აღწერა
Compare() (სტატიკური)	int	ადარებს TimeSpan კლასის ორ ობიექტს. გასცემს შესაბამის რიცხვს
Equals()	bool	ადარებს TimeSpan კლასის ორ ობიექტს. გასცემს შესაბამის ლოგიკურ მნიშვნელობას

ცხრილი 13.6. (გაგრძელება)

FromDays() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს დღეების მითითებულ რაოდენობას
FromHours() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს საათების მითითებულ რაოდენობას
FromMilliseconds() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს მილიწამების მითითებულ რაოდენობას
FromMinutes() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს წუთების მითითებულ რაოდენობას
FromSeconds() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს წამების მითითებულ რაოდენობას
FromTicks() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს წიკების მითითებულ რაოდენობას
Parse() (სტატიკური)	TimeSpan	დროის ინტერვალის მითითებულ სტრიქონულ წარმოდგენას გარდაქმნის TimeSpan კლასის ეკვივალენტურ ობიექტად
Add()	TimeSpan	TimeSpan კლასის მოცემულ ობიექტს უმატებს ამავე კლასის ობიექტს
CompareTo()	int	ადარებს TimeSpan კლასის ორ ობიექტს. გასცემს შესაბამის რიცხვს
Duration()	TimeSpan	გასცემს დადებითი ხანგრძლივობის TimeSpan კლასის ობიექტს, რომელიც სიდიდით მიმდინარე ობიექტის ხანგრძლივობის ტოლია
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს
Negate()	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომლის მნიშვნელობა მიმდინარე ობიექტის მნიშვნელობის საწინააღმდეგოა
Subtract()	TimeSpan	TimeSpan კლასის მოცემულ ობიექტს აკლებს ამავე კლასის ობიექტს
ToString()	string	TimeSpan კლასის მოცემულ ობიექტს გარდაქმნის ეკვივალენტურ სტრიქონად

TotalDays, TotalHours, TotalMinutes, TotalSeconds და **TotalMilliseconds** თვისებებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
{
//      პროგრამა 13.4
```

```
//      პროგრამაში ხდება TotalDays, TotalHours, TotalMinutes, TotalSeconds და
//      TotalMilliseconds თვისებებთან მუშაობის დემონსტრირება
TimeSpan Drois_Intervali1 = new TimeSpan(5, 15, 45);
label1.Text  = Drois_Intervali1.TotalDays.ToString() + '\n';
label1.Text += Drois_Intervali1.TotalHours.ToString() + '\n';
label1.Text += Drois_Intervali1.TotalMinutes.ToString() + '\n';
label1.Text += Drois_Intervali1.TotalSeconds.ToString() + '\n';
label1.Text += Drois_Intervali1.TotalMilliseconds.ToString() + '\n';
}
```

FromDays(), FromHours(), FromMinutes(), FromSeconds(), FromMilliseconds() და **FromTicks()** მეთოდების პარამეტრებს აქვთ double ტიპი. ამ მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
{
//      პროგრამა 13.5
//      პროგრამაში ხდება FromDays(), FromHours(), FromMinutes(), FromSeconds(),
//      FromMilliseconds() და FromTicks() მეთოდებთან მუშაობის დემონსტრირება
TimeSpan Drois_Intervali1 = TimeSpan.FromDays(7);
TimeSpan Drois_Intervali2 = TimeSpan.FromHours(7);
TimeSpan Drois_Intervali3 = TimeSpan.FromMinutes(7);
TimeSpan Drois_Intervali4 = TimeSpan.FromSeconds(7);
TimeSpan Drois_Intervali5 = TimeSpan.FromMilliseconds(7);
TimeSpan Drois_Intervali6 = TimeSpan.FromTicks(7);
label1.Text  = Drois_Intervali1.ToString() + '\n';
label1.Text += Drois_Intervali2.ToString() + '\n';
label1.Text += Drois_Intervali3.ToString() + '\n';
label1.Text += Drois_Intervali4.ToString() + '\n';
label1.Text += Drois_Intervali5.ToString() + '\n';
label1.Text += Drois_Intervali6.ToString() + '\n';
}
```

ცხრილი 13.7. Parse() მეთოდის ფორმატის ელემენტები.

ელემენტი	აღწერა
ws	არააუცილებელი " " (ინტერვალის სიმბოლო)
□	არააუცილებელი "-", რომელიც აღნიშნავს, რომ ინტერვალი უარყოფითია
d	დღეების არააუცილებელი რაოდენობა
hh	საათი ოცდაოთხსაათიან ფორმატში
mm	წუთები
ss	წამები
ff	წამის არააუცილებელი ნაწილები. იგი შეიძლება შეიცავდეს 1□7 სიმბოლოს

Parse() მეთოდი. მისთვის პარამეტრად გადაცემულ სტრიქონს უნდა ჰქონდეს შემდეგი ფორმატი:

[ws] [-] [d.]hh:mm:ss[.ff][ws]

კვადრატულ ფრჩხილებში მოთავსებული ელემენტების მითითება არ არის

აუცილებელი.

ფორმატის ელემენტები მოყვანილია ცხრილში 13.7.

მაგალითი.

```
{
//      პროგრამაში ხდება Parse() მეთოდის ფორმატის ელემენტებთან მუშაობის
//      დემონსტრირება
//      იქმნება Drois_Intervali1 ობიექტი, რომლის ხანგრძლივობაა 9 საათი, 15 წუთი და 45 წამი
TimeSpan Drois_Intervali1 = TimeSpan.Parse("9:15:45");
//      იქმნება Drois_Intervali2 ობიექტი, რომლის ხანგრძლივობაა
//      2 დღე, 9 საათი, 15 წუთი და 45.50 წამი
TimeSpan Drois_Intervali2 = TimeSpan.Parse("2.9:15:45.50");
label1.Text  = Drois_Intervali1.ToString() + '\n';
label1.Text += Drois_Intervali2.ToString() + '\n';
}
```

Add() და **Subtract()** მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
{
TimeSpan Drois_Intervali1 = new TimeSpan(2, 15, 25);
TimeSpan Drois_Intervali2 = new TimeSpan(12, 20, 30);
TimeSpan Drois_Intervali3 = Drois_Intervali2.Add(Drois_Intervali1);
TimeSpan Drois_Intervali4 = Drois_Intervali2.Subtract(Drois_Intervali1);
label1.Text  = Drois_Intervali3.ToString() + '\n';          // შედეგი - 14:35:55
label1.Text += Drois_Intervali4.ToString() + '\n';          // შედეგი - 10:05:05
}
```

Duration() და **Negate()** მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
{
TimeSpan Drois_Intervali1 = new TimeSpan(2, 15, 25);
TimeSpan Drois_Intervali2 = Drois_Intervali1.Duration();
TimeSpan Drois_Intervali3 = Drois_Intervali1.Negate();
label1.Text  = Drois_Intervali2.ToString() + '\n';          // შედეგი - 02:15:25
label1.Text += Drois_Intervali3.ToString() + '\n';          // შედეგი - -02:15:25
}
```

თავი 14. კოლექციები

მე-4 თავში ჩვენ განვიხილეთ მასივები. ისინი ფართოდ გამოიყენებიან სხვადასხვა ხასიათის ამოცანების გადაწყვეტად. ამავე დროს, მათ აქვთ შემდეგი შეზღუდვები:

- შექმნის შემდეგ მასივს ენიჭება ფიქსირებული ზომა, რომელსაც ვეღარ შევცვლით.
- შეუძლებელია მასივში ელემენტის ჩამატება ან წაშლა.
- მასივის ელემენტებთან მიმართვა შესაძლებელია მხოლოდ ინდექსის საშუალებით.
- მასივის ელემენტებს ერთი და იგივე ტიპი აქვთ.

ეს შეზღუდვები მოხსნილია კოლექციებში. ჩვენ განვიხილავთ შემდეგ კოლექციებს: დინამიური მასივები, ჰეშ-ცხრილები, დახარისხებული სიები, რიგები, სტეკები და ბიტების მასივები. კოლექციების აღმწერი კლასები გამოცხადებულია System.Collections სახელების სივრცეში.

ამ განყოფილებაში მოყვანილი ყველა პროგრამის დასაწყისში უნდა მოვათავსოთ დირექტივა using System.Collections;

დინამიური მასივები

ჩვეულებრივი მასივებისაგან განსხვავებით, დინამიური მასივი მასში ელემენტების დამატების ან წაშლის შემთხვევაში ზომას იცვლის. დინამიურ მასივებში ელემენტების ნუმერაცია ნულიდან იწყება. ArrayList კლასში განსაზღვრულია თვისებები და მეთოდები დინამიური მასივების ელემენტებთან სამუშაოდ. ზოგიერთი მათგანი მოყვანილია ცხრილებში 14.1 და 14.2.

დინამიური მასივის შექმნა. ArrayList კონსტრუქტორს სამი გადატვირთული ვერსია აქვს. შესაბამისად, დინამიური სტრიქონი სამი გზით შეგვიძლია შევქმნათ. ამ ვერსიების სინტაქსია:

ArrayList()

ArrayList(ICollection *კოლექცია)*

ArrayList(int *ტევადობა)*

მაგალითი.

```
{
string[] Striqoni = { "ანა", "საბა", "ლიკა", "რომანი" };
int[] masivi = new int[] { 1, 2, 3, 4, 5 };
ArrayList DinamiuriMasivi1 = new ArrayList();
ArrayList DinamiuriMasivi2 = new ArrayList(masivi);
ArrayList DinamiuriMasivi3 = new ArrayList(Striqoni);
ArrayList DinamiuriMasivi4 = new ArrayList(50);
foreach (int x in DinamiuriMasivi2) label1.Text += x.ToString() + " ";
foreach (string x in DinamiuriMasivi3) label2.Text += x + " ";
}
```

ცხრილი 14.1. ArrayList კლასის თვისებები

თვისება	ტიპი	აღწერა
Capacity	int	განსაზღვრავს დინამიური მასივის ტევადობას, ანუ ელემენტების მაქსიმალურ რაოდენობას
Count	int	შეიცავს დინამიური მასივში ელემენტების რაოდენობას
IsFixedSize	bool	მიუთითებს, არის თუ არა დინამიური მასივი ფიქსირებული ზომის
IsReadOnly	bool	მიუთითებს, არის თუ არა დინამიური მასივი მხოლოდ წაკითხვადი
Item	object	გასცემს ან აყენებს მითითებული ინდექსის მქონე ელემენტის მნიშვნელობას

ცხრილი 14.2. ArrayList კლასის მეთოდები

მეთოდი	დაბრუნებული მნიშვნელობის ტიპი	აღწერა
ReadOnly()	ArrayList	გასცემს დინამიურ მასივს მხოლოდ წაკით-ხვისათვის
Repeat()	ArrayList	გასცემს დინამიური მასივს, რომლის ყველა ელემენტი გადაცემული მნიშვნელობების ტოლია
Add()	int	დინამიურ მასივს ბოლოში უმატებს ელე-მენტს
AddRange()	void	დინამიურ მასივს ბოლოში უმატებს სხვა კო-ლექციის ელემენტებს
BinarySearch()	int	დახარისხებულ მასივში ასრულებს მითითებული ელემენტის ორობით ძებნას.
Clear()	void	დინამიური მასივიდან შლის ყველა ელემენტს
Contains()	bool	განსაზღვრავს შეიცავს თუ არა დინამიური მასივი მითითებულ ელემენტს
CopyTo()	void	დინამიური მასივის ელემენტებს გადაწერს ერთგანზომილებიან მასივში
Equals()	bool	განსაზღვრავს ტოლია თუ არა ორი დინამიური მასივი
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს
GetRange()	ArrayList	გასცემს დინამიური მასივს, რომელიც შეიცავს საწყისი მასივის ელემენტების დიაპაზონს
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს
IndexOf()	int	გასცემს ნაპოვნი ელემენტის პოზიციის ნომერს

ცხრილი 14.2 (გაგრძელება)

Insert()	void	ახდენს დინამიურ მასივში ელემენტის ჩასმას მითითებული პოზიციიდან
InsertRange()	void	ახდენს დინამიურ მასივში კოლექციის ელემენტების ჩასმას მითითებული პოზიციიდან
LastIndexOf()	int	გასცემს ნაპოვნი ელემენტის უკანასკნელი პოზიციის ნომერს
Remove()	void	დინამიური მასივიდან შლის პირველ ნაპოვნ ელემენტს
RemoveAt()	void	დინამიური მასივიდან შლის მითითებული პოზიციის მქონე ელემენტს
RemoveRange()	void	დინამიური მასივიდან შლის ელემენტების დიაპაზონს დაწყებული მითითებული პოზიციიდან
Reverse()	void	დინამიური მასივის ელემენტებს უკუ მიმდევრობით ალაგებს
SetRange()	void	დინამიური მასივის შესაბამის დიაპაზონში ახდენს კოლექციის ელემენტების ჩასმას
Sort()	void	ახდენს დინამიური მასივის ან მისი დიაპაზონის დახარისხებას
ToArray()	object[]	დინამიური მასივის ელემენტებს გადაწერს ჩვეულებრივ მასივში
ToString()	string	მიმდინარე ობიექტს გარდაქმნის სტრიქონად
StrimToSize()	void	ამცირებს დინამიური მასივის ტევადობას ელემენტების იმ რაოდენობამდე, რომელიც აქვს ამ მასივს მეთოდის გამოძახების მომენტში

ელემენტების დამატება და ჩამატება. **Add()** მეთოდი დინამიურ მასივს ბოლოში უმატებს მითითებულ ელემენტს, რომლის ტიპია object. მისი სინტაქსია:
arraList.Add*(სიდიდე)*
აქ *arrayList* აღნიშნავს ArrayList ტიპის ობიექტს.

მოყვანილ პროგრამაში ხდება Add მეთოდთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 14.1
//      პროგრამაში ხდება Add მეთოდთან მუშაობის დემონსტრირება
ArrayList DinamiuriMasivi = new ArrayList();
//
DinamiuriMasivi.Add("საბა");
DinamiuriMasivi.Add("ანა");
DinamiuriMasivi.Add("ლიკა");
DinamiuriMasivi.Add("რომანი");
```

```
//
for ( int indexi = 0; indexi < DinamiuriMasivi.Count; indexi++ )
        label1.Text += "DinamiuriMasivi [" + indexi.ToString() + "] = " +
    DinamiuriMasivi[indexi].ToString() + '\n';
}
```

დინამიურ მასივს შეგვიძლია დავუმატოთ სხვადასხვა ტიპის მონაცემები. მაგალითი.

```
{
//
ArrayList DinamiuriMasivi = new ArrayList();
//
DinamiuriMasivi.Add(1);
DinamiuriMasivi.Add(2.2);
DinamiuriMasivi.Add(true);
DinamiuriMasivi.Add("რომანი");
for (int indexi = 0; indexi < DinamiuriMasivi.Count; indexi++)
   label1.Text += "DinamiuriMasivi[" + indexi.ToString() + "] = " +
   DinamiuriMasivi[indexi].ToString() + '\n';
}
```

AddRange() მეთოდი დინამიურ მასივს ბოლოში უმატებს ელემენტების დიაპაზონს. მისი სინტაქსია:

arrayList.Insert*(კოლექცია)*

მეთოდის შესულების შედეგად კოლექცია დაემატება arrayList დინამიურ მასივს.

მაგალითი.

```
{
string[] striqoni1 = { "ანა", "საბა", "ლიკა", "რომანი", "გიორგი" };
string[] striqoni2 = { "ნატა", "ბექა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni1);
DinamiuriMasivi1.AddRange(striqoni2); // DinamiuriMasivi1 = "ანა", "საბა", "ლიკა",
                                      // "რომანი", "გიორგი", "ნატა", "ბექა"
for (int index = 0; index < DinamiuriMasivi1.Count; index++)
   label1.Text += DinamiuriMasivi1[index] + "  ";
}
```

Insert() მეთოდი ახდენს მითითებული ელემენტის ჩასმას მითითებულ პოზიციაში. მისი სინტაქსია:

arrayList.Insert*(ინდექსი, სიდიდე)*

მაგალითი.

```
{
string[] striqoni1 = { "ანა", "საბა", "ლიკა", "რომანი", "გიორგი" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni1);
DinamiuriMasivi1.Insert(2, "ლევანი"); // DinamiuriMasivi1 = "ანა", "საბა",
                                      // "ლევანი", "ლიკა", "რომანი", "გიორგი"
for ( int index = 0; index < DinamiuriMasivi1.Count; index++ )
   label1.Text += DinamiuriMasivi1[index] + "  ";
}
```

InsertRange() მეთოდი ახდენს კოლექციის ჩამატებას მითითებული პოზიციიდან. მისი სინტაქსია:

arrayList.InsertRange*(ინდექსი, კოლექცია)*

მაგალითი.

```
{
string[] striqoni1 = { "ანა", "საბა", "ლიკა", "რომანი", "გიორგი" };
string[] striqoni2 = { "ნატა", "ბექა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni1);
DinamiuriMasivi1.InsertRange(2, striqoni2); // DinamiuriMasivi1 = "ანა", "საბა",
                                            // "ნატა", "ბექა", "ლიკა", "რომანი", "გიორგი"
for ( int index = 0; index < DinamiuriMasivi1.Count; index++ )
   label2.Text += DinamiuriMasivi1[index] + "  ";
}
```

SetRange() მეთოდი ახდენს დინამიური მასივის ელემენტების დიაპაზონის შეცვლას კოლექციის ელემენტებით დაწყებული მითითებული ინდექსიდან. მისი სინტაქსია:

arrayList.SetRange*(ინდექსი, კოლექცია)*

მაგალითი.

```
{
string[] striqoni1 = { "ანა", "საბა", "ლიკა", "რომანი", "გიორგი" };
string[] striqoni2 = { "ნატა", "ბექა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni1);
DinamiuriMasivi1.SetRange(2, striqoni2);     // DinamiuriMasivi1 = "ანა", "საბა",
                                             // "ნატა", "ბექა", "გიორგი"
for ( int index = 0; index < DinamiuriMasivi1.Count; index++ )
   label1.Text += DinamiuriMasivi1[index] + "  ";
}
```

ელემენტების ძებნა. **Contains()** მეთოდი ამოწმებს შეიცავს თუ არა დინამიური მასივი მითითებულ ელემენტს. თუ ელემენტი სიაში არსებობს, მაშინ მეთოდი გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას. მისი სინტაქსია:

arrayList.Contains*(სიდიდე)*

მაგალითი.

```
{
bool elementi_aris;
string[] striqoni = { "რომანი", "ანა", "საბა", "ლიკა", " რომანი ", "საბა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
elementi_aris = DinamiuriMasivi1.Contains("ანა");
if ( elementi_aris == true ) label1.Text = "ელემენტი არის მასივში";
      else label1.Text = "ელემენტი არ არის მასივში";
}
```

IndexOf() მეთოდი დინამიურ მასივში ეძებს მითითებულ ელემენტს. თუ ეს ელემენტი რამდენიმეა მასივში, მაშინ გაიცემა პირველი მათგანის ინდექსი, წინააღმდეგ შემთხვევაში კი - -1. მისი სინტაქსია:

arrayList.IndexOf*(სიდიდე)*

მაგალითი.

```
{
int index1;
string[] striqoni = { "რომანი", "ანა", "საბა", "ლიკა", " რომანი ", "საბა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
index1 = DinamiuriMasivi2.IndexOf("საბა");                                    // index1 = 2
label1.Text = " პირველი " + DinamiuriMasivi2[index1] +
                " ელემენტის ინდექსია " + index1.ToString();
}
```

LastIndexOf() მეთოდი დინამიურ მასივში ეძებს მითითებულ ელემენტს. თუ ეს ელემენტი რამდენიმეა მასივში, მაშინ გაიცემა უკანასკნელი მათგანის ინდექსი, წინააღმდეგ შემთხვევაში კი - -1. მისი სინტაქსია:

arrayList.LastIndexOf*(სიდიდე)*

მაგალითი.

```
{
int index1;
string[] striqoni = { "რომანი", "ანა", "საბა", "ლიკა", " რომანი ", "საბა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
index1 = DinamiuriMasivi1.LastIndexOf("რომანი");                              // index1 = 4
label1.Text = " უკანასკნელი " + DinamiuriMasivi1[index1] +
                " ელემენტის ინდექსია " + index1.ToString();
}
```

ელემენტების წაშლა. **RemoveAt()** მეთოდი. იგი დინამიური მასივიდან შლის მითითებული ინდექსის მქონე ელემენტს. მისი სინტაქსია:

arrayList.RemoveAt*(ინდექსი)*

მაგალითი.

```
{
string[] striqoni = { "ანა", "საბა", "ლიკა", "რომანი" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
DinamiuriMasivi1.RemoveAt(1);          // DinamiuriMasivi1 = "ანა", "ლიკა", "რომანი"
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + "  ";
}
```

Remove() მეთოდი. იგი დინამიური მასივიდან შლის მითითებულ ელემენტს. მისი სინტაქსია:

arrayList.Remove*(სიდიდე)*

მაგალითი.

```
{
string[] striqoni = { "ანა", "საბა", "ლიკა", "რომანი" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
DinamiuriMasivi1.Remove("რომანი");    // DinamiuriMasivi1 = "ანა", "საბა", "ლიკა"
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + "  ";
}
```

RemoveRange() მეთოდი. იგი დინამიური მასივიდან შლის ელემენტების დიაპაზონს. მისი სინტაქსია:

arrayList.RemoveRange*(საწყისი_ინდექსი, ელემენტების_რაოდენობა)*

მაგალითი.

```
{
string[] striqoni = { "ანა", "საბა", "ლიკა", "რომანი" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
DinamiuriMasivi1.RemoveRange(1, 2);       // DinamiuriMasivi1 = "ანა", "რომანი"
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + "  ";
}
```

ელემენტების დახარისხება, ძებნა და მიმდევრობის შებრუნება. **Sort()** მეთოდი დინამიური მასივის ელემენტებსს ზრდადობის მიხედვით ალაგებს. მისი სინტაქსია:
arrayList.Sort()

მაგალითი.

```
{
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
DinamiuriMasivi1.Sort();     // DinamiuriMasivi1 = "ანა", "ლიკა", "რომანი", "საბა"
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + "  ";
}
```

BinarySearch() მეთოდი დახარისხებულ დინამიურ მასივში ეძებს მითითებულ ელემენტს. პოვნის შემთხვევაში გაიცემა მისი ინდექსი. მისი სინტაქსია:
arrayList.BinarySearch*(ინდექსი)*

მაგალითი.

```
{
int indexi;
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
DinamiuriMasivi1.Sort();
indexi = DinamiuriMasivi1.BinarySearch("რომანი");                    // indexi = 2
label1.Text = indexi.ToString();
}
```

Reverse() მეთოდი ახდენს დინამიური მასივის ელემენტების მიმდევრობის შებრუნებას. მისი სინტაქსია:
arrayList.Reverce()

მაგალითი.

```
{
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + "  ";
DinamiuriMasivi1.Reverse();          // DinamiuriMasivi1 = "ანა", "რომანი", "ლიკა", "საბა"
foreach ( string str1 in DinamiuriMasivi1 ) label2.Text += str1 + "  ";
}
```

დინამიური მასივის ტევადობის შემცირება და ელემენტების დიაპაზონის მიღება. **TrimToSize()** მეთოდი დინამიური მასივის ტევადობას ამცირებს მასივში რეალურად არსებული ელემენტების რაოდენობამდე. მისი სინტაქსია:
arrayList.TrimToSize()

GetRange() მეთოდი გასცემს დინამიური მასივის მითითებული რაოდენობის ელემენტს დაწყებული მითითებული პოზიციიდან. მისი სინტაქსია:

arrayList.GetRange*(საწყისი_ინდექსი, ელემენტების_რაოდენობა)*

მაგალითი.

```
{
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა", "ნატა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
ArrayList DinamiuriMasivi2 = DinamiuriMasivi1.GetRange(1, 3);
                                        // DinamiuriMasivi2 = "ლიკა", "რომანი", "ანა"
foreach ( string str1 in DinamiuriMasivi2 ) label1.Text += str1 + "  ";
}
```

ჩამომთვლელების გამოყენება. ვნახოთ თუ როგორ შეიძლება ჩამომთვლელების გამოყენება დინამიური მასივიდან ელემენტების წაკითხვისათვის. ზემოთ აღნიშნული ყველა სახის კოლექცია ახდენს IEnumerable ინტერფეისის რეალიზებას. ამ ინტერფეისში განსაზღვრულია Current თვისება და GetEnumerator(), MoveNext() და Reset() მეთოდები.

GetEnumerator() მეთოდი გასცემს ჩამომთვლელს, რომელიც IEnumerator ობიექტს წარმოადგენს და გამოიყენება კოლექციის ელემენტებთან მიმართვისათვის. ჩვენ განვიხილავთ მისი ორი ვერსიის სინტაქსს:

arrayList.GetEnumerator()

arrayList.GetEnumerator(საწყისი_ინდექსი, ელემენტების_რაოდენობა)

MoveNext() მეთოდი ჩამომთვლელს ათავსებს კოლექციის მომდევნო ელემენტზე და გასცემს true მნიშვნელობას თუ მომდევნო ელემენტი არსებობს, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.

Reset() მეთოდი ჩამომთვლელს ათავსებს საწყის მდგომარეობაში, ანუ კოლექციის პირველი ელემენტის წინ.

მოყვანილ პროგრამაში ხდება დინამიურ მასივთან ჩამომთვლელის საშუალებით მუშაობის დემონსტრირება.

```
{
// პროგრამა 14.2
// პროგრამაში ხდება დინამიურ მასივთან ჩამომთვლელის საშუალებით მუშაობის დემონსტრირება
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა", "ნატა" };
ArrayList DinamiuriMasivi = new ArrayList(striqoni);
IEnumerator Chamomtvleli = DinamiuriMasivi.GetEnumerator();
while ( Chamomtvleli.MoveNext() )
        label1.Text += "myEnumerator.Current = " +
                        Chamomtvleli.Current.ToString() + '\n';
//
Chamomtvleli.Reset();
Chamomtvleli.MoveNext();
label1.Text += "myEnumerator.Current = " +
          Chamomtvleli.Current.ToString() + '\n';                  // გამოჩნდება - “საბა”
}
```

საკუთარი კლასების ობიექტების დინამიური მასივები. დინამიური მასივები ჩვეულებრივი ტიპის მქონე ცვლადების გარდა შეიძლება შეიცავდეს ჩვენი საკუთარი კლასების ობიექტებს. თუ ვქმნით დინამიურ მასივს, რომელიც ჩვენს მიერ შექმნილი კლასის ობიექტებს შეიცავს, და ვაპირებთ მათ დახარისხებას, მაშინ ჩვენს მიერ

შექმნილი კლასი უნდა უზრუნველყოფდეს IComparable ინტერფეისს. ეს ინტერფეისი ჩვენს მიერ შექმნილი კლასისაგან ითხოვს CompareTo() მეთოდის რეალიზებას. გარდა ამისა, ჩვენი კლასი უნდა ახდენდეს IComparer ინტერფეისის Compare() მეთოდის რეალიზებას. მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```
//      პროგრამა 14.3
//      პროგრამაში ხდება საკუთარი კლასის ობიექტების დინამიურ მასივთან მუშაობის
//      დემონსტრირება
public class Manqana : IComparable
{
public string Modeli;
public int GamoshvebisWeli;
public Manqana(string par_model, int par_gamoshvebisweli)
{
  this.Modeli = par_model;
  this.GamoshvebisWeli = par_gamoshvebisweli;
}
//      გადატვირთული ToString() მეთოდი
public override string ToString()
{
  return "მოდელი არის " + Modeli + ", გამოშვების წელი არის " + GamoshvebisWeli;
}
//      IComparer ინტერფეისის Compare()მეთოდის რეალიზაცია
public int Compare(object obj1, object obj2)
{
Manqana obj1Manqana = (Manqana)obj1;
Manqana obj2Manqana = (Manqana)obj2;
if (obj1Manqana.GamoshvebisWeli < obj2Manqana.GamoshvebisWeli)
{
return -1;
}
else if (obj1Manqana.GamoshvebisWeli > obj2Manqana.GamoshvebisWeli)
{
return 1;
}
else
{
return 0;
}
}
//      IComparer ინტერფეისის CompareTo() მეთოდის რეალიზება
public int CompareTo(object obj1)
{
    return Compare(this, obj1);
}
}
```

```
public void MasivisGamotana(string DinamiuriMasivisSaxeli,
                                    ArrayList DinamiuriMasivi)
{
for (int indexi = 0; indexi < DinamiuriMasivi.Count; indexi++)
   label1.Text += DinamiuriMasivisSaxeli + "[" + indexi.ToString() + "] = " +
                          DinamiuriMasivi[indexi].ToString() + '\n';
}
private void button1_Click(object sender, EventArgs e)
{
ArrayList DinamiuriMasivi = new ArrayList();
//        სიისთვის ოთხი ელემენტის დამატება
Manqana myOpel   = new Manqana("Opel", 2006);
Manqana myFord   = new Manqana("Ford", 2007);
Manqana myBMW    = new Manqana("BMW" , 2005);
Manqana myToiota = new Manqana("Toiota", 2000);
DinamiuriMasivi.Add(myOpel);
DinamiuriMasivi.Add(myFord);
DinamiuriMasivi.Add(myBMW);
DinamiuriMasivi.Add(myToiota);
MasivisGamotana("DinamiuriMasivi", DinamiuriMasivi);
//
if ( DinamiuriMasivi.Contains(myFord) )
{
int index = DinamiuriMasivi.IndexOf(myFord) + '\n';                //        index = 11
label1.Text += "myFord ობიექტის ინდექსია = " + index.ToString() + '\n';
}
//
DinamiuriMasivi.Remove(myFord);
MasivisGamotana("DinamiuriMasivi", DinamiuriMasivi);
//
DinamiuriMasivi.Sort();
MasivisGamotana("DinamiuriMasivi", DinamiuriMasivi);
//
int index2 = DinamiuriMasivi.BinarySearch(myBMW);             // index2 = 1
label1.Text += "myBMW ობიექტის ინდექსია = " + index2.ToString() + '\n';
//
ArrayList MeoreDinamiuriMasivi = DinamiuriMasivi.GetRange(1, 2);
MasivisGamotana("MeoreDinamiuriMasivi", MeoreDinamiuriMasivi);
//
        IEnumerator Chamomtvleli = DinamiuriMasivi.GetEnumerator();
        while ( Chamomtvleli.MoveNext() )
           label1.Text += " Chamomtvleli.Current = " + Chamomtvleli.Current.ToString() + '\n';
//
Chamomtvleli.Reset();
Chamomtvleli.MoveNext();
```

```
label1.Text += "Chamomtvleli.Current = " +
                Chamomtvleli.Current.ToString() + '\n';
}
```

ჰეშ-ცხრილები

ჰეშ-ცხრილები ინახავენ წყვილებს - გასაღები-მნიშვნელობა, ანუ ჰეშ-ცხრილის თითოეული ელემენტი შედგება გასაღებისა და მნიშვნელობისაგან. გასაღები გამოიყენება შესაბამისი მნიშვნელობის მისაღებად. ჰეშ-ცხრილები განსაზღვრულია Hashtable კლასში. ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.3 და 14.4. ჰეშ-ცხრილში გასაღები და მნიშვნელობა შეიძლება იყოს ნებისმიერი ტიპის ობიექტი. თუმცა, ჩვეულებრივ, გასაღების როლში ხშირად სტრიქონებს იყენებენ.

ცხრილი 14.3. Hashtable კლასის თვისებები

თვისებები	ტიპი	აღწერა
Count	int	შეიცავს ჰეშ-ცხრილში შენახული ელემენტების რაოდენობას
IsFixedSize	bool	ამოწმებს აქვს თუ არა ჰეშ-ცხრილს ფიქსირებული სიგრძე
IsReadOnly	bool	ამოწმებს არის თუ არა ჰეს-ცხრილი მხოლოდ წაკითხვადი
Item	object	გასცემს ან აყენებს მითითებული გასაღების მქონე ელემენტის მნიშვნელობას
Keys	ICollection	გასცემს კოლექციას, რომელიც ჰეშ-ცხრილის გასაღებებისაგან შედგება
Values	ICollection	გასცემს კოლექციას, რომელიც ჰეშ-ცხრილის მნიშვნელობებისაგან შედგება

ცხრილი 14.4. Hashtable კლასის მეთოდები

მეთოდები	დაბრუნებული ტიპი	აღწერა
Add()	void	ჰეშ-ცხრილს უმატებს მითითებული გასაღებისა და მნიშვნელობის მქონე ელემენტს
Clear()	void	ჰეშ-ცხრილიდან შლის ყველა ელემენტს
Clone()	object	ქმნის ჰეშ-ცხრილის ასლს
Contains()	bool	განსაზღვრავს, შეიცავს თუ არა ჰეშ-ცხრილი მითითებულ გასაღებს
ContainsKey()	bool	განსაზღვრავს, შეიცავს თუ არა ჰეშ-ცხრილი მითითებულ გასაღებს
ContainsValue()	bool	განსაზღვრავს, შეიცავს თუ არა ჰეშ-ცხრილი მითითებულ მნიშვნელობას
CopyTo()	void	ერთგანზომილებიან მასივში ჰეშ-ცხრილიდან გადაწერს გასაღებებს ან მნიშვნელობებს

ცხრილი 14.4. (გაგრძელება)

Equals()	bool	ამოწმებს, ტოლია თუ არა ორი ჰეშ-ცხრილი
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს, რომელიც გამოიყენება ჰეშ-ცხრილის ელემენტებთან სამუშაოდ
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს
Remove()	void	ჰეშ-ცხრილიდან შლის მითითებული გასაღების მქონე ელემენტს
ToString()	string	გასცემს მიმდინარე ობიექტის შესაბამის სტრიქონს

ელემენტების დამატება, წაშლა და მიღება. **Add()** მეთოდი გამოიყენება ჰეშ-ცხრილში ელემენტების დასამატებლად. მისი სინტაქსია:

Hashtable.Add*(გასაღები, მნიშვნელობა)*

Remove() მეთოდი გამოიყენება ჰეშ-ცხრილში ელემენტების წასაშლელად. მისი სინტაქსია:

Hashtable.Remove*(გასაღები)*

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 14.4
//      პროგრამაში ხდება Add() და Remove() მეთოდებთან მუშაობის დემონსტრირება
Hashtable HeshCxrili = new Hashtable();
//
HeshCxrili.Add("კა", "კახეთი");
HeshCxrili.Add("იმ", "იმერეთი");
HeshCxrili.Add("თუ", "თუშეთი");
HeshCxrili.Add("ხე", "ხევსურეთი");
foreach ( string Gasagebi in HeshCxrili.Keys )
   label1.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in HeshCxrili.Values )
   label1.Text += Mnishvneloba.ToString() + '\n';
//
HeshCxrili.Remove("თუ");
foreach ( string Gasagebi in HeshCxrili.Keys )
   label2.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in HeshCxrili.Values )
   label2.Text += Mnishvneloba.ToString() + '\n';
string Regioni = (string)HeshCxrili["იმ"];
label3.Text = Regioni;
}
```

როგორც პროგრამიდან ჩანს, ჰეშ-ცხრილიდან საჭირო მნიშვნელობის მისაღებად კვადრატულ ფრჩხილებში უნდა მივუთითოთ მისი შესაბამისი გასაღები, მაგალითად, HeshCxrili["იმ"]. გასაღებების მისაღებად უნდა გამოვიყენოთ Keys თვისება, ხოლო მნიშვნელობების მისაღებად კი - Values თვისება.

გასაღებისა და მნიშვნელობის ძებნა. **ContainsKey()** მეთოდი გამოიყენება ჰეშ-

ცხრილში მითითებული გასაღების მოსაძებნად. პოვნის შემთხვევაში გაიცემა true, წინააღმდეგ შემთხვევაში - false. მისი სინტაქსია:
Hashtable.ContainsKey*(გასაღები)*

ContainsValue() მეთოდი გამოიყენება ჰეშ-ცხრილში მითითებული მნიშვნელობის მოსაძებნად. პოვნის შემთხვევაში გაიცემა true, წინააღმდეგ შემთხვევაში - false. მისი სინტაქსია:
Hashtable.ContainsValue*(მნიშვნელობა)*

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 14.5
//      პროგრამაში ხდება ContainsKey() და ContainsValue() მეთოდებთან მუშაობის დემონსტრირება
Hashtable HeshCxrili = new Hashtable();
//
HeshCxrili.Add("კა", "კახეთი");
HeshCxrili.Add("იმ", "იმერეთი");
HeshCxrili.Add("თუ", "თუშეთი");
HeshCxrili.Add("ხე", "ხევსურეთი");
foreach ( string Gasagebi in HeshCxrili.Keys )
      label1.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in HeshCxrili.Values )
      label1.Text += Mnishvneloba.ToString() + '\n';
//
if ( HeshCxrili.ContainsKey("კა") )
         label2.Text = "ჰეშ-ცხრილი შეიცავს მითითებულ გასაღებს";
else     label2.Text = "ჰეშ-ცხრილი არ შეიცავს მითითებულ გასაღებს";
//
if ( HeshCxrili.ContainsValue("ხევსურეთი") )
         label3.Text = "ჰეშ-ცხრილი შეიცავს მითითებულ მნიშვნელობას";
else     label3.Text = "ჰეშ-ცხრილი არ შეიცავს მითითებულ მნიშვნელობას";
}
```

გასაღებებისა და მნიშვნელობების გადაწერა ერთგანზომილებიან მასივში. **CopyTo()** მეთოდი გამოიყენება ჰეშ-ცხრილიდან გასაღებებისა და მნიშვნელობების გადასაწერად ერთგანზომილებიან მასივში. მისი სინტაქსია:
Hashtable.Keys.CopyTo*(მასივი, ინდექსი)*
მეთოდი დაწყებული ინდექსი პოზიციიდან მითითებულ მასივში გადაწერს ჰეშ-ცხრილის გასაღებებს.
Hashtable.Values.CopyTo*(მასივი, ინდექსი)*
მეთოდი დაწყებული ინდექსი პოზიციიდან მითითებულ მასივში გადაწერს ჰეშ-ცხრილის მნიშვნელობებს.

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 14.6
//      პროგრამაში ხდება CopyTo() მეთოდთან მუშაობის დემონსტრირება
Hashtable HeshCxrili = new Hashtable();
```

```
HeshCxrili.Add("კა", "კახეთი");
HeshCxrili.Add("იმ", "იმერეთი");
HeshCxrili.Add("თუ", "თუშეთი");
HeshCxrili.Add("ხე", "ხევსურეთი");
string[] Gasagebebi     = new string[HeshCxrili.Count];
string[] Mnishvnelobebi = new string[HeshCxrili.Count];

foreach ( string Gasagebi in HeshCxrili.Keys )
   label1.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in HeshCxrili.Values )
   label1.Text += Mnishvneloba.ToString() + '\n';
//
HeshCxrili.Keys.CopyTo(Gasagebebi, 0);
foreach ( string Gasagebi in Gasagebebi )
   label2.Text += Gasagebi + '\n';
//
HeshCxrili.Values.CopyTo(Mnishvnelobebi, 0);
foreach ( string Mnishvneloba in Mnishvnelobebi )
   label3.Text += Mnishvneloba + '\n';
}
```

დახარისხებული სიები

დახარისხებული სია არის დინამიური მასივისა და ჰეშ-ცხრილის კომბინაცია. დახარისხებული სიის ელემენტის მიღება შეიძლება როგორც ინდექსის, ისე გასაღების მიხედვით. გასაღებები ასეთ მასივებში დახარისხებულია ზრდადობის მიხედვით. მასივში ახალი ელემენტის ჩამატებისას იგი იმ პოზიციაში მოთავსთდება, რომ მასივი დახარისხებული დარჩეს. დახარისხებული სია განსაზღვრულია SortedList კლასში. ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.5 და 14.6.

ცხრილი 14.5. SortedList კლასის თვისებები

თვისება	ტიპი	აღწერა
Capacity	int	განსაზღვრავს დახარისხებული სიის ტევადობას
Count	int	შეიცავს დახარისხებულ სიაში რეალურად არსებული ელემენტების რაოდენობას
IsFixedSize	bool	ამოწმებს, აქვს თუ არა დახარისხებულ მასივს ფიქსირებული სიგრძე
IsReadOnly	bool	ამოწმებს, არის თუ არა დახარისხებული სია მხოლოდ წაკითხვადი
Item	object	გასცემს ან აყენებს მითითებული გასაღების მქონე ელემენტის მნიშვნელობას
Keys	ICollection	გასცემს დახარისხებული სიის გასაღებებისაგან შემდგარ კოლექციას
Values	ICollection	გასცემს დახარისხებული სიის მნიშვნელობებისაგან შემდგარ კოლექციას

ცხრილი 14.6. SortedList კლასის მეთოდები

მეთოდი	დაბრუნებული ტიპი	აღწერა
Add()	void	დახარისხებულ სიას ელემენტს უმატებს
Clear()	void	დახარისხებული სიიდან შლის ყველა ელემენტს
Clone()	object	ქმნის დახარისხებული სიის ასლს
Contains()	bool	მითითებულ გასაღებს ეძებს დახარისხებულ სიაში
ContainsKey()	bool	მითითებულ გასაღებს ეძებს დახარისხებულ სიაში
ContainsValue()	bool	მითითებულ მნიშვნელობას ეძებს დახარისხებულ სიაში
CopyTo()	void	ერთგანზომილებიან მასივში გადაწერს დახარისხებული სიის მნიშვნელობებს ან გასაღებებს
Equals()	bool	ამოწმებს, ტოლია თუ არა ორი ობიექტი
GetByIndex()	object	დახარისხებული სიიდან გასცემს მითითებული ინდექსის მქონე მნიშვნელობას
GetEnumerator()	IDictionaryEnumerator	გასცემს ჩამომთვლელს დახარისხებული სიისათვის
GetKey()	object	გასცემს დახარისხებული სიის მითითებული პოზიციის გასაღებს
GetKeyList()	IList	გასცემს დახარისხებული სიის ყველა გასაღებს
GetType()	Type	გასცემს მოცემული ობიექტის ტიპს
GetValueList()	IList	გასცემს დახარისხებული სიის ყველა მნიშვნელობას
IndexOfKey()	int	გასცემს მითითებული გასაღების ინდექსს
IndexOfValue()	int	გასცემს მითითებული მნიშვნელობის ინდექსს
Remove()	void	დახარისხებული სიიდან შლის მითითებული გასაღების შემცველ ელემენტს
RemoveAt()	void	დახარისხებული სიიდან შლის მითითებული მნიშვნელობის შემცველ ელემენტს
SetByIndex()	void	ცვლის მნიშვნელობას ინდექსით მითითებულ პოზიციაში
ToString()	string	გასცემს მიმდინარე ობიექტის შესაბამის სტრიქონს
TrimToSize()	void	დახარისხებული სიის ზომას ამცირებს მასში რეალურად არსებული ელემენტების რაოდენობამდე

ელემენტების დამატება, წაშლა და მიღება. **Add()** მეთოდი გამოიყენება დახარისხებულ სიაში ელემენტების დასამატებლად. მისი სინტაქსია:
sortedList.Add*(გასაღები, მნიშვნელობა)*

Remove() მეთოდი გამოიყენება დახარისხებული სიიდან ელემენტების წასაშლელად. მისი სინტაქსია:
sortedList.Remove*(გასაღები)*

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 14.7
//      პროგრამაში ხდება Add() და Remove() მეთოდებთან მუშაობის დემონსტრირება
SortedList DaxarisxebuliSia = new SortedList ();
//
DaxarisxebuliSia.Add("კა", "კახეთი");
DaxarisxebuliSia.Add("იმ", "იმერეთი");
DaxarisxebuliSia.Add("თუ", "თუშეთი");
DaxarisxebuliSia.Add("ხე", "ხევსურეთი");
foreach ( string Gasagebi in DaxarisxebuliSia.Keys )
   label1.Text += Gasagebi.ToString() + '\n';
foreach ( string MniSvneloba in DaxarisxebuliSia.Values )
   label1.Text += MniSvneloba.ToString() + '\n';
//
DaxarisxebuliSia.Remove("თუ");
foreach ( string Gasagebi in DaxarisxebuliSia.Keys )
   label2.Text += Gasagebi.ToString() + '\n';
foreach ( string MniSvneloba in DaxarisxebuliSia.Values )
   label2.Text += MniSvneloba.ToString() + '\n';
string Regioni = (string)DaxarisxebuliSia["იმ"];
label3.Text += Regioni;
}
```

როგორც პროგრამიდან ჩანს, დახარისხებული სიიდან საჭირო მნიშვნელობის მისაღებად კვადრატულ ფრჩხილებში უნდა მივუთითოთ მისი შესაბამისი გასაღები, მაგალითად, DaxarisxebuliSia["იმ"]. გასაღებების მისაღებად უნდა გამოვიყენოთ Keys თვისება, ხოლო მნიშვნელობების მისაღებად კი - Values თვისება.

მნიშვნელობის მიღება და შეცვლა ინდექსის მიხედვით. **GetByIndex()** მეთოდი გამოიყენება დახარისხებული სიიდან საჭირო მნიშვნელობის მისაღებად ინდექსის მიხედვით. მისი სინტაქსია:
sortedList.GetByIndex*(ინდექსი)*

მაგალითი.

```
string regioni1 = ( string ) DaxarisxebuliSia.GetByIndex(2);
```

SetByIndex() მეთოდი გამოიყენება დახარისხებული სიაში მნიშვნელობის შესაცვლელად ინდექსის მიხედვით. მისი სინტაქსია:
sortedList.SetByIndex*(ინდექსი, მნიშვნელობა)*

მაგალითი.

```
DaxarisxebuliSia.SetByIndex(2, “რაჭა”);
```

გასაღების, გასაღებისა და მნიშვნელობის ინდექსის მიღება. **GetKey()** მეთოდი გამოიყენება დახარისხებული სიიდან გასაღების მისაღებად ინდექსის მიხედვით. მისი სინტაქსია:

sortedList.GetKey*(ინდექსი)*

მაგალითი.

```
string Gasagebi = ( string ) DaxarisxebuliSia. GetKey (2);
```

IndexOfKey() მეთოდი გამოიყენება დახარისხებული სიიდან მითითებული გასაღების ინდექსის მისაღებად. მისი სინტაქსია:

sortedList.IndexOfKey*(გასაღები)*

მაგალითი.

```
int GasagebisIndeqsi = DaxarisxebuliSia.IndexOfKey("იმ");
```

IndexOfValue() მეთოდი გამოიყენება დახარისხებული სიიდან მითითებული მნიშვნელობის ინდექსის მისაღებად. მისი სინტაქსია:

sortedList.IndexOfValue*(მნიშვნელობა)*

მაგალითი.

```
int MnishvnelobisIndeqsi = DaxarisxebuliSia.IndexOfValue("იმერეთი");
```

გასაღებებისა და მნიშვნელობების სიის მიღება. **GetKeyList()** მეთოდი გამოიყენება დახარისხებული სიიდან გასაღებების სიის გასაცემად. მეთოდი გასცემს IList ინტერფეისის ობიექტს. მეთოდის სინტაქსია:

sortedList.GetKeyList()

მაგალითი.

```
{
IList GasagebebisSia = DaxarisxebuliSia.GetKeyList();
foreach ( string Gasagebi in GasagebebisSia )
        label1.Text += Gasagebi + "   ";
}
```

GetValueList() მეთოდი გამოიყენება დახარისხებული სიიდან მნიშვნელობების სიის გასაცემად. მეთოდი გასცემს IList ინტერფეისის ობიექტს. მეთოდის სინტაქსია:

sortedList. GetValueList()

მაგალითი.

```
{
IList MnishvnelobebisSia = DaxarisxebuliSia.GetValueList();
foreach ( string Gasagebi in MnishvnelobebisSia )
        label1.Text += Gasagebi + "   ";
}
```

გასაღებისა და მნიშვნელობის ძებნა. **ContainsKey()** მეთოდი გამოიყენება დახარისხებულ სიაში მითითებული გასაღების მოსაძებნად. პოვნის შემთხვევაში გაიცემა true მნიშვნელობა, წინააღმდეგ შემთხვევაში კი - false. მისი სინტაქსია:

sortedList.ContainsKey*(გასაღები)*

ContainsValue() მეთოდი გამოიყენება დახარისხებულ სიაში მითითებული მნიშვნელობის მოსაძებნად. პოვნის შემთხვევაში გაიცემა true, წინააღმდეგ შემთხვევაში - false. მისი სინტაქსია:

sortedList.ContainsValue*(მნიშვნელობა)*

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```
{
```

```
//      პროგრამა 14.8
//      პროგრამაში ხდება ContainsKey() და ContainsValue()მეთოდებთან მუშაობის დემონსტრირება
SortedList DaxarisxebuliSia = new SortedList();
//
DaxarisxebuliSia.Add("კა", "კახეთი");
DaxarisxebuliSia.Add("იმ", "იმერეთი");
DaxarisxebuliSia.Add("თუ", "თუშეთი");
DaxarisxebuliSia.Add("ხე", "ხევსურეთი");
foreach ( string Gasagebi in DaxarisxebuliSia.Keys )
   label1.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in DaxarisxebuliSia.Values )
   label1.Text += Mnishvneloba.ToString() + '\n';
//
if ( DaxarisxebuliSia.ContainsKey("კა") )
        label2.Text = "დახარისხებული სია შეიცავს მითითებულ გასაღებს";
else    label2.Text = "დახარისხებული სია არ შეიცავს მითითებულ გასაღებს";
//
if ( DaxarisxebuliSia.ContainsValue("ხევსურეთი") )
        label2.Text = "დახარისხებული სია შეიცავს მითითებულ მნიშვნელობას";
else    label2.Text = "დახარისხებული სია არ შეიცავს მითითებულ მნიშვნელობას";
}
```

გასაღებებისა და მნიშვნელობების გადაწერა ერთგანზომილებიან მასივში. **CopyTo()** მეთოდი გამოიყენება დახარისხებული სიიდან გასაღებებისა და მნიშვნელობების გადასაწერად ერთგანზომილებიან მასივში. მისი სინტაქსია:

sortedList.Keys.CopyTo*(მასივი, ინდექსი)*
sortedList.Values.CopyTo*(მასივი, ინდექსი)*

Keys თვისების CopyTo მეთოდი დაწყებული ინდექსი პოზიციიდან მითითებულ მასივში გადაწერს დახარისხებული სიის გასაღებებს. Values თვისების CopyTo მეთოდი დაწყებული ინდექსი პოზიციიდან მითითებულ მასივში გადაწერს დახარისხებული სიის მნიშვნელობებს.

მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```
{
//      პროგრამა 14.9
//      პროგრამაში ხდება CopyTo() მეთოდთან მუშაობის დემონსტრირება
SortedList DaxarisxebuliSia = new SortedList();
//
DaxarisxebuliSia.Add("კა", "კახეთი");
DaxarisxebuliSia.Add("იმ", "იმერეთი");
DaxarisxebuliSia.Add("თუ", "თუშეთი");
DaxarisxebuliSia.Add("ხე", "ხევსურეთი");
string[] Gasagebebi = new string[DaxarisxebuliSia.Count];
string[] Mnishvnelobebi = new string[DaxarisxebuliSia.Count];
foreach ( string Gasagebi in DaxarisxebuliSia.Keys )
   label1.Text += Gasagebi.ToString() + '\n';
```

```
foreach ( string Mnishvneloba in DaxarisxebuliSia.Values )
   label1.Text += Mnishvneloba.ToString() + '\n';
//
DaxarisxebuliSia.Keys.CopyTo(Gasagebebi, 0);
foreach ( string Gasagebi in Gasagebebi )
   label2.Text += Gasagebi + '\n';
//
DaxarisxebuliSia.Values.CopyTo(Mnishvnelobebi, 0);
foreach ( string Mnishvneloba in Mnishvnelobebi )
   label3.Text += Mnishvneloba + '\n';
}
```

რიგები

რიგი არის მეხსიერების უბანი, რომელიც მუშაობს პრინციპით "პირველი მოვიდა - პირველი წავიდა" (FIFO - First In, First Out). რიგი განსაზღვრულია Queue კლასში. ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.7 და 14.8.

ცხრილი 14.7. Queue კლასის თვისებები

თვისება	ტიპი	აღწერა
Count	int	შეიცავს რიგში რეალურად არსებული ელემენტების რაოდენობას
IsReadOnly	bool	ამოწმებს, არის თუ არა რიგი მხოლოდ წაკითხვადი

ცხრილი 14.8. Queue კლასის მეთოდები

თვისება	დაბრუნებული ტიპი	აღწერა
Clear()	void	რიგიდან ყველა ელემენტს შლის
Clone()	object	ქმნის რიგის ასლს
Contains()	bool	ამოწმებს, არის თუ არა რიგში მითითებული ელემენტი
CopyTo	void	რიგის ელემენტებს გადაწერს ერთგანზომილებიან მასივში
Dequeu()	object	გასცემს რიგის პირველი ელემენტის მნიშვნელობას და მას რიგიდან შლის
Enqueue()	void	რიგს ბოლოში უმატებს ელემენტს
Equals()	bool	ამოწმებს, ტოლია თუ არა ორი რიგი
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს რიგის ელემენტების გასწვრივ იტერაციისათვის
GetType()	Type	გასცემს რიგის მიმდინარე ელემენტის ტიპს
Peek()	object	გასცემს რიგის პირველი ელემენტის მნიშვნელობას, მაგრამ არ შლის მას რიგიდან
ToArray()	object[]	რიგის ელემენტებს გადაწერს მასივში
ToString()	string	გასცემს რიგის მიმდინარე ელემენტის შესაბამის სტრიქონს

მოყვანილ პროგრამაში ხდება **Enqueue(), Dequeu()** და **Peek()** მეთოდებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 14.10
//      პროგრამაში ხდება Enqueue(), Dequeu() და Peek() მეთოდებთან მუშაობის დემონსტრირება
// რიგის შექმნა
Queue Rigi = new Queue();
// რიგში ელემენტების დამატება
Rigi.Enqueue("ანა");
Rigi.Enqueue("საბა");
Rigi.Enqueue("ლიკა");
Rigi.Enqueue("რომანი");
// რიგის ელემენტების გამოტანა
foreach ( string striqoni in Rigi )
      label1.Text += striqoni + '\n';
// რიგში ელემენტების რაოდენობის მიღება
int ElementebisRaodenoba = Rigi.Count;
for ( int count = 0; count < ElementebisRaodenoba; count++ )
   {
     label2.Text += "Peek()   = " + Rigi.Peek()    + '\n';
     label2.Text += "Dequeu() = " + Rigi.Dequeue() + '\n';
   }
}
```

სტეკები

სტეკი არის მეხსიერების უბანი, რომელიც მუშაობს პრინციპით “უკანასკნელი მოვიდა - პირველი გავიდა” (LIFO - Last In, First Out). სტეკი განსაზღვრულია Stack კლასში. ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.9 და 14.10.

ცხრილი 14.9. Stack კლასის თვისებები

თვისება	ტიპი	აღწერა
Count	int	შეიცავს სტეკში რეალურად არსებული ელემენტების რაოდენობას
IsReadOnly	bool	ამოწმებს, არის თუ არა სტეკი მხოლოდ წაკითხვადი

ცხრილი 14.10. Stack კლასის მეთოდები

თვისება	დაბრუნებული ტიპი	აღწერა
Clear()	void	სტეკიდან ყველა ელემენტს შლის
Clone()	object	ქმნის სტეკის ასლს
Contains()	bool	ამოწმებს, არის თუ არა სტეკში მითითებული ელემენტი
CopyTo	void	სტეკის ელემენტებს გადაწერს ერთგანზომილებიან მასივში
Equals()	bool	ამოწმებს, ტოლია თუ არა ორი სტეკი
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს სტეკის ელემენტების გასწვრივ იტერაციისათვის
GetType()	Type	გასცემს სტეკის მიმდინარე ელემენტის ტიპს
Peek()	object	გასცემს სტეკის უკანასკნელ (მწვერვალზე მყოფ) ობიექტს, მაგრამ არ შლის მას სტეკიდან
Pop()	object	გასცემს სტეკის უკანასკნელ (მწვერვალზე მყოფ) ობიექტს და შლის მას სტეკიდან
Push()	void	ამატებს ელემენტს სტეკის მწვერვალზე
ToArray()	object[]	სტეკის ელემენტებს გადაწერს მასივში
ToString()	string	გასცემს სტეკის მიმდინარე ელემენტის შესაბამის სტრიქონს

მოყვანილ პროგრამაში ხდება **Push(), Pop()** და **Peek()** მეთოდებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 14.11
//      პროგრამაში ხდება Push(), Pop() და Peek() მეთოდებთან მუშაობის დემონსტრირება
//      სტეკის შექმნა
Stack Steki = new Stack();
//      სტეკში ელემენტების დამატება
Steki.Push("ანა");
Steki.Push("საბა");
Steki.Push("ლიკა");
Steki.Push("რომანი");
//      სტეკის ელემენტების გამოტანა
foreach ( string striqoni in Steki )
      label1.Text += striqoni + '\n';
//      სტეკის ელემენტების რაოდენობის მიღება
int ElementebisRaodenoba = Steki.Count;
label1.Text += "ელემენტების რაოდენობა - " +
            ElementebisRaodenoba.ToString() + '\n';
for ( int count = 0; count < ElementebisRaodenoba; count++ )
{
   label2.Text += "Peek() = " + Steki.Peek() + '\n';
```

```
   label2.Text += "Pop()  = " + Steki.Pop()  + '\n';
}
}
```

ბიტების მასივები

ბიტების მასივი არის ბულის ცვლადების (true და false) მასივი. მასში true მნიშვნელობას შეესაბამება 1, false მნიშვნელობას კი - 0. ბიტების მასივი განსაზღვრულია BitArray კლასში. ამ კლასში წარმოდგენილ მეთოდებს შორის აღსანიშნავია და, ან, არა და გამომრიცხავი ან მეთოდები. BitArray კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.11 და.14.12.

განვიხილოთ ზოგიერთი მეთოდი.

Not() მეთოდი გამოიყენება ბიტების მასივის ელემენტების ინვერტირებისათვის (იხ. ბიტობრივი ოპერატორები, გვ. 62). მისი სინტაქსია:

bitArray1.Not()

bitArray1 არის BitArray ტიპის ობიექტი.

Or() მეთოდი გამოიყენება ან ოპერაციის შესასრულებლად მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის. მისი სინტაქსია:

bitArray1.Or(bitArray2)

And() მეთოდი გამოიყენება და ოპერაციის შესასრულებლად მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის. მისი სინტაქსია:

bitArray1.And(bitArray2)

Xor() მეთოდი გამოიყენება გამომრიცხავი ან ოპერაციის შესასრულებლად მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის. მისი სინტაქსია:

bitArray1.Xor(bitArray2)

ცხრილი 14.11. BitArray კლასის თვისებები

თვისება	ტიპი	აღწერა
Count	int	შეიცავს ბიტების მასივში რეალურად არსებული ელემენ-ტების რაოდენობას
IsReadOnly	bool	ამოწმებს, არის თუ არა ბიტების მასივი მხოლოდ წაკით-ხვადი
Item	object	განსაზღვრავს ელემენტის მნიშვნელობას მითითებული ინდექსის მიხედვით. ეს თვისება არის BiTArray კლასის ინდექსატორი
Length	object	განსაზღვრავს ელემენტების რაოდენობას, რომლებიც შეიძლება შევინახოთ ბიტების მასივში

ცხრილი 14.12. BitArray კლასის მეთოდები

მეთოდი	დაბრუნებული ტიპი	აღწერა
And()	BitArray	ასრულებს და ოპერაციას მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის
Clone()	object	ქმნის ბიტების მასივის ასლს
CopyTo()	void	ბიტების მასივის ელემენტებს ერთგანზომილებიან მასივში გადაწერს
Equals()	bool	ამოწმებს ტოლია თუ არა ბიტების ორი მასივი
Get()	bool	გასცემს ბიტების მასივის მითითებული პოზიციის ბიტის მნიშვნელობას
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს ბიტების მასივისათვის
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს
Not()	BitArray	ახდენს ბიტების მასივის ელემენტების ინვერსიას
Or()	BitArray	ასრულებს ან ოპერაციას მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის
Set()	void	ბიტების მასივის მითითებული პოზიციის ბიტს ანიჭებს მითითებულ მნიშვნელობას
SetAll()	void	მითითებულ მნიშვნელობას ანიჭებს ბიტების მასივის ყველა ელემენტს
ToString()	string	გასცემს მიმდინარე ელემენტის შესაბამის სტრიქონს
Xor()	BitArray	ასრულებს გამომრიცხავ ან ოპერაციას მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის

მოყვანილ პროგრამაში ხდება ამ ფუნქციების მუშაობის დემონსტრირება.

```
{
//      პროგრამა 14.12
//      პროგრამაში ხდება Not(),Or(), And() და Xor() მეთოდებთან მუშაობის დემონსტრირება
label1.Text = "BitebisMasivi";
label2.Text = "BitebisMasivi1";
label3.Text = "NOT";
label4.Text = "OR";
label5.Text = "AND";
label6.Text = "XOR";
BitArray BitebisMasivi = new BitArray(4);
BitebisMasivi[0] = true;
BitebisMasivi[1] = false;
BitebisMasivi[2] = true;
BitebisMasivi[3] = false;
//      ელემენტების გამოტანა
```

```
label1.Text += '\n';
label2.Text += '\n';
label3.Text += '\n';
label4.Text += '\n';
label5.Text += '\n';
label6.Text += '\n';
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
   label1.Text += "BitebisMasivi[" + indexi.ToString() + "] = " +
                  BitebisMasivi[indexi].ToString() + '\n';
//
BitArray BitebisMasivi1 = new BitArray(BitebisMasivi);
BitebisMasivi1[0] = true;
BitebisMasivi1[1] = true;
BitebisMasivi1[2] = false;
BitebisMasivi1[3] = false;
//       ელემენტების გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi1.Count; indexi++ )
   label2.Text += "BitebisMasivi1[" + indexi.ToString() + "] = " +
          BitebisMasivi1[indexi].ToString() + '\n';
//       Not() მეთოდი ახდენს BitebisMasivi მასივის ელემენტების ინვერსიას
BitebisMasivi.Not();          //       BitebisMasivi = { false, true, false, true } ელემენტების გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
   label3.Text += "BitebisMasivi[" + indexi.ToString() + "] = " +
          BitebisMasivi[indexi].ToString() + '\n';
//       Or() მეთოდი BitebisMasivi და BitebisMasivi1 მასივის ელემენტებს შორის ასრულებს ან
//       ოპერაციას
BitebisMasivi.Or(BitebisMasivi1);  // BitebisMasivi = { true,true,false,true } ელემენტების გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
   label4.Text += "BitebisMasivi[" + indexi.ToString() + "] = " +
          BitebisMasivi[indexi].ToString() + '\n';
//       And() მეთოდი BitebisMasivi და BitebisMasivi1 მასივის ელემენტებს შორის ასრულებს და
//       ოპერაციას
BitebisMasivi.And(BitebisMasivi1);       // BitebisMasivi = { true,true,false,false } ელემენტების
                                         // გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
   label5.Text += "BitebisMasivi[" + indexi.ToString() + "] = " +
          BitebisMasivi[indexi].ToString() + '\n';
//       Xor() მეთოდი BitebisMasivi და BitebisMasivi1 მასივის ელემენტებს შორის ასრულებს
//       გამომრიცხავ ან ოპერაციას
BitebisMasivi.Xor(BitebisMasivi1);       // BitebisMasivi = {false,false,false,false} ელემენტების
                                         //  გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
  label6.Text += "BitebisMasivi[" + indexi.ToString() + "] = " +
          BitebisMasivi[indexi].ToString() + '\n';
}
```

თავი 15. შესრულების ნაკადები

შესრულების ნაკადის განსაზღვრა

პროცესი არის შესრულების სტადიაში მყოფი პროგრამა (პროგრამა-დანართი). მაგალითად, თუ ჩვენ გავუშვებთ Word ტექსტურ რედაქტორს, მაშინ შეიქმნება მისი შესაბამისი პროცესი. ოპერაციული სისტემები პროცესებს იყენებენ სხვადასხვა პროგრამა-დანართების, როგორიცაა Word, Excel და ა.შ., განცალკევებისათვის. თითოეული პროცესი ქმნის ერთ ან მეტ ნაკადს პროგრამის კოდის ნაწილების შესრულების მართვის მიზნით და ა.შ. ნაკადი არის ძირითადი ერთეული, რომლისთვისაც ოპერაციული სისტემა პროცესორის დროს გამოყოფს. პროცესები და ნაკადები დაწვრილებითაა განხილული ლიტერატურაში [8] და [9].

.NET Framework სისტემა პროცესს ყოფს ადვილად სამართავ ქვეპროცესებად, რომლებსაც პროგრამა-დანართების დომენები ეწოდებათ. ეს არის იზოლირებული გარემო, რომელშიც პროგრამა-დანართი სრულდება. პროგრამა-დანართების დომენები, რომლებიც წარმოდგენილი არიან AppDomain ობიექტებით, უზრუნველყოფენ იზოლირებისა და უსაფრთხოების საზღვრებს შესრულების პროცესში მყოფი კოდისთვის. ერთი პროცესის შიგნით ერთი ან მეტი ნაკადი შეიძლება შესრულდეს ერთ ან მეტ პროგრამა-დანართის დომენში.

ნაკადებთან სამუშაო კლასები განსაზღვრულია System.Threading სახელების სივრცეში. ნაკადები ორგვარია: შესრულების ნაკადები და მონაცემების ნაკადები. მონაცემების ნაკადები ესაა ნაკადები, რომელთა გადაადგილება ხდება ერთი მოწყობილობიდან მეორეზე. ამ თავში დაწვრილებით განვიხილავთ შესრულების ნაკადებს.

ხშირ შემთხვევაში კომპიუტერში რამოდენიმე პროგრამა ერთდროულად სრულდება. მაგალითად, ჩვენ შეგვიძლია ერთმანეთის მიყოლებით შესასრულებლად გავუშვათ Excel, Paint და Internet Explorer პროგრამები. თითოეული მათგანი წარმოდგენილი იქნება შესაბამისი პროცესით. პროცესორი ამ პროცესებს შორის ისე ახდენს სწრაფად გადართვას, რომ ჩვენ გვეჩვენება თითქოს სამივე პროგრამა ერთდროულად სრულდება. სინამდვილეში, პროცესორი ასრულებს პირველი პროცესის კოდის მცირე ნაწილს, შემდეგ მეორე პროცესის კოდის მცირე ნაწილს და ა.შ. თითოეული პროცესი წარმოქმნის ძირითად ნაკადს, რომელშიც ხდება შესაბამისი პროგრამის შესრულება.

ახლა დავუშვათ, რომ Excel პროგრამასთან მუშაობის დროს ჩვენ შევასრულეთ პრინტერზე ბეჭდვის ბრძანება. ამ შემთხვევაში პროცესი წარმოქმნის მეორე ნაკადს, რომელშიც შესრულდება პრინტერზე მონაცემების ბეჭდვა. ამავე დროს ძირითად ნაკადში ჩვენ შეგვიძლია გავაგრძელოთ ცხრილის რედაქტირება. პროცესორი მოახდენს ამ ორ ნაკადს შორის გადართვას.

Thread კლასში განსაზღვრული თვისებები და მეთოდები მოყვანილია ცხრილებში 15.1 და 15.2.

ცხრილი 15.1. Thread კლასის თვისებები

თვისება	ტიპი	აღწერა
ApartmentState	ApartmentState	განსაზღვრავს ნაკადი ერთნაკადურ თუ მრავალნაკადურ გარემოში სრულდება
CurrentCulture	CultureInfo	ნაკადისათვის აყენებს ან გასცემს რეგიონალურ პარამეტრებს
CurrentThread	Thread	გასცემს ნაკადს, რომელიც მოცემულ მომენტში სრულდება
IsAlive	bool	გასცემს true მნიშვნელობას თუ ნაკადი იქნა გაშვებული და არ იქნა დამთავრებული
IsBackground	bool	ნაკადისთვის აყენებს ან გასცემს ფონური შესრულების ატრიბუტს
IsThreadPoolThread	bool	გასცემს true მნიშვნელობას თუ ნაკადი იმყოფება ნაკადების სისტემურ პულში
Name	string	აყენებს ან გასცემს ნაკადის სახელს. თუ სახელი მითითებული არ არის, მაშინ ის არის null-ის ტოლი
Priority	ThreadPriority	გასცემს ან აყენებს ნაკადის პრიორიტეტს
ThreadState	ThreadState	გასცემს ნაკადის მდგომარეობას

შესრულების ნაკადის შექმნა

ნაკადის შესაქმნელად უნდა შევქმნათ Thread კლასის ობიექტი. ამ კლასის კონსტრუქტორს უნდა გადავცეთ დელეგატი. ეს დელეგატი მიმართავს იმ მეთოდს, რომელიც უნდა შესრულდეს ნაკადის ქვეშ. პროგრამის საწყისი კოდი, რომელშიც სრულდება ნაკადებთან მუშაობა, უნდა იწყებოდეს დირექტივით:

using System.Threading;

ცხრილი 15.2. Thread კლასის მეთოდები

მეთოდი	აღწერა
Abort()	იწვევს ნაკადის შესრულების შეწყვეტას
AllocateDataSlot() (სტატიკურია)	ყველა ნაკადისთვის გამოყოფს მონაცემების არასახელდებულ უბანს
AllocateNamedDataSlot() (სტატიკურია)	ყველა ნაკადისთვის გამოყოფს მონაცემების სახელდებულ უბანს
Equal()	ამოწმებს, ტოლია თუ არა ორი ობიექტი
FreeNamedDataSlot() (სტატიკურია)	ათავისუფლებს ადრე გამოყოფილ მონაცემების სახელდებულ უბანს
GetData() (სტატიკურია)	გასცემს მონაცემების უბნის მნიშვნელობას
GetDomain() (სტატიკურია)	გასცემს დომენს, რომელშიც ნაკადი სრულდება
GetDomainID() (სტატიკურია)	გასცემს იმ დომენის იდენტიფიკატორს, რომელშიც ნაკადი სრულდება

ცხრილი 15.2 (გაგრძელება)

GetNamedDataSlot() (სტატიკურია)	გასცემს მონაცემების სახელდებული უბნის მნიშვნელობას
GetType()	გასცემს მიმდინარე ობიექტის ტიპს
Interrupt()	წყვეტს ნაკადის შესრულებას
Join()	ბლოკავს ნაკადის შესრულებას სხვა ნაკადის დამთავრებამდე
ResetAbort() (სტატიკურია)	აუქმებს მიმდინარე ნაკადის შეწყვეტის მოთხოვნას
Resume()	აგრძელებს შეჩერებული ნაკადის შესრულებას
SetData() (სტატიკურია)	მონაცემების უბანს მნიშვნელობას ანიჭებს
Sleep()	აჩერებს ნაკადის შესრულებას მითითებული დროის განმავლობაში
SpinWait() (სტატიკურია)	ნაკადს აიძულებს გამოტოვოს ციკლების მითითებული რაოდენობა
Start()	იწყებს ნაკადის შესრულებას
Suspend()	დროებით აჩერებს ნაკადის შესრულებას
ToString()	გასცემს მიმდინარე ნაკადის შესაბამის სტრიქონს

ამ განყოფილებაში შევადგენთ კონსოლურ პროგრამა-დანართებს, რადგან ასეთ პროგრამებთან მუშაობის დროს ნაკადების შესრულების შედეგები უფრო მკაფიოდ ჩანს. კონსოლური პროგრამის შესაქმნელად ნახ. 2.2-ზე ნაჩვენები ფანჯრის Templates ზონაში უნდა მოვნიშნოთ Console Application ელემენტი, Name: ველში შევიტანოთ პროგრამის სახელი, კატალოგი ავირჩიოთ Browse კლავიშის საშუალებით და დავაჭიროთ OK კლავიშს. გახსნილ ფანჯარაში შეგვაქვს ჩვენი პროგრამის კოდი ისე, როგორც ეს ქვემოთაა ნაჩვენები.

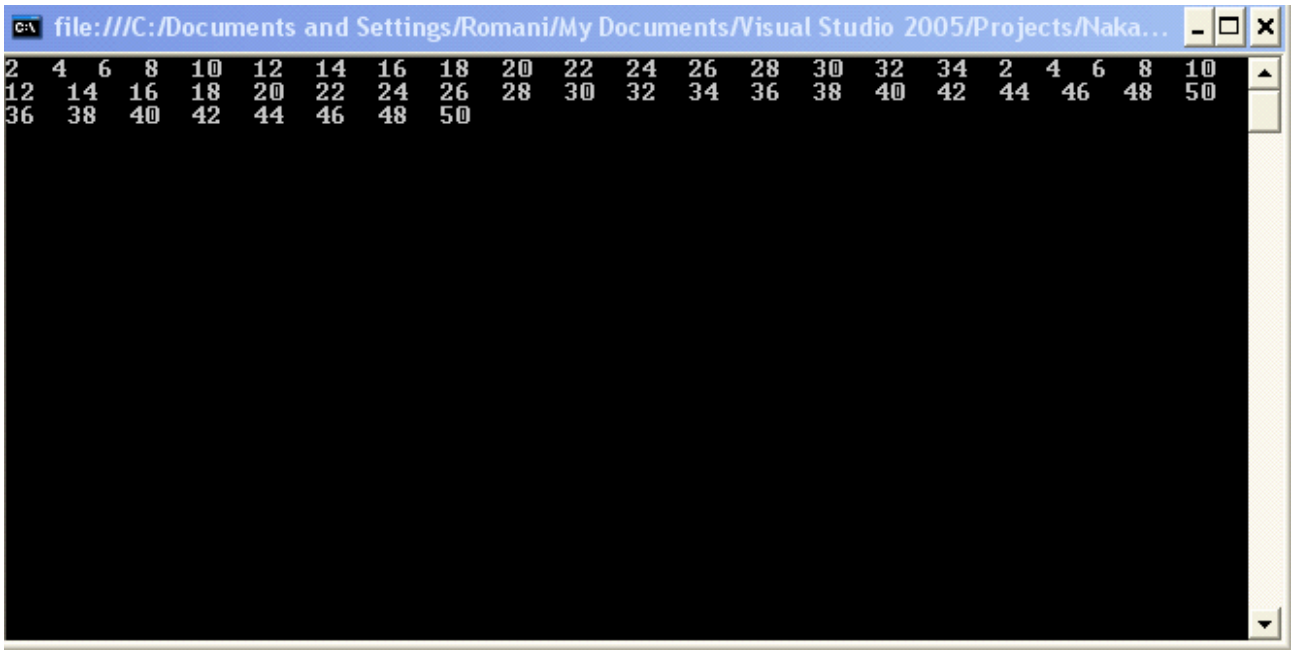
```
//      პროგრამა 15.1
//      პროგრამაში ხდება ორ ნაკადთან მუშაობის დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Ori_Nakadi
{
   class ChemiKlasi
   {
      //  ნაკადში შესასრულებელი მეთოდის გამოძახება
      public static void Datvla_Luwi()
      {
         for (int index2 = 2; index2 <= 50; index2 += 2)
            Console.Write(index2.ToString() + "  ");
         Thread.Sleep(5000);
      }
```

```
        static void Main(string[] args)
        {
            //  Nakadi1 ნაკადის შექმნა, რომელშიც შესრულდება Datvla_Luwi() მეთოდი
            Thread Nakadi1 = new Thread(new ThreadStart(Datvla_Luwi));
            //  Nakadi1 ნაკადში შესრულებას იწყებს Datvla_Luwi() მეთოდი
            Nakadi1.Start();
            //  პირველ, ძირითად ნაკადში შესრულებას იწყებს Datvla_Luwi() მეთოდი
            Datvla_Luwi();
        }
    }
}
```

პროგრამაში იქმნება ობიექტი-ნაკადი Nakadi1. კონსტრუქტორს გადაეცემა Datvla_Luwi დელეგატი, რომელიც წარმოადგენს Datvla_Luwi() მეთოდზე მიმთითებელს. ამ მეთოდის შესრულების დასაწყებად გამოიძახება Nakadi1 ობიექტის Start() მეთოდი. ამის შემდეგ, მომდევნო სტრიქონში ვიძახებთ Datvla_Luwi() მეთოდს, რომელიც პირველ ძირითად ნაკადში შესრულდება. შედეგად, ორ სხვადასხვა ნაკადში ერთდროულად სრულდება Datvla_Luwi() მეთოდი. პროგრამის მუშაობის ერთ-ერთი შესაძლო შედეგი ნაჩვენებია ნახ. 15.1-ზე.

აქ ორი ნაკადის მუშაობის შედეგები ერთმანეთშია არეული. ეს იმიტომ მოხდა, რომ ორივე ნაკადს შედეგები ერთ მოწყობილობაზე გამოაქვს. ჩვენს შემთხვევაში ეს მოწყობილობაა - მონიტორი. როგორც ვხედავთ, პირველმა ნაკადმა, რომელშიც Datvla_Luwi() მეთოდი სრულდება, მონაცემები გამოიტანა 2-დან 34-ის ჩათვლით. შემდეგ პროცესორი გადაერთო Nakadi1-ის შესრულებაზე. ამ ნაკადში მუშაობს Datvla_Luwi() მეთოდი. ამ ნაკადმა მონაცემები თავიდან ბოლომდე გამოიტანა ეკრანზე. შემდეგ პროცესორმა ისევ გააგრძელა პირველი ნაკადის შესრულება, რომელმაც დაამთავრა ეკრანზე მონაცემების გამოტანა.



ნახ. 15.1.

აქვე შევნიშნოთ, რომ ამ პროგრამის ყოველი გაშვებისას ეკრანზე შეიძლება

სხვადასხვა შედეგები მივიღოთ. საქმე ისაა, რომ წინასწარ უცნობია თუ როდის მოახდენს პროცესორი ერთი ნაკადიდან მეორეზე გადართვას.

რაც შეეხება Thread.Sleep(5000) მეთოდს, იგი საშუალებას გვაძლევს შედეგები ეკრანზე დავაყოვნოთ 5 წამის განმავლობაში (1000 შეესაბამება 1 წამს).

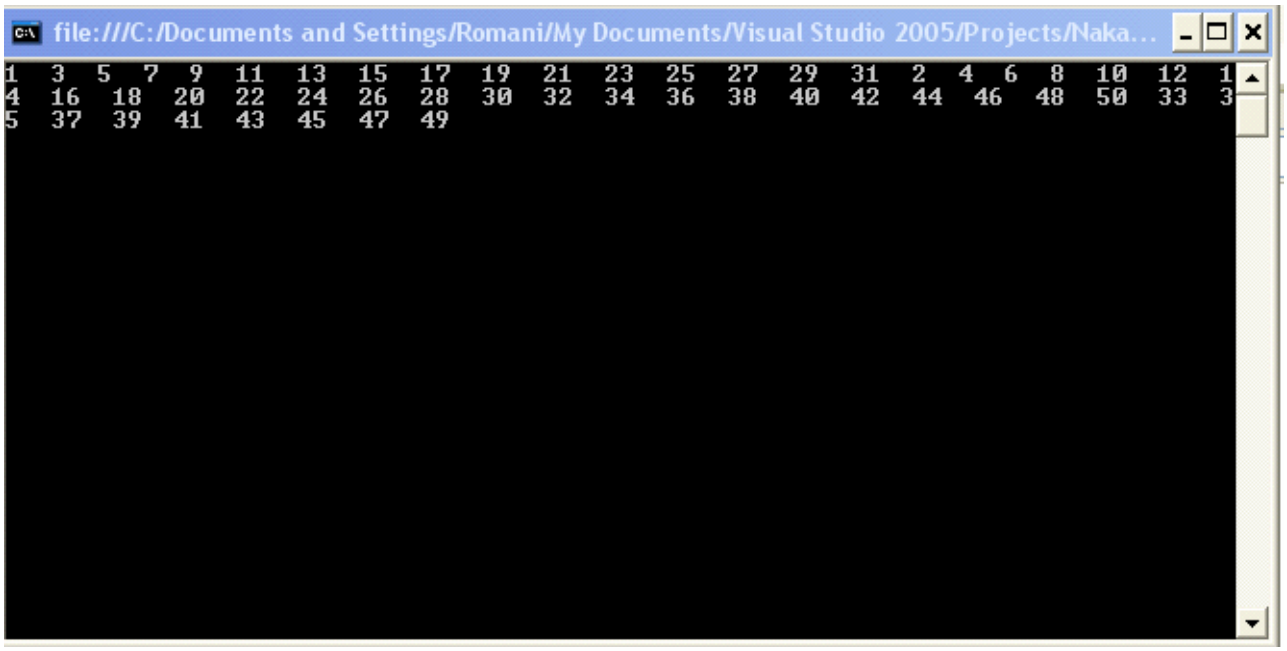
იმისათვის, რომ პროგრამის შესრულების შედეგები უფრო თვალსაჩინო იყოს, ქვემოთ მოყვანილი პროგრამის ChemiKlasi კლასში გამოვაცხადოთ ორი მეთოდი: Datvla_Kenti() და Datvla_Luwi(), რომლებიც შესაბამისად, კენტ და ლუწ რიცხვებს გასცემენ.

```
//      პროგრამა 15.2
//      პროგრამაში ხდება ორი სხვადასხვა მეთოდის ორ სხვადასხვა ნაკადში შესრულების დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Ori_Sxvadasxva_Metodi
{
   class ChemiKlasi
   {
      public static void Datvla_Kenti()
      {
         for (int index1 = 1; index1 <= 50; index1 += 2)
            Console.Write(index1.ToString() + "  ");
         Thread.Sleep(5000);
      }
      public static void Datvla_Luwi()
      {
         for (int index2 = 2; index2 <= 50; index2 += 2)
            Console.Write(index2.ToString() + "  ");
         Thread.Sleep(5000);
      }
      static void Main()
      {
         //
         Thread Nakadi1 = new Thread(new ThreadStart(Datvla_Luwi));
         //  Datvla_Luwi() მეთოდი შესრულებას იწყებს Nakadi1 ნაკადში
         Nakadi1.Start();
         //  Datvla_Kenti() მეთოდი შესრულებას იწყებს ძირითად ნაკადში
         Datvla_Kenti();
      }
   }
}
```

ძირითად პროგრამაში იქმნება ობიექტი-ნაკადი Nakadi1. კონსტრუქტორს გადაეცემა დელეგატი - Datvla_Luwi, რომელიც წარმოადგენს Datvla_Luwi() მეთოდზე

მიმთითებელს. შემდეგ, Datvla_Luwi მეთოდის შესრულების დასაწყებად გამოიძახება Nakadi1 ობიექტის Start() მეთოდი. ამის შემდეგ, მომდევნო სტრიქონში ვიძახებთ Datvla_Kenti() მეთოდს, რომელიც პირველ, ძირითად ნაკადში შესრულდება. შედეგად, ორ სხვადასხვა ნაკადში ერთდროულად სრულდება Datvla_Kenti() და Datvla_Luwi() მეთოდები. პროგრამის მუშაობის ერთ-ერთი შესაძლო შედეგი ნაჩვენებია ნახ. 15.2.-ზე.



ნახ. 15.2.

შესრულების ნაკადის პრიორიტეტები

შექმნისას თითოეულ ნაკადს ენიჭება Normal პრიორიტეტი. სულ არსებობს პრიორიტეტების 5 დონე:

Lowest - უდაბლესი
BelowNormal - ნორმალურზე დაბალი
Normal - ნორმალური
AboveNormal - ნორმალურზე მაღალი
Highest - უმაღლესი

მოყვანილ პროგრამაში ხდება ნაკადებისათვის პრიორიტეტების მინიჭების დემონსტრირება.

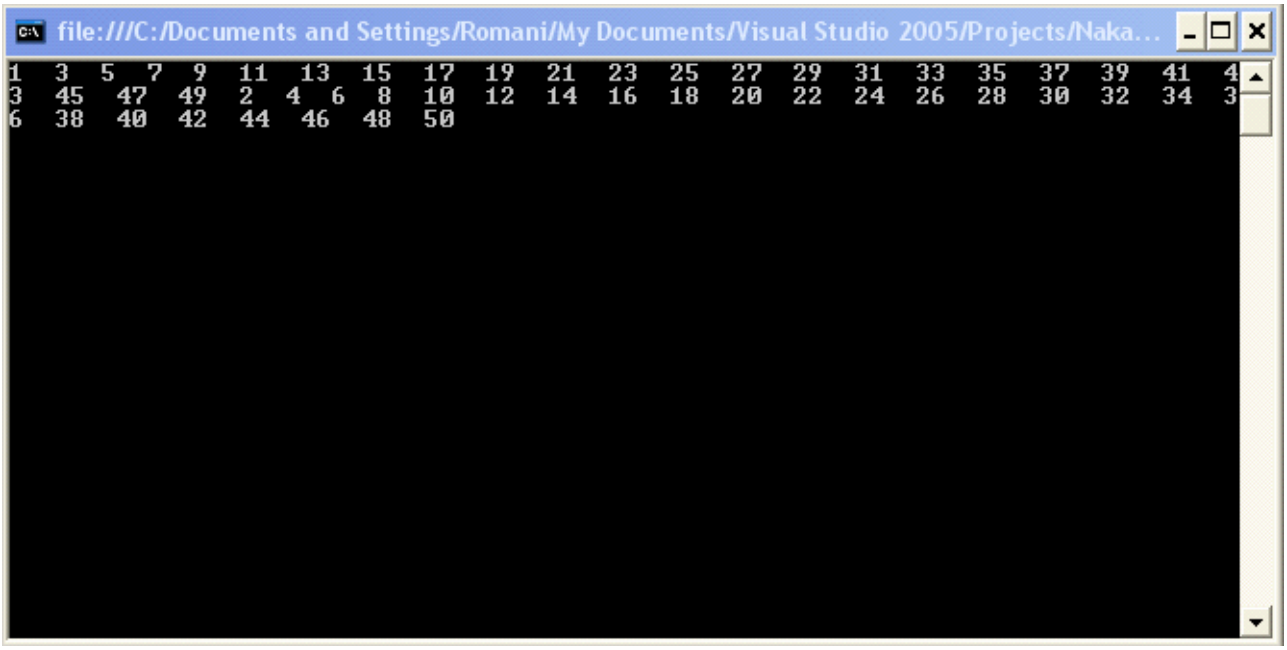
```
//      პროგრამა 15.3
//      პროგრამაში ხდება ნაკადებისათვის პრიორიტეტების მინიჭების დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Ori_Metodi_Prioriteti
{
```

```
class ChemiKlasi
{
   public static void Datvla_Kenti()
   {
      for (int index1 = 1; index1 <= 50; index1 += 2)
         Console.Write(index1.ToString() + "  ");
      Thread.Sleep(5000);
   }
   public static void Datvla_Luwi()
   {
      for (int index2 = 2; index2 <= 50; index2 += 2)
         Console.Write(index2.ToString() + "  ");
      Thread.Sleep(5000);
   }
   static void Main(string[] args)
   {
      //  Nakadi1 ნაკადს შექმნის დროს ენიჭება Normal პრიორიტეტი
      Thread Nakadi1 = new Thread(new ThreadStart(Datvla_Kenti));
      //  მიმდინარე ნაკადს ენიჭება Lowest პრიორიტეტი
      Thread.CurrentThread.Priority = ThreadPriority.Lowest;
      //  Nakadi1 ნაკადს შექმნის დროს ენიჭება Highest პრიორიტეტი
      Nakadi1.Priority = ThreadPriority.Highest;
      Nakadi1.Start();
      Datvla_Luwi();
   }
}
}
```

პროგრამაში Nakadi1 ნაკადს უმაღლესი პრიორიტეტი ენიჭება, ხოლო ძირითად ნაკადს, რომელშიც Datvla_Luwi() მეთოდი სრულდება - უდაბლესი. აქ Thread კლასის CurrentThread სტატიკური თვისება გასცემს მიმდინარე ნაკადს, ანუ იმ ნაკადს, რომელიც მოცემულ მომენტში სრულდება, ხოლო Thread ობიექტის CurrentThread თვისება კი - მიმდინარე ნაკადზე მიმთითებელს.

როგორც პროგრამის შესრულების შედეგებიდან ჩანს (ნახ. 15.3), ჯერ მუშაობას ამთავრებს პრიორიტეტული ნაკადი, შემდეგ კი - დაბალი პრიორიტეტის მქონე ნაკადი. თუმცა არ არსებობს იმის გარანტია, რომ მაღალპრიორიტეტული ნაკადი უფრო ადრე დაამთავრებს შესრულებას, ვიდრე დაბალპრიორიტეტული ნაკადი. საქმე ის არის, რომ, ჯერ ერთი, ყველა ოპერაციული სისტემა რესურსებს ერთნაირად არ ანაწილებს. მეორეც, პრიორიტეტები ხშირად განსაზღვრავენ პროცესორული დროის ხანგრძლივობას და არა პროცესების შესრულების რიგითობას. შედეგად, მაღალპრიორიტეტულმა პროცესმა, რომელიც დიდი მოცულობის გამოთვლებს ასრულებს, მუშაობა შეიძლება უფრო გვიან დაამთავროს, ვიდრე დაბალპრიორიტეტულმა პროცესმა, რომელიც მცირე მოცულობის გამოთვლებს ასრულებს.



ნახ. 15.3.

შესრულების ნაკადის მდგომარეობები

ნაკადის მიმდინარე მდგომარეობის მისაღებად გამოიყენება ThreadState თვისება. ნაკადი არსებობის მანძილზე შეიძლება რამდენიმე მდგომარეობაში იმყოფებოდეს. ეს მდგომარეობები მოყვანილია ცხრილში 15.3.

ცხრილი 15.3. ნაკადის მდგომარეობები.

მდგომარეობა	მდგომარეობის აღწერა
Unstarted	ნაკადს შესრულება არ დაუწყია
Running	ნაკადი სრულდება
Background	ნაკადი ფონურ რეჟიმში სრულდება
WaitSleepJoin	ნაკადი ბლოკირებულია Wait, Sleep ან Join მეთოდის მიერ
SuspendRequested	მოთხოვნილია ნაკადის დროებით შეჩერება
Suspended	ნაკადი დროებით შეჩერებულია
StopRequested	მოთხოვნილია ნაკადის გაჩერება
Stopped	ნაკადი გაჩერებულია
AbortRequested	მოთხოვნილია ნაკადის გაუქმება
Aborted	ნაკადი გაუქმებულია

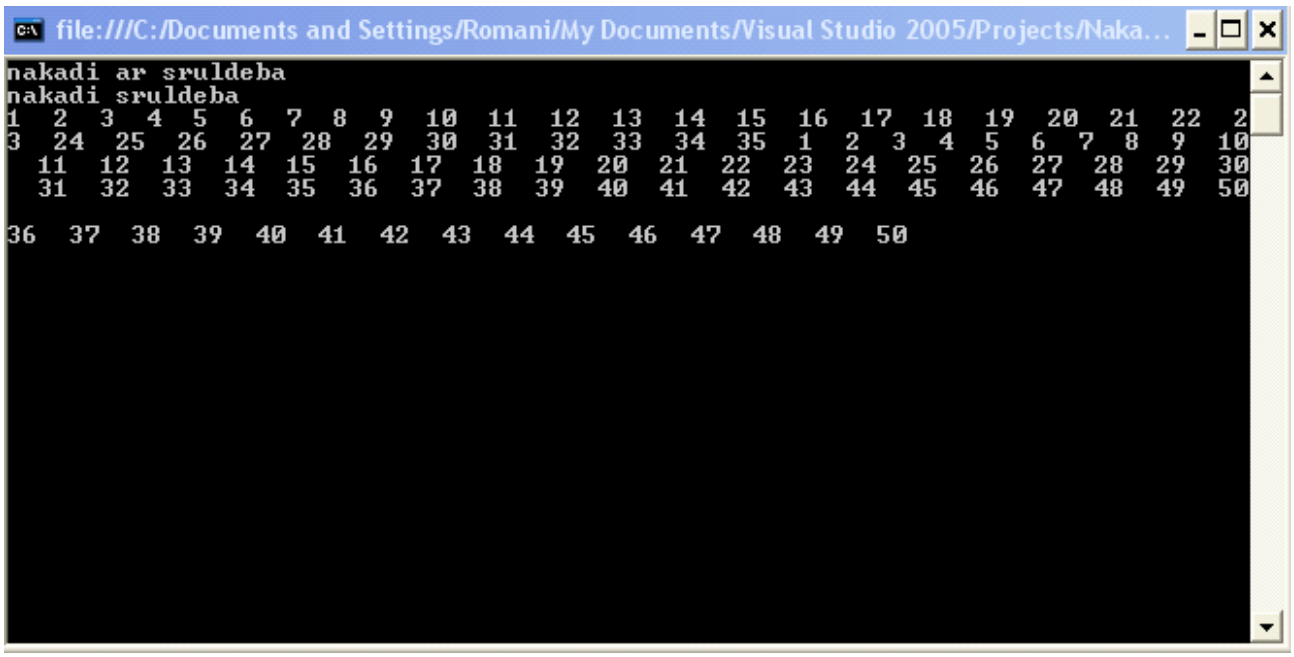
შექმნის შემდეგ ნაკადს აქვს Unstarted მდგომარეობა. Start() მეთოდის გამოძახების შემდეგ ნაკადი Running მდგომარებაში გადადის. თუ ნაკადის IsBackground თვისებას true მნიშვნელობას მივანიჭებთ, მაშინ ის ფონურ რეჟიმში იმუშავებს. ნაკადი დროის ერთსა და იმავე მომენტში შეიძლება რამდენიმე მდგომარეობაში იმყოფებოდეს, მაგალითად, WaitSleepJoin და AbortRequested მდგომარეობებში.

მოყვანილ პროგრამაში ხდება ThreadState თვისებასთან მუშაობის დემონსტრირება.

```
//      პროგრამა 15.4
//      პროგრამაში ხდება ThreadState თვისებასთან მუშაობის დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Nakadis_Mdgomareobis_Analizi
{
class ChemiKlasi
{
public static void Datvla()
{
for (int index = 1; index <= 50; index++)
    Console.Write(index.ToString() + "  ");
Console.WriteLine();
Thread.Sleep(5000);
}
public static void MdgomareobisNaxva(Thread Nakadi)
{
if ( ( Nakadi.ThreadState & ThreadState.Aborted ) == ThreadState.Aborted )
     Console.WriteLine("ნაკადი შეწყვეტილია");
if ( ( Nakadi.ThreadState &
    (ThreadState.Stopped | ThreadState.Unstarted | ThreadState.Aborted)) == 0 )
    Console.WriteLine("ნაკადი სრულდება");
       else Console.WriteLine("ნაკადი არ სრულდება");
}
static void Main(string[] args)
{
//
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
MdgomareobisNaxva(Nakadi1);
// ნაკადის გაშვება
Nakadi1.Start();
MdgomareobisNaxva(Nakadi1);
// ამავე დროს სრულდება მეორე ნაკადი
Datvla();
// nakadi1 ნაკადის შეწყვეტა
Nakadi1.Abort();
MdgomareobisNaxva(Nakadi1);
}
}
}
```

ნახ. 15.4.-ზე ნაჩვენებია შედეგის ერთ-ერთი შესაძლო ვარიანტი.



ნახ. 15.4.

შესრულების ნაკადის ლოკალური მონაცემები

გადასაწყვეტი ამოცანიდან გამომდინარე ნაკადებმა შეიძლება გამოიყენონ ოპერატიული მეხსიერების ერთი და იგივე ან სხვადასხვა უბნები. იმისათვის, რომ რამოდენიმე ნაკადმა ერთობლივად გამოიყენოს მონაცემები, საჭიროა ამ მონაცემების გამოცხადება ნაკადებისათვის საერთო ხილვადობის უბანში. იმისათვის, რომ პროცესებმა თავიანთ ლოკალურ მონაცემებთან იმუშაონ, ანუ გამოიყენონ ოპერატიული მეხსიერების სხვადასხვა უბნები, უნდა გამოვიყენოთ LocalDataStoreSlot ობიექტები.

მოყვანილ პროგრამაში ხდება ორი პროცესის მიერ საკუთარ ლოკალურ უბნებთან მუშაობის დემონსტრირება.

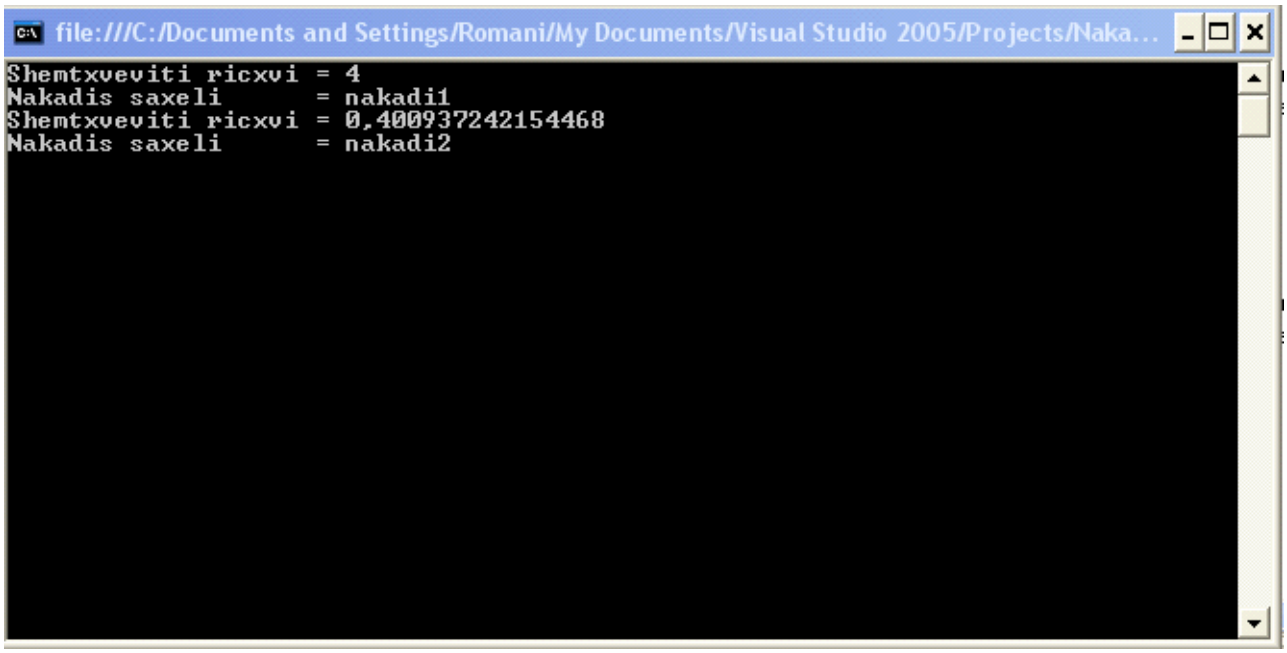
```
//      პროგრამა 15.5
//      პროგრამაში ხდება ორი პროცესის მიერ საკუთარ ლოკალურ უბნებთან მუშაობის დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Lokaluri_Monacemebi
{
class ChemiKlasi
{
public static void MonacemebisGamotana()
{
Console.WriteLine("შემთხვევითი რიცხვი = " +
        Thread.GetData(Thread.GetNamedDataSlot("შემთხვევითი რიცხვი - ")));
```

```
Console.WriteLine("ნაკადის სახელი     = " +
        Thread.GetData(Thread.GetNamedDataSlot("ნაკადის სახელი     - ")));
Thread.Sleep(5000);
}
public static void MonacemebisChatsera1()
{
Random Shemtxveviti_Ricxvi = new Random();
Thread.SetData(Thread.GetNamedDataSlot("შემთხვევითი რიცხვი - "),
        Shemtxveviti_Ricxvi.Next(100));
Thread.SetData(Thread.GetNamedDataSlot("ნაკადის სახელი      - "),
        Thread.CurrentThread.Name);
MonacemebisGamotana();
}
public static void MonacemebisChatsera2()
{
Random Shemtxveviti_Ricxvi = new Random();
Thread.SetData(Thread.GetNamedDataSlot("შემთხვევითი რიცხვი - "),
        Shemtxveviti_Ricxvi.NextDouble());
Thread.SetData(Thread.GetNamedDataSlot("ნაკადის სახელი     - "),
        Thread.CurrentThread.Name);
MonacemebisGamotana();
}
static void Main(string[] args)
{
//      სახელდებული უბნების გამოყოფა
Thread.AllocateNamedDataSlot("შემთხვევითი რიცხვი - ");
Thread.AllocateNamedDataSlot("ნაკადის სახელი      - ");
//      მეორე ნაკადის შექმნა და გაშვება
Thread nakadi1 = new Thread(new ThreadStart(MonacemebisChatsera1));
nakadi1.Name = "nakadi1";
nakadi1.Start();
//      მესამე ნაკადის შექმნა და გაშვება
Thread nakadi2 = new Thread(new ThreadStart(MonacemebisChatsera2));
nakadi2.Name = "nakadi2";
nakadi2.Start();
// მონაცემების უბნის გასუფთავება
Thread.FreeNamedDataSlot("შემთხვევითი რიცხვი- ");
Thread.FreeNamedDataSlot("ნაკადის სახელი     - ");
}
}
}
```

ძირითად პროგრამაში ხდება მეხსიერების ორი უბნის გამოყოფა AllocateNamedDataSlot() მეთოდის საშუალებით. ამ უბნების სახელებია: "შემთხვევითი რიცხვი - " და "ნაკადის სახელი   - ". შემდეგ იქმნება ორი ნაკადი: nakadi1 და nakadi2. მათ ენიჭებათ შესაბამისი სახელები. nakadi1 ნაკადში ხდება MonacemebisChatsera1()

მეთოდის შესრულება, nakadi2 ნაკადში კი - MonacemebisChatsera2() მეთოდის შესრულება. თითოეული მეთოდი "შემთხვევითი რიცხვი - " უბანში წერს შემთხვევით რიცხვს, "ნაკადის სახელი - " უბანში კი - შესაბამისი ნაკადის სახელს. შემდეგ ხდება MonacemebisGamotana() მეთოდის გამოძახება, რომელსაც ეკრანზე გამოაქვს "შემთხვევითი რიცხვი - " და "ნაკადის სახელი - " უბნებში მოთავსებული მონაცემები. ძირითადი პროგრამის დასასრულს ხდება მეხსიერების გამოყოფილი უბნების გათავისუფლება და ოპერაციული სისტემისათვის გადაცემა. პროგრამის შესრულების შედეგები ნაჩვენებია ნახ. 15.5-ზე.



ნახ. 15.5.

მიუხედავად იმისა, რომ ორივე ნაკადი იყენებს ერთი და იგივე სახელის მქონე მეხსიერების უბნებს, მათ მაინც გამოეყოფათ მეხსიერების სხვადასხვა ლოკალური უბნები.

შესრულების ნაკადების მართვა

ჩვეულებრივ, ჩვენ არ ვიცით თუ რომელი ნაკადი როდის შესრულდება და ვერ განვსაზღვრავთ მათი შესრულების მიმდევრობას. ამ განყოფილებაში ჩვენ განვიხილავთ ნაკადების მართვის ხერხებს.

Timer კლასი

Timer კლასი საშუალებას გვაძლევს ნაკადი დროის გარკვეული გამეორებადი ინტერვალების შემდეგ, ანუ გარკვეული პერიოდულობით შევასრულოთ. Timer ტიპის ობიექტის შექმნისას კონსტრუქტორს ოთხი პარამეტრი უნდა გადავცეთ:

- callback. ესაა TimeCallback ტიპის დელეგატი, რომელიც მიმართავს იმ მეთოდს, რომელსაც ტაიმერი გამოიძახებს.
- state. ესაა ობიექტი, რომელიც TimeCallback მეთოდს გადაეცემა. ეს პარამეტრი იმიტომ ეთითება, რომ ტაიმერისათვის მისაწვდომი იყოს რაიმე მუდმივი

მდგომარეობა, წინააღმდეგ შემთხვევაში ეთითება null.

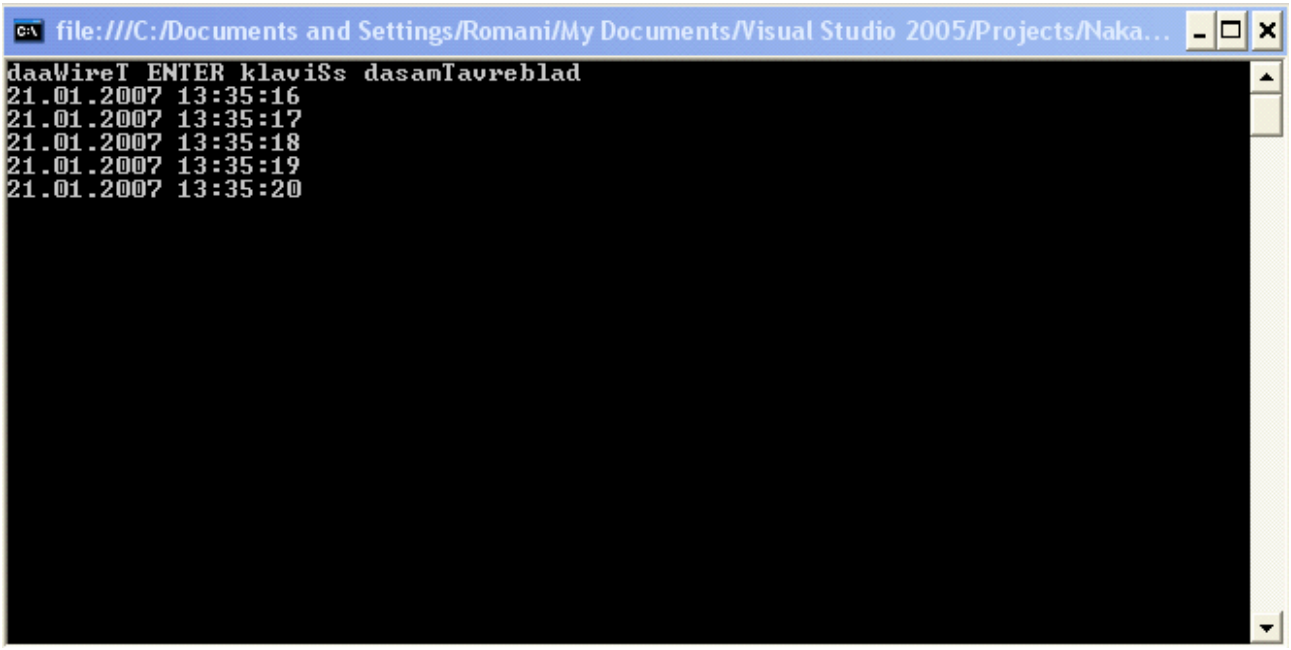
- dueTime. ესაა მილიწამების რაოდენობა ტაიმერის პირველ ამუშავებამდე.
- Period. ესაა მილიწამების რაოდენობა ტაიმერის ამუშავებებს შორის.

მოყვანილ პროგრამაში ხდება Timer კლასთან მუშაობის დემონსტრირება.

```
//      პროგრამა 15.6
//      პროგრამაში ხდება Timer კლასთან მუშაობის დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Timer
{
class ChemiKlasi
{
public static void Drois_Gacema(Object Mdgomareoba)
{
Console.WriteLine(DateTime.Now);
}
static void Main(string[] args)
{
//      დელეგატის შექმნა, რომელსაც Timer ობიექტი გამოიძახებს
TimerCallback delegati = new TimerCallback(Drois_Gacema);
//      ტაიმერის შექმნა, რომელიც წამში ორჯერ იმუშავებს
//      პირველი გაშვება ერთი წამის შემდეგ შესრულდება
System.Threading.Timer Taimeri = new
        System.Threading.Timer(delegati, null, 1000, 2000);
//      პროგრამის შესრულების დასამთავრებლად ვაჭერთ Enter კლავიშს
Console.WriteLine("დააჭირეთ ENTER კლავიშს დასამთავრებლად");
int ricxvi = Console.Read();
//       რესურსების გათავისუფლება
Taimeri.Dispose();
Taimeri = null;
}
}
}
```

პროგრამის შესრულების შედეგი ნაჩვენებია ნახ. 15.6-ზე.



ნახ. 15.6.

პროგრამაში იქმნება delegati დელეგატი, რომელიც გამოიყენება Drois_Gacema() მეთოდის გამოსაძახებლად. შემდეგ, იქმნება Taimeri ობიექტი-ტაიმერი, რომელიც პირველად ამუშავდება 1 წამის შემდეგ, შემდეგ კი იმუშავებს წამში ორჯერ, ანუ წამში ორჯერ მოხდება Drois_Gacema() მეთოდის გამოძახება. პროგრამის ბოლოს Dispose() მეთოდი ათავისუფლებს ოპერაციული სისტემის რესურსებს, რომელიც ნაკადს გამოეყო შექმნის დროს.

Join() მეთოდი

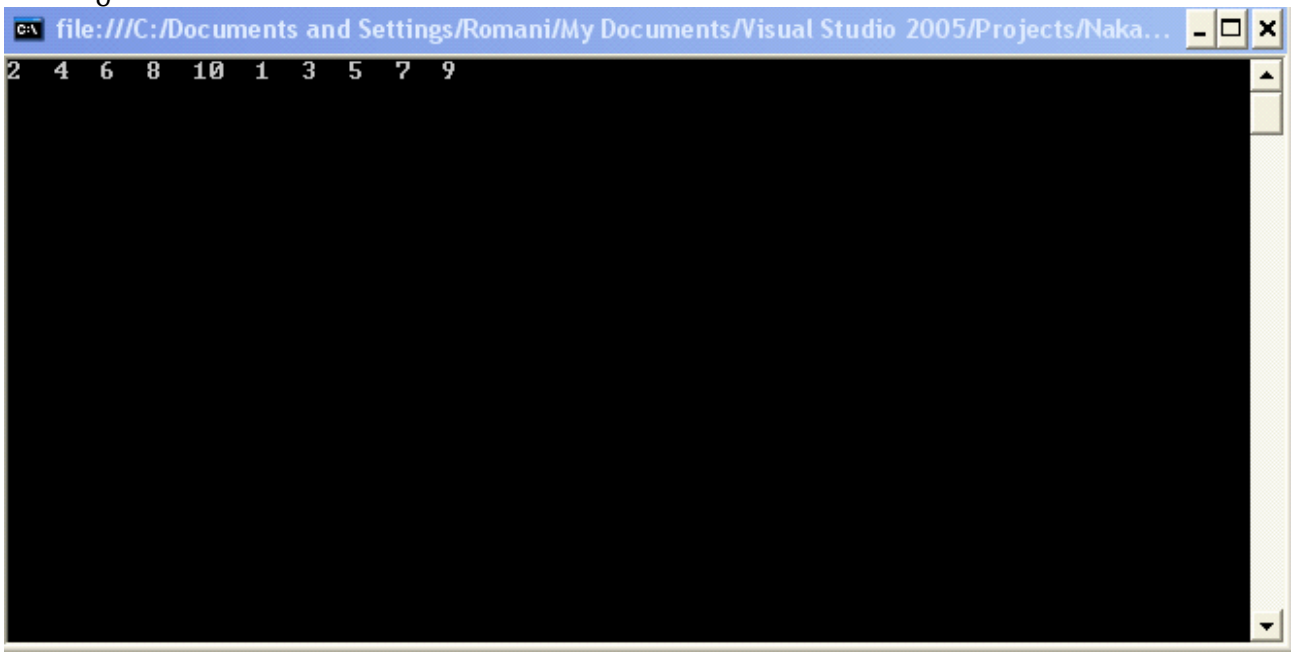
Join() მეთოდი გამოიყენება ერთი ნაკადის მისაბმელად მეორე ნაკადისათვის. შედეგად, მიბმული ნაკადი დაელოდება პირველი ნაკადის შესრულების დამთავრებას და ამის შემდეგ დაიწყებს მუშაობას. ამის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
//      პროგრამა 15.7
//      პროგრამაში ხდება ერთი ნაკადის მეორისათვის მიბმის დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Join
{
class ChemiKlasi
{
public static void Datvla_Kenti()
{
for (int indexi = 1; indexi <= 10; indexi += 2)
        Console.Write(indexi.ToString() + "  ");
Thread.Sleep(3000);
```

```
}
public static void Datvla_Luwi()
{
for (int indexi = 2; indexi <= 10; indexi += 2)
        Console.Write(indexi.ToString() + "  ");
Thread.Sleep(3000);
}
static void Main(string[] args)
{
Thread nakadi2 = new Thread(new ThreadStart(Datvla_Kenti));
//
nakadi2.Start();
//      nakadi2-ის ბლოკირება პირველი ნაკადის დამთავრებამდე
nakadi2.Join();
Datvla_Luwi();
}
}
}
```

პროგრამაში ხდება nakadi2 ნაკადის ბლოკირება მანამ, სანამ არ დამთავრდება Datvla_Luwi() მეთოდის შესრულება. პროგრამის შესრულების შედეგი ნაჩვენებია ნახ. 15.7-ზე.



ნახ. 15.7.

Join() მეთოდი საშუალებას გვაძლევს, აგრეთვე, მივუთითოთ ლოდინის მაქსიმალური დრო. მაგალითად, nakadi2.Join(4000) მეთოდის შესრულების შედეგად nakadi2 ნაკადი მეორე ნაკადის შესრულების დამთავრებას დაელოდება არა უმეტეს 4 წამისა.

lock საკვანძო სიტყვა

ჩვეულებრივ, მრავალნაკადიანი კოდის შესრულებისას ადგილი აქვს ბევრ

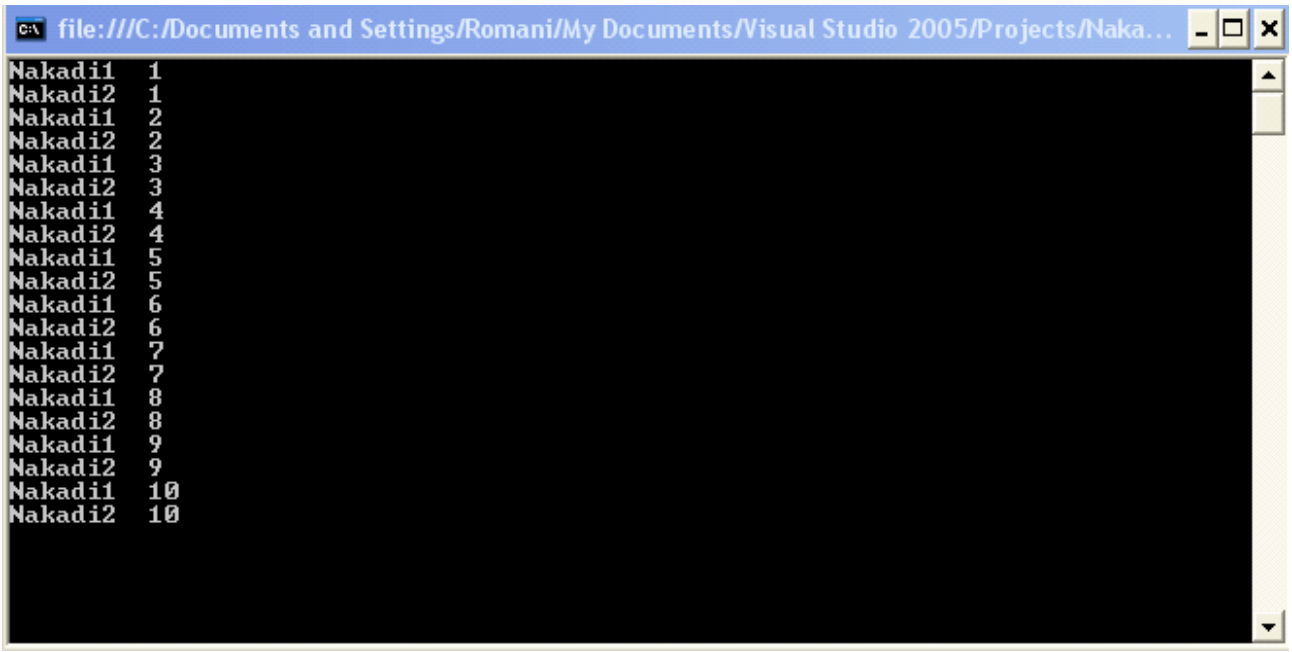
პრობლემას. ერთ-ერთი მათგანი ნაჩვენებია მოყვანილ პროგრამაში, რომელშიც ორი ნაკადი ერთსა და იმავე ცვლადთან მუშაობს.

```
//      პროგრამა 15.8
//      პროგრამაში ხდება პრობლემის დემონსტრირება, როცა ორი ნაკადი
//      ერთსა და იმავე ცვლადთან მუშაობს
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Lock
{
class ChemiKlasi
{
private int Saerto_Mtvleli = 0;
public void Datvla()
{
while ( Saerto_Mtvleli < 10 )
{
        int Temp = Saerto_Mtvleli;
    Temp++;
    Console.WriteLine(Thread.CurrentThread.Name + "  " + Temp);
    Thread.Sleep(500);
    Saerto_Mtvleli = Temp;
}
Thread.Sleep(5000);
}
static void Main(string[] args)
{
ChemiKlasi obieqti = new ChemiKlasi();
obieqti.Nakadebis_Shesruleba();
}
public void Nakadebis_Shesruleba()
{
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}
```

პროგრამაში Nakadi1 და Nakadi2 ნაკადები მუშაობენ Saerto_Mtvleli ცვლადთან. ამ ცვლადის მნიშვნელობას ზრდის Datvla() მეთოდი. შესაბამისად, შედეგი უნდა იყოს 1, 2,

3, ..., 10. რადგან ორი ნაკადი ერთდროულად ცვლის ერთი და იგივე ცვლადის მნიშვნელობას, ამიტომ შედეგი შეიძლება იყოს ჩვენთვის არასასურველი. მაგალითად, ასეთი:



ნახ. 15.8.

არასწორი შედეგი მიიღება იმ მოქმედებების შედეგად, რომლებსაც პროგრამა ასრულებს:

1. Nakadi1 ნაკადი Temp ცვლადს ანიჭებს Saerto_Mtvleli ცვლადის მნიშვნელობას, ანუ 0-ს. შემდეგ, Temp ცვლადის მნიშვნელობა ერთით იზრდება და ხდება 1-ის ტოლი.
2. Nakadi1 ნაკადი დროებით აჩერებს შესრულებას 0,5 წამის განმავლობაში.
3. ოპერაციული სისტემის დამგეგმავი იწყებს Nakadi2 ნაკადის შესრულებას.
4. Nakadi2 ნაკადი Temp ცვლადს ანიჭებს Saerto_Mtvleli ცვლადის მნიშვნელობას, ანუ 0-ს. როგორც ვხედავთ, Temp ცვლადის მნიშვნელობა, რომელიც 1-ის ტოლი გახდა Nakadi1 ნაკადის შესრულების დროს, ისევ 0-ის ტოლი ხდება Nakadi2 ნაკადის შესრულების შედეგად. შემდეგ, Temp ცვლადის მნიშვნელობა ერთით იზრდება და ხდება 1-ის ტოლი.
5. შემდეგ, Nakadi2 ნაკადი დროებით აჩერებს შესრულებას 0,5 წამის განმავლობაში.
6. Nakadi1 ნაკადი ისევ აგრძელებს შესრულებას და Saerto_Mtvleli ცვლადს ანიჭებს Temp ცვლადის მნიშვნელობას, კერძოდ, 1-ს.
7. Nakadi2 ნაკადი აგრძელებს შესრულებას და Saerto_Mtvleli ცვლადს ახალ მნიშვნელობას (Temp ცვლადის მნიშვნელობას) ანიჭებს, ცვლის რა Nakadi1 ნაკადის მიერ ამ ცვლადისთვის მინიჭებულ მნიშვნელობას.
8. ციკლი მეორდება.

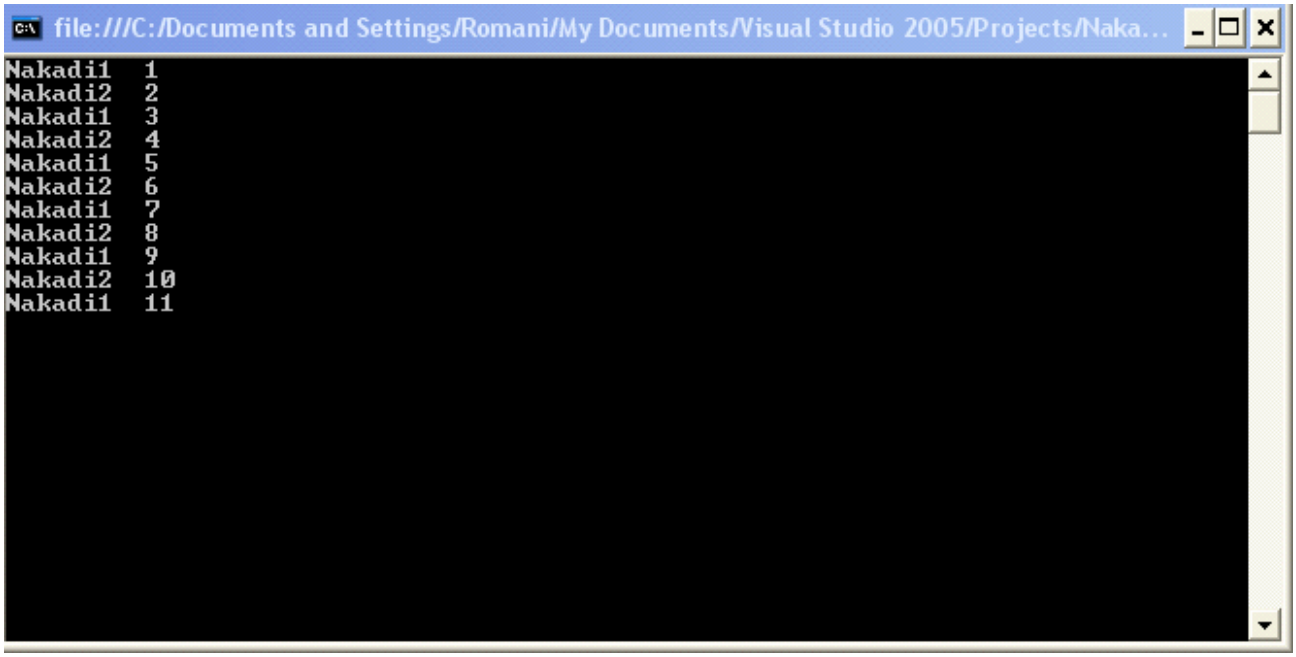
როგორც ვხედავთ, პრობლემას ადგილი აქვს იმის გამო, რომ სხვა პროცესის შესრულებაზე გადართვა ხდება ციკლის შუა ნაწილში. ამის თავიდან ასაცილებლად შეგვიძლია გამოვიყენოთ lock საკვანძო სიტყვა. იგი განსაზღვრავს კოდის იმ ფრაგმენტს, რომლის ბლოკირებაც უნდა მოხდეს. ბლოკირებული ფრაგმენტის შესრულების დროს არ მოხდება სხვა პროცესის შესრულებაზე გადართვა. lock სიტყვის გამოყენებით

შეიძლება ნებისმიერი ობიექტის ბლოკირება. მასთან ერთად მოყვანილ პროგრამაში გამოყენებულია this სიტყვა. შედეგად, მოხდება მთელი კლასის კოდის ბლოკირება.

```
//        პროგრამა 15.9
//        პროგრამაში ხდება lock სიტყვის გამოყენების დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Lock
{
class ChemiKlasi
{
private int Saerto_Mtvleli = 0;
public void Datvla()
{
while ( Saerto_Mtvleli < 10 )
{
lock (this)
{
     int Temp = Saerto_Mtvleli;
    Temp++;
    Console.WriteLine(Thread.CurrentThread.Name + "  " + Temp);
    Thread.Sleep(500);
    Saerto_Mtvleli = Temp;
}
}
Thread.Sleep(5000);
}
static void Main(string[] args)
{
ChemiKlasi obieqti = new ChemiKlasi();
obieqti.Nakadebis_Shesruleba();
}
public void Nakadebis_Shesruleba()
{
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}
```

პროგრამის შესრულების შედეგი იქნება ისეთი, როგორიც ჩვენ გვჭირდება (ნახ. 15.9).



ნახ. 15.9.

უკანასკნელი მნიშვნელობა 11 მივიღეთ იმიტომ, რომ როცა Nakadi1 ნაკადმა აღმოაჩინა რა Saerto_Mtvleli ცვლადის ახალი მნიშვნელობა 9, დაიწყო ახალი ციკლი და შემდეგ გადავიდა ბლოკირებულ მდგომარეობაში. ამასობაში Nakadi2 ნაკადმა ამ ცვლადის მნიშვნელობა შეცვალა და გახადა 10-ის ტოლი. როცა Nakadi1 ნაკადს მოეხსნება ბლოკირება, მაშინ ის მუშაობას აგრძელებს უკვე ამ ცვლადის შეცვლილ მნიშვნელობასთან.

Interlocked კლასი

Interlocked კლასი გამოიყენება ცვლადების უსაფრთხო ინკრემენტისა და დეკრემენტისათვის. ამ კლასის მეთოდები ნაჩვენებია ცხრილში 15.3.

ცხრილი 15.3. Interlocked კლასის მეთოდები.

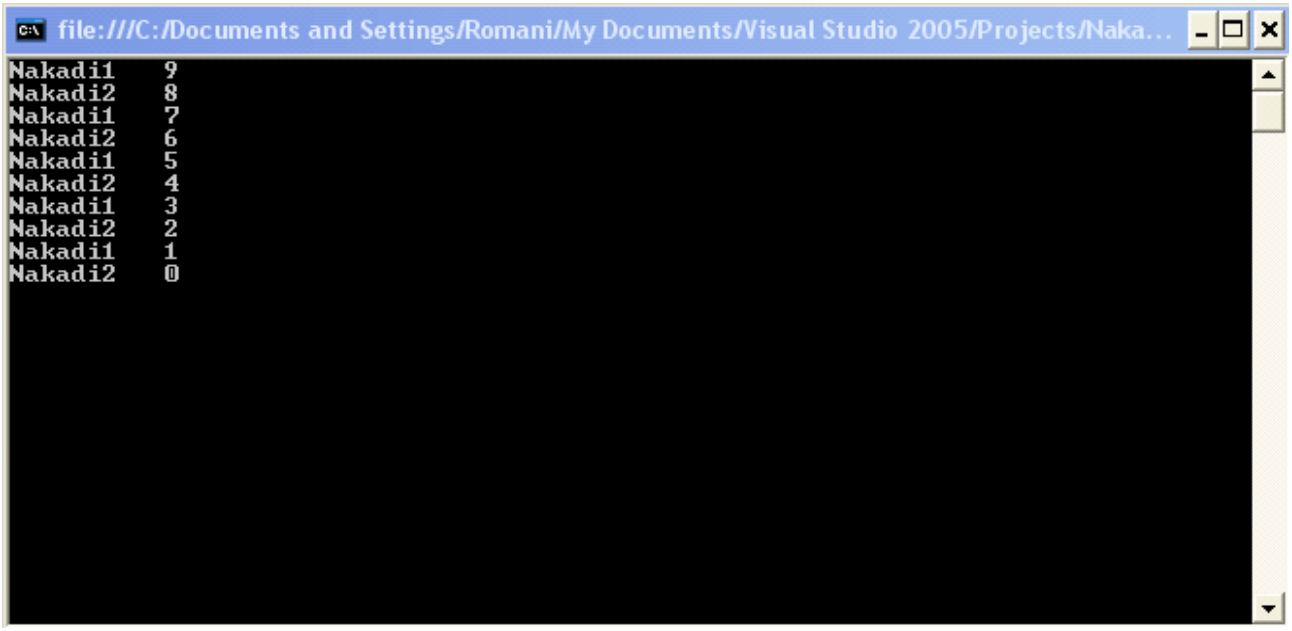
მეთოდი	აღწერა
Decrement() (სტატიკური)	ცვლადის მნიშვნელობას ერთით ამცირებს
Exchange() (სტატიკური)	ცვლადს მნიშვნელობას ანიჭებს
Increment() (სტატიკური)	ცვლადის მნიშვნელობას ერთით ზრდის

მოყვანილ პროგრამაში ხდება Decrement() მეთოდის გამოყენების დემონსტრირება.

```
//      პროგრამა 15.10
//      პროგრამაში ხდება Decrement() მეთოდის გამოყენების დემონსტრირება
using System;
using System.Collections.Generic;
```

```
using System.Text;
using System.Threading;

namespace Interlocked
{
class ChemiKlasi
{
private int Saerto_Mtvleli = 10;
public void Datvla()
{
while ( Saerto_Mtvleli > 0 )
{
        System.Threading.Interlocked.Decrement(ref Saerto_Mtvleli);
        Console.WriteLine(Thread.CurrentThread.Name + "    " + Saerto_Mtvleli);
    Thread.Sleep(500);
}
Thread.Sleep(5000);
}
static void Main(string[] args)
{
ChemiKlasi obieqti = new ChemiKlasi();
obieqti.Nakadebis_Shesruleba();
}
public void Nakadebis_Shesruleba()
{
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}
```



ნახ. 15.10.

პროგრამის შესრულების შედეგი ნაჩვენებია ნახ. 15.10-ზე.

თუ Datvla() მეთოდში Decrement() მეთოდს შევცვლით Increment() მეთოდით, მაშინ Saerto_Mtvleli ცვლადის მნიშვნელობა ერთით გაიზრდება ამ მეთოდის ყოველი შესრულებისას. Exchange() მეთოდის გამოყენების შემთხვევაში Datvla() მეთოდი შემდეგ სახეს მიიღებს.

```
public void Datvla()
{
Console.WriteLine(Thread.CurrentThread.Name + "   " + Saerto_Mtvleli);
System.Threading.Interlocked.Exchange(ref Saerto_Mtvleli, 55);
Console.WriteLine(Thread.CurrentThread.Name + "   " + Saerto_Mtvleli);
Thread.Sleep(5000);
}
```

Monitor კლასი

Monitor კლასი გამოიყენება კოდის ნებისმიერ ობიექტთან სინქრონიზებული მიმართვის განხორციელებისათვის. ამ კლასის მეთოდები მოყვანილია ცხრილში 15.4.

ცხრილი 15.4. Monitor კლასის მეთოდები.

მეთოდი	აღწერა
Enter() (სტატიკური)	ბლოკავს მონიტორისათვის გადაცემულ ობიექტს. თუ სხვა ნაკადმა უკვე დაბლოკა ეს ობიექტი, მაშინ მიმდინარე ნაკადის შესრულება დროებით შეწყდება მანამ, სანამ სხვა ნაკადი არ გაათავისუფლებს ამ ობიექტს
Exit() (სტატიკური)	ახდენს ობიექტის დებლოკირებას
Pulse() (სტატიკური)	მომდევნო მომლოდინე ნაკადს აუწყებს იმის შესახებ, რომ მონიტორმა დროებით დაამთავრა მუშაობა ობიექტთან და ნაკადს შეუძლია მუშაობის გაგრძელება

ცხრილი 15.4 (გაგრძელება)

PulseAll() (სტატიკური)	ყველა ნაკადს აუწყებს იმის შესახებ, რომ ობიექტი მალე გათავისუფლდება
TryEnter() (სტატიკური)	ცდილობს დაბლოკოს გადაცემული ობიექტი. თუ ობიექტის ბლოკირება შესაძლებელია, მაშინ გაიცემა true, წინააღმდეგ შემთხვევაში - false
Wait() (სტატიკური)	ათავისუფლებს ყველა ბლოკირებას და დროებით აჩერებს მიმდინარე ნაკადის შესრულებას მანამ, სანამ სხვა ნაკადი არ გამოიძახებს Pulse() მეთოდს

მოყვანილ პროგრამაში ხდება Monitor კლასის გამოიყენების დემონსტრირება.

```
//      პროგრამა 15.11
//      პროგრამაში ხდება Monitor კლასის გამოიყენების დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Monitor
{
class ChemiKlasi
{
private int Saerto_Mtvleli = 0;
public void Datvla()
{
while ( Saerto_Mtvleli < 10 )
{
        System.Threading.Monitor.Enter(this);
    int Temp = Saerto_Mtvleli;
    Temp++;
    Console.WriteLine(Thread.CurrentThread.Name + "   " + Temp);
    Thread.Sleep(500);
    Saerto_Mtvleli = Temp;
    System.Threading.Monitor.Exit(this);
}
Thread.Sleep(5000);
}
static void Main(string[] args)
{
ChemiKlasi obieqti = new ChemiKlasi();
obieqti.Nakadebi_Shesruleba();
}
public void Nakadebi_Shesruleba()
{
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
```

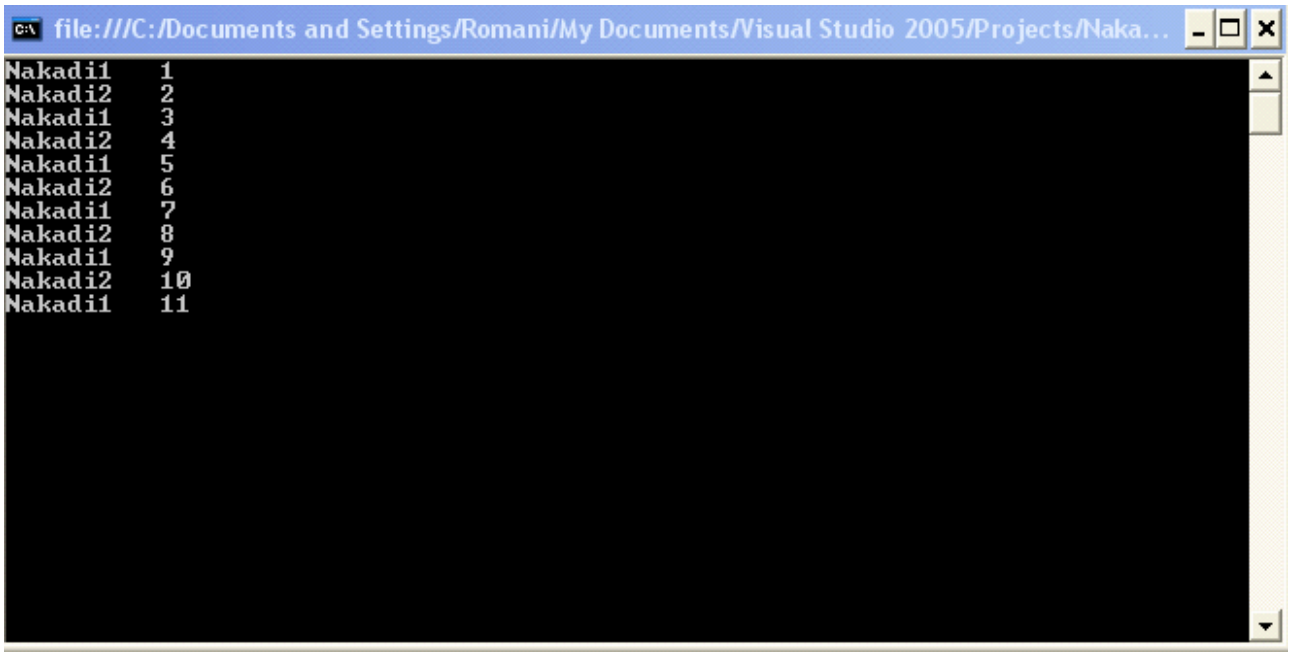
```
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}
```

პროგრამის შესრულების შედეგები ნაჩვენებია ნახ. 15.11-ზე.

Mutex კლასი

ნაკადების სინქრონიზება შესაძლებელია, აგრეთვე, Mutex კლასის, ანუ სემაფორების გამოიყენებით. სემაფორი არის ობიექტი, რომელსაც დროის განსაზღვრულ მომენტში შეიძლება იყენებდეს მხოლოდ ერთი ნაკადი. ჩვენ გამოვიყენებთ Mutex კლასის კონსტრუქტორს, რომელსაც ორი პარამეტრი აქვს. პირველი პარამეტრი ბულის ტიპისაა და განსაზღვრავს ობიექტი-სემაფორი უნდა ეკუთვნოდეს თუ არა მის შემქმნელ ნაკადს. მეორე პარამეტრი არის სემაფორის სახელი.

ნაკადს შეუძლია WaitOne() მეთოდის გამოძახება სემაფორის მისაღებად. ეს ნაკადი რჩება ბლოკირებულ მდგომარეობაში მანამ, სანამ სემაფორი სხვა ნაკადს უკავია. როგორც კი ნაკადი სემაფორს გაათავისუფლებს, სემაფორი მის მომთხოვნ ნაკადს დაეთმობა. როცა ნაკადი სემაფორთან მუშაობას დაამთავრებს, მაშინ მას შეუძლია ReleaseMutex() მეთოდის გამოძახება სემაფორის გასათავისუფლებლად.



ნახ. 15.11.

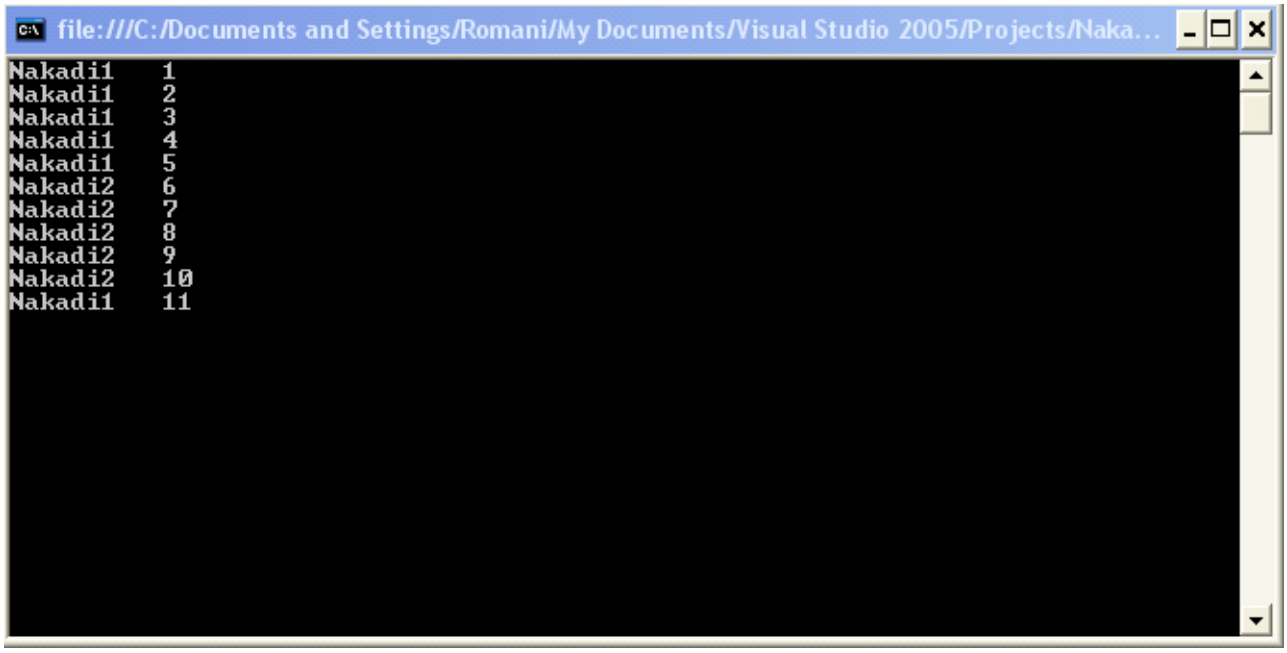
მოყვანილ პროგრამაში ხდება ნაკადების სინქრონიზებისათვის სემაფორის გამოყენების დემონსტრირება.

```
//      პროგრამა 15.12
//      პროგრამაში ხდება სემაფორების გამოყენების დემონსტრირება
using System;
using System.Collections.Generic;
```

```
using System.Text;
using System.Threading;

namespace Mutex
{
class ChemiKlasi
{
private static int Saerto_Mtvleli = 0;
//        სემაფორის შექმნა
static System.Threading.Mutex Semapori;
public static void Datvla()
{
while ( Saerto_Mtvleli < 10 )
{
    //   სემაფორის მიღება
    Semapori.WaitOne();
    int Temp = Saerto_Mtvleli;
    Temp++;
    Console.WriteLine(Thread.CurrentThread.Name + "    " + Temp);
    Thread.Sleep(500);
    Saerto_Mtvleli = Temp;
    //   სემაფორის გათავისუფლება
    Semapori.ReleaseMutex();
}
Thread.Sleep(5000);
}
static void Main()
{
Semapori = new System.Threading.Mutex(false, "SemaporisMushaoba");
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}
```

პროგრამის მუშაობის შედეგი ნაჩვენებია ნახ. 15.12-ზე.


file:///C:/Documents and Settings/Romani/My Documents/Visual Studio 2005/Projects/Naka...
Nakadi1 1
Nakadi1 2
Nakadi1 3
Nakadi1 4
Nakadi1 5
Nakadi2 6
Nakadi2 7
Nakadi2 8
Nakadi2 9
Nakadi2 10
Nakadi1 11

ნახ. 15.12.

თავი 16. საბაზო ბიბლიოთეკის სხვა კლასები

გლობალიზაცია

.NET გარემო შეიცავს გლობალიზაციის გაფართოებულ უზრუნველყოფას. System.Globalization სახელების სივრცეში განსაზღვრულია კლასები (ცხრილი 16.1), რომლებიც საშუალებას გვაძლევენ პროგრამის კოდი სწრაფად გავაწყოთ ჩვენთვის საჭირო ეროვნული კულტურის ელემენტების გამოყენებაზე. ეროვნული კულტურის ელემენტებია: ეროვნული ენა, ვალუტის სიმბოლო, დროის, თარიღისა და რიცხვის ფორმატი და ა.შ.

.NET გარემოში თითოეული ქვეყნის კულტურას შეესაბამება აღინიშვნა, რომელიც შედგება კულტურის კოდისგან, რომელსაც მოსდევს ერთი ან მეტი ქვეკულტურის კოდი. შეთანხმების თანახმად კულტურის კოდები პატარა ასოებით აღინიშნება, ხოლო ქვეკულტურის კოდები - დიდი ასოებით. მაგალითად, en-CA კოდი განსაზღვრავს ინგლისურ კულტურას კანადაში. ეს კულტურა შეიცავს ინფორმაციას კანადაში მცხოვრები ინგლისელებისათვის. pt-BR კოდი განსაზღვრავს პორტუგალიურ კულტურას ბრაზილიაში, ka-GE კოდი განსაზღვრავს ქართულ კულტურას, bs-BA-Latn კოდი განსაზღვრავს ბოსნიის კულტურას იმ მომხმარებლებისათვის, რომლებიც ლათინურ ანბანს იყენებენ და ა.შ. აქვე შევნიშნოთ, რომ .NET გარემოში ამერიკული ინგლისური ენა და ბრიტანული ინგლისური ენა სხვადასხვა კულტურებს მიეკუთვნება, რადგან ამ ქვეყნების ვალუტის სიმბოლოები და თარიღის ფორმატები ერთმანეთისაგან განსხვავდება.

ცხრილი 16.1. System.Globalization სახელების სივრცის კლასები.

კლასი	აღწერა
Calendar	აბსტრაქტული კლასია, რომელიც გვაძლევს საერთო წარმოდ- გე- ნას კულტურისათვის დამახასიათებელი კალენდრის შესახებ.
CompareInfo	სტრიქონების შედარების მეთოდების კულტურისათვის დამახასიათებელი კოლექციაა
CultureInfo	შეიცავს ინფორმაციას კულტურის ყველა ასპექტის შესახებ
DateTimeFormatInfo	შეიცავს ინფორმაციას დროისა და თარიღის ფორმატის შესახებ
DaylightTime	შეიცავს ინფორმაციას ზაფხულისა და ზამთრის დროზე გადასვლის შესახებ
NumberFormatInfo	შეიცავს ინფორმაციას რიცხვის წარმოდგენის ფორმატის შესახებ
RegionInfo	შეიცავს ინფორმაციას განსაზღვრული რეგიონის ან ქვეყნის შესახებ
SortKey	შეიცავს ინფორმაციას, რომელიც აუცილებელია სტრიქონების დახარისხებისათვის
StringInfo	წარმოგვიდგენს მეთოდებს სტრიქონების დაყოფისათვის ელემენტებად და ამ ელემენტების გადარჩევისათვის
TextInfo	შეიცავს ინფორმაციას ტექსტის დაფორმატების შესახებ

ცხრილი 16.2. CultureInfo კლასის თვისებები.

თვისება	ტიპი	აღწერა
Calendar	Calendar	შეიცავს კალენდარს, რომელიც მოცემულ კულტურაში ავტომატურად გამოიყენება
CompareInfo	CompareInfo	შეიცავს ინფორმაციას მოცემულ კულტურაში სტრიქონების შედარების შესახებ
CurrentCulture (სტატიკურია)	CurrentCulture	შეიცავს მოცემული ნაკადის მიმდინარე კულტურას
CurrentUICulture (სტატიკურია)	CultureInfo	შეიცავს სამომხმარებლო ინტერფეისის მიმდინარე კულტურას
DateTimeFormat	DateTimeFormat-Info	შეიცავს ინფორმაციას მოცემული კულტურის თარიღისა და დროის შესახებ
DisplayName	string	შეიცავს კულტურის ასახვად სახელს
EnglishName	string	შეიცავს კულტურის ინგლისურ დასახელებას
InstalledUICulture (სტატიკურია)	CultureInfo	შეიცავს ოპერაციულ სისტემასთან ერთად დაინსტალირებულ კულტურას
InvariantCulture (სტატიკურია)	CultureInfo	შეიცავს ინვარიანტულ კულტურას
IsNeutralCulture	bool	მიუთითებს CultureInfo ობიექტი წარმოადგენს თუ არა ნეიტრალურ კულტურას
IsReadOnly	bool	მიუთითებს CultureInfo ობიექტი არის თუ არა მხოლოდ წაკითხვადი
LCID	string	შეიცავს კულტურის იდენტიფიკატორს CultureInfo ობიექტისთვის
Name	string	შეიცავს კულტურის დასახელებას
NativeName	string	შეიცავს კულტურის ეროვნულ დასახელებას
NumberFormat	NumberFormat- -Info	შეიცავს ინფორმაციას რიცხვის ფორმატის შესახებ მოცემული კულტურისათვის
OptionalCalendars	Calendar[]	შეიცავს ალტერნატიული კალენდრების სიას მოცემული კულტურისათვის
Parent	CultureInfo	შეიცავს კულტურას, რომელიც არის მშობელი მოცემული კულტურისათვის
TextInfo	TextInfo	შეიცავს ინფორმაციას ტექსტის დაფორმატების შესახებ მოცემული კულტურისათვის
ThreeLetterISO-LanguageName	string	შეიცავს ISO ენის კოდს მოცემული კულტურის ენისათვის
TwoLetterISO-LanguageName	string	შეიცავს ISO ენის ორ ასოიან კოდს მოცემული კულტურის ენისათვის
UseUserOverride	bool	მიუთითებს CultureInfo ობიექტი იყენებს თუ არა მომხმარებლის მიერ არჩეულ გაწყობებს

ცხრილი 16.3. CultureInfo კლასის მეთოდები.

მეთოდი	აღწერა
ClearCachedData()	აახლებს ქეშირებულ ინფორმაციას კულტურის შესახებ
CreateSpecificCulture() (სტატიკურია)	კულტურის დასახელებიდან ქმნის CultureInfo ობიექტს
GetCultures() (სტატიკურია)	გასცემს კულტურების სიას
ReadOnly() (სტატიკურია)	გასცემს CultureInfo ობიექტის ვერსიას, რომლის მხოლოდ წაკითხვაა შესაძლებელი

ამ განყოფილებაში მოყვანილი ყველა პროგრამის დასაწყისში უნდა მოვათავსოთ დირექტივა using System.Globalization;

CultureInfo კლასის თვისებები და მეთოდები მოყვანილია ცხრლებში 16.2 და 16.3.

მოყვანილ პროგრამაში ხდება CultureInfo და მასთან დაკავშირებული კლასების გამოყენების დემონსტრირება.

```
{
//      პროგრამა 16.1
//      პროგრამაში სრულდება ქართულ კულტურასთან მუშაობა
//      იქმნება ქართული კულტურა
CultureInfo QartuliKultura = new CultureInfo("ka-GE");
label1.Text = QartuliKultura.NativeName;          //      გაიცემა - ქართული(საქართველო)
label2.Text = QartuliKultura.EnglishName;         //      გაიცემა - Georgian(Georgia)
//      ქართული კულტურისათვის განისაზღვრება თარიღისა და დროის ფორმატი
DateTimeFormatInfo TarigiDroisFormati = QartuliKultura.DateTimeFormat;
label8.Text += TarigiDroisFormati.LongDatePattern;
                                                  // გაიცემა - yyyy'წლის'dd MM, dddd
//      ქართული კულტურისათვის განისაზღვრება ვალუტისა და რიცხვის ფორმატი
NumberFormatInfo RicxvisFormati = QartuliKultura.NumberFormat;
label3.Text = RicxvisFormati.CurrencySymbol;      //      გაიცემა - Lari
label4.Text = RicxvisFormati.NumberDecimalSeparator;  //      გაიცემა - ,
}
```

პროგრამაში იქმნება CultureInfo კლასის QartuliKultura ობიექტი, რომელიც ქართულ კულტურას წარმოადგენს. შემდეგ, გაიცემა ამ კულტურის სახელი ქართულ და ინგლისურ ენებზე, თარიღის გრძელი ფორმატი, ვალუტის სიმბოლო და მთელისა და წილადის გამყოფი.

CurrentCulture და CurrentUICulture თვისებები. CurrentCulture თვისება გამოიყენება კულტურის ისეთი მონაცემების ფორმატირებისათვის, როგორიცაა რიცხვების, თარიღისა და დროის ფორმატი, ვალუტის სიმბოლო, დახარისხების რიგითობა და სტრიქონების შედარების წესები. CurrentUICulture თვისება გამოიყენება სამომხმარებლო ინტერფეისის ენისათვის. იგი ახდენს შესაბამისი ფაილიდან კულტურაზე დამოკიდებული რესურსების, პირველ რიგში კი სამომხმარებლო ინტერფეისის ტექსტის, მიღებას. მოყვანილ პროგრამაში ხდება ამ თვისებებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამა 16.2
//      პროგრამაში ხდება CurrentCulture და CurrentUICulture თვისებებთან მუშაობის
//      დემონსტრირება
//      იქმნება ქართული კულტურა
Thread.CurrentThread.CurrentCulture = new CultureInfo("ka-GE");
label1.Text = Thread.CurrentThread.CurrentCulture.Name + '\n';
// იქმნება დიდი ბრიტანეთის კულტურა
CultureInfo InglisisKultura1 = new CultureInfo("en-GB");
label1.Text += CultureInfo.CurrentCulture.Name + '\n';
//      სამომხმარებლო ინტერფეისისათვის მიმდინარე კულტურის დაყენება
//      მიმდინარე ნაკადისათვის განისაზღვრება იტალიის კულტურა
Thread.CurrentThread.CurrentUICulture = new CultureInfo("it-IT");
label1.Text += Thread.CurrentThread.CurrentUICulture.Name + '\n';
//      იქმნება დიდი ბრიტანეთის კულტურა
CultureInfo InglisisKultura2 = new CultureInfo("en-GB");
label1.Text += CultureInfo.CurrentUICulture.Name;
}
```

.NET გარემო ორ სხვადასხვა ნაკადში ახდენს ორ სხვადასხვა კულტურასთან მუშაობას. როგორც პროგრამიდან ჩანს, მიმდინარე ნაკადისათვის მიმდინარე კულტურად განისაზღვრა საქართველო - “ka-GE”, CultureInfo ობიექტის შექმნის გზით. ამ ობიექტზე მიმართვა ენიჭება Thread.CurrentThread.CurrentCulture თვისებას. შემდეგ, იქმნება დამოუკიდებელი InglisisKultura1 ობიექტი, რომელიც განსაზღვრავს ინგლისის კულტურას - “en-GB”. მაგრამ, ეს კულტურა არ არის მიმდინარე, ამიტომ ეკრანზე ისევ გამოჩნდება “ka-GE” კულტურა.

კულტურების ჩამონათვალი. CultureInfo ობიექტის GetCultures() მეთოდი გასცემს კულტურების სიას. მეთოდს პარამეტრად გადაეცემა CultureTypes პარამეტრი, რომელიც შეიცავს კულტურების ნაკრებს. ცხრილში 16.4 მოყვანილია ამ პარამეტრის მნიშვნელობები.

ცხრილი 16.4. CultureTypes პარამეტრის მნიშვნელობები.

მნიშვნელობა	აღწერა
AllCultures	ყველა კულტურა, ამოცნობილი .NET გარემოს მიერ
InstalledWin32Cultures	ყველა კულტურა, დაყენებული ოპერაციულ სისტემაში
NeutralCultures	ყველა ნეიტრალური კულტურა
SpecificCultures	ყველა სპეციფიური კულტურა

მოყვანილ პროგრამაში ხდება თითოეული კულტურის დასახელების ეკრანზე გამოტანა.

```
{
//      პროგრამა 16.3
//      პროგრამაში ხდება თითოეული კულტურის დასახელების ეკრანზე გამოტანა
foreach ( CultureInfo Kultura in
                                CultureInfo.GetCultures(CultureTypes.AllCultures) )
    listBox1.Items.Add(Kultura.EnglishName + "-" + Kultura.Name);
```

}

პროგრამაში listBox1 კომპონენტი გამოიყენება კულტურების სახელების გამოსატანად. იგი შეგიძლიათ ჩასვათ Toolbox პანელიდან.

ინვარიანტული კულტურა. ინვარიანტული კულტურა (Invariant Language, Invariant Country) გამოიყენება იმ მონაცემებთან სამუშაოდ, რომლებიც არ იქნება ასახული რომელიმე კულტურის ელემენტების მიხედვით. ის დაკავშირებულია ინგლისურ ენასთან და არა რომელიმე ქვეყანასთან ან რეგიონთან. CultureInfo ობიექტი, რომელიც წარმოადგენს ინვარიანტულ კულტურას, შეგვიძლია ორი სხვადასხვა საშუალებით შევქმნათ:

```
CultureInfo InvariantuliKultura1 = new CultureInfo(“”);
CultureInfo InvariantuliKultura2 = CultureInfo.InvariantCulture;
```

მოყვანილ პროგრამაში ხდება თარიღისა და ვალუტის დაფორმატება საქართველოსა და დიდი ბრიტანეთის კულტურებისათვის.

```
{
//      პროგრამა 16.4
//      პროგრამაში ხდება თარიღისა და ვალუტის დაფორმატება
DateTime Tarigi = DateTime.Now;
Double   Valuta = 12345.67;
// თარიღისა და ვალუტის დაფორმატება ქართული კულტურის პარამეტრების
შესაბამისად
CultureInfo SaqartvelosKultura = new CultureInfo("ka-GE");
string QartuliTarigi = Tarigi.ToString("d", SaqartvelosKultura);
string QartuliValuta = Valuta.ToString("c", SaqartvelosKultura);
label10.Text = QartuliTarigi + '\n' + QartuliValuta + '\n';
// თარიღისა და ვალუტის დაფორმატება ინგლისის კულტურის პარამეტრების
შესაბამისად
CultureInfo InglisisKultura = new CultureInfo("en-GB");
string InglisuriTarigi = Tarigi.ToString("d", InglisisKultura);
string InglisuriValuta = Valuta.ToString("c", InglisisKultura);
label10.Text += InglisuriTarigi + '\n' + InglisuriValuta + '\n';
}
```

პროგრამაში გამოიყენება ToString() მეთოდის გადატვირთული ვერსია, რომელიც ახდენს თარიღისა და ვალუტის დაფორმატებას.

გრაფიკა

.NET გარემოში განსაზღვრულია გრაფიკის კლასები, რომლებიც განკუთვნილია GDI+ ინტერფეისთან სამუშაოდ (Graphic Device Interface, გრაფიკული მოწყობილობების ინტერფეისი). ეს კლასები საშუალებას გვაძლევენ დავხატოთ გეომეტრიული ფიგურები, ვიმუშაოთ გამოსახულების შემცველ ფაილებთან და ტიპოგრაფიულ შრიფტებთან. GDI+ არის კლასების ბიბლიოთეკა, რომლის დანიშნულებაა გრაფიკული ობიექტების ასახვა ჩვენი კომპიუტერის აპარატურული საშუალებებით. მისი მიზანია პროგრამა-დანართების შემუშავებლების იზოლირება სხვადასხვა გრაფიკული მოწყობილობების რეალიზების დეტალებისაგან. ჩვენს კოდს შეუძლია გამოიძახოს GDI+ ბიბლიოთეკაში

შემავალი კლასები. იგი ინსტალირდება Windows XP, Windows .NET Server ან .NET გარემოს ინსტალირების დროს. GDI+ კლასები სამ ფუნქციას ასრულებენ:
- ვექტორული გრაფიკის ისეთი ელემენტების ასახვა, როგორიცაა წერტილები, ხაზები და გეომეტრიული ფიგურები.
- გრაფიკული ობიექტების შემცველი ფაილების ასახვა, მაგალითად, bmp, gif, tif, jpg და png ფორმატის მქონე.
- შრიფტების ასახვა.

ცხრილი 16.5. Graphics კლასის თვისებები.

თვისება	ტიპი	აღწერა
Clip	Region	განსაზღვრავს ხატვის უბანს
ClipBounds	Rectangle	შეიცავს Rectangle ობიექტს, რომელიც განსაზღვრავს ხატვის უბანს
CompositingMode	CompositingMode	შედგენილი გამოსახულებებისათვის მიუთითებს ხატვის რეჟიმს
CompositingQuality	CompositingQuality	შეიცავს გამოსახულებების ხარისხს
DpiX	float	განსაზღვრავს ჰორიზონტალური გარჩევის უნარს Graphics ობიექტისთვის
DpiY	float	განსაზღვრავს ვერტიკალური გარჩევის უნარს Graphics ობიექტისთვის
InterpolatingMode	InterpolatingMode	განსაზღვრავს შუალედური წერტილების გამოთვლის რეჟიმს
IsClipEmpty	bool	მიუთითებს, არის თუ არა კვეთის უბანი ცარიელი
PageScale	float	შეიცავს მასშტაბირების კოეფიციენტს Graphics ობიექტისთვის
PageUnit	GraphicsUnit	განსაზღვრავს გამოყენებულ ზომის ერთეულს
PixelOffsetMode	PixelOffsetMode	განსაზღვრავს პიქსელების წანაცვლების საშუალებას ხატვის დროს
RenderingOrigin	Point	განსაზღვრავს კოორდინატების დასაწყისს ფუნჯისა და შტრიხებისათვის
SmoothingMode	SmoothingMode	მართავს ხაზის სიგლუვეს Graphics ობიექტზე მათი ხატვის დროს
TextContrast	int	განსაზღვრავს კონტრასტის მნიშვნელობას ტექსტისათვის
TextRenderingHint	TextRenderingHint	განსაზღვრავს შემხსენებელ შეტყობინებას ტექსტისათვის
Transform	Matrix	განსაზღვრავს გეომეტრიულ გარდაქმნას
VisibleClipbounds	Rectangle	შეიცავს კვეთის ხილულ უბანს Graphics ობიექტისთვის

ცხრილი 16.6. Graphics კლასის მეთოდები.

მეთოდი	აღწერა
AddMetafileComment()	Metafile ობიექტს უმატებს კომენტარს
BeginContainer()	ხსნის ახალ გრაფიკულ კონტეინერს
Clear()	ასუფთავებს სახატავ ზედაპირს
DrawArc()	ხატავს რკალს
DrawBezier()	ხატავს ბაზიეს მრუდს
DrawBeziers()	ხატავს ბაზიეს მრუდების სერიას
DrawClosedCurve()	ხატავს დახურულ ფუნდამენტურ მრუდს
DrawCurve()	ხატავს ფუნდამენტურ მრუდს
DrawEllipse()	ხატავს ელიფსს
DrawIcon()	ხატავს პიქტოგრამას
DrawIconUnstretched()	ხატავს პიქტოგრამას მასშტაბირების გარეშე
DrawImage()	ხატავს გამოსახულებას
DrawImageUnscaled()	ხატავს გამოსახულებას მასშტაბირების გარეშე
DrawLine()	ხატავს ხაზს
DrawLines()	ხატავს ხაზების სერიას
DrawPath()	ხატავს GraphicsPath ობიექტს
DrawPie()	ხატავს სექტორს
DrawPolygon()	ხატავს მრავალკუთხედს
DrawRectangle()	ხატავს სწორკუთხედს
DrawRectangles()	ხატავს სწორკუთხედების სერიას
DrawString()	ხატავს ტექსტის სტრიქონს
EndContainer()	ხურავს გრაფიკულ კონტეინერს
EnumerateMetafile()	ახდენს მეტაფაილის ვიზუალიზებას
ExcludeClip()	სწორკუთხედს უმატებს კვეთის უბანს
FillClosedCurve()	ფერით ავსებს ჩაკეტილ მრუდს
FillEllipse()	ფერით ავსებს ელიფსს
FillPath()	ფერით ავსებს GraphicsPath ობიექტს
FillPie()	ფერით ავსებს სექტორს
FillPolygon()	ფერით ავსებს მრავალკუთხედს
FillRectangle()	ფერით ავსებს სწორკუთხედს
Fill Rectangles()	ფერით ავსებს სწორკუთხედების სერიას
FillRegion()	ფერით ავსებს Region ობიექტს
Flush()	იწვევს რიგში მყოფი ოპერაციების იძულებით შესრულებას
FromImage() (სტატიკურია)	Image ობიექტიდან ქმნის Graphics ობიექტს
GetNearestColor()	გასცემს მითითებულ ფერთან ახლოს მყოფ ფერს
IntersectClip()	ცვლის კვეთის უბანს სწორკუთხედთან მისი გადაკვეთის გზით
IsVisible()	გასცემს true-ს თუ მითითებული წერტილი ჩანს, წინააღმდეგ შემთხვევაში კი - false-ს

ცხრილი 16.6. (გაგრძელება)

MeasureCharacter-Ranges()	გასცემს ინფორმაციას ტექსტური სტრიქონის შესახებ
MeasureString()	გასცემს სტრიქონის ზომას, თუ ის დახატულია მითითებული შრიფტით
ResetClip()	აუქმებს კვეთის უბანს
ResetTransform()	აუქმებს Graphics ობიექტის გარდაქმნას
Restore()	აღადგენს GraphicsState ობიექტში შენახულ მდგომარეობას
RotateTransform()	ახდენს Graphics ობიექტის მობრუნებას
Save()	მდგომარეობას ინახავს GraphicsState ობიექტში
ScaleTransform()	ცვლის Graphics ობიექტის გარდაქმნის მასშტაბს
SetClip()	აყენებს კვეთის უბანს

ცხრილი 16.7. Pen კლასის თვისებები.

თვისება	ტიპი	აღწერა
Alignment	PenAlignment	შეიცავს Pen ობიექტის საზღვარზე გასწორების სახეს
Brush	Brush	შეიცავს Brush ობიექტს, რომელსაც იყენებს მოცემული Pen ობიექტი
Color	Color	შეიცავს Pen ობიექტის ფერს
CustomEndCap	CustomLineCap	შეიცავს ხაზის დაბოლოების სტილს
CustomStartCap	CustomLineCap	შეიცავს ხაზის დასაწყისის სტილს
DashCap	DashCap	შეიცავს პუნქტირის მონაკვეთების დაბოლოებების სტილს
DashOffset	Float	შეიცავს მანძილს ხაზის დასაწყისიდან პუნქტირის შაბლონამდე
DashPattern	Float [ ]	პუნქტირის მონაკვეთებისა და მათ შორის მანძილების მასივია
DashStyle	DashStyle	შეიცავს პუნქტირის მონაკვეთების სტილს
EndCap	LineCap	შეიცავს ხაზის დაბოლოების სტილს
LineJoin	LineJoin	შეიცავს რამდენიმე ხაზს შორის შეერთების სტილს
MiterLimit	Float	შეიცავს ჩამოჭრილი კუთხეების სისქეს რამდენიმე ხაზის შეერთებისას
PenType	PenType	შეიცავს სტილს, რომელიც გამოიყენება Pen ობიექტის მიერ ხაზების გავლებისას
StartCap	LineCap	შეიცავს ხაზის დასაწყისის სტილს
Width	Float	ხაზის სისქეა პიქსელებში

ცხრილი 16.8. Pen კლასის მეთოდები.

მეთოდი	აღწერა
Clone()	ქმნის Pen ობიექტის ასლს
ResetTransform()	ახდენს გარდაქმნის დაყვანას საწყის მდგომარეობაზე
RotateTransform()	ახდენს გარდაქმნის მობრუნებას
ScaleTransform()	ახდენს გარდაქმნის მასშტაბირებას
SetLineCap()	აყენებს StartCap და EndCap თვისებებს
TranslateTransform()	ახდენს გარდაქმნის ტრანსლაციას

Graphics კლასი განსაზღვრავს სახატავ უბანს (ფურცელს, ტილოს), Pen კლასი გამოიყენება ხაზებისა და გრაფიკული ფიგურების დასახატად, Brush კლასი გამოიყენება ფიგურების შესავსებად არჩეული ფერით. Rectangle კლასი განსაზღვრავს სახატავი უბნის საზღვრებს. Graphics კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 16.5 და 16.6. Pen კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 16.7 და 16.8.

სიგრძის ერთეული ყველა გრაფიკული ობიექტისათვის არის პიქსელი. კოორდინატების დასაწყისად ითვლება Graphics ობიექტის ზედა მარცხენა კუთხე.

Graphics ობიექტი წარმოადგენს ხატვის ზედაპირს და შეგვიძლია განვიხილოთ როგორც სახატავი ფურცელი (ტილო). Pen ობიექტი შეიცავს დასახატი ხაზის თვისებებს.

გეომეტრიული ფიგურების ხატვა

განვიხილოთ Graphics კლასის ზოგიერთი მეთოდი.

Graphics კლასის ობიექტი იქმნება CreateGraphics() მეთოდის საშუალებით. მაგალითად,

```
Graphics SaxataviPurceli = CreateGraphics();
```

DrawArc() მეთოდი გამოიყენება ელიფსის ნაწილის (რკალის) დასახატად. მისი სინტაქსია:

```
void Graphics.DrawArc(Pen ფუნჯი, Rectangle სწორკუთხედი,
                      float საწყისი_კუთხე, float მოღუნვის_კუთხე);
```

მაგალითი.

```
{
//      სახატავი ფურცლის შექმნა
Graphics SaxataviPurceli = CreateGraphics();
//      კალმის შექმნა
Pen Kalami = new Pen(Color.Red, 2);
//      ხატვის უბნის შექმნა
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
//      რკალის ხატვა
SaxataviPurceli.DrawArc(Kalami, Otxkutxedi, 0, 90);
}
```

როგორც პროგრამიდან ჩანს, ჯერ იქმნება სახატავი ფურცელი. შემდეგ იქმნება წითელი ფერის კალამი, რომლის სისქეა 2 პიქსელი. შემდეგ იქმნება ხატვის უბანი,

რომელშიც უნდა მოთავსთდეს რკალი. ამ უბნის საწყისი წერტილია (0,0), საბოლოო წერტილი კი - (200,300). ბოლოს სრულდება რკალის ხატვა.

DrawRectangle() მეთოდი გამოიყენება სწორკუთხედის დასახატად. მისი სინტაქსია:

void Graphics.DrawRectangle(Pen *ფუნჯი,* Rectangle *სწორკუთხედი);*

მაგალითი.

```
{
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
SaxataviPurceli.DrawRectangle(Kalami, Otxkutxedi);
}
```

DrawEllipse() მეთოდი გამოიყენება ელიფსის დასახატად. მისი სინტაქსია:

void Graphics.DrawEllipse(Pen *ფუნჯი,* Rectangle *სწორკუთხედი);*

მაგალითი.

```
{
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
SaxataviPurceli.DrawEllipse(Kalami, Otxkutxedi);
}
```

DrawPie() მეთოდი გამოიყენება სექტორის დასახატად. მისი სინტაქსია:

void Graphics.DrawPie(Pen *ფუნჯი,* Rectangle *სწორკუთხედი,*
float *საწყისი_კუთხე,* float *მოღუნვის_კუთხე);*

მაგალითი.

```
{
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
SaxataviPurceli.DrawPie(Kalami, Otxkutxedi, 0, 90);
}
```

DrawLine() მეთოდი გამოიყენება ხაზის გასავლებად საწყისი წერტილიდან საბოლოო წერტილამდე. მისი სინტაქსია:

void Graphics.DrawLine(Pen *ფუნჯი,* float x1, float y1, float x2, float y2);

აქ x1 და y1 არის საწყისი წერტილის კოორდინატები, ხოლო x2 და y2 კი - საბოლოო წერტილის. მაგალითი.

```
{
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
SaxataviPurceli.DrawLine(Kalami, 0, 0, 100, 100);
}
```

DrawPolygon() მეთოდი გამოიყენება მრავალკუთხედის ასაგებად. მისი სინტაქსია:

void Graphics.DrawPolygon(Pen *ფუნჯი,* Point [ ] *წერტილები);*

მაგალითი.

```
{
// წერტილების შექმნა, რომლებიც მრავალკუთხედს განსაზღვრავენ
Point wertili1 = new Point(50, 50);
Point wertili2 = new Point(100, 25);
Point wertili3 = new Point(130, 5);
Point wertili4 = new Point(170, 50);
Point wertili5 = new Point(210, 100);
Point wertili6 = new Point(50, 50);
Point[] WertilebisMasivi =  {
                                        wertili1,
                                        wertili2,
                                        wertili3,
                                        wertili4,
                                        wertili5,
                                        wertili6
                        };
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
SaxataviPurceli.DrawPolygon(Kalami, WertilebisMasivi);
}
```

DrawLines() მეთოდი გამოიყენება ხაზების გასავლებად საწყისი წერტილიდან საბოლოო წერტილამდე. მისი სინტაქსია:

void Graphics.DrawLines(Pen *ფუნჯი,* Point [ ] *წერტილები);*

მაგალითი.

```
{
//      წერტილების შექმნა ხაზებისათვის
Point wertili1 = new Point(50, 50);
Point wertili2 = new Point(100, 25);
Point wertili3 = new Point(130, 5);
Point wertili4 = new Point(170, 50);
Point wertili5 = new Point(210, 100);
Point wertili6 = new Point(50, 50);
Point[] WertilebisMasivi =  {
                                        wertili1,
                                        wertili2,
                                        wertili3,
                                        wertili4,
                                        wertili5,
                                        wertili6
                                };
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
SaxataviPurceli.DrawLines(Kalami, WertilebisMasivi);
}
```

Pen კლასის კონსტრუქტორს აქვს 4 გადატვირთული ვერსია. პირველი ვერსიის

სინტაქსია:
Pen *ობიექტის_სახელი* = new Pen(Brush *ფუნჯი);*
მაგალითი.
{
Brush PunjiSolid = new SolidBrush(Color.Red);
Pen Kalami = new Pen(PunjiSolid);
}
მეორე ვერსიის სინტაქსია:
Pen *ობიექტის_სახელი* = new Pen(Color *ფერი);*
მაგალითი.
{
Pen Kalami = new Pen(Color.Red);
}
მესამე ვერსიის სინტაქსია:
Pen *ობიექტის_სახელი* = new Pen(Brush *ფუნჯი,* float *სიგანე);*
მაგალითი.
{
Brush PunjiSolid = new SolidBrush(Color.Red);
Pen Kalami = new Pen(PunjiSolid, 3);
}
მეოთხე ვერსიის სინტაქსია:
Pen *ობიექტის_სახელი* = new Pen(Color *ფერი,* float *სიგანე);*
მაგალითი.
{
Pen Kalami = new Pen(Color.Red, 3);
}
მოყვანილ პროგრამაში ხდება ხაზისა და ისრის გავლების დემონსტრირება.
{
// პროგრამა 16.5
// პროგრამაში ხდება ხაზის გავლების დემონსტრირება
int x1 = 5, y1 = 5, x2 = 200, y2 = 200;
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 5);
SaxataviPurceli.DrawLine(Kalami, x1, y1, x2, y2);
//
x1 = 5; y1 = 200; x2 = 200; y2 = 5;
Kalami.StartCap = LineCap.ArrowAnchor; // ხაზის დასაწყისს ექნება ისრის ფორმა
Kalami.EndCap = LineCap.Round; // ხაზის დაბოლოება იქნება მომრგვალებული
SaxataviPurceli.DrawLine(Kalami, x1, y1, x2, y2);
// რესურსების გათავისუფლება
Kalami.Dispose();
SaxataviPurceli.Dispose();
}
პროგრამაში ჯერ იქმნება Graphics კლასის SaxataviPurceli ობიექტი. შემდეგ

იქმნება Pen კლასის Kalami ობიექტი, რომელსაც ექნება წითელი ფერი, სისქე კი - 5 პიქსელი. თუ სისქე არ არის მითითებული, მაშინ იგი აიღება 1-ის ტოლი. DrawLine მეთოდი (0,0) წერტილიდან გაავლებს წითელ ხაზს (200, 200) წერტილამდე. შემდეგ პირველი წერტილის კოორდინატი ხდება (5, 200), მეორე წერტილის კი - (200, 5). ხაზის დასაწყისს ექნება ისრის ფორმა, ბოლოს კი - მომრგვალებული სახე. ამის შემდეგ, ისარი გაივლება (5, 200) წერტილიდან (200, 5) წერტილამდე. პროგრამის ბოლოს ხდება Kalami და SaxataviPurceli ობიექტების მიერ დაკავებული რესურსების გათავისუფლება.

ფიგურების შევსება ფერით

Brush კლასი გამოიყენება სხვადასხვა ფერით და შაბლონით ფიგურების შესავსებად. ის აბსტრაქტულია და ამიტომ ამ კლასის ობიექტებს ვერ შევქმნით. მისგან წარმოებული კლასები მოყვანილია ცხრილში 16.9.

ცხრილი 16.9. Brush კლასიდან წარმოებული კლასები.

კლასი	გამოყენება
HatchBrush	გამოიყენება უბნის შესავსებად სტანდარტული შაბლონების ნაკრებიდან ამორჩეული შაბლონის მიხედვით
LinearGradientBrush	გამოიყენება უბნის შესავსებად ორ ფერს შორის თანაბარი გადასვლით სწორი ხაზის გასწვრივ
PathGradientBrush	გამოიყენება უბნის შესავსებად ორ ფერს შორის თანაბარი გადასვლით ნებისმიერი ხაზის გასწვრივ
SolidBrush	გამოიყენება უბნის შესავსებად ერთტონიანი ფერით
TextureBrush	გამოიყენება უბნის შესავსებად ფაილიდან წაკითხული გამოსახულებით

მოყვანილ პროგრამაში ხდება სხვადასხვა გეომეტრიული ფიგურის სხვადასხვა სტილით შევსების დემონსტრირება, კერძოდ, სწორკუთხედისთვის გამოიყენება გრადიენტული შევსება, ელიფსისთვის - შაბლონის მიხედვით შევსება, ხოლო სექტორისთვის კი - თანაბარი ფერით შევსება.

```
{
//      პროგრამა 16.6
//      პროგრამაში ხდება  სხვადასხვა გეომეტრიულ ფიგურის სხვადასხვა სტილით შევსება
int x = 0;
int y = 0;
int Sigane  = 200;
int Simagle = 300;
//      სახატავი ფურცლის შექმნა
Graphics SaxataviPurceli = CreateGraphics();
//      სახატავი უბნის შექმნა
Rectangle Otxkutxedi = new Rectangle(x, y, Sigane, Simagle);
//      ფუნჯის შექმნა თანაბარი ფერით შევსებისათვის
Brush PunjiSolid = new SolidBrush(Color.Red);
//      ფუნჯის შექმნა შაბლონის მიხედვით შევსებისათვის
```

```
Brush PunjiHatch = new HatchBrush(HatchStyle.Horizontal, Color.Blue,
                                        Color.Yellow);
//      ფუნჯის შექმნა გრადიენტული შევსებისათვის
Brush PunjiGradient = new LinearGradientBrush(Otxkutxedi, Color.Black,
                                                Color.Aqua, 45, false);
//      ოთხკუთხედის, ელიფსისა და სექტორის შევსება ფერით
SaxataviPurceli.FillRectangle(PunjiGradient, Otxkutxedi);
SaxataviPurceli.FillEllipse(PunjiHatch, 5, 110, 100, 100);
SaxataviPurceli.FillPie(PunjiSolid, 60, 60, 120, 120, 270, 90);
//      რესურსების გათავისუფლება
PunjiSolid.Dispose();
PunjiHatch.Dispose();
PunjiGradient.Dispose();
SaxataviPurceli.Dispose();
}
```

სტრიქონების ხატვა

DrawString() მეთოდი გამოიყენება სტრიქონების დასახატად. მისი სინტაქსია:

void Graphics.DrawString(string *სტრიქონი,* Font *შრიფტი,* Brush *ფუნჯი,* float *x*, float *y*);

მოყვანილ პროგრამაში ხდება ჰორიზონტალური სტრიქონის ხატვის დემონსტრირება.

```
{
//      პროგრამა 16.7
//      პროგრამაში ხდება ჰორიზონტალური სტრიქონის ხატვა
float x = 10.0f;
float y = 10.0f;
string DasaxatiStriqoni = "ანა და საბა";
//  სახატავი ფურცლის შექმნა
Graphics SaxataviPurceli = CreateGraphics();
//      შრიფტის განსაზღვრა სტრიქონისთვის
Font Shrifti = new Font("AcadNusx", 14);
//      ფუნჯის შექმნა
SolidBrush Punji = new SolidBrush(Color.Black);
//  ჰორიზონტალური სტრიქონის ხატვა
SaxataviPurceli.DrawString(DasaxatiStriqoni, Shrifti, Punji, x, y);
//  რესურსების გათავისუფლება
Shrifti.Dispose();
Punji.Dispose();
SaxataviPurceli.Dispose();
}
```

მოყვანილ პროგრამაში ხდება ვერტიკალური სტრიქონის ხატვის დემონსტრირება.

```
{
//      პროგრამა 16.8
//      პროგრამაში ხდება ვერტიკალური სტრიქონის ხატვა
```

```
float x = 10.0f;
float y = 10.0f;
string DasaxatiStriqoni = "საბა და ანა";
// სახატავი ფურცლის შექმნა
Graphics SaxataviPurceli = this.CreateGraphics();
// შრიფტის განსაზღვრა სტრიქონისთვის
Font Shrifti = new Font("AcadNusx", 14);
// ფუნჯის შექმნა
SolidBrush Punji = new SolidBrush(Color.Black);
// სტრიქონისათვის ვერტიკალური მიმართულების არჩევა
StringFormat StriqonisFormati = new
                                        StringFormat(StringFormatFlags.DirectionVertical);
// ვერტიკალური სტრიქონის ხატვა
SaxataviPurceli.DrawString(DasaxatiStriqoni, Shrifti, Punji,
                                    x, y, StriqonisFormati);
// რესურსების გათავისუფლება
Shrifti.Dispose();
Punji.Dispose();
SaxataviPurceli.Dispose();
}
```

გამოსახულებასთან მუშაობა

გამოსახულებასთან სამუშაოდ გამოიყენება Bitmap ობიექტი. იგი მუშაობს bmp, gif, jpg, png და tif ფაილებთან. GDI+ ინტერფეისი ავტომატურად ცვლის გამოსახულების ზომას ისე, რომ ის მოთავსთდეს ხატვის უბანში. ცხრილებში 16.10 და 16.11 მოყვანილია Bitmap კლასის თვისებები და მეთოდები.

ცხრილი 16.10. Bitmap კლასის თვისებები.

თვისება	ტიპი	აღწერა
Flags	int	ალამი-ატრიბუტების ნაკრები
FrameDimensionsList	Guid [ ]	მასივი, რომელიც მიუთითებს გამოსახულებაში კადრების განზომილებებს
Height	int	გამოსახულების სიმაღლე მასშტაბირების გარეშე
HorizontalResolution	float	ჰორიზონტალური გარჩევის უნარი პიქსე-ლი/დუიმზე
Palette	ColorPalette	ფერების პალიტრა
PhysicalDimension	Size	გამოსახულების სიგანე და სიმაღლე
PixelFormat	PixelFormat	გამოსახულების პიქსელების ფორმატი
PropertyIDList	int [ ]	თვისებების იდენტიფიკატორების მასივი, რომლებიც გამოსახულებასთან ერთადაა შენახული
PropertyItems	PropertyItem [ ]	PropertyItem ობიექტების მასივი, რომელიც მოცემულ გამოსახულებას აღწერს

ცხრილი 16.10 (გაგრძელება)

RawFormat	ImageFormat	გამოსახულების ფორმატი
Size	Size	გამოსახულების სიგანე და სიმაღლე
VerticalResolution	float	ვერტიკალური გარჩევის უნარი პიქსე-ლი/დუიმზე
Width	int	გამოსახულების სიგანე მასშტაბირების გარეშე

ცხრილი 16.11. Bitmap კლასის მეთოდები.

მეთოდი	აღწერა
Clone() (სტატიკური)	ქმნის Bitmap ობიექტის ასლს
FromHicon()	ქმნის Bitmap ობიექტს Windows-ის პიქტოგრამის დესკრიპტორიდან
FromResource()	ქმნის Bitmap ობიექტს Windows-ის რესურსიდან
GetBounds()	გასცემს სწორკუთხა უბანს, რომელშიც უნდა მოთავსთდეს გამოსახულება
GetEncoderParameterList()	გასცემს ინფორმაციას გამოსახულების შენახვის პარამეტრების შესახებ
GetFrameCount()	გასცემს გამოსახულებაში კადრების რაოდენობას
GetHbitmap()	გასცემს Windows-ის ბიტების კარტის დესკრიპტორს მოცემული გამოსახულებისათვის
GetHicon()	გასცემს Windows-ის პიქტოგრამის დესკრიპტორს მოცე-მული გამოსახულებისათვის
GetPixel()	გასცემს გამოსახულების მითითებული პიქსელის ფერს
GetPropertyItem()	გასცემს PropertyItem ობიექტს
GetThumbnailImage()	გასცემს მოცემული გამოსახულების პიქტოგრამას
LockBits()	გამოსახულებას აფიქსირებს ოპერატიულ მეხსიერებაში
MakeTransparent()	გამოსახულების ნაწილებს გამჭვირვალეს ხდის
RemovePropertyItem()	მოცემული გამოსახულებიდან შლის PropertyItem ობიექტს
RotateFlip()	გამოსახულებს აბრუნებს ან სარკისებურად ასახავს
Save()	გამოსახულებას ნაკადში ინახავს
SaveAdd()	ახდენს ორი გამოსახულების კომბინირებას
SelectActiveFrame()	მრავალკადრიანი გამოსახულებიდან ირჩევს ერთ კადრს
SetPixel()	აყენებს გამოსახულების მითითებული პიქსელის ფერს
SetPropertyItem()	აყენებს ობიექტის მნიშვნელობას
SetResolution()	აყენებს გამოსახულების გარჩევის უნარს
UnlockBits()	აუქმებს ოპერატიულ მეხსიერებაში გამოსახულების ფიქსაციას

Bitmap კლასის კონსტრუქტორს აქვს 12 გადატვრთული ვერსია. ჩვენ მოვიყვანთ ორი მათგანის სინტაქსს:
Bitmap(string *ფაილის_სახელი);*

Bitmap(int *სიგანე,* int *სიმაღლე);*

მოყვანილ პროგრამაში ხდება მითითებული ფაილიდან გამოსახულების ასახვის დემონსტრირება.

```
{
//      პროგრამა 16.9
//      პროგრამაში ხდება გამოსახულების ასახვა მითითებული ფაილიდან
Graphics SaxataviPurceli = CreateGraphics();
Bitmap Gamosaxuleba2 = new Bitmap("ani2.jpg");
SaxataviPurceli.DrawImage(Gamosaxuleba2, 5, 5, 250, 250);
}
```

მოყვანილ პროგრამაში ხდება დიალოგურ ფანჯარაში არჩეული ფაილიდან გამოსახულების ასახვის დემონსტრირება.

```
{
//      პროგრამა 16.10
//      პროგრამაში ხდება გამოსახულების ასახვა დიალოგურ ფანჯრაში არჩეული ფაილიდან
Graphics SaxataviPurceli = CreateGraphics();
OpenFileDialog Surati_Failidan = new OpenFileDialog();
Bitmap Gamosaxuleba;
if ( Surati_Failidan.ShowDialog() == DialogResult.OK )
{
  Gamosaxuleba = new Bitmap(Surati_Failidan.FileName);
  SaxataviPurceli.DrawImage(Gamosaxuleba, 5, 5, 250, 250);
}
}
```

SetPixel() მეთოდი გამოიყენება მითითებული პიქსელისთვის (მონიტორის ეკრანის წერტილისთვის) ფერის მისანიჭებლად. მისი სინტაქსია:

void SetPixel(int x, int y, Color *ფერი);*

მოყვანილ პროგრამაში ხდება SetPixel() მეთოდის გამოყენებით გამოსახულების ასახვის დემონსტრირება.

```
{
//      პროგრამა 16.11
//      პროგრამაში ხდება გამოსახულების ასახვა SetPixel() მეთოდის გამოყენებით
Graphics SaxataviPurceli = CreateGraphics();
Bitmap Gamosaxuleba1 = new Bitmap(90, 90);              // სიგანე = სიმაღლე = 90
//      თეთრი ოთხკუთხედის ხატვა
for ( int x = 0; x < Gamosaxuleba1.Height; ++x )
   for ( int y = 0; y < Gamosaxuleba1.Width; ++y )
      Gamosaxuleba1.SetPixel(x, y, Color.White);
//  წითელი ხაზის ხატვა
for ( int x = 0; x < Gamosaxuleba1.Height; ++x )
   Gamosaxuleba1.SetPixel(x, x, Color.Red);
//  გამოსახულების გამოტანა
SaxataviPurceli.DrawImage(Gamosaxuleba1, 0, 0, 100, 100);
}
```

მონაცემების დაშიფვრა და გაშიფვრა

მონაცემების დაცვის ერთ-ერთი სახეა მათი დაშიფვრა. .NET გარემო შეიცავს კლასებს, რომლებიც უზრუნველყოფენ როგორც სიმეტრიულ, ისე ასიმეტრიულ კრიპტოგრაფიას. სიმეტრიულ კრიპტოგრაფიას, ზოგჯერ უწოდებენ კრიპტოგრაფიას დახურული გასაღებით, ხოლო ასიმეტრიულ კრიპტოგრაფიას კი - კრიპტოგრაფიას ღია გასაღებით.

სიმეტრიული კრიპტოგრაფიის შემთხვევაში მონაცემების დაშიფვრისა და გაშიფვრისათვის ერთი და იგივე გასაღები გამოიყენება. .NET გარემოში გამოყენებულია სიმეტრიული კრიპტოგრაფიის ოდნავ გართულებული ფორმა, რომელსაც დაშიფრული ბლოკების გადაბმა ეწოდება. მონაცემების გასაშიფრად ამ ფორმის გამოყენების შემთხვევაში უნდა ვიცოდეთ არა მხოლოდ გასაღები, არამედ მაინიაცილებელი ვექტორიც. სიმეტრიული კრიპტოგრაფიის მიმართულებით შექმნილია ბევრი ალგორითმი, რომლებიც ორიენტირებულია იმაზე, რომ რთული იყოს გასაღების გარეშე მონაცემების გაშიფვრა. .NET გარემოში გამოყენებულია სიმეტრიული კრიპტოგრაფიის ოთხი ალგორითმი: DES, RC2, Rijndael და Triple DES.

ასიმეტრიული კრიპტოგრაფიის შემთხვევაში მონაცემების დაშიფვრისა და გაშიფვრისათვის გამოიყენება ორი გასაღები: ღია და დახურული. ღია გასაღების გამოყენებით დაშიფრული მონაცემების გაშიფვრა შესაძლებელია მხოლოდ შესაბამისი დახურული გასაღებით. .NET გარემოში გამოყენებულია ასიმეტრიული კრიპტოგრაფიის ორი ალგორითმი: RSA და DSA.

თუ გასაღები არ გვაქვს, მაშინ ასიმეტრიული შიფრის “გატეხვა” უფრო რთულია, ვიდრე სიმეტრიულის. “გატეხვა” არის პროცესი, როცა გასაღები არ გვაქვს და სხვადასხვა ხერხებით ვცდილობთ დაშიფრული მონაცემების გაშიფვრას. გარდა ამისა, მონაცემების ასიმეტრიული დაშიფვრა მოითხოვს უფრო მეტ გამოთვლებს, ვიდრე სიმეტრიული. კრიპტოგრაფიის კლასები მოთავსებულია System.Security.Cryptography სახელების სივრცეში.

სიმეტრიული კრიპტოგრაფია

სიმეტრიული კრიპტოგრაფიისათვის გამოიყენება TripleDESCryptoServiceProvider კლასი. მისი თვისებები მოყვანილია ცხრილში 16.12, მეთოდები კი - ცხრილში 16.13.

მონაცემების სიმეტრიული დაშიფვრისათვის გამოიყენება ნაკადები. დაშიფვრა ხორციელდება CryptoStream ობიექტის საშუალებით. რადგან ერთი ნაკადის გამოსასვლელი შეიძლება მივაწოდოთ მეორე ნაკადის შესასვლელს, ამიტომ CryptoStream კლასი შეგვიძლია გამოვიყენოთ FileStream კლასთან ერთად ფაილში დაშიფრული მონაცემების ჩასაწერად და წასაკითხად, აგრეთვე, NetworkStream კლასთან ერთად ქსელში გადასაცემი მონაცემების დასაშიფრად.

ცხრილი 16.12. TripleDESCryptoServiceProvider კლასის თვისებები

თვისება	ტიპი	აღწერა
BlockSize	Int	დასაშიფრი ან გასაშიფრი ბაიტების რაოდენობა ერთი ოპერაციის შესრულებისას
FeedbackSize	Int	მონაცემების რაოდენობა თითოეული ბლოკიდან, რომლებიც გამოიყენება მომდევნო ბლოკის შიფრაციისათვის
IV	Byte	მაინიციალიზებელი ვექტორი მოცემული ალგორითმისთვის
Key	Byte	დაშიფვრის გასაღები მოცემული ალგორითმისთვის
KeySize	Int	გასაღებში ბიტების რაოდენობა
LegalBlockSizes	KeySizes[]	ბლოკების ზომები, რომლებსაც მოცემული ალგორითმი უზრუნველყოფს
LegalKeySizes	KeySizes[]	გასაღებების ზომები, რომლებსაც მოცემული ალგორითმი უზრუნველყოფს
Mode	CipherMode	განსაზღვრავს სიმეტრიული ალგორითმის ოპერაციის რეჟიმს
Padding	PaddingMode	განსაზღვრავს ბაიტებს, რომლებიც დაემატება უკანასკნელ ბლოკს, რომელსაც არ აქვს საკმარისი სიგრძე

ცხრილი 16.13. TripleDESCryptoServiceProvider კლასის მეთოდები

მეთოდი	აღწერა
Clear()	ათავისუფლებს ალგორითმის მიერ დაკავებულ რესურსებს
CreateDecryptor()	ქმნის ობიექტს, რომელიც შეიძლება გამოყენებული იყოს გაშიფვრის ოპერაციის შესასრულებლად
CreateEncryptor()	ქმნის ობიექტს, რომელიც შეიძლება გამოყენებული იყოს დაშიფვრის ოპერაციის შესასრულებლად
GenerateIV()	ახდენს ახალი შემთხვევითი მაინიციალიზებელი ვექტორის გენერირებას
GenerateKey()	ახდენს ახალი შემთხვევითი გასაღების გენერირებას
ValidateKeySize()	განსაზღვრავს, აქვს თუ არა მიმდინარე გასაღებს კორექტული ზომა

მოყვანილი პროგრამა ასრულებს ფაილში ჩასაწერი მონაცემების დაშიფვრას.

```
{
//      პროგრამა 16.11
//      პროგრამაში ხდება ფაილში ჩასაწერი მონაცემების დაშიფვრა
FileStream SafailoNakadi1 = File.Create("temp.txt");
// ახალი კრიპტოგრაფიული პროვაიდერის შექმნა
TripleDESCryptoServiceProvider CriptoProvaideri = new TripleDESCryptoServiceProvider();
// დამშიფრავი ნაკადის შექმნა საფაილო ნაკადში
CryptoStream CriptoNakadi = new CryptoStream(SafailoNakadi1,
```

```
CriptoProvaideri.CreateEncryptor(), CryptoStreamMode.Write);
StreamWriter ChawerisNakadi = new StreamWriter(CriptoNakadi);
ChawerisNakadi.WriteLine("საბა და ანა");
ChawerisNakadi.WriteLine("ლიკა და რომანი");
ChawerisNakadi.Close();
//      გასაღებისა და მაინიციალიზებელი ვექტორის შენახვა
FileStream SafailoNakadi2  = File.Create("encrypted.key");
BinaryWriter OrobitiNakadi = new BinaryWriter(SafailoNakadi2);
OrobitiNakadi.Write(CriptoProvaideri.Key);
OrobitiNakadi.Write(CriptoProvaideri.IV);
OrobitiNakadi.Close();
}
```

პროგრამაში ჯერ იქმნება TripleDESCryptoServiceProvider კლასის ობიექტი მონაცემების დამშიფრავი სამსახურის უზრუნველყოფისათვის. შემდეგ იქმნება CryptoStream კლასის ობიექტი, რომელიც მოახდენს მონაცემების დაშიფვრას. CryptoStream კლასის კონსტრუქტორს სამი პარამეტრი გადაეცემა. პირველია ობიექტი-ნაკადი, რომელიც გამოიყენება ფაილში მონაცემების ჩასაწერად ან წასაკითხად. მეორე პარამეტრია მეთოდი, რომელიც ახდენს მონაცემების დაშიფვრას ან გაშიფვრას. მესამე პარამეტრია რეჟიმი, რომელიც განსაზღვრავს CryptoStream ობიექტი კითხულობს თუ წერს მონაცემებს.

პროგრამაში StreamWriter ობიექტიდან მონაცემები გადაეცემა CryptoStream ობიექტს, რომელიც ახდენს ამ მონაცემების დაშიფვრას, შემდეგ კი დაშიფრული მონაცემები გადაეცემა FileStream ობიექტს, რომელიც ამ მონაცემებს ფაილში ჩაწერს. ბოლოს პროგრამა ფაილში წერს გასაღებსა და მაინიაციალიზებელ ვექტორს.

მოყვანილი პროგრამა ასრულებს ფაილიდან წაკითხული მონაცემების გაშიფვრას.

```
{
//      პროგრამა 16.12
//      პროგრამაში ხდება ფაილიდან წაკითხული მონაცემების გაშიფვრა
TripleDESCryptoServiceProvider CriptoProvaideri = new
                                                  TripleDESCryptoServiceProvider();
FileStream   SafailoNakadi1   = File.OpenRead("encrypted.key");
BinaryReader OrobitiWamkitxavi = new BinaryReader(SafailoNakadi1);
//      ფაილიდან გასაღებისა და ვექტორის წაკითხვა
CriptoProvaideri.Key = OrobitiWamkitxavi.ReadBytes(24);
CriptoProvaideri.IV  = OrobitiWamkitxavi.ReadBytes(8);
FileStream   SafailoNakadi2 = File.OpenRead("temp.txt");
//      ფაილიდან წაკითხული მონაცემების გაშიფვრა
CryptoStream CriptoNakadi = new CryptoStream(SafailoNakadi2,
      CriptoProvaideri.CreateDecryptor(), CryptoStreamMode.Read);
StreamReader WakitxvisNakadi = new StreamReader(CriptoNakadi);
label1.Text = WakitxvisNakadi.ReadToEnd().ToString() + '\n' +
                "გასაღების ზომა = " + CriptoProvaideri.KeySize.ToString() +
                '\n' + "ბლოკის ზომა = " + CriptoProvaideri.BlockSize.ToString();
label2.Text = "რეჟიმი = " + CriptoProvaideri.Mode.ToString();
WakitxvisNakadi.Close();
```

}

პროგრამაში ხდება დაშიფრული მონაცემების წაკითხვა FileStream ობიექტის მიერ, მათი გაშიფვრა CryptoStream ობიექტის მიერ, შემდეგ კი მათი გადაცემა StreamReader ობიექტისათვის და ეკრანზე გამოტანა.

ასიმეტრიული კრიპტოგრაფია

ასიმეტრიული კრიპტოგრაფიის ალგორითმები განსაზღვრულია არა ნაკადებთან, არამედ, მცირე ზომის მონაცემებთან სამუშაოდ. მოყვანილ პროგრამაში ხდება მონაცემების დაშიფვრისა და გაშიფვრის დემონსტრირება ასიმეტრიული კრიპტოგრაფიის მეთოდების გამოყენებით.

```
{
//      პროგრამა 16.13
//      პროგრამაში ხდება მონაცემების დაშიფვრისა და გაშიფვრის დემონსტრირება
//      ასიმეტრიული კრიპტოგრაფიის მეთოდების გამოყენებით
label1.Text = "";
// ახალი კრიპტოგრაფიული პროვაიდერის შექმნა
RSACryptoServiceProvider CriptoProvaideri = new RSACryptoServiceProvider();
Byte[] DasashifriMasivi = { 1, 2, 3, 4, 5 };
// მონაცემების შიფრაცია
Byte[] DashifruliMasivi = CriptoProvaideri.Encrypt(DasashifriMasivi, false);
label1.Text = "დაშიფრული მონაცემები: \n";
for ( int indexi = 0; indexi < DashifruliMasivi.GetLength(0); indexi++ )
    label1.Text += DashifruliMasivi[indexi].ToString() + "   ";
label1.Text += '\n';
// მონაცემების დეშიფრაცია
Byte[] GashifruliMasivi = CriptoProvaideri.Decrypt(DashifruliMasivi, false);
label1.Text += "გაშიფრული მონაცემები: \n";
for ( int indexi = 0; indexi < GashifruliMasivi.GetLength(0); indexi++ )
   label1.Text += GashifruliMasivi[indexi].ToString() + "   ";
label2.Text += "ალგორითმის სახელი - " +
CriptoProvaideri.KeyExchangeAlgorithm.ToString() + '\n'
       + "გასაღების ზომა = "  + CriptoProvaideri.KeySize.ToString() + '\n';
}
```

III ნაწილი. მონაცემთა ბაზები

თავი 17. შესავალი მონაცემთა ბაზებში

მონაცემთა ბაზების აგების პრინციპები

მონაცემთა ბაზები გამოიყენება ყოველთვის, როცა საჭიროა მონაცემების დიდ მასივებთან მუშაობა. მონაცემთა ბაზა არის ცხრილების ერთობლიობა. ცხრილი არის ჩანაწერების (სტრიქონების) ერთობლიობა (ცხრილი 17.1). ჩანაწერი არის ველების (სვეტების) ერთობლიობა. ველი შეიძლება შეიცავდეს სხვადასხვა ტიპის მონაცემებს, როგორიცაა მთელი და წილადი რიცხვები, სტრიქონები, ლოგიკური მონაცემები, ნახატები, თარიღი და დრო და ა.შ.

ცხრილის თითოეული ჩანაწერი შეესაბამება ერთ ობიექტს და შეიცავს ინფორმაციას ამ ობიექტის შესახებ. ცხრილის ველები შეესაბამებიან ობიექტის ამა თუ იმ მახასიათებლებს (თვისებებს). ველს აქვს სახელი და მასში მოთავსებული მონაცემის ტიპი. ველის სახელი არის იდენტიფიკატორი, რომელიც პროგრამაში გამოიყენება მონაცემებთან სამუშაოდ, უნდა იწყებოდეს ასოთი და შეიცავდეს ასოებს, ციფრებსა და ხაზგასმის სიმბოლოს.

როგორც წესი, მონაცემთა ბაზაში თითოეულ ცხრილს აქვს ერთი ან მეტი ველი, რომელიც ცალსახად განსაზღვრავს თითოეულ ჩანაწერს ცხრილში. ამ ველს (ან ველების ჯგუფს) ცხრილის პირველადი გასაღები ეწოდება. პირველადი გასაღების მნიშვნელობა ცხრილის თითოეული ჩანაწერისათვის უნდა იყოს უნიკალური. სწორედ პირველადი გასაღები გამოიყენება მონაცემთა ბაზაში ინფორმაციის არაწინააღმდეგობრიობის უზრუნველსაყოფად.

ხშირად საჭირო ხდება ცხრილებს შორის ბმების (კავშირების) დამყარება. ამ შემთხვევაში ერთი ცხრილი არის მთავარი (მშობელი), მეორე კი დეტალების (შვილობილი). როცა მთავარ ცხრილში ავირჩევთ რომელიმე ჩანაწერს, დეტალების ცხრილიდან ამოირჩევა შესაბამისი ჩანაწერები (ნახ. 17.1). ეს ორი ცხრილი ერთმანეთს გარე გასაღების საშუალებით უკავშირდება. ცხრილს, რომელიც გარე გასაღებს შეიცავს დეტალების ცხრილი ეწოდება, ხოლო ცხრილს, რომელსაც გარე გასაღები მიმართავს - მთავარი ცხრილი. ამრიგად, დეტალების ცხრილის გარე გასაღები უკავშირდება მთავარი ცხრილის პირველად გასაღებს. კავშირს განხორციელებულს გარე გასაღების საშუალებით ეწოდება “მთავარი-დეტალი” ან “მშობელი-შვილი”. ცხრილებს შორის კავშირის ასეთ ტიპს ეწოდება “ერთი-ბევრთან”.

Produkti ცხრილში პირველადი გასაღებია Id ველი. Gakidvebi ცხრილში გარე გასაღებია Id_Produkti ველი (ნახ. 17.1). Gakidvebi ცხრილის Id_Produkti ველი უკავშირდება Produkti ცხრილის Id ველს.

ბმული ცხრილების ერთობლიობას მონაცემთა რელაციური ბაზა ეწოდება.

მონაცემთა ბაზასთან მუშაობა ხორციელდება ბაზების მართვის სისტემის (მბმს) საშუალებით. იგი არის პროგრამული უზრუნველყოფა, რომელიც მართავს მონაცემთა ბაზაში ინფორმაციის ჩამატებას, შეცვლას, წაშლას და ა.შ. ესენია: Paradox, dBase, FoxPro, Microsoft Access, InterBase, SyBase, Oracle, Microsoft SQL და ა.შ. სხვადასხვა მბმს-ები სხვადასხვანაირად ინახავენ მონაცემთა ბაზებს. მაგალითად, Paradox და dBase ცხრილს

ერთ ფაილში ინახავენ, მონაცემთა ბაზას კი კატალოგში. ამ შემთხვევაში მონაცემთა ბაზა არის კატალოგი. Microsoft Access და InterBase რამდენიმე ცხრილს ერთ ფაილში ინახავენ. ამ შემთხვევაში მონაცემთა ბაზა არის ფაილი. კლიენტ/სერვერი ტიპის სისტემები, როგორიცაა Oracle, SyBase, Microsoft SQL მონაცემებს ცალკე კომპიუტერზე ინახავენ და კლიენტთან ურთიერთობა აქვთ SQL (Structured Query Language, სტრუქტურირებული მოთხოვნების ენა) ენის საშუალებით.

მონაცემთა ბაზის სქემა ეწოდება მონაცემთა სტრუქტურას, რომელიც შეიცავს იმ ცხრილებისა და ველების აღწერას, რომლებისგანაც მონაცემთა ბაზა შედგება.

ცხრილი 17.1. Produkti.

Id	Dasaxeleba	Tarigi		Darcha	Fasi	
1	კალამი	20/03/03		73	0.50	I ჩანაწერი
2	საშლელი	29/12/03		145	0.20	II ჩანაწერი
3	ფანქარი	15/11/03		111	0.45	
4	მელანი (შავი)	05/04/03		92	0.60	.
5	რვეული	21/05/03		204	1.00	.
6	ფაილი	01/02/03	. . .	26	3.00	.
7	ფანქარი	11/01/03		204	0.50	
8	ჭიკარტი	17/07/03		389	1.20	
9	ფურცელი A4	27/08/03		18	6.50	
10	ალბომი	16/09/03		90	2.20	
11	საშლელი	13/06/03		135	0.30	
. . .	. . .	. . .		. . .	. . .	i-ური ჩანაწერი
I ველი	II ველი	III ველი			j-ური ველი	

მონაცემთა ბაზების ტიპები

არსებობს მონაცემთა ბაზების ოთხი მოდელი:

- ავტონომური;
- ფაილ/სერვერი;
- კლიენტ/სერვერი;
- მრავალდონიანი.

ავტონომური მონაცემთა ბაზა ინახება ლოკალურ კომპიუტერზე. ამავე კომპიუტერზე ინახება მონაცემთა ბაზების მართვის სისტემა. ქსელი არ გამოიყენება. მომხმარებელს არ სჭირდება სხვა კომპიუტერზე მოთავსებულ მონაცემთა ბაზებთან მიმართვა.

ფაილ/სერვერული მონაცემთა ბაზები ქსელური მონაცემთა ბაზებია. თითოეული კლიენტისათვის იქმნება მონაცემების ლოკალური ასლი, რომელთანაც ის მუშაობს. ამ დროს რამდენიმე მომხმარებელი ერთდროულად მუშაობს ერთ მონაცემთა ბაზასთან. ასეთი მონაცემთა ბაზების ნაკლია არამწარმოებლური დატვირთვა. კლიენტის მხრიდან ყოველი მიმართვისას მის ლოკალურ ასლში სრულდება მონაცემების გაახლება მონაცემთა ბაზიდან, რომელიც სერვერზეა მოთავსებული. იმ შემთხვევაშიც კი, როცა

მოთხოვნა ერთ ჩანაწერს ეხება, სრულდება ყველა ჩანაწერის გაახლება. თუ მონაცემთა ბაზაში ბევრი ჩანაწერია და ცოტა კლიენტი, ქსელი მაინც ზედმეტად იქნება დატვირთული, რაც მნიშვნელოვნად აქვეითებს მოთხოვნების შესრულების სიჩქარეს. მეორე ნაკლია ის, რომ მონაცემების მთლიანობა უნდა უზრუნველყოს კლიენტის პროგრამამ.

მონაცემთა ბაზებისთვის, რომლებთანაც ბევრი მომხმარებელი მუშაობს გამოიყენება კლიენტ/სერვერული მონაცემთა ბაზები. ამ დროს მონაცემთა ბაზასთან მიმართვა ხორციელდება სერვერის საშუალებით. კლიენტი სერვერს აძლევს დავალებას შეასრულოს ძებნის ან გაახლების ოპერაცია. სერვერი (ესაა მძლავრი კომპიუტერი) ასრულებს ამ მოთხოვნებს და კლიენტს უგზავნის შედეგს. მუშაობის ასეთი ორგანიზაცია ზრდის პროგრამების შესრულების ეფექტურობას, განტვირთავს ქსელს და უზრუნველყოფს მონაცემების მთლიანობას.

მრავალდონიანი მონაცემთა ბაზები არის ქსელში მონაცემების დამუშავების ახალი პერსპექტიული გზა. მას ზოგჯერ მონაცემთა ბაზების მრავალგანშტოებიან ორგანიზაციას (multitier) უწოდებენ. გავრცელებულია მისი სამდონიანი ვარიანტი:

პირველ (ქვედა) დონეზე მომხმარებლების კომპიუტერებზე მოთავსებულია კლიენტების პროგრამები, რომლებიც უზრუნველყოფენ სამომხმარებლო ინტერფეისს.

მეორე დონეზე მოთავსებულია პროგრამების სერვერი, რომელიც უზრუნველყოფს მონაცემების გაცვლას მომხმარებლებსა და განაწილებულ მონაცემთა ბაზას შორის.

მესამე დონეზე მოთავსებულია მონაცემთა ბაზების დაშორებული სერვერი, რომელიც იღებს ინფორმაციას პროგრამების სერვერისგან და მართავს მათ.

ესაა მონაცემთა ბაზების ყველაზე რთული და მოქნილი ორგანიზაცია.

შენახული პროცედურა ესაა პროცედურა, რომელიც ინახება და სრულდება მონაცემთა ბაზაში. ენა, რომელზეც იწერება შენახული პროცედურა დამოკიდებულია მონაცემთა ბაზების მართვის სისტემის მწარმოებელზე. მაგალითად, SQL-სერვერზე შენახული პროცედურები იწერება Transact_SQL ენაზე.

ცხრილი Produkti.

Id	Dasaxeleba	Tarigi_Migebis	Mivige	Darcha	Fasi
10	ალბომი	16/09/03	130	90	2.20
6	ფაილი	01/02/03	50	26	3.00
7	ფანქარი	11/01/03	300	204	0.50
3	ფანქარი	15/11/03	180	111	0.45
9	ფურცელი (A4)	27/08/03	70	18	6.50
1	კალამი	20/03/03	100	73	0.50
. . .	. . .	. . .	. . .	. . .	. . .

ცხრილი Gakidvebi.

Id	Id_Produkti	Dasaxeleba	Tarigi_ gakidvis	Raodenoba	Tanxa
1	10	ალბომი	17/09/03	2	4.40
2	10	ალბომი	20/09/03	5	11.00
3	10	ალბომი	23/09/03	1	2.20

ნახ. 17.1.

თავი18. SQL-სერვერი

SQL-მოთხოვნის ტიპები

SQL (Structured Query Language, სტრუქტურული მოთხოვნების ენა) არის სტანდარტული ენა მონაცემთა რელაციურ ბაზებთან სამუშაოდ. SQL-მოთხოვნა არის დავალება (შეკითხვა) SQL-სერვერისადმი, რომელიც ასრულებს ამ დავალებას და აბრუნებს შესაბამის ჩანაწერებს.

არსებობს SQL-მოთხოვნების ბევრი ტიპი, მაგრამ ყველაზე ხშირად გამოიყენება შემდეგი SQL-მოთხოვნები:

მონაცემების დამუშავების ენის მოთხოვნები (Data Manipulation Language - DML).

მონაცემების განსაზღვრის ენის მოთხოვნები (Data Definition Language - DDL).

DML-მოთხოვნები საშუალებას გვაძლევენ ამოვიღოთ, დავუმატოთ, შევცვალოთ ან წავშალოთ ჩანაწერები მონაცემთა ბაზების ცხრილებში. არსებობს DML-მოთხოვნის ოთხი ტიპი:

SELECT მოთხოვნა გამოიყენება ერთი ან მეტი ცხრილიდან ჩანაწერების მისაღებად.

INSERT მოთხოვნა გამოიყენება ცხრილში ერთი ან მეტი ჩანაწერის დასამატებლად.

UPDATE მოთხოვნა გამოიყენება ცხრილში ერთი ან მეტი არსებული ჩანაწერის შესაცვლელად.

DELETE მოთხოვნა გამოიყენება ცხრილიდან ერთი ან მეტი ჩანაწერის წასაშლელად.

DDL-მოთხოვნები საშუალებას გვაძლევენ მონაცემთა ბაზებში შევქმნათ სტრუქტურები, მაგალითად, ცხრილები.

DML-მოთხოვნების დაწერის საფუძვლების შესწავლამდე აუცილებელია ვიცოდეთ თუ როგორ უნდა შევიტანოთ და შევასრულოთ SQL-მოთხოვნები. SQL-სერვერთან მუშაობისას ეს შეგვიძლია გავაკეთოთ Query Analyzer უტილიტის (პროგრამის) საშუალებით.

SQL-მოთხოვნების ასაგებად შეგვიძლია, აგრეთვე, გამოვიყენოთ მოთხოვნების კონსტრუქტორი - Query Builder. ესაა გრაფიკული ინსტრუმენტი, რომელიც საშუალებას გვაძლევს ვიზუალურად შევქმნათ SQL-მოთხოვნები.

Query Analyzer უტილიტთან მუშაობა

Query Analyzer უტილიტი შეგვიძლია გამოვიყენოთ SQL-მოთხოვნების შესატანად და შესასრულებლად. მისი გაშვება შემდეგნაირად ხდება: Start→All Programs→Microsoft SQL Server→ Query Analyzer.

SQL-სერვერთან შეერთება.

Query Analyzer უტილიტის გაშვებისას გამოჩნდება SQL-სერვერთან შეერთების დიალოგური ფანჯარა (ნახ. 18.1). SQL Server გაშლადი სიიდან ვირჩევთ SQL-სერვერის სახელს.

თუ ჩავრთავთ Windows authentication (Windows-ის ავტორიზაცია) გადამრთველს, მაშინ SQL-სერვერი შეერთების ავტორიზაციისათვის გამოიყენებს Windows 2000/NT/XP/2003-ის მიმდინარე მომხმარებლის ინფორმაციას. თუ ჩავრთავთ

SQL Server authentication (ავტორიზაცია SQL-სერვერის საშუალებებით) გადამრთველს, მაშინ აუცილებელია მომხმარებლის სახელისა და პაროლის შეტანა. Login name ველში უნდა შევიტანოთ sa (system administrator) სახელი, Password ველში კი პაროლი, რომელიც შეტანილი იყო SQL-სერვერის ინსტალირების დროს.

თუ SQL Server ველში შევიტანთ localhost მნიშვნელობას, მაშინ იგი მიუთითებს იმაზე, რომ SQL-სერვერი მუშაობს ლოკალურ კომპიუტერზე.

ნახ. 18.1.

SQL-მოთხოვნების შეტანა და შესრულება.

როგორც კი Query Analyzer უტილიტის საშუალებით დავუკავშირდებით SQL-სერვერს შეგვიძლია გამოვიყენოთ Object Browser უტილიტი მონაცემთა ბაზის ნაწილების სანახავად, აგრეთვე, Query ფანჯარაში SQL-მოთხოვნების შესატანად და მათ შესასრულებლად. ნახ. 18.2-ზე ნაჩვენებია Object Browser (ობიექტების ბრაუზერი) ფანჯარა და Query ფანჯარა მოთხოვნის შედეგით, რომელიც აბრუნებს Northwind მონაცემთა ბაზის Customers ცხრილის CustomerID და CompanyName ველების მნიშვნელობებს. Northwind მონაცემთა ბაზა ინახება Microsoft SQL-სერვერზე და შეიცავს ინფორმაციას ვირტუალური Northwind კომპანიის შესახებ, რომელიც კვების პროდუქტებს ყიდის.

SQL-მოთხოვნის შეტანა ხდება Query ფანჯრის ზედა ნაწილში, ხოლო მოთხოვნის შედეგი კი აისახება ფანჯრის ქვედა ნაწილში. მონაცემთა მუშა ბაზა ეთითება USE მოთხოვნის საშუალებით, ხოლო ჩანაწერების ამოღება მონაცემთა ბაზიდან სრულდება SELECT მოთხოვნის საშუალებით (ნახ. 18.2). მაგალითისათვის შევიტანოთ USE-მოთხოვნა:

```
USE Northwind
```

ეს მოთხოვნა მიუთითებს იმაზე, რომ ჩვენ გვინდა Northwind მონაცემთა ბაზის გამოყენება. მომდევნო სტრიქონში შევიტანოთ SELECT-მოთხოვნა (მას მოგვიანებით განვიხილავთ):

```
SELECT CustomerID, CompanyName FROM Customers;
```

ეს მოთხოვნა მიუთითებს იმაზე, რომ ჩვენ გვინდა მივიღოთ Customers ცხრილის CustomerID და CompanyName ველების მნიშვნელობები.

SQL ენაში სიმბოლოების რეგისტრი არ არის განსხვავებული. ამიტომ,

დარეზერვებული სიტყვების, ცხრილებისა და ველების სახელებისათვის შეგვიძლია გამოვიყენოთ როგორც ზედა, ისე ქვედა რეგისტრის სიმბოლოები. ჩვენ ზედა რეგისტრის სიმბოლოებს დარეზერვებული სიტყვებისათვის გამოვიყენებთ, ხოლო შერეული რეგისტრის სიმბოლოებს ცხრილებისა და ველების სახელებისთვის.

Query ფანჯარაში შეტანილი SELECT-მოთხოვნა შეგვიძლია ოთხი გზით შევასრულოთ:

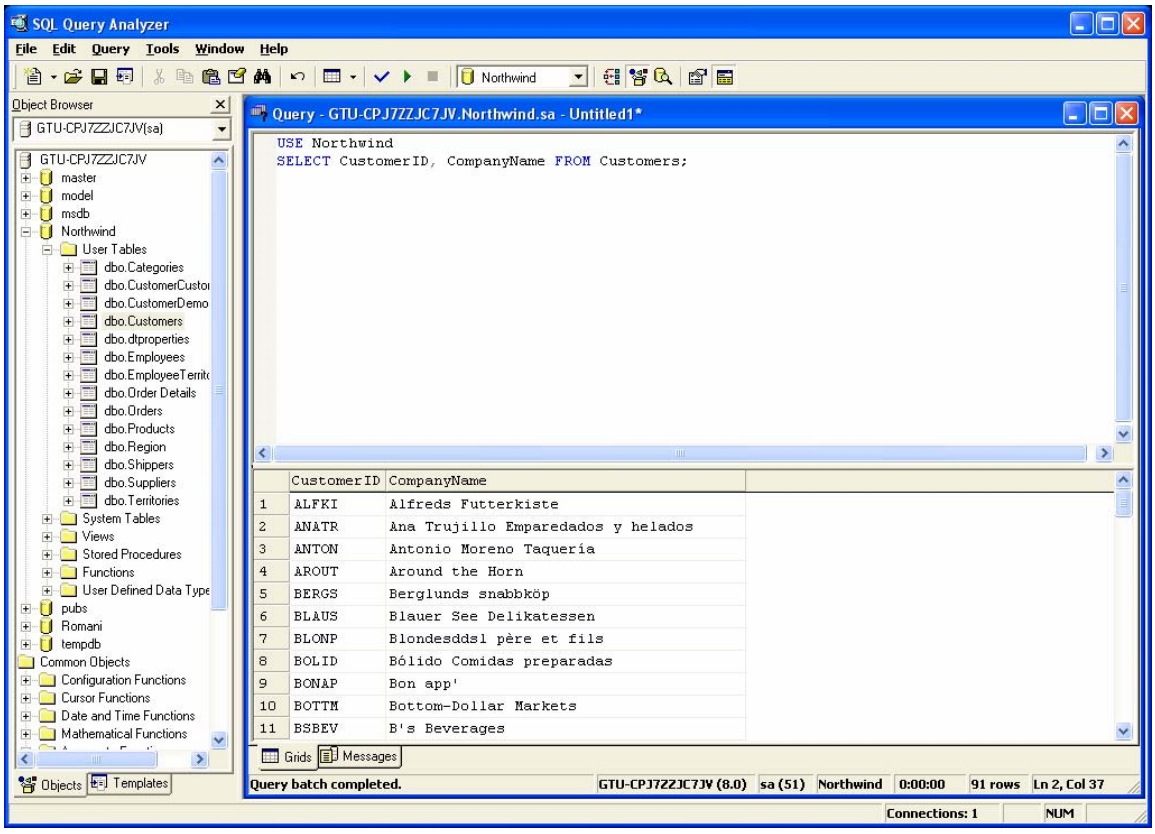
Query მენიუში ვირჩევთ Execute ბრძანებას.

ინსტრუმენტების პანელში ვაჭერთ Execute Query კლავიშს (მწვანე სამკუთხედი).

ვაჭერთ კლავიატურის F5 კლავიშს.

ვაჭერთ Ctrl+E კლავიშებს.

მონაცემთა ბაზის მართვის სისტემა იღებს SQL-მოთხოვნას, ასრულებს მას და გასცემს მის შედეგს. მოთხოვნის შედეგი აისახება Query ფანჯრის ქვედა ნაწილში.



ნახ. 18.2

SQL-მოთხოვნის შენახვა და ჩატვირთვა.

არსებობს SQL-მოთხოვნის ტექსტის შენახვის შესაძლებლობა ტექსტურ ფაილში. შემდგომში შესაძლებელი იქნება ამ ფაილიდან მოთხოვნის წაკითხვა და შესრულება. SQL-მოთხოვნის შესანახად ვასრულებთ შემდეგი მოქმედებებიდან ერთ-ერთს:

File□Save As;

ვაჭერთ ინსტრუმენტების პანელის Query/Result კლავიშს;

Ctrl+S.

ყველა ამ შემთხვევაში გაიხსნება Save Query (მოთხოვნის შენახვა) ფანჯარა. მოთხოვნა შევინახოთ CustomerSelect.sql ფაილში. მოთხოვნის შენახვის შემდეგ ფაილი

შეგვიძლია გავხსნათ ერთ-ერთი საშუალებით:

File□Open;

ვაჭერთ ინსტრუმენტების პანელის Load SQL Script (SQL-სკრიპტის ჩატვირთვა) კლავიშს;

Ctrl+Shift+P.

ყველა ამ შემთხვევაში გაიხსნება Open Query File (მოთხოვნის შემცველი ფაილის გახსნა) ფანჯარა. ფაილიდან მოთხოვნის ჩატვირთვის შემდეგ შეგვეძლება მისი შესრულება.

მონაცემთა ტიპები

მონაცემთა ბაზაში ცხრილის თითოეულ ველს აქვს განსაზღვრული ტიპი. იგი მიუთითებს თუ რა მნიშვნელობები იქნება მოთავსებული ცხრილის ველებში. ცხრილში 18.1 მოყვანილია SQL-სერვერის მონაცემების ზოგიერთი ტიპი.

ცხრილი 18.1. Microsoft SQL-სერვერის მონაცემთა ტიპები.

ტიპი	აღწერა
bigint	მთელი რიცხვი დიაპაზონში $-2^{63} \div 2^{63}-1$
int	მთელი რიცხვი დიაპაზონში $-2^{31} \div 2^{31}-1$
smallint	მთელი რიცხვი დიაპაზონში $-2^{15} \div 2^{15}-1$
tinyint	მთელი რიცხვი დიაპაზონში 0 ÷ 255
bit	მთელი რიცხვი მნიშვნელობით 0 ან 1
decimal	მოცემული სიზუსტისა და მასშტაბის რიცხვითი მნიშვნელობა დიაპაზონში $2^{38}+1 \div 2^{38}-1$
numeric	იგივეა რაც decimal
money	ფულის ერთეული დიაპაზონში $-2^{63} \div 2^{63}-1$
smallmoney	ფულის ერთეული დიაპაზონში $-2^{31} \div 2^{31}-1$
float	მცურავწერტილიანი მნიშვნელობა დიაპაზონში -1,79E+308 □ -1,79E+308
real	მცურავწერტილიანი მნიშვნელობა დიაპაზონში -3,40E+38 ÷ -3,40E+38
datetime	თარიღი და დრო 1753 წლის 1 იანვრიდან 9999 წლის 31 დეკემბრამდე
smalldatetime	თარიღი და დრო 1900 წლის 1 იანვრიდან 2079 წლის 6 ივნისამდე
char	ფიქსირებული სიგრძის სტრიქონი 8000 სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს
varchar	ცვლადი სიგრძის სტრიქონი 8000 სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს

ცხრილი 18.1 (გაგრძელება)

text	ცვლადი სიგრძის სტრიქონი სიმბოლოების მაქსიმალური რაოდენობით 2^{31} -1, რომელიც არ შეიცავს Unicode სიმბოლოებს
nchar	ფიქსირებული სიგრძის Unicode სიმბოლოების სტრიქონი 4000 სიმბოლომდე
nvarchar	ცვლადი სიგრძის Unicode სიმბოლოების სტრიქონი 4000 სიმბოლომდე
ntext	ცვლადი სიგრძის Unicode სიმბოლოების სტრიქონი 2^{30} -1 სიმბოლომდე
binary	ფიქსირებული სიგრძის ორობითი მონაცემები 8000 ბაიტამდე
varbinary	ცვლადი სიგრძის ორობითი მონაცემები 8000 ბაიტამდე
image	ცვლადი სიგრძის ორობითი მონაცემები 2^{31} -1 ბაიტამდე

მონაცემთა ბაზისა და ცხრილების შექმნა

SQL-სერვერზე მონაცემთა ბაზის შესაქმნელად შემდეგ მოქმედებებს ვასრულებთ:

1. Start→All Programs→Microsoft SQL Server→Enterprise Manager.
2. Console Root ფანჯრის მარცხენა ზონაში ვაჭერთ Microsoft SQL Servers სტრიქონის მარცხნივ მყოფ + კლავიშს.
3. ვაჭერთ SQL Server სტრიქონის მარცხნივ მყოფ + კლავიშს.
4. ვაჭერთ (local)(Windows NT) სტრიქონის მარცხნივ მყოფ + კლავიშს.
5. მოვნიშნავთ Databases სტრიქონს.
6. ფანჯრის მარჯვენა ზონაში ვხსნით კონტექსტურ მენიუს და ვასრულებთ New Database ბრძანებას.
7. გახსნილი ფანჯრის Name ველში შეგვაქვს მონაცემთა ბაზის სახელი, მაგალითად, Student და ვაჭერთ OK კლავიშს.

SQL-სერვერზე ცხრილის შესაქმნელად შემდეგ მოქმედებებს ვასრულებთ:

1. ვაჭერთ Databases სტრიქონის მარცხნივ მყოფ + კლავიშს.
2. მოვნიშნავთ Student სტრიქონს.
3. ფანჯრის მარჯვენა ზონაში ვხსნით კონტექსტურ მენიუს და ვასრულებთ New ქვემენიუს Table ბრძანებას.
4. გაიხსნება ფანჯარა, რომლის Column Name ველში შეგვაქვს ველის სახელი, Data Type ველში შეგვაქვს ველის ტიპი, Length ველში შეგვაქვს ველის სიგრძე, Allow Nulls ველში მოვნიშნავთ ალამს იმ შემთხვევაში თუ აუცილებელია მოცემულ ველში მნიშვნელობის შეტანა.
5. შევქმნათ პირველადი გასაღები. ამისათვის, ცხრილის პირველი ველის სახელზე მოვათავსოთ კურსორი, გავხსნათ კონტექსტური მენიუ და შევასრულოთ Set Primary Key ბრძანება.
6. ყველა ველის შეტანის შემდეგ ამ ფანჯრის ზედა ნაწილში ვხსნით კონტექსტურ მენიუს და ვასრულებთ Save ბრძანებას. გახსნილ ფანჯარაში შეგვაქვს ცხრილის სახელი, მაგალითად, Salespeople და ვაჭერთ OK კლავიშს.

ცხრილის შექმნა და წაშლა შესაძლებელია, აგრეთვე, SQL-ენის ბრძანებების გამოყენებით. მონაცემთა ბაზის შესაქმნელად გამოიყენება CREATE DATABASE ბრძანება. მისი სინტაქსია:

CREATE DATABASE *მონაცემთა_ბაზის_სახელი*

მაგალითად, CREATE DATABASE Studenti ბრძანება შექმნის მონაცემთა ბაზას, რომლის სახელია Studenti.

მონაცემთა ბაზის წასაშლელად გამოიყენება DROP DATABASE ბრძანება. მისი სინტაქსია:

DROP DATABASE *მონაცემთა_ბაზის_სახელი*

მაგალითად, DROP DATABASE Studenti ბრძანება წაშლის მონაცემთა ბაზას, რომლის სახელია Studenti.

ცხრილის შესაქმნელად გამოიყენება CREATE TABLE ბრძანება. მისი სინტაქსია:

CREATE TABLE *ცხრილის_სახელი (ველის_სახელი ველის_ტიპი(ზომა))*

ზომა ეთითება სტრიქონული და ზოგიერთი სხვა ტიპის ველისათვის.

PRIMARY KEY მიეთითება ველის გამოცხადების შემდეგ და იმაზე მიუთითებს, რომ მოცემული ველი წარმოადგენს პირველად გასაღებს. NOT NULL მიუთითებს, რომ მოცემული ველის მნიშვნელობა აუცილებლად უნდა იყოს მითითებული. მაგალითად,

```
CREATE TABLE Table1
(       ID  integer NOT NULL PRIMARY KEY,
        Dep char(15),
        Fam char(20) NOT NULL,
        Nam float,
        Dat datetime
)
```

UNIQUE სიტყვა მიუთითებს, რომ მოცემული ველის მნიშვნელობა უნდა იყოს უნიკალური (არ უნდა მეორდებოდეს). DEFAULT სიტყვა მიუთითებს იმ მნიშვნელობას, რომელიც მოცემულ ველს ავტომატურად მიენიჭება იმ შემთხვევაში, როცა ველის მნიშვნელობა არ არის მითითებული. მაგალითად,

```
CREATE TABLE Table2
(       ID  integer NOT NULL PRIMARY KEY,
        Dep char(15) UNIQUE,
        Fam integer DEFAULT 2005,
        Nam float,
        Dat datetime,
        Dap char(10) DEFAULT 'samxaraZe'
)
```

CHECK სიტყვა გამოიყენება ველის მნიშვნელობაზე შეზღუდვების დასადებად. მაგალითად,

```
CREATE TABLE Table3
(       ID integer NOT NULL PRIMARY KEY,
        Dep char(15) UNIQUE,
        Fam integer DEFAULT 2005 CHECK ((Fam > 2000) AND (Fam < 2050)),
        Nam float,
        Dat datetime,
        Dap char(10) DEFAULT 'samxaraZe'
```

)

ცხრილის წასაშლელად გამოიყენება DROP TABLE ბრძანება. მისი სინტაქსია:

DROP TABLE *ცხრილის_სახელი*

მაგალითად, Pers ცხრილის წასაშლელად უნდა შევიტანოთ ბრძანება:

DROP TABLE Table1

SELECT ბრძანება

SELECT ბრძანება გამოიყენება მონაცემთა ბაზის ცხრილებიდან ინფორმაციის ამოსაღებად. ამ ბრძანებას შევისწავლით სამი ცხრილის საფუძველზე: Salespeople (სავაჭრო აგენტები), Customers (შემკვეთები) და Orders (შეკვეთები). ამისათვის, ჯერ შევქმნათ Student მონაცემთა ბაზა, მასში კი ეს სამი ცხრილი. ქვემოთ მოყვანილია ეს ცხრილები.

ცხრილი Salespeople.

Snum	Sname	Qalaqi	Kodi
101	სამხარაძე	თბილისი	12
102	კაპანაძე	ქუთაისი	13
103	გაჩეჩილაძე	ბათუმი	10
104	შენგელია	თელავი	11
105	მჭედლიშვილი	გორი	15

ცხრილი Customers.

Cnum	Cname	Qalaqi	Rating	Snum
201	ჯეონეთი	თბილისი	100	101
202	სანეტი	ფოთი	200	103
203	თიაი-სერვისი	ქობულეთი	200	102
204	ტაო	ხაშური	300	102
205	ჯეოსელი	სოხუმი	100	101
206	მაგთი	რუსთავი	100	104
207	გლობალ-ერთი	ბათუმი	300	105

ცხრილი Orders.

Onum	Tanxa	Tarigi	Cnum	Snum
301	981.41	1999/10/03	207	105
302	1900.10	1999/10/03	206	104
303	767.19	1999/10/03	201	101
304	5160.45	1999/10/03	203	102
305	1098.16	1999/10/03	202	105
306	75.75	1999/10/04	204	102
307	4723.00	1999/10/05	205	101
308	1713.23	1999/10/04	202	103
309	1309.95	1999/10/06	204	102
310	9891.88	1999/10/06	205	101

მარტივ შემთხვევაში SELECT ბრძანება მონაცემთა ბაზის ცხრილებიდან იღებს ინფორმაციას. მაგალითად, Salespeople (სავაჭრო აგენტები) ცხრილის ჩანაწერების მისაღებად SELECT ბრძანება უნდა შევიტანოთ შემდეგნაირად:

```
SELECT snum, sname, qalaqi, kodi FROM Salespeople
```

ბრძანების შესრულების შედეგია:

snum	sname	qalaqi	kodi
101	სამხარაძე	თბილისი	12
102	კაპანაძე	ქუთაისი	13
103	გაჩეჩილაძე	ბათუმი	10
104	შენგელია	თელავი	11
105	მჭედლიშვილი	გორი	15

აქ FROM არის საკვანძო სიტყვა, რომლის შემდეგ ეთითება ცხრილის სახელი. ცხრილის ყველა ველის გამოსატანად შეგვიძლია მივუთითოთ *. მაგალითად,

```
SELECT * FROM Salespeople
```

ბრძანების შესრულების შედეგი იქნება ისეთივე, როგორც წინა შემთხვევაში.

შეგვიძლია მივიღოთ, აგრეთვე, ჩვენთვის საჭირო ველები. მაგალითად,

```
SELECT sname, kodi FROM Salespeople
```

ბრძანების შესრულების შედეგია:

sname	kodi
სამხარაძე	12
კაპანაძე	13
გაჩეჩილაძე	10
შენგელია	11
მჭედლიშვილი	15

DISTINCT საკვანძო სიტყვა ზღუდავს გამეორებადი მნიშვნელობების გამოტანას. დავუშვათ, გვაინტერესებს მოცემულ მომენტში რომელ სავაჭრო აგენტებს აქვთ შეკვეთები Orders ცხრილში (შეკვეთების ცხრილში). ჩვენ არ გვაინტერესებს თითოეულ აგენტს რამდენი შეკვეთა აქვს. გვაინტერესებს მხოლოდ სავაჭრო აგენტების კოდების სია snum. ამისათვის, შეგვიძლია შევიტანოთ ბრძანება

```
SELECT snum FROM Orders
```

შედეგად მივიღებთ:

snum
105
104
101
102
105
102
101
103
102
101

როგორც ვხედავთ სავაჭრო აგენტების კოდები მეორდება. თუ გვინდა მივიღოთ სია გამეორებადი მნიშვნელობების გარეშე, მაშინ უნდა გამოვიყენოთ DISTINCT სიტყვა:
SELECT DISTINCT snum FROM Orders

შედეგად მივიღებთ ასეთ სიას:

snum
101
102
103
104
105

WHERE საკვანძო სიტყვა გამოიყენება ცხრილში ჩანაწერების ფილტრაციისათვის. იგი განსაზღვრავს პირობას, რომელიც შეიძლება იყოს ჭეშმარიტი (true) ან მცდარი (false) ცხრილის ნებისმიერი სტრიქონისათვის. SELECT-მოთხოვნა ამოირჩევს ცხრილის მხოლოდ იმ სტრიქონებს, რომლებისთვისაც ეს პირობა იღებს ჭეშმარიტ მნიშვნელობას. დავუშვათ გვაინტერესებს თბილისელი სავაჭრო აგენტების სახელები და მათი საკომისიო კოდი. SELECT-მოთხოვნას ექნება სახე:
SELECT sname, kodi FROM Salespeople WHERE qalaqi = 'Tbilisi'
შედეგად, ამოირჩევა ჩანაწერები:

Sname	Kodi
სამხარაძე	12

როცა მოთხოვნა შეიცავს WHERE სიტყვას, მაშინ SQL-სერვერი განიხილავს მთელ ცხრილს და ამოწმებს პირობა სრულდება თუ არა თითოეული ჩანაწერისათვის. აქვე შევნიშნოთ, რომ სტრიქონული კონსტანტების მოთავსება ხდება აპოსტროფებში 'თბილისი' ან ბრჭყალებში "თბილისი".

ახლა შევადგინოთ მოთხოვნა, რომლის WHERE სიტყვაში მითითებულია რიცხვითი ველი, მაგალითად, Customers ცხრილის rating ველი. ეს ველი გამოიყენება იმისათვის, რომ დავყოთ შემკვეთები ჯგუფებად. ამოვირჩიოთ ყველა შემკვეთი, რომელთა რეიტინგია 100. SELECT მოთხოვნას ექნება სახე:
SELECT * FROM Customers WHERE rating = 100

რადგან rating რიცხვითი ველია, ამიტომ ბრჭყალები არ გამოიყენება. შედეგად, ამოირჩევა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
201	ჯეონეთი	თბილისი	100	101
205	ჯეოსელი	სოხუმი	100	101
206	მაგთი	რუსთავი	100	104

შედარებისა და ლოგიკის ოპერატორები

შედარების ოპერატორებია:
= - ტოლია

> - მეტია
< - ნაკლებია
>= - მეტია ან ტოლი
<= - ნაკლებია ან ტოლი
!= - არაა ტოლი

დავუშვათ გვაინტერესებს შემკვეთები, რომელთა რეიტინგი აღემატება 200-ს. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Customers WHERE rating > 200
```

შედეგად, ამოირჩევა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
204	ტაო	ხაშური	300	102
207	გლობალ-ერთი	ბათუმი	300	105

ლოგიკის ოპერატორებია:

AND - ლოგიკური “და”
OR - ლოგიკური “ან”
NOT - ლოგიკური “არა”

AND ოპერატორი ადარებს ორ არგუმენტს (გამოსახულებას) და აბრუნებს true მნიშვნელობას თუ ორივე გამოსახულების მნიშვნელობაა true, წინააღმდეგ შემთხვევაში აბრუნებს false მნიშვნელობას. OR ოპერატორი ადარებს ორ არგუმენტს (გამოსახულებას) და აბრუნებს true მნიშვნელობას თუ ერთ-ერთი ლოგიკური გამოსახულების მნიშვნელობაა true. NOT ოპერატორი აბრუნებს true მნიშვნელობას თუ მისი არგუმენტის მნიშვნელობაა false და პირიქით.

ლოგიკის ოპერატორების გამოყენება მნიშვნელოვნად ზრდის SELECT-ბრძანების შესაძლებლობებს. დავუშვათ, გვაინტერესებს ბათუმელი შემკვეთები, რომელთა რეიტინგი აღემატება 200-ს. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Customers WHERE qalaqi = ‘baTumi’  AND rating > 200
```

შედეგად, გაიცემა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
207	გლობალ-ერთი	ბათუმი	300	105

ახლა დავუშვათ, გვაინტერესებს ის შემკვეთები, რომლებიც მდებარეობენ ბათუმში ან რომელთა რეიტინგი აღემატება 200-ს. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Customers WHERE qalaqi = ‘baTumi’  OR rating > 200
```

შედეგად, გაიცემა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
204	ტაო	ხაშური	300	102
207	გლობალ-ერთი	ბათუმი	300	105

იმ შემკვეთების მისაღებად, რომლებიც არ მდებარეობენ ბათუმში SELECT-მოთხოვნა უნდა შევიტანოთ შემდეგნაირად:

```
SELECT * FROM Customers WHERE NOT ( qalaqi = ‘baTumi’ )
```

სპეციალური ოპერატორები IN, BETWEEN, LIKE, IS NULL

IN ოპერატორი განსაზღვრავს იმ მნიშვნელობების სიას, რომელსაც უნდა შეიცავდეს ველის მნიშვნელობა. მაგალითად, თუ გვაინტერესებს ინფორმაცია თბილისელი ან ქუთაისელი სავაჭრო აგენტების შესახებ, უნდა გამოვიყენოთ შემდეგი SELECT-მოთხოვნა:

SELECT * FROM Salespeople WHERE qalaqi = 'Tbilisi' OR qalaqi = 'quTaisi'

მაგრამ არსებობს უფრო მარტივი და ეფექტური საშუალება:

SELECT * FROM Salespeople WHERE qalaqi IN ('Tbilisi', 'quTaisi')

შედეგად, გაიცემა ჩანაწერები:

snum	sname	qalaqi	kodi
101	სამხარაძე	თბილისი	12
102	კაპანაძე	ქუთაისი	13

BETWEEN ოპერატორი განსაზღვრავს მნიშვნელობების დიაპაზონს. BETWEEN სიტყვის შემდეგ ეთითება საწყისი მნიშვნელობა, შემდეგ AND სიტყვა და საბოლოო მნიშვნელობა. საწყისი და საბოლოო სიდიდეები შედიან დიაპაზონში. დავუშვათ გვაინტერესებს ის სავაჭრო აგენტები, რომელთა საკომისიო კოდი მოთავსებულია 10-სა და 12-ს შორის. SELECT-მოთხოვნას ექნება სახე:

SELECT * FROM Salespeople WHERE kodi BETWEEN 10 AND 12

შედეგად გაიცემა ჩანაწერები:

snum	sname	qalaqi	kodi
101	სამხარაძე	თბილისი	12
103	გაჩეჩილაძე	ბათუმი	10
104	შენგელია	თელავი	11

თუ გვაინტერესებს ის სავაჭრო აგენტები, რომელთა საკომისიო კოდი არ არის მოთავსებული 10-სა და 12-ს შორის, მაშინ SELECT-მოთხოვნას ექნება სახე:

SELECT * FROM Salespeople WHERE kodi NOT BETWEEN 10 AND 12

LIKE ოპერატორი გამოიყენება სიმბოლურ ველებში ქვესტრიქონების საპოვნელად. ამისათვის, იგი იყენებს შაბლონებს, რომლებიც სპეციალური სიმბოლოებისაგან შედგება (ცხრილი 18.2).

ხაზგასმის სიმბოლო _ ცვლის ნებისმიერ ერთ სიმბოლოს. მაგალითად, 'ბა_ი' შაბლონს შეესაბამება 'ბალი', 'ბარი', 'ბაკი' და არა 'ბავშვი'.

ცხრილი 18.2. სპეციალური სიმბოლოები.

#	სიმბოლო	აღწერა
1	_	შეესაბამება ნებისმიერ ერთ სიმბოლოს
2	%	შეესაბამება სიმბოლოების ნებისმიერ მიმდევრობას
3	[ ]	შეესაბამება ფრჩხილებში ჩამოთვლილ ნებისმიერ ერთ სიმბოლოს
4	[^]	შეესაბამება ნებისმიერ ერთ სიმბოლოს, გარდა ფრჩხილებში ჩამოთვლილი სიმბოლოებისა
5	[-]	შეესაბამება სიმბოლოების დიაპაზონს
6	#	შეესაბამება ნებისმიერ ერთ ციფრს

სიმბოლო % ცვლის სიმბოლოების ნებისმიერ მიმდევრობას. მაგალითად, ‘ბა%ი’ შაბლონს შეესაბამება ‘ბავშვი’, ‘ბარტყი’, ‘ბარი’ და ა.შ.

დავუშვათ გვაინტერესებს შემკვეთები, რომელთა სახელები ‘ჯ’ ასოთი იწყება. SELECT-მოთხოვნას ექნება სახე:

SELECT * FROM Customers WHERE cname LIKE ‘j%’

შედეგად, გაიცემა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
201	ჯეონეთი	თბილისი	100	101
205	ჯეოსელი	სოხუმი	100	101

[ ] ფრჩხილებში ეთითება სიმბოლოები. შედეგად გაიცემა ის ჩანაწერები, რომლებშიც მითითებული ველის მნიშვნელობები იწყება ამ ჩამოთვლილი სიმბოლოებიდან ერთ-ერთით. მაგალითად,

SELECT * FROM Customers WHERE qalaqi LIKE ‘[თს]%’

შედეგად, გაიცემა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
201	ჯეონეთი	თბილისი	100	101
205	ჯეოსელი	სოხუმი	100	101

[^] ფრჩხილებში ეთითება სიმბოლოები. შედეგად გაიცემა ის ჩანაწერები, რომლებშიც მითითებული ველის მნიშვნელობები არ იწყება ამ ჩამოთვლილი სიმბოლოებიდან არც ერთით. მაგალითად,

SELECT * FROM Customers WHERE cname LIKE ‘[^თს]%’

შედეგად, გაიცემა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
201	ჯეონეთი	თბილისი	100	101
205	ჯეოსელი	სოხუმი	100	101
206	მაგთი	რუსთავი	100	104
207	გლობალ-ერთი	ბათუმი	300	107

[-] შეესაბამება სიმბოლოების დიაპაზონს. მაგალითად,

SELECT * FROM Customers WHERE cname LIKE '[ა-ჯ]%'

ბრძანების შესრულების შედეგად გაიცემა ჩანაწერები, რომლებშიც cname ველის მნიშვნელობები იწყება ა-დან ჯ-მდე ასოებით:

Cnum	Cname	Qalaqi	Rating	Snum
201	ჯეონეთი	თბილისი	100	101
205	ჯეოსელი	სოხუმი	100	101
207	გლობალ-ერთი	ბათუმი	300	107

შეესაბამება ნებისმიერ ერთ ციფრს. მაგალითად, ‘ა#’-ს შეესაბამება ა1, ა2 და ა.შ.

LIKE ოპერატორის გამოყენება მოხერხებულია მაშინ, როცა ზუსტად არ ვიცით სტრიქონის მნიშვნელობა.

IS NULL ოპერატორი გამოიყენება იმისათვის, რომ ვიპოვოთ NULL მნიშვნელობის შემცველი ჩანაწერები. NULL მიუთითებს მნიშვნელობის არ არსებობაზე. მაგალითად, Customers ცხრილიდან ამოვირჩიოთ ის ჩანაწერები, რომელთა qalaqi ველი შეიცავს NULL მნიშვნელობას. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Customers WHERE qalaqi IS NULL
```

აგრეგატული ფუნქციები

აგრეგატული ფუნქციებია:

COUNT - ასრულებს იმ ჩანაწერების დათვლას, რომლებიც აკმაყოფილებენ მოთხოვნის პირობებს.

SUM - გამოთვლის ველის მნიშვნელობების ჯამს.

AVG - გამოთვლის ველის მნიშვნელობების საშუალო არითმეტიკულს.

MAX - განსაზღვრავს ამორჩეულ მნიშვნელობებს შორის მაქსიმალურს.

MIN - განსაზღვრავს ამორჩეულ მნიშვნელობებს შორის მინიმალურს.

SUM და AVG ფუნქციები გამოიყენება მხოლოდ რიცხვითი ველების მიმართ. COUNT, MAX და MIN ფუნქციები გამოიყენება როგორც რიცხვითი, ისე სიმბოლური ველების მიმართ. MAX და MIN ფუნქციები სიმბოლურ ველებთან მუშაობის დროს მნიშვნელობებს ადარებენ ანბანური მიმდევრობით. აგრეგატული ფუნქციები ახდენენ NULL მნიშვნელობების იგნორირებას.

ვიპოვოთ Orders ცხრილის ყველა შეკვეთის თანხების ჯამი. SELECT-მოთხოვნას ექნება სახე:

```
SELECT SUM(tanxa) FROM Orders
```

შედეგად, გაიცემა სტრიქონი:

SUM of tanxa
26658.40

COUNT ფუნქცია თვლის მნიშვნელობების რაოდენობას მოცემულ ველში ან სტრიქონების საერთო რაოდენობას ცხრილში. დავუშვათ, გვაინტერესებს Orders ცხრილში სავაჭრო აგენტების ნომრების რაოდენობა. SELECT-მოთხოვნას ექნება სახე:

```
SELECT COUNT(DISTINCT snum) FROM Orders
```

შედეგად, გაიცემა სტრიქონი:

COUNT of snum
5

ყურადღება მივაქციოთ იმას, რომ SELECT-მოთხოვნაში გამოყენებულია DISTINCT სიტყვა, შედეგად, snum ველში დაითვლება უნიკალური მნიშვნელობები. თუ DISTINCT სიტყვას არ მივუთითებთ, მაშინ გაიცემა სტრიქონი:

COUNT of snum
10

ცხრილში ჩანაწერების რაოდენობის დასათვლელად COUNT ფუნქციას არგუმენტად უნდა მივუთითოთ *. SELECT-მოთხოვნას ექნება სახე:

```
SELECT COUNT(*) FROM Customers
```

შედეგად, გაიცემა სტრიქონი:

COUNT (*)
7

GROUP BY ოპერატორი საშუალებას გვაძლევს განვსაზღვროთ მნიშვნელობების ქვესიმრავლე, რომლის მიმართაც გამოყენებული იქნება აგრეგატული ფუნქცია. დავუშვათ, გვაინტერესებს შეკვეთის მაქსიმალური თანხა თითოეული სავაჭრო აგენტისათვის. SELECT-მოთხოვნას ექნება სახე:

```
SELECT snum, MAX(tanxa) FROM Orders GROUP BY snum
```

შედეგად, გაიცემა ჩანაწერები:

```
snum  MAX of tanxa
---------------------------
101     9891.88
102     5160.45
103     1713.23
104     1900.10
105     1098.16
```

GROUP BY ოპერატორი აგრეგატულ ფუნქციებს იყენებს ჩანაწერების ჯგუფების მიმართ. ჯგუფის ფორმირების პირობაა ველის ერთნაირი მნიშვნელობები. ჩვენს შემთხვევაში ესაა snum ველი.

GROUP BY ოპერატორი შეგვიძლია გამოვიყენოთ რამოდენიმე ველის მიმართ. დავუშვათ, გვაინტერესებს სავაჭრო აგენტების კოდები და მათი შეკვეთების მაქსიმალური თანხები თითოეული თარიღისათვის. SELECT-მოთხოვნას ექნება სახე:

```
SELECT snum, tarigi, MAX(tanxa) FROM Orders
GROUP BY snum, tarigi  ORDER BY snum
```

შედეგად, გაიცემა ჩანაწერები:

Snum	Tarigi	MAX of tanxa
101	1999/10/03	767.19
101	1999/10/05	4723.00
101	1999/10/06	9891.88
102	1999/10/03	5160.45
102	1999/10/04	75.75
102	1999/10/06	1309.95
103	1999/10/04	1713.23
104	1999/10/03	1900.10
105	1999/10/03	1098.16

ORDER BY ოპერატორი გამოიყენება Orders ცხრილის დასახარისხებლად snum ველის მნიშვნელობების ზრდადობის მიხედვით.

ახლა დავუშვათ გვაინტერესებს თითოეული აგენტის შეკვეთების მაქსიმალური თანხა, რომელიც აღემატება 3000-ს. ამ ამოცანის გადასაწყვეტად უნდა გამოვიყენოთ HAVING ოპერატორი, რომელიც განსაზღვრავს კრიტერიუმს, რომელიც გამოიყენება მიღებული შედეგიდან ჯგუფის გამოსაყოფად. SELECT-მოთხოვნას ექნება სახე:

```
SELECT snum, tarigi, MAX(tanxa) FROM Orders
GROUP BY snum, tarigi
HAVING MAX(tanxa) > 3000
```

შედეგად, გაიცემა ჩანაწერები:

Snum	Tarigi	MAX of tanxa
102	1999/10/03	5160.45
101	1999/10/05	4723.00
101	1999/10/06	9891.88

ჩანაწერების დახარისხება

ORDER BY ოპერატორი გამოიყენება ცხრილის დასახარისხებლად ამა თუ იმ ველის მნიშვნელობების ზრდის ან კლების მიხედვით. მაგალითად,

SELECT * FROM Orders ORDER BY Tarigi

ბრძანების შესრულების შედეგად გაიცემა Tarigi ველის მიხედვით ზრდადობით დალაგებული ჩანაწერები.

ASC და DESC საკვანძო სიტყვების გამოყენებით შეგვიძლია აშკარად მივუთითოთ თუ რა რიგითობით უნდა შესრულდეს ველების დახარისხება. ASC გამოიყენება ველების დასახარისხებლად ზრდადობით, DESC კი - კლებადობით. მაგალითად,

SELECT * FROM Orders ORDER BY Tarigi DESC

ბრძანების შესრულების შედეგად გაიცემა Tarigi ველის მიხედვით კლებადობით დალაგებული ჩანაწერები.

თუ ASC საკვანძო სიტყვა არ არის მითითებული, მაშინ დახარისხება შესრულდება ზრდადობის მიხედვით.

ORDER BY ოპერატორის შემდეგ შეგვიძლია მივუთითოთ ერთი ან მეტი ველის სახელი, მაგალითად,

SELECT * FROM Orders ORDER BY tarigi, tanxa DESC

ამ შემთხვევაში Orders ცხრილის ჩანაწერები ჯერ დახარისხდება tarigi ველის მნიშვნელობების მიხედვით ზრდადობით, შემდეგ კი tanxa ველის მნიშვნელობების მიხედვით კლებადობით.

რამოდენიმე ცხრილიდან ჩანაწერების ამორჩევა

აქამდე ჩვენი მოთხოვნები ერთ ცხრილს ეხებოდა. ამიტომ მოთხოვნებში ველის სახელის წინ არ ვუთითებდით ცხრილის სახელს. SQL ენა საშუალებას გვაძლევს ერთი მოთხოვნის შიგნით მივმართოთ რამდენიმე ცხრილს. ამ შემთხვევაში ველის სახელის წინ ეთითება ცხრილის სახელი. მაგალითად,

Salespeople.snum, Salespeople.qalaqi, Orders.tarigi.

დავუშვათ, გვაინტერესებს ის სავაჭრო აგენტები და შემკვეთები, რომლებიც ერთ ქალაქში მდებარეობენ. SELECT-მოთხოვნას ექნება სახე:

SELECT Customers.cname, Salespeople.sname, Salespeople.qalaqi

FROM Salespeople, Customers

WHERE Salespeople.qalaqi = Customers.qalaqi

შედეგად, გაიცემა ჩანაწერები:

Cname	Sname	Qalaqi
ჯეონეთი	სამხარაძე	თბილისი
გლობალ-ერთი	გაჩეჩილაძე	ბათუმი

ამ მოთხოვნის შესრულებისას SQL-სერვერი იღებს ამ ორი ცხრილის ჩანაწერების თითოეულ კომბინაციას და ამოწმებს მათ WHERE სიტყვაში მითითებული პირობის შესრულებაზე. ამოირჩევა ის ჩანაწერები, რომლებისთვისაც ეს პირობა სრულდება.

ახლა დავუშვათ გვაინტერესებს იმ შემკვეთების შეკვეთები, რომლებიც არ იმყოფებიან სავაჭრო სავაჭრო აგენტებთან ერთად ერთ ქალაქში. SELECT-მოთხოვნას ექნება სახე:

```
SELECT Orders.onum, Customers.cname, Orders.cnum, Orders.snum
FROM Salespeople, Customers, Orders
WHERE Customers.qalaqi <> Salespeople.qalaqi AND
      Orders.cnum = Customers.cnum AND Orders.snum = Salespeople.snum
```

შედეგად, გაიცემა ჩანაწერები:

Onum	Cname	Cnum	Snum
301	გლობალ-ერთი	207	105
302	მაგთი	206	104
305	თიაი-სერვისი	203	102
306	სანეტი	202	105
307	ტაო	204	102
308	ჯეოსელი	205	101
309	სანეტი	202	103
310	ტაო	204	102
311	ჯეოსელი	205	101

ჩადგმული მოთხოვნები

ერთ მოთხოვნას შეუძლია მეორე მოთხოვნის მართვა. ეს ხდება ერთი მოთხოვნის შიგნით მეორის მოთავსებით. შიდა მოთხოვნის მიერ გაცემული შედეგები მოწმდება გარე (ძირითადი) მოთხოვნის პირობაში, რომელიც განსაზღვრავს ეს შედეგები ჭეშმარიტია (true) თუ მცდარი (false). მაგალითად, დავუშვათ ვიცით სავაჭრო აგენტის გვარი - 'შენგელია', მაგრამ არ ვიცით მისი კოდი - snum და გვაინტერესებს მისი ყველა შეკვეთა. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Orders
WHERE snum =    (
                SELECT snum FROM Salespeople WHERE sname = 'Sengelia'
                )
```

შედეგად, გაიცემა ჩანაწერი:

Onum	TANXA	Tarigi	Cnum	Snum
302	1900.10	1999/10/03	206	104

ჯერ შესრულდება შიდა მოთხოვნა, შემდეგ კი გარე. შიდა მოთხოვნის

შესრულების დროს Salespeople ცხრილიდან ამოირჩევა ჩანაწერები, რომლებშიც sname ველის მნიშვნელობაა ‘შენგელია’, შემდეგ ამოირჩევა snum ველის მნიშვნელობა. ასეთია ერთი სტრიქონი, რომელშიც snum = 104. ეს შედეგი მოთავსთდება გარე მოთხოვნის პირობაში შიდა მოთხოვნის ნაცვლად. შედეგად, გარე მოთხოვნის პირობა მიიღებს სახეს:

```
WHERE snum = 104
```

შემდეგ შესრულდება გარე მოთხოვნა.

იმისათვის, რომ გარე მოთხოვნა შესრულდეს საჭიროა, რომ შიდა მოთხოვნამ გასცეს ერთი ან მეტი ჩანაწერი. წინააღმდეგ შემთხვევაში, გარე მოთხოვნა არ შესრულდება. თუ ვიყენებთ შიდა მოთხოვნას, რომელიც რამდენიმე ჩანაწერს გასცემს, მაშინ უმჯობესია IN ოპერატორის გამოყენება. მაგალითად, გვაინტერესებს თბილისელი სავაჭრო აგენტების შეკვეთები. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Orders
WHERE snum IN (SELECT snum FROM Salespeople WHERE qalaqi = ‘Tbilisi’)
```

ამ შემთხვევაში, შიდა მოთხოვნის გამოყენება აუმჯობესებს კითხვადობასა და მწარმოებლურობას ქვემოთ მოყვანილ მოთხოვნასთან შედარებით:

```
SELECT Orders.* FROM Orders, Salespeople
WHERE Orders.snum = Salespeople.snum AND Salespeople.qalaqi = ‘Tbilisi’
```

მართალია, ეს ორი მოთხოვნა ეკვივალენტურია, მაგრამ SQL-სერვერი მეორე მოთხოვნის შესრულებისას განიხილავს ორივე ცხრილის ჩანაწერების შესაძლო კომბინაციებს და მათთვის პირობის შესრულებას. გაცილებით მარტივი და ეფექტურია Salespeople ცხრილიდან სავაჭრო აგენტების კოდების ამორჩევა და შემდეგ მათი ძებნა Orders ცხრილში. ამიტომ, SQL-სერვერის ოპტიმიზატორი მეორე მოთხოვნას გარდაქმნის ჩადგმულ მოთხოვნად (პირველ მოთხოვნად).

ჩვენს მიერ განხილული შიდა მოთხოვნები გასცემენ ერთ ველს. SELECT * ტიპის ბრძანება შიდა მოთხოვნებში აკრძალულია.

შიდა მოთხოვნებს შეიძლება ჰქონდეთ საკუთარი GROUP BY და HAVING საკვანძო სიტყვები. დავუშვათ გვაინტერესებს იმ შემკვეთების რაოდენობა, რომელთა რეიტინგი მეტია თბილისის საშუალო რეიტინგზე. SELECT-მოთხოვნას ექნება სახე:

```
SELECT rating, COUNT(DISTINCT cnum)  FROM Customers GROUP BY rating
HAVING rating >     (
                    SELECT AVG(rating) FROM Customers WHERE qalaqi = ‘Tbilisi’
                    )
```

შედეგად გაიცემა ჩანაწერები:

```
rating    COUNT(DISTINCT cnum)
-----------------------------------------------------
200             2
300             2
```

ბმული მოთხოვნები (ქვემოთხოვნები)

ბმული მოთხოვნების გამოყენებისას შიდა მოთხოვნიდან შეგვიძლია მივმართოთ გარე მოთხოვნის ცხრილს. ამ დროს შიდა მოთხოვნა სრულდება გარე მოთხოვნაში მითითებული ცხრილის თითოეული ჩანაწერისათვის. გარე მოთხოვნის ჩანაწერს,

რომლისთვისაც უნდა შესრულდეს შიდა მოთხოვნა, ეწოდება ჩანაწერი-კანდიდატი.

ბმული მოთხოვნების შესრულების პროცედურაა:

1. ჩანაწერი-კანდიდატის ამორჩევა.
2. შიდა მოთხოვნის შესრულება.
3. პირობის შეფასება გარე მოთხოვნაში შიდა მოთხოვნის შედეგების საფუძველზე. განისაზღვრება აირჩევა თუ არა ჩანაწერი-კანდიდატი გამოტანისათვის.
4. პროცედურა მეორდება ყველა სტრიქონისათვის.

დავუშვათ, გვაინტერესებს 3 ოქტომბრის შეკვეთები. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Customers C
WHERE '1999/10/03' IN      (
                                SELECT tarigi FROM Orders O WHERE O.cnum = C.cnum
                                )
```

შედეგად, გაიცემა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
201	ჯეონეთი	თბილისი	100	101
202	სანეტი	ფოთი	200	103
203	თიაი-სერვისი	ქობულეთი	200	102
206	მაგთი	რუსთავი	100	104
207	გლობალ-ერთი	ბათუმი	300	105

მოთხოვნებში გამოყენებული C და O სახელები არის ცხრილების ფსევდონიმები. კერძოდ, C არის Customers ცხრილის ფსევდონიმი, O კი Orders ცხრილის ფსევდონიმი. ეს იმას ნიშნავს, რომ Customers და Orders სახელების ნაცვლად შეგვიძლია გამოვიყენოთ C და O სახელები. ამ მაგალითში შიდა მოთხოვნა შესრულდება გარე მოთხოვნის თითოეული ჩანაწერისათვის, ანუ Customers ცხრილის თითოეული ჩანაწერისათვის. ჯერ გარე მოთხოვნაში აირჩევა Customers ცხრილის პირველი ჩანაწერი. შემდეგ სრულდება შიდა მოთხოვნა. შედეგად, Orders ცხრილიდან ამოირჩევა ის ჩანაწერები, რომელშიც ერთმანეთს ემთხვევა Customers და Orders ცხრილების cnum ველების მნიშვნელობები. ეს მნიშვნელობაა 201. შედეგად, შიდა მოთხოვნა გასცემს ჩანაწერს:

303 767.19 1999/10/03 201 101

ამ ჩანაწერში tarigi ველის მნიშვნელობა ემთხვევა გარე მოთხოვნის პირობაში მითითებულ მნიშვნელობას, ამიტომ ეს სტრიქონი გაიცემა როგორც საბოლოო შედეგი. შემდეგ, Customers ცხრილიდან ამოირჩევა ამოირჩევა მეორე ჩანაწერი და მეორდება აღნიშნული პროცესი.

ახლა დავუშვათ გვაინტერესებს იმ სავაჭრო აგენტების სახელები და ნომრები, რომლებსაც ჰყავთ ერთზე მეტი შემკვეთი. SELECT-მოთხოვნას ექნება სახე:

```
SELECT snum, sname FROM Salespeople S
WHERE 1 < ( SELECT COUNT(*) FROM Customer C WHERE C.snum = S.snum )
```

შედეგად, გაიცემა ჩანაწერები:

Snum	Sname
101	სამხარაძე
102	კაპანაძე

ბმული მოთხოვნები შეგვიძლია გამოვიყენოთ ცხრილის თავის თავთან

შედარებისათვის. მაგალითად, გვაინტერესებს ყველა შეკვეთა, რომელთა თანხა აღემატება შეკვეთების თანხების საშუალო არითმეტიკულს. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Orders O
WHERE tanxa > ( SELECT AVG(tanxa) FROM Orders O1 WHERE O1.cnum = O.cnum )
```

შედეგად, გაიცემა ჩანაწერები:

Onum	Tanxa	Tarigi	Cnum	Snum
308	1713.23	1999/10/04	202	103
309	1309.95	1999/10/06	204	102
310	9891.88	1999/10/06	205	101

EXISTS ოპერატორი

EXISTS ოპერატორი არგუმენტად იღებს ქვემოთხოვნას და აფასებს მას როგორც ნამდვილს (true), თუ მოთხოვნა აბრუნებს ჩანაწერებს და როგორც არანამდვილს (false) წინააღმდეგ შემთხვევაში. მაგალითად, ჩვენ შეგვიძლია გადავწყვიტოთ ამოვიღოთ თუ არა ჩანაწერები Customers ცხრილიდან თუ ერთი ან მეტი შემკვეთი იმყოფება თბილისში. SELECT-მოთხოვნას ექნება სახე:

```
SELECT cnum, cname, qalaqi FROM Customers
WHERE EXISTS ( SELECT * FROM Customers WHERE qalaqi = 'Tbilisi' )
```

შედეგად, გაიცემა ჩანაწერები:

Cnum	Cname	Qalaqi
201	ჯეონეთი	თბილისი
202	სანეტი	ფოთი
203	თიაი-სერვისი	ქობულეთი
204	ტაო	ხაშური
205	ჯეოსელი	სოხუმი
206	მაგთი	რუსთავი
207	გლობალ-ერთი	ბათუმი

შიდა მოთხოვნა ამოირჩევს ყველა ჩანაწერს თბილისელი სავაჭრო აგენტისათვის. EXISTS ოპერატორი აფიქსირებს, რომ შიდა მოთხოვნამ დააბრუნა ჩანაწერები. შედეგად, პირობა ნამდვილია. შიდა მოთხოვნა ერთხელ შესრულდება გარე მოთხოვნისათვის და აქვს ერთი მნიშვნელობა ყველა შემთხვევაში. ამიტომ, EXISTS ოპერატორი პირობას ხდის ნამდვილს ან არანამდვილს ყველა ჩანაწერისათვის ერთდროულად.

ბმულ ქვემოთხოვნებში EXISTS ოპერატორი შეფასდება გარე მოთხოვნის ცხრილის თითოეული ჩანაწერისათვის. ეს საშუალებას გვაძლევს EXISTS ოპერატორი გამოვიყენოთ როგორც პირობა, რომელიც ახდენს სხვადასხვა პასუხების გენერირებას გარე მოთხოვნაში მითითებული ცხრილის თითოეული ჩანაწერისათვის. დავუშვათ, გვაინტერესებს სავაჭრო აგენტები, რომლებსაც რამდენიმე შემკვეთი ჰყავთ. SELECT-მოთხოვნას ექნება სახე:

```
SELECT DISTINCT snum FROM Customers couter
WHERE EXISTS ( SELECT * FROM Customers cinner
```

WHERE cinner.snum = couter.snum AND cinner.cnum <> couter.cnum)

შედეგად, გაიცემა ჩანაწერები:

Snum
101
102

გარე მოთხოვნის თითოეული ჩანაწერი-კანდიდატისათვის (ესაა სავაჭრო აგენტი, რომელსაც მოცემულ მომენტში ვამოწმებთ), შიდა მოთხოვნა პოულობს ჩანაწერებს, რომლებიც ემთხვევა snum ველის მნიშვნელობას (რომელიც ჰქონდა სავაჭრო აგენტს), და არ ემთხვევა cnum ველის მნიშვნელობას (რომელიც სხვა შემკვეთებს შეესაბამება). თუ შიდა მოთხოვნის მიერ მოიძებნა ასეთი ჩანაწერები, ეს იმას ნიშნავს, რომ არის ორი სხვადასხვა შემკვეთი, რომლებსაც ერთი სავაჭრო აგენტი ემსახურება. ამიტომ, EXISTS ოპერატორი ნამდვილია მიმდინარე ჩანაწერისათვის და სავაჭრო აგენტის ნომერი (snum) იქნება გამოტანილი. DISTINCT სიტყვა რომ არ იყოს მითითებული, მაშინ თითოეული ეს სავაჭრო აგენტი ამორჩეული იქნება იმდენჯერ, რამდენი შემკვეთიც ჰყავს.

უფრო სასარგებლო იქნება მეტი ინფორმაცია გამოვიტანოთ სავაჭრო აგენტების შესახებ და არა მხოლოდ მათი ნომრები. ეს შეიძლება გავაკეთოთ შემკვეთებისა და სავაჭრო აგენტების ცხრილების გაერთიანებით. SELECT-მოთხოვნას ექნება სახე:

```
SELECT DISTINCT first.snum, first.sname, first.qalaqi
FROM Salespeople first, Customers second
WHERE EXISTS ( SELECT * FROM Customers third WHERE second.snum = third.snum
                    AND second.cnum <> third.cnum AND first.snum = second.snum )
```

შედეგად, გაიცემა ჩანაწერები:

Snum	Sname	Qalaqi
101	სამხარაძე	თბილისი
102	კაპანაძე	ქუთაისი

შიდა მოთხოვნა ისეთივეა, როგორიც წინა ვარიანტში. გარე მოთხოვნა - ესაა სავაჭრო აგენტებისა და შემკვეთების ცხრილების გაერთიანება.

წინა მაგალითმა გვიჩვენა, რომ EXISTS ოპერატორი შეიძლება გამოვიყენოთ ლოგიკის ოპერატორებთან ერთად. თუ გვაინტერესებს სავაჭრო აგენტები, რომლებსაც ერთი შემკვეთი ჰყავთ, მაშინ საჭიროა წინა მოთხოვნის ინვერტირება (ინვერსია). SELECT-მოთხოვნას ექნება სახე:

```
SELECT DISTINCT snum  FROM Customers couter
WHERE NOT EXISTS ( SELECT * FROM Customers cinner
                    WHERE cinner.snum = couter.snum AND cinner.cnum <> couter.cnum )
```

შედეგად, გაიცემა ჩანაწერები:

Snum
103
104
105

ANY და ALL ოპერატორები

დავუშვათ გვაინტერესებს სავაჭრო აგენტები, რომლებიც შემკვეთებთან ერთად არიან ერთ ქალაქში. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Salespeople WHERE qalaqi = ANY (SELECT qalaqi FROM Customers)
```

შედეგად, გაიცემა ჩანაწერები:

Snum	Sname	Qalaqi	Kodi
101	სამხარაძე	თბილისი	12
103	გაჩეჩილაძე	ბათუმი	10

ANY ოპერატორი პირობას თვლის ჭეშმარიტად (true) თუ ერთ-ერთი მნიშვნელობა აკმაყოფილებს გარე მოთხოვნის პირობას.

ALL ოპერატორი პირობას თვლის ჭეშმარიტად (true), თუ ყველა მნიშვნელობა, ამორჩეული შიდა მოთხოვნის მიერ, აკმაყოფილებს გარე მოთხოვნის პირობას. დავუშვათ გვაინტერესებს ის შემკვეთები, რომელთა რეიტინგი მეტია თითოეული თბილისელი შემკვეთის რეიტინგზე. SELECT-მოთხოვნას ექნება სახე:

```
SELECT * FROM Customers
WHERE rating > ALL ( SELECT rating FROM Customers WHERE qalaqi = 'Tbilisi' )
```

შედეგად, გაიცემა ჩანაწერები:

Cnum	Cname	Qalaqi	Rating	Snum
202	სანეტი	ფოთი	200	103
203	თიაი-სერვისი	ქობულეთი	200	102
204	ტაო	ხაშური	300	102
207	გლობალ-ერთი	ბათუმი	300	105

UNION ბრძანება

UNION ბრძანება აერთიანებს რამდენიმე მოთხოვნის შედეგს. მაგალითად, გვაინტერესებს თბილისელი სავაჭრო აგენტები და შემკვეთები. მოთხოვნას ექნება სახე:

```
SELECT snum, sname FROM Salespeople WHERE qalaqi = 'Tbilisi'
UNION
SELECT cnum, cname FROM Customers WHERE qalaqi = 'Tbilisi'
```

შედეგად, გაიცემა ჩანაწერები:

```
snum     sname
----------------------
101      ჯეონეთი
101      სამხარაძე
```

UNION ბრძანების გამოყენების დროს ორი წესი უნდა დავიცვათ:

- ყველა მოთხოვნაში ველების რაოდენობა და მიმდევრობა უნდა იყოს ერთნაირი.
- ველების ტიპები უნდა ემთხვეოდეს.

ტიპების თავსებადობა განისაზღვრება შემდეგი ცხრილის მიხედვით:

ცხრილი 18.3. ტიპების თავსებადობა

ველის მონაცემთა ტიპები	შედეგის ტიპი
ორივე ველი char ტიპისაა L1 და L2 ფიქსირებული სიგრძეებით	char ტიპი, რომლის სიგრძე L1-სა და L2-ს შორის უდიდესის ტოლია
ორივე ველი binary ტიპისაა L1 და L2 ფიქსირებული სიგრძეებით	binary ტიპი, რომლის სიგრძე L1-სა და L2-ს შორის უდიდესის ტოლია
ერთი ან ორივე ველი varchar ტიპისაა	varchar ტიპი, რომლის სიგრძე L1-სა და L2-ს შორის უდიდესის ტოლია
ერთი ან ორივე ველი varbinary ტიპისაა	varbinary ტიპი, რომლის სიგრძე L1-სა და L2-ს შორის უდიდესის ტოლია
ორივე ველი რიცხვითი ტიპისაა (smallint, money, float)	უდიდესი სიზუსტის მქონე მონაცემის ტიპი

UNION ბრძანება ავტომატურად გამორიცხავს ერთნაირ ჩანაწერებს. მაგრამ, თუ გვინდა ყველა ჩანაწერის მიღება, მაშინ უნდა გამოვიყენოთ UNION ALL ბრძანება:

```
SELECT snum, qalaqi FROM Customers
UNION ALL
SELECT snum, qalaqi FROM Salespeople
```

შედეგად, გაიცემა ჩანაწერები:

Snum	Qalaqi
101	თბილისი
103	ფოთი
102	ქობულეთი
102	ხაშური
101	სოხუმი
104	რუსთავი
105	ბათუმი
101	თბილისი
102	ქუთაისი
103	ბათუმი
104	თელავი
105	გორი

UNION ბრძანებასთან ერთად შეგვიძლია ORDER BY ოპერატორის გამოყენება შედეგის მოსაწესრიგებლად. ამასთან, ORDER BY სიტყვა ეთითება მხოლოდ UNION ბრძანების უკანასკნელი მოთხოვნის შემდეგ:

```
SELECT a.snum, sname, onum, 'udidesi', tarigi FROM Salespeople a, Orders b
WHERE a.snum = b.snum AND
        b.tanxa = ( SELECT MAX(tanxa) FROM Orders c WHERE c.tarigi = b.tarigi )
UNION
SELECT a.snum, sname, onum, 'umciresi', tarigi FROM Salespeople a, Orders b
WHERE a.snum = b.snum AND
        b.tanxa = ( SELECT MIN(tanxa) FROM Orders c WHERE c.tarigi = b.tarigi )
```

ORDER BY 3

შედეგად, გაიცემა ჩანაწერები:

snum	sname	onum		tarigi
105	მჭედლიშვილი	301	უმცირესი	1999/10/03
102	კაპანაძე	304	უდიდესი	1999/10/03
102	კაპანაძე	306	უმცირესი	1999/10/04
101	სამხარაძე	307	უმცირესი	1999/10/05
101	სამხარაძე	307	უდიდესი	1999/10/05
103	გაჩეჩილაძე	308	უდიდესი	1999/10/04
102	კაპანაძე	309	უმცირესი	1999/10/06
101	სამხარაძე	310	უდიდესი	1999/10/06

ORDER BY 3 საკვანძო სიტყვა შედეგებს დაალაგებს onum ველის ზრდადობის მიხედვით, რადგან ეს ველი რიგით მესამეა. ORDER BY 3-ის ნაცვლად შეგვიძლია, აგრეთვე, მივუთითოთ ORDER BY onum.

ჩანაწერების ჩამატება, წაშლა და შეცვლა

ჩანაწერების ჩამატება, წაშლა და შეცვლა სრულდება შემდეგი ბრძანებებით:

INSERT - ჩამატება

UPDATE - შეცვლა

DELETE - წაშლა

ჩანაწერების ჩამატება. ჩანაწერების ჩამატება სრულდება INSERT ბრძანების გამოყენებით. მისი სინტაქსია:

INSERT INTO table_name VALUES (value1, value2, ...)

აქ table_name ცხრილის სახელია, value1, value2 კი ჩასამატებელი სიდიდეები. მაგალითად, სავაჭრო აგენტების ცხრილში ჩანაწერების დასამატებლად უნდა გამოვიყენოთ ბრძანება:

INSERT INTO Salespeople VALUES (6, 'მჭედლიშვილი', 'ონი', 12)

თითოეული მნიშვნელობის ტიპი, VALUE სიტყვის შემდეგ მოთავსებულ ფრჩხილებში, უნდა ემთხვეოდეს ველის ტიპს, რომელშიც ხდება მისი ჩასმა. პირველი მნიშვნელობა ჩაიწერება პირველ ველში, მეორე მნიშვნელობა მეორე ველში და ა.შ. ცარიელი მნიშვნელობის ჩასასმელად ვუთითებთ NULL სიტყვას. მაგალითად,

INSERT INTO Salespeople VALUES (7, 'გასიტაშვილი', NULL, 12)

შეგვიძლია აშკარად მივუთითოთ ველები, რომლებშიც გვინდა მნიშვნელობების ჩასმა. ეს საშუალებას გვაძლევს მნიშვნელობები ჩავსვათ ნებისმიერი მიმდევრობით. მაგალითად,

INSERT INTO Customers (qalaqi, cname, cnum) VALUES ('ზესტაფონი', 'ნონიაშვილი', 8)

აქ გამოტოვებულია rating და snum ველები. მათ მიენიჭებათ ავტომატურად განსაზღვრული მნიშვნელობები. ჩვეულებრივ, ესაა NULL ან მნიშვნელობა, მითითებული ცხრილის შექმნისას.

INSERT ბრძანება შეგვიძლია, აგრეთვე, გამოვიყენოთ ცხრილში მოთხოვნის შედეგების ჩასასმელად. ამისათვის, VALUES სიტყვას ვცვლით შესაბამისი მოთხოვნით.

მაგალითად,
INSERT INTO TbilisiStaff SELECT * FROM Salespeople WHERE qalaqi = ‘თბილისი’

აქ ამოირჩევა Salespeople ცხრილის ყველა ჩანაწერი, რომლებშიც qalaqi = ‘თბილისი’ და ისინი მოთავსთდებიან TbilisiStaff ცხრილში. ამასთან, TbilisiStaff ცხრილი უნდა აკმაყოფილებდეს შემდეგ პირობებს:
- უნდა იყოს შექმნილი CREATE TABLE ბრძანებით.
- უნდა ჰქონდეს ოთხი ველი, რომლებიც ემთხვევა Salespeople ცხრილის ველების ტიპებს.

ცხრილიდან ჩანაწერების წაშლა. ცხრილიდან ჩანაწერების წასაშლელად გამოიყენება DELETE ბრძანება. ცხრილიდან ყველა ჩანაწერის წასაშლელად უნდა გამოვიყენოთ ბრძანება:
DELETE FROM Salespeople

კონკრეტული ჩანაწერების წასაშლელად უნდა მივუთითოთ პირობა. მაგალითად, 105 ნომრის მქონე სავაჭრო აგენტის წასაშლელად უნდა შევიტანოთ ბრძანება:
DELETE FROM Salespeople WHERE snum = 105

ველის მნიშვნელობის შეცვლა. UPDATE ბრძანებით შეგვიძლია შევცვალოთ ერთი ან მეტი ველის მნიშვნელობა. მაგალითად, ყველა შემკვეთის რეიტინგის 200-ით შესაცვლელად უნდა შევიტანოთ ბრძანება:
UPDATE Customers SET rating = 200

UPDATE ბრძანებაში შეგვიძლია პირობის მითითებაც. მაგალითად, 101 კოდის მქონე სავაჭრო აგენტის რეიტინგის შესაცვლელად უნდა შევიტანოთ ბრძანება:
UPDATE Customers SET rating = 300 WHERE snum = 101

SET სიტყვის შემდეგ შეიძლება მძიმეებით გამოყოფილი რამდენიმე ველის მითითებაც. მაგალითად,
UPDATE Customers SET qalaqi = ‘samtredia’, rating = 400 WHERE snum = 105

ტრანზაქცია

პრაქტიკული ამოცანების გადაწყვეტისას, ხშირად საჭირო ხდება რამდენიმე SQL-მოთხოვნის ერთდროულად შესრულება. მაგალითად, საბანკო გადარიცხვის შემთხვევაში საჭიროა ერთი ანგარიშიდან მოვხსნათ თანხა და დავდოთ ის სხვა ანგარიშზე. თუ ჩვენ გვაქვს ორი სხვადასხვა UPDATE-მოთხოვნა, რომლებიც მოხსნიან და დადებენ თანხას, მაშინ საჭიროა, რომ ორივე მოთხოვნის შედეგი გაფორმებული იყოს როგორც ერთი მთლიანი. თუ რაიმე მიზეზის გამო ერთ-ერთი მოთხოვნა ვერ შესრულდა, აუცილებელია ორივე მოთხოვნის შედეგების გაუქმება.

SQL-მოთხოვნების მუშაობის შედეგების ჩაწერა ცნობილია როგორც SQL-მოთხოვნების ფიქსაცია (commit ან committing). SQL-მოთხოვნის შედეგების გაუქმება ცნობილია როგორც უკუქცევა (rollback ან rolling back).

SQL-მოთხოვნები შეგვიძლია დავაჯგუფოთ ტრანზაქციებში, შემდეგ კი დავაფიქსიროთ ან უკუვაქციოთ ეს ტრანზაქცია როგორც ერთი მთლიანი. ამრიგად, ტრანზაქცია არის პროცესი, როცა ან შესრულდება მასში მოთავსებული ყველა SQL-მოთხოვნა ან არ შესრულდება არც ერთი მათგანი. მაგალითად, საბანკო მაგალითში ორი UPDATE-მოთხოვნა შეგვიძლია მოვათავსოთ ტრანზაქციაში, შემდეგ კი დავაფიქსიროთ

ან უკუვაქციოთ ეს ტრანზაქცია როგორც ერთი მთლიანი, იმაზე დამოკიდებულებით, წარმატებით შესრულდა თუ არა ორივე SQL-მოთხოვნა.

ტრანზაქციის დასაწყებად გამოიყენება BEGIN TRANSACTION ოპერატორი (შემოკლებით BEGIN TRANS). შემდეგ მოდიან SQL-მოთხოვნის შემადგენელი ოპერატორები. ტრანზაქციის დასაფიქსირებლად გამოიყენება COMMIT TRANSACTION მოთხოვნა (შემოკლებით COMMIT TRANS). ტრანზაქციის უკუქცევისათვის გამოიყენება ROLLBACK TRANSACTION მოთხოვნა (შემოკლებით ROLLBACK TRANS ან ROLLBACK).

განვიხილოთ მაგალითი. მოყვანილი ტრანზაქცია შედგება ორი INSERT-მოთხოვნისაგან: პირველი ჩანაწერს ამატებს Customers ცხრილში, მეორე კი - Orders ცხრილში. ტრანზაქციას ექნება სახე:

```
BEGIN TRANSACTION;
INSERT INTO Customers (Cnum, Qalaqi) VALUES (208, 'ბორჯომი');
INSERT INTO Orders (Onum, Cnum)      VALUES (311, 208);
COMMIT;
```

ნახ. 18.3-ზე ნაჩვენებია ამ ტრანზაქციის მუშაობის შედეგები, ორ SELECT-მოთხოვნასთან ერთად, რომლებიც გვიჩვენებენ ორ ახალ ჩანაწერს.

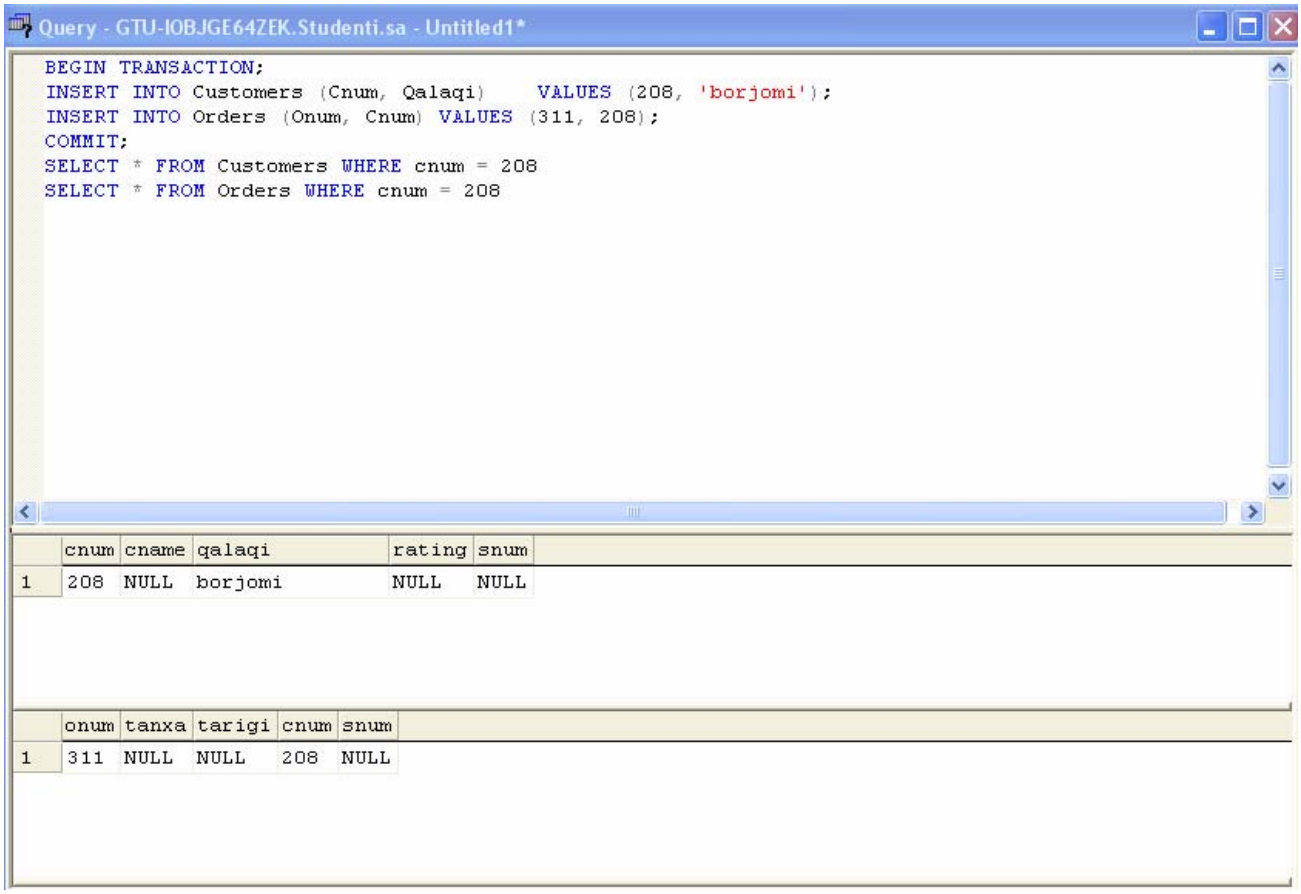
მომდევნო ტრანზაქცია შედგება ანალოგიური INSERT-მოთხოვნებისაგან, მაგრამ ეს ტრანზაქცია უკუიქცევა ROLLBACK მოთხოვნის მიერ:

```
BEGIN TRANSACTION;
INSERT INTO Customers (Cnum, Qalaqi)    VALUES (209, 'ბაკურიანი');
INSERT INTO Orders (Onum, Cnum)      VALUES (312, 209);
ROLLBACK;
```

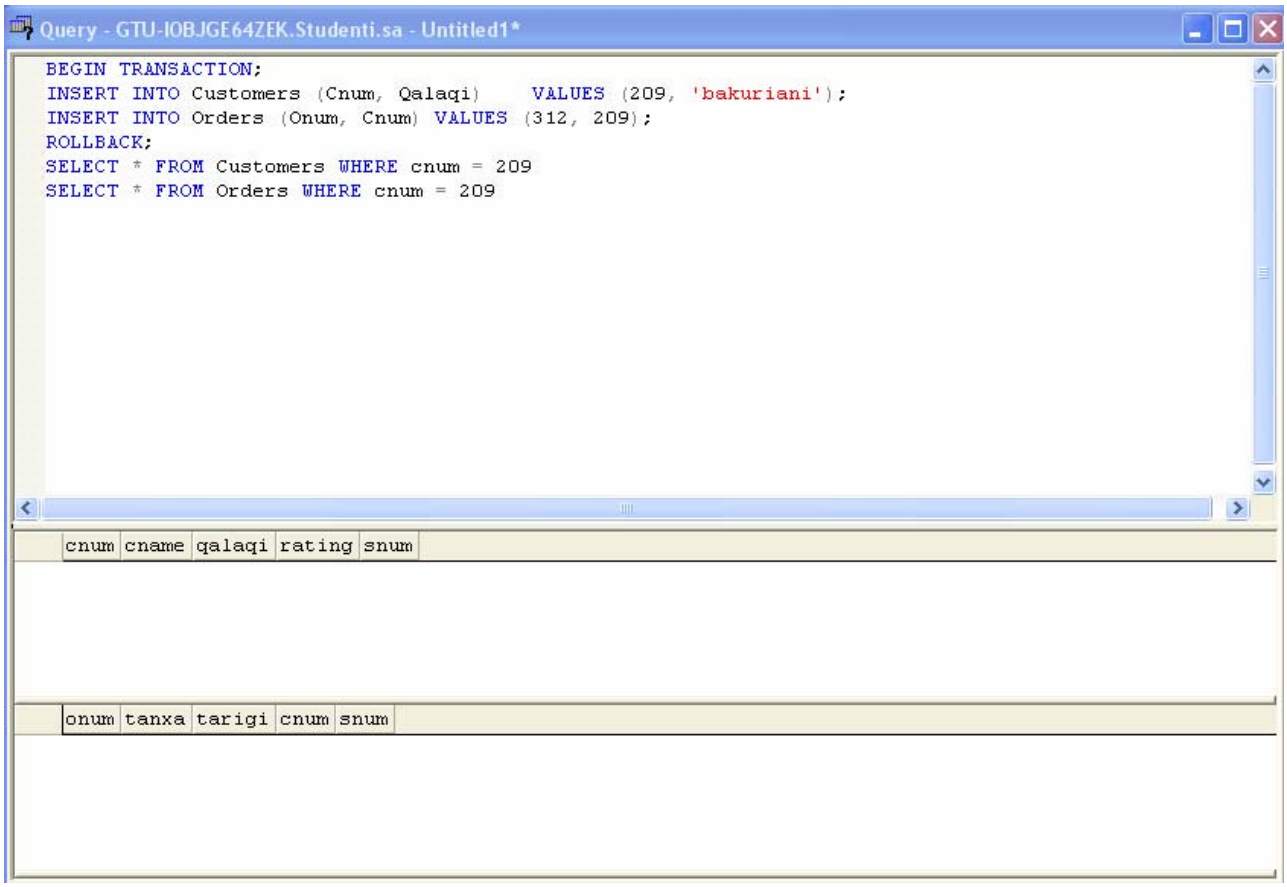
რადგან ტრანზაქცია უკუიქცევა, ამიტომ ორი ჩანაწერი, დამატებული INSERT-მოთხოვნის მიერ, წაიშლება. ნახ. 18.4-ზე ნაჩვენებია ეს ტრანზაქცია ორ SELECT-მოთხოვნასთან ერთად, საიდანაც ჩანს, რომ ეს ჩანაწერები არ დაემატა.

ტრანზაქცია უნდა შევამოწმოთ შეცდომის არსებობაზე, სანამ დავაფიქსირებთ ან უკუვაქცევთ მას, რადგან შეცდომები ყოველთვის არ წყვეტენ მიმდინარე ჩანაწერის დამუშავებას. ამისათვის, SQL-სერვერში გამოიყენება @@ERROR ცვლადი. ამ ცვლადის მნიშვნელობა ნულის ტოლია ყოველთვის, როცა SQL-მოთხოვნა შესრულდა და არ მიგვიყვანა შეცდომამდე. თუ @@ERROR ცვლადი შეიცავს არანულოვან მნიშვნელობას, მაშინ ეს იმას ნიშნავს, რომ ადგილი ჰქონდა შეცდომას. შესაბამისად, ტრანზაქცია უნდა დავაფიქსიროთ მაშინ, როცა @@ERROR ცვლადის მნიშვნელობა ნულის ტოლია. წინააღმდეგ შემთხვევაში, უნდა მოვახდინოთ ტრანზაქციის უკუქცევა.

BEGIN TRANSACTION მოთხოვნაში ტრანზაქციას შეგვიძლია სახელი დავარქვათ. ეს სასარგებლოა, რადგან შეგვიძლია გავიგოთ თუ რომელ ტრანზაქციაში ვმუშაობთ მიმდინარე მომენტში.



ნახ. 18.3.

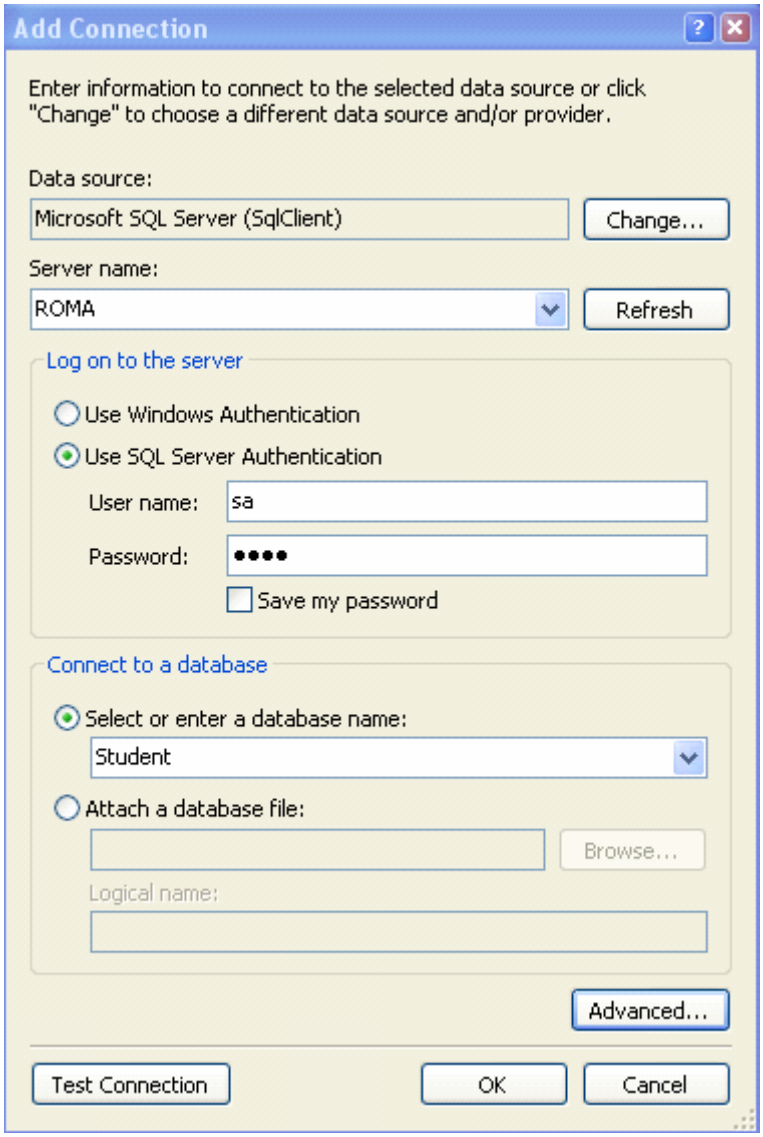


ნახ. 18.4.

ქვემოთ მოყვანილია ტრანზაქციისათვის სახელის მინიჭებისა და @@ERROR ცვლადის გამოყენების მაგალითი:

```
BEGIN TRANSACTION MyTransaction;
INSERT INTO Customers (Cnum, Qalaqi) VALUES (210, 'ზესტაფონი');
INSERT INTO Orders (Onum, Cnum)      VALUES (313, 210);
IF @@ERROR = 0
        COMMIT TRANSACTION MyTransaction;
ELSE    ROLLBACK TRANSACTION MyTransaction;
```

ყურადღება მივაქციოთ იმას, რომ ტრანზაქციის სახელი MyTransaction გამოიყენება COMMIT და ROLLBACK მოთხოვნებში.



ნახ. 18.5.

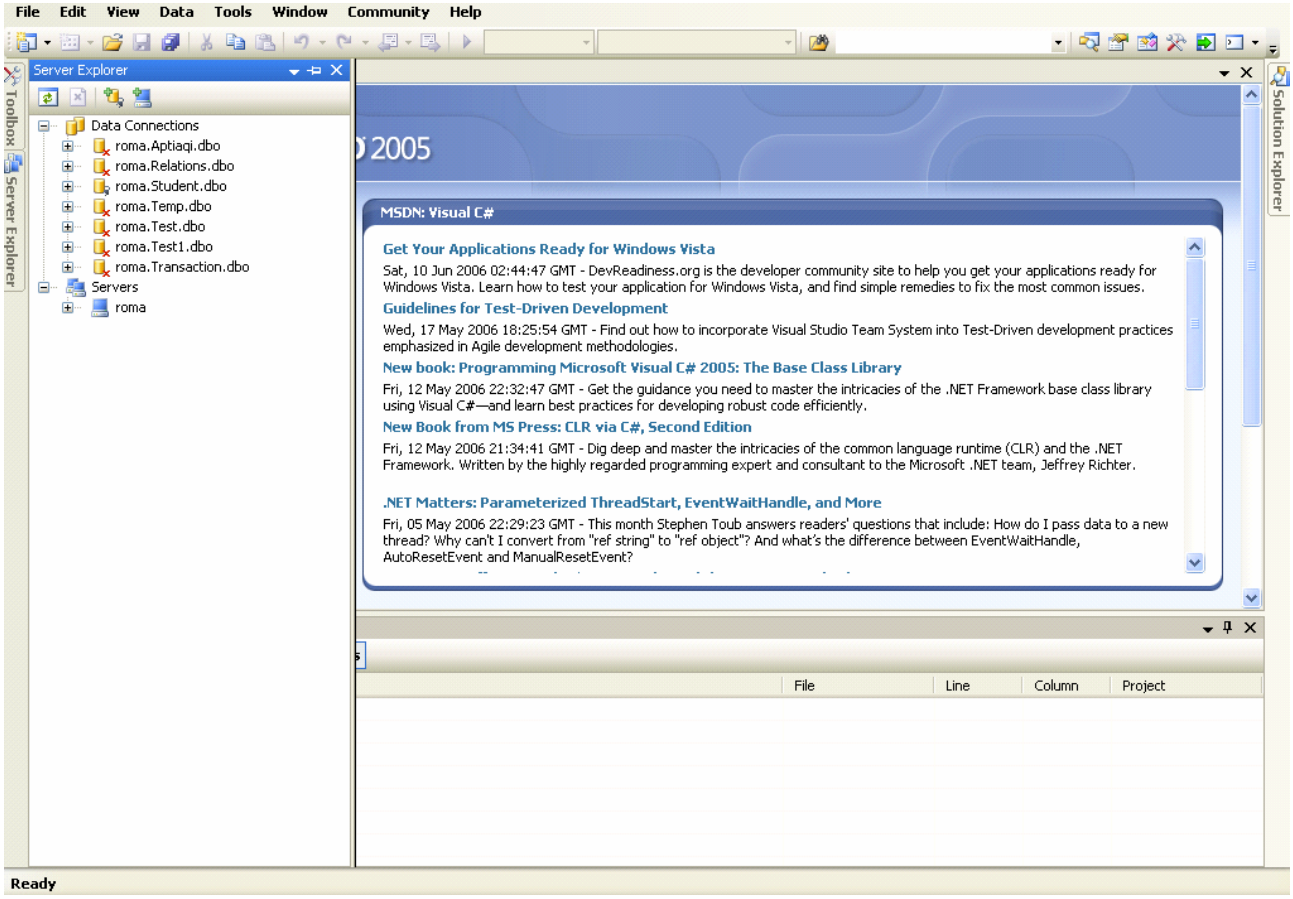
მონაცემთა ბაზასთან მიმართვა Visual Studio .NET-ის საშუალებებით

Visual Studio .NET გარემოს აქვს ფუნქციები, რომლებიც საშუალებას იძლევიან დავუკავშირდეთ მონაცემთა ბაზას და ვიმუშაოთ მის ცხრილებთან. ეს ფუნქციები თავმოყრილია Server Explorer ფანჯარაში.

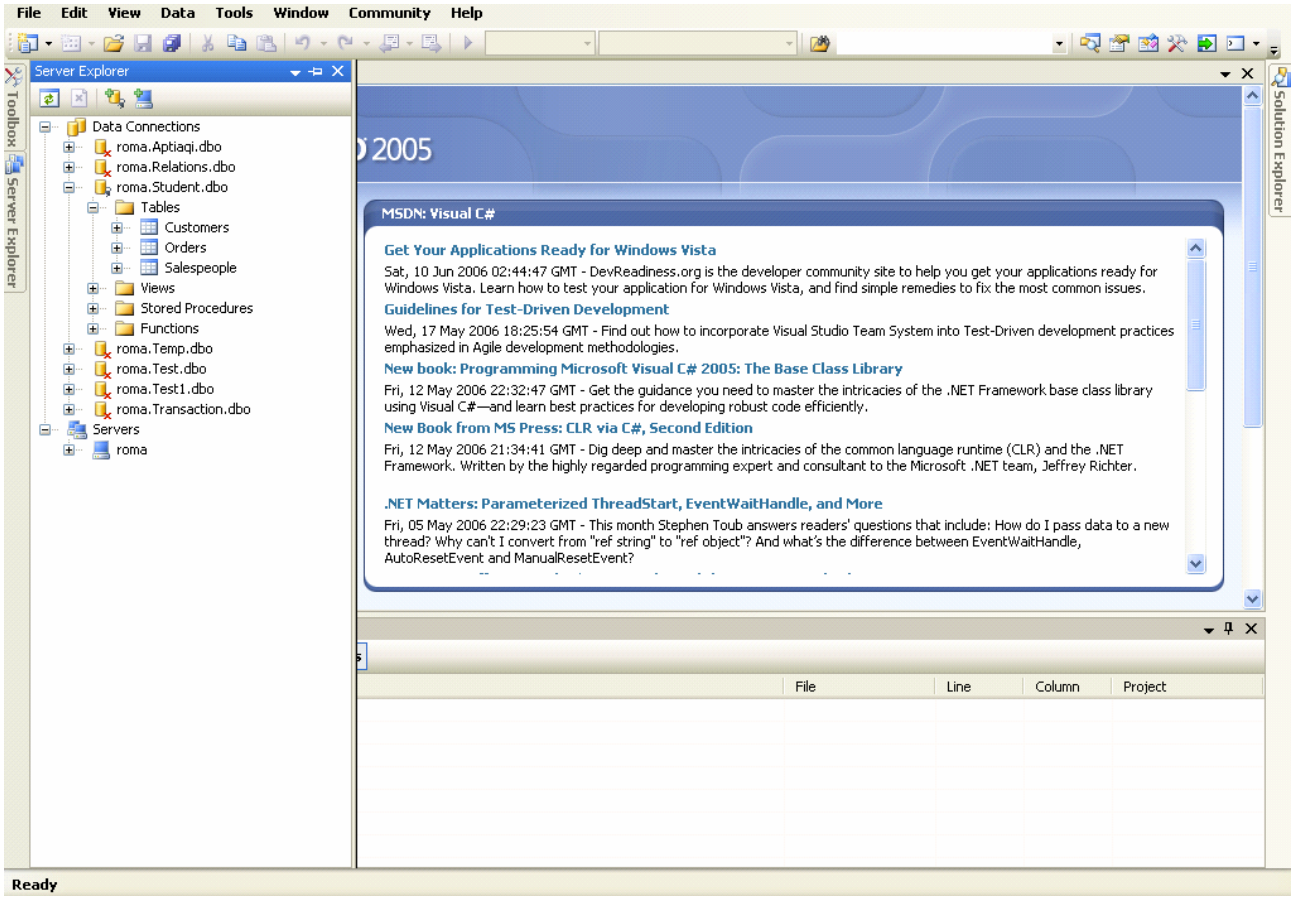
უწინარეს ყოვლისა, მონაცემთა ბაზასთან უნდა დავამყაროთ შეერთება (კავშირი). ამისათვის, ჯერ უნდა გავუშვათ Visual Studio .NET სისტემა, შემდეგ კი შევასრულოთ Tools მენიუს Connect to Database ბრძანება. გაიხსნება Add Connection ფანჯარა (ნახ. 18.5). Server name ველში შეგვაქვს სერვერის (კომპიუტერის) სახელი. შემდეგ უნდა ჩავრთოთ Use SQL Server Authentication გადამრთველი და User name და Password ველებში შევიტანოთ შესაბამისი ინფორმაცია. საჭირო ბაზას ვირჩევთ Select or enter a database name გაშლადი სიიდან. ვაჭერთ Ok კლავიშს.

დაბოლოს, როცა მონაცემთა ბაზასთან შეერთება დამყარებულია, შეგვიძლია ცხრილების ნახვა და ცვლილება. ცხრილებთან მიმართვისათვის ვხსნით Server Explorer ფანჯარას (ნახ. 18.6). რისთვისაც ერთხელ ვაჭერთ გახსნილი ფანჯრის მარცხნივ მოთავსებულ Server Explorer პანელს. ამ ფანჯრის კატალოგების ხის უბანში ჯერ ვხსნით Roma.Student.dbo განშტოებას, შემდეგ კი Tables განშტოებას (ნახ. 18.7). Customers ცხრილიდან ჩანაწერების მისაღებად ისარს ვათავსებთ ამ ცხრილის პიქტოგრამაზე, ვხსნით კონტექსტურ მენიუს და ვასრულებთ Show Table Data ბრძანებას. ნახ. 18.8-ზე ნაჩვენებია ჩანაწერები Customers ცხრილიდან.

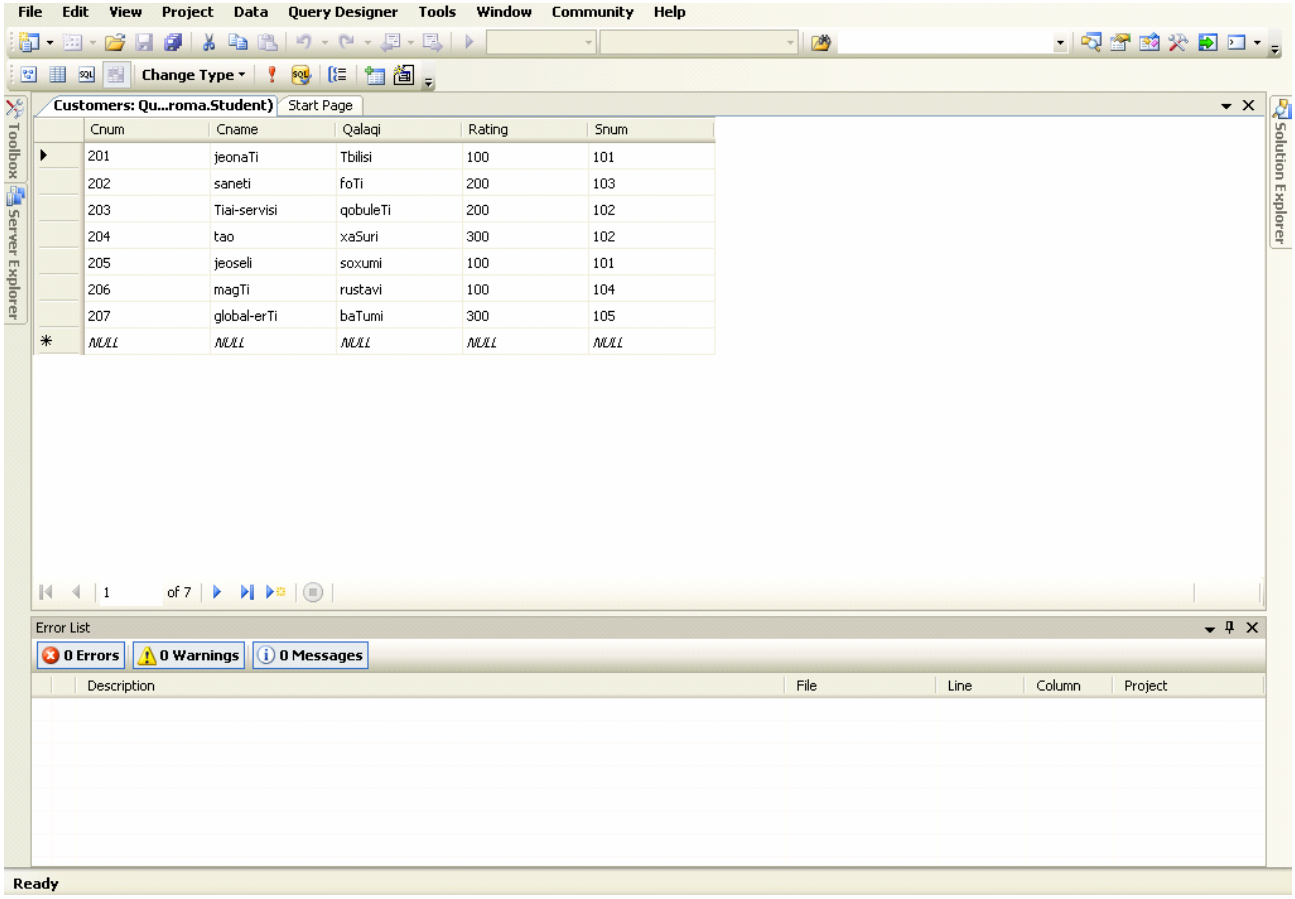
SQL-მოთხოვნის შესატანად ვაჭერთ ინსტრუმენტების პანელის Show SQL Pane კლავიშს (ნახ. 18.9). SQL-მოთხოვნა შეგვიძლია ავაგოთ ვიზუალურად თუ დავაჭერთ Show Diagram Pane კლავიშს (ნახ. 18.10). შეგვიძლია ველების თვისებების ნახვა თუ მოვნიშნავთ საჭირო ველს, გავხსნით კონტექსტურ მენიუს და შევასრულებთ Properties ბრძანებას.



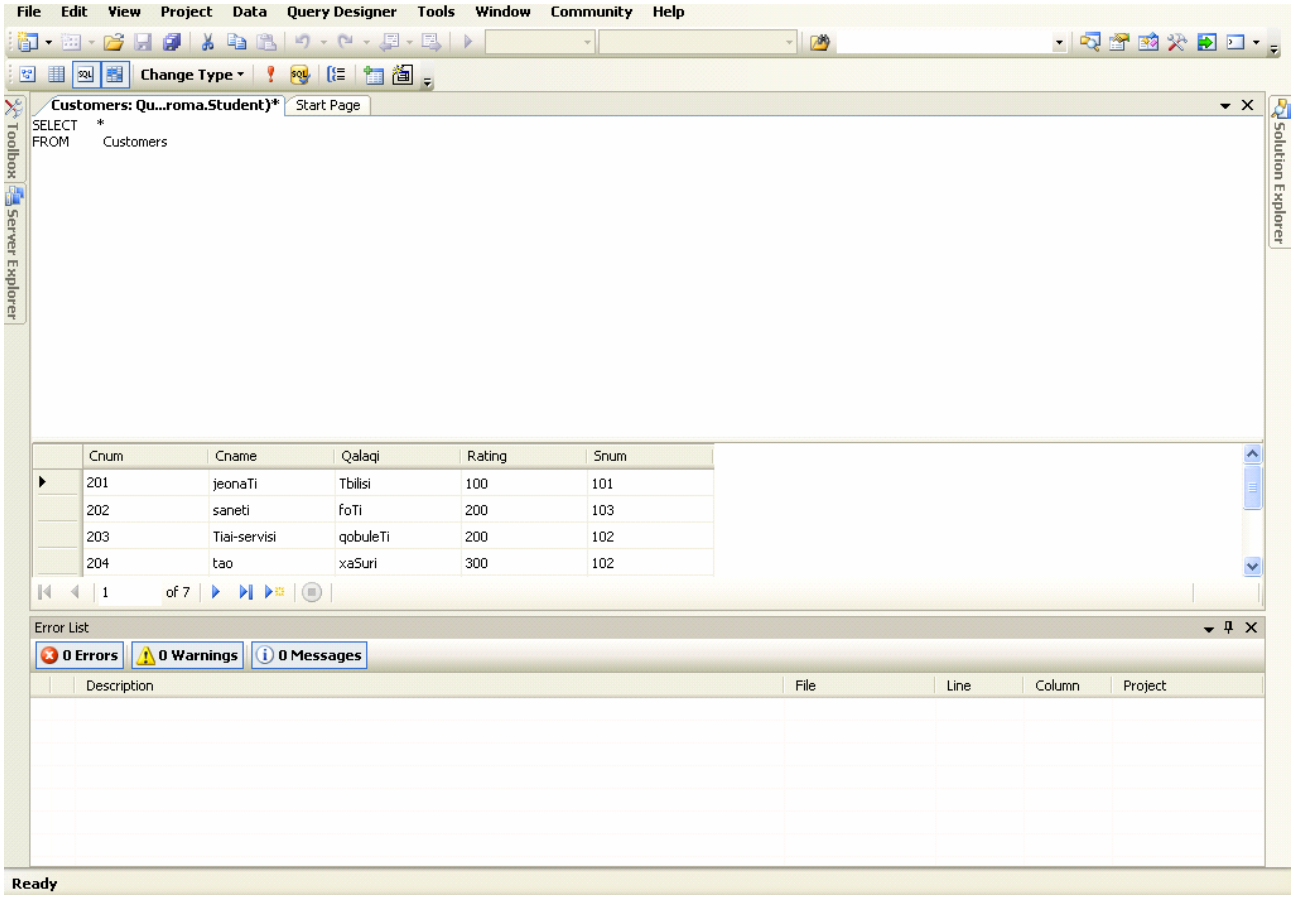
ნახ. 18.6.



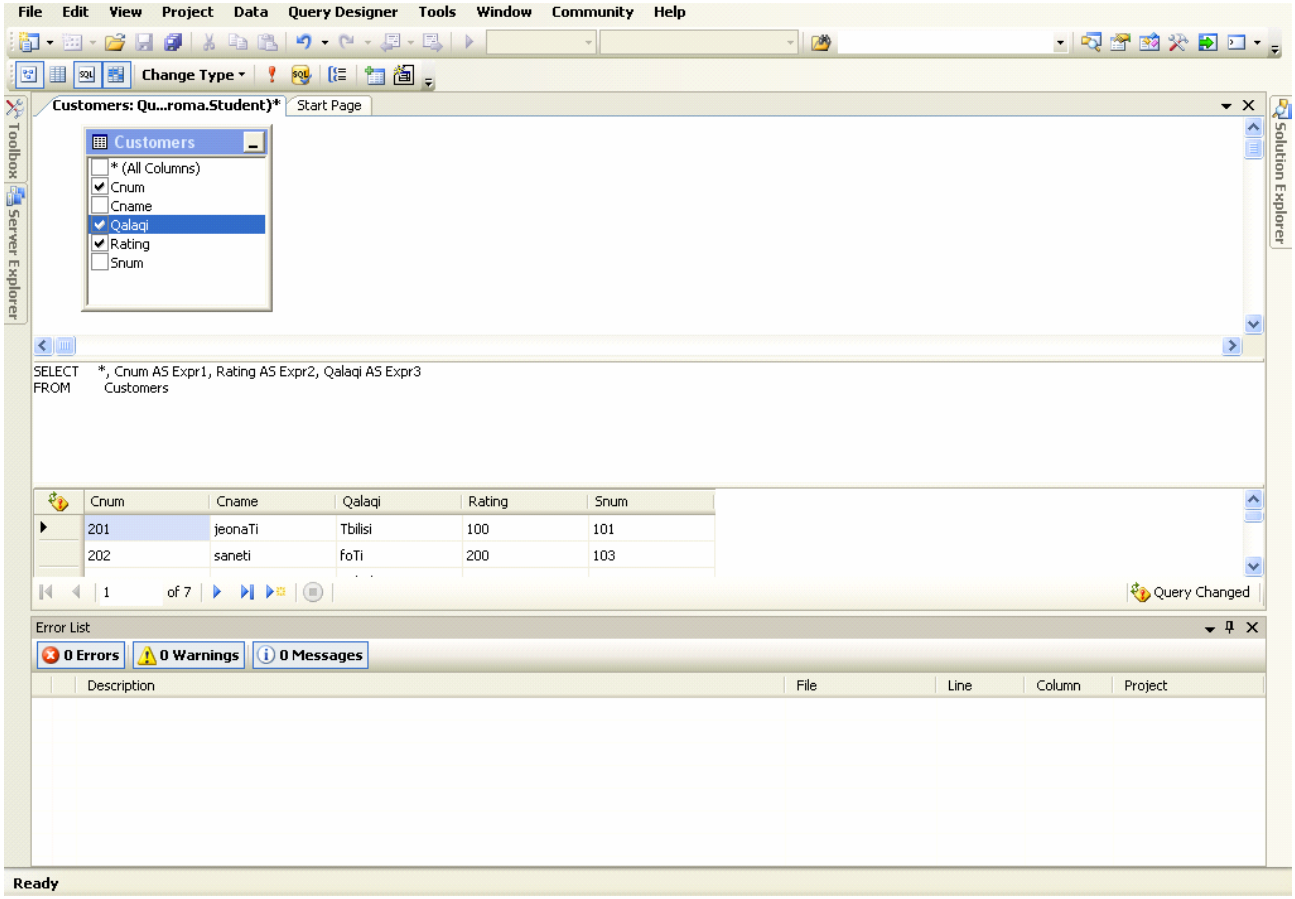
ნახ. 18.7.



ნახ. 18.8.



ნახ. 18.9.


File Edit View Project Data Query Designer Tools Window Community Help
Change Type
Customers: Qu...roma.Student)*
Start Page
Customers
* (All Columns)
Cnum
Cname
Qalaqi
Rating
Snum
SELECT *, Cnum AS Expr1, Rating AS Expr2, Qalaqi AS Expr3
FROM Customers
Cnum Cname Qalaqi Rating Snum
201 jeonaTi Tbilisi 100 101
202 saneti foTi 200 103
1 of 7
Query Changed
Error List
0 Errors
0 Warnings
0 Messages
Description File Line Column Project
Ready

ნახ. 18.10.

თავი 19. ADO.NET ბიბლიოთეკა

შესავალი

ამ თავში ვნახავთ თუ როგორ უნდა ვიმუშაოთ მონაცემთა ბაზასთან C#-პროგრამიდან Active Data Objects ბიბლიოთეკის საშუალებით .NET Framework პლატფორმისათვის (შემოკლებით ADO.NET).

მონაცემთა ბაზასთან მიმართვა ADO.NET ბიბლიოთეკის საშუალებით შემდეგნაირად ხორციელდება: C#-პროგრამა უერთდება მონაცემთა ბაზას, იღებს საჭირო ინფორმაციას და ამ ინფორმაციას ათავსებს იმ კომპიუტერის მეხსიერებაში, რომელზეც ეს პროგრამა მუშაობს. როცა ინფორმაცია იმყოფება მეხსიერებაში, მასზე შეგვიძლია სხვადასხვა მანიპულაციები შევასრულოთ. მაგალითად, შეგვიძლია ავსახოთ, დავუმატოთ, შევცვალოთ ან წავშალოთ არსებული ჩანაწერები. ამასთან, ყველაფერი ეს შეგვიძლია გავაკეთოთ არა ერთ, არამედ რამოდენიმე ცხრილთან. პროგრამა პერიოდულად დაუკავშირდება მონაცემთა ბაზას იმ ცვლილებების შესანახად, რომელიც ჩვენ შევიტანეთ მონაცემების ლოკალურ ასლში. ლოკალური ასლი არის ის ინფორმაცია, რომელიც აღებულია სერვერიდან და დროებით ინახება იმ კომპიუტერის მეხსიერებაში, რომელზეც ჩვენი პროგრამა მუშაობს.

ADO.NET უზრუნველყოფს მონაცემთა ბაზასთან მიმართვის გამორთულ მოდელს. სხვა სიტყვებით რომ ვთქვათ, ჩვენთვის საჭირო ინფორმაცია, ინახება იმ კომპიუტერზე, რომელზეც C#-პროგრამა მუშაობს.

ჩვეულებრივ, მონაცემთა ბაზა იმყოფება იმ კომპიუტერზე, რომელზეც ჩვენი პროგრამა მუშაობს. თუმცა, ყველაფერი დამოკიდებულია გადასაწყვეტი ამოცანის მოთხოვნებზე. უმჯობესია მონაცემთა ბაზის მოთავსება უფრო მძლავრ კომპიუტერზე (სერვერზე), რომელსაც შეეძლება ერთდროულად მოემსახუროს იმ პროგრამებსა და მომხმარებლებს, რომლებიც SQL-მოთხოვნებს აგზავნიან.

ADO.NET კლასები

მონაცემთა ბაზასთან მიმართვის გამორთული მოდელის უზრუნველსაყოფად ADO.NET ბიბლიოთეკაში განსაზღვრულია კლასების ორი სიმრავლე:

- საერთო კლასები მონაცემებთან სამუშაოდ;
- მართვადი კლასები-პროვაიდერები.

პირველი სიმრავლის კლასების ობიექტები გამოიყენება მონაცემთა ბაზიდან მიღებული ინფორმაციის ლოკალური ასლის შესანახად. აქ ძირითადია System.Data.DataSet კლასი.

მეორე სიმრავლის კლასების ობიექტები გამოიყენება მონაცემთა ბაზასთან კავშირის უზრუნველყოფისათვის, DataSet ობიექტიდან ინფორმაციის წაკითხვისათვისა და ჩაწერისათვის. მართვადი ობიექტები უზრუნველყოფენ პირდაპირ მიმართვას მონაცემთა ბაზასთან და იძლევიან ბაზაში არსებულ მონაცემებთან ლოკალური მონაცემების სინქრონიზების საშუალებას.

მონაცემებთან სამუშაო კლასები და ობიექტები

განვიხილოთ მონაცემებთან სამუშაო ზოგიერთი კლასი.

DataSet კლასი. ამ კლასის ობიექტი გამოიყენება მონაცემთა ბაზაში მოთავსებული ინფორმაციის ლოკალური ასლის შესანახად. მასში შესაძლებელია მონაცემების ისეთი სტრუქტურების წარმოდგენა, როგორიცაა ცხრილები, ჩანაწერები, ველები და ა.შ. DataSet ობიექტები შეიძლება, აგრეთვე, გამოვიყენოთ მონაცემების წარმოსადგენად XML-ფაილებიდან.

DataTable კლასი. ამ კლასის ობიექტი გამოიყენება ცხრილის წარმოსადგენად. ერთ DataSet ობიექტში შეიძლება რამდენიმე DataTable ობიექტის მოთავსება.

DataRow კლასი. ამ კლასის ობიექტი გამოიყენება ჩანაწერების წარმოსადგენად. ერთ DataTable ობიექტში შეიძლება რამდენიმე DataRow ობიექტის მოთავსება.

DataColumn კლასი. ამ კლასის ობიექტი გამოიყენება ველის წარმოსადგენად. ერთ DataRow ობიექტში შეიძლება რამდენიმე DataColumn ობიექტის მოთავსება.

DataRelation კლასი. ამ კლასის ობიექტი გამოიყენება ორ DataTable ობიექტს შორის კავშირების წარმოსადგენად (დასამყარებლად). DataRelation ობიექტები შეგვიძლია გამოვიყენოთ მონაცემთა ბაზაში ორ ცხრილს შორის "მშობელი-შვილი" მიმართების შესაქმნელად. ერთ DataSet ობიექტში შეიძლება რამდენიმე DataRelation ობიექტის მოთავსება.

Constraint კლასი. ამ კლასის ობიექტი გამოიყენება მონაცემთა ბაზის შეზღუდვების წარმოსადგენად, როგორიცაა ველის მნიშვნელობის შეზღუდვა უნიკალურობაზე ან გარე გასაღების შეზღუდვა, რომელიც სხვა ცხრილზე მიუთითებს. ერთ DataTable ობიექტში შეიძლება მოვათავსოთ რამდენიმე Constraint ობიექტი.

DataView კლასი. ამ კლასის ობიექტი გამოიყენება DataTable ობიექტის ველების ქვესიმრავლის სანახავად, რომლებიც მიიღება ფილტრაციის შედეგად. ერთი DataSet ობიექტი შეიძლება შეიცავდეს რამდენიმე DataView ობიექტს.

DataSet, DataTable, DataRow, DataColumn, DataRelation, Constraint და DataView კლასები განსაზღვრულია System.Data სახელების სივრცეში.

მართვადი კლასები და ობიექტები

განვიხილოთ ზოგიერთი მართვადი კლასი.

SqlConnection, OleDbConnection, OdbcConnection კლასები. SqlConnection კლასის ობიექტი გამოიყენება SQL-სერვერზე მონაცემთა ბაზასთან შეერთების დასამყარებლად. OleDbConnection კლასის ობიექტი გამოიყენება იმ მონაცემთა ბაზის მართვის სისტემასთან მისაერთებლად, რომელიც უზრუნველყოფს OLEDB ტექნოლოგიას (Object Linking and Embedding for Databases - ობიექტის დაკავშირება და ჩადგმა მონაცემთა ბაზებისათვის), მაგალითად, Oracle და Access. OdbcConnection კლასის ობიექტი გამოიყენება იმ მონაცემთა ბაზის მართვის სისტემასთან მისაერთებლად, რომელიც უზრუნველყოფს ODBC (Open Database Connectivity - მონაცემთა ბაზასთან მიმართვის ღია ინტერფეისი) ტექნოლოგიას. ყველა ძირითადი მონაცემთა ბაზა უზრუნველყოფს ODBC ტექნოლოგიას, მაგრამ მისი საშუალებით მონაცემთა ბაზასთან მიმართვა .NET

სისტემაში ჩვეულებრივ ნელა სრულდება, ვიდრე წინა ორი საშუალებით.

SqlCommand, OleDbCommand, OdbcCommand კლასები. ამ კლასების ობიექტები გამოიყენება SQL-მოთხოვნის ფორმირებისათვის ან შენახული პროცედურის გამოძახებისათვის. ფორმირებული მოთხოვნა ან პროცედურის გამოძახება შეიძლება შესრულდეს წინა აბზაცში ჩამოთვლილი კლასებიდან ერთ-ერთის საშუალებით.

SqlDataReader, OleDbDataReader, OdbcDataReader კლასები. ამ კლასების ობიექტები გამოიყენება ინფორმაციის წასაკითხად მონაცემთა ბაზიდან, რომელიც მოთავსებულია Microsoft SQL-სერვერზე ან მუშაობს იმ მონაცემთა ბაზის მართვის სისტემის ქვეშ, რომელიც უზრუნველყოფს OLEDB ან ODBC ტექნოლოგიას. ეს ობიექტები გამოიყენება მონაცემების მხოლოდ წასაკითხად მონაცემების წყაროდან და წარმოადგენენ DataSet ობიექტის ალტერნატივას. მონაცემების წაკითხვა ამ ობიექტების საშუალებით, როგორც წესი, უფრო სწრაფად ხდება, ვიდრე DataSet ობიექტის გამოყენებით.

SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter კლასები. ამ კლასების ობიექტები გამოიყენება ჩანაწერების გადასაადგილებლად DataSet ობიექტსა და მონაცემთა ბაზას შორის, რომელიც მოთავსებულია SQL-სერვერზე, ან მუშაობს იმ მონაცემთა ბაზის მართვის სისტემის ქვეშ, რომელიც უზრუნველყოფს OLEDB ან ODBC ტექნოლოგიას.

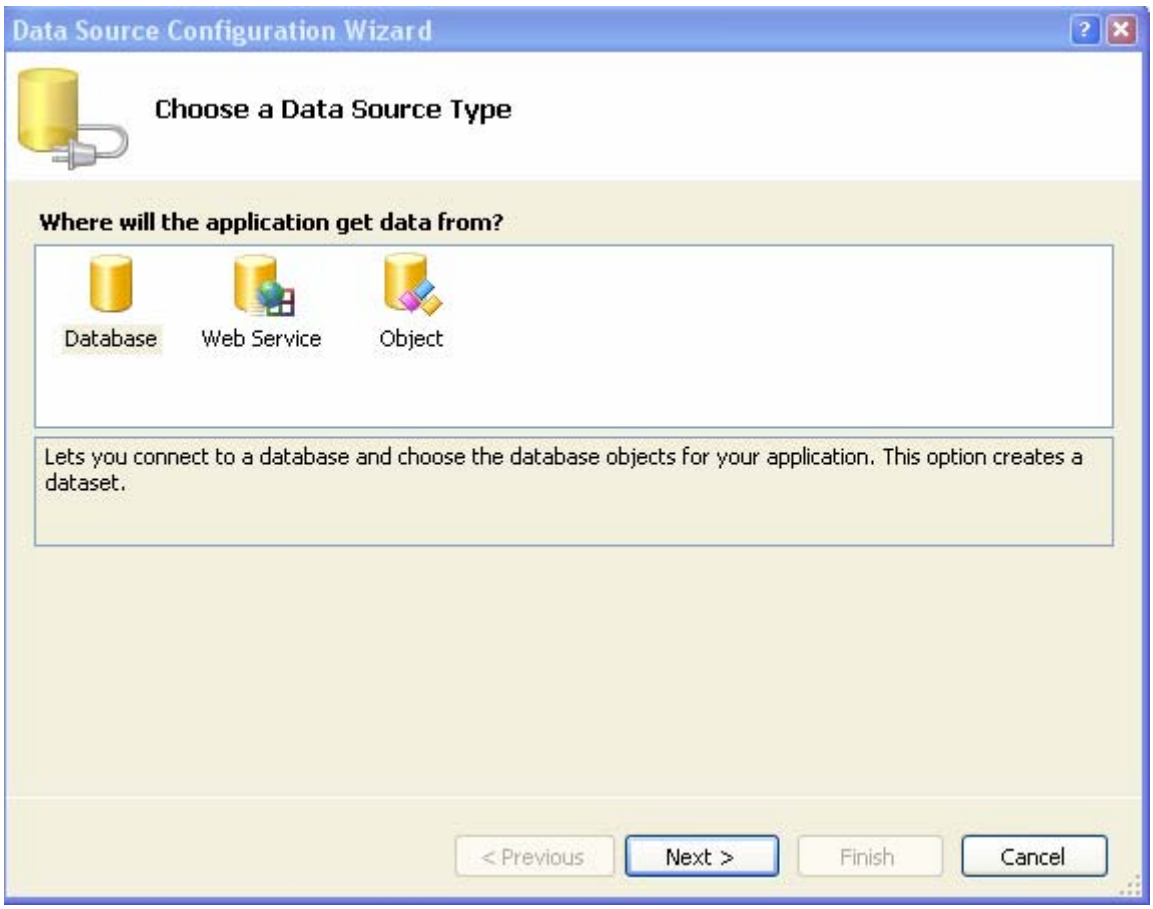
მართვადი კლასები Microsoft SQL-სერვერისათვის გამოცხადებულია System.Data.SqlClient სახელების სივრცეში. კლასები მონაცემთა ბაზებისათვის, რომლებიც უზრუნველყოფენ ODBC ტექნოლოგიას, გამოცხადებულია System.Data.Odbc სახელების სივრცეში.

სანამ დაწვრილებით განვიხილავდეთ ADO.NET კლასებს, განვიხილოთ მარტივი მაგალითი, რომელშიც ნაჩვენებია SELECT-მოთხოვნის შესრულება.

მონაცემთა ბაზასთან მუშაობა ვიზუალური და არავიზუალური კომპონენტების გამოყენებით

ამ განყოფილებაში მოყვანილ მაგალითში ნაჩვენებია თუ როგორ უნდა დავამყაროთ კავშირი SQL-სერვერზე მოთავსებულ Student მონაცემთა ბაზსთან და როგორ ავსახოთ ცხრილების სტრიქონები ფორმაზე მოთავსებულ კომპონენტებში.

გავხსნათ ახალი პროექტი. გამოჩნდება Form1 კომპონენტი. ToolBox ფანჯრის AllWindowsForms განყოფილებიდან BindingSource კომპონენტი მოვათავსოთ ფორმაზე. მას დაერქმევა bindingSource1 და გამოჩნდება ფორმის ქვედა ზონაში, რადგან ის არავიზუალური კომპონენტია. მოვნიშნოთ იგი. გავხსნათ Properties ფანჯრის DataSource სია და დავაჭიროთ Add Project Data Source მიმართვას.



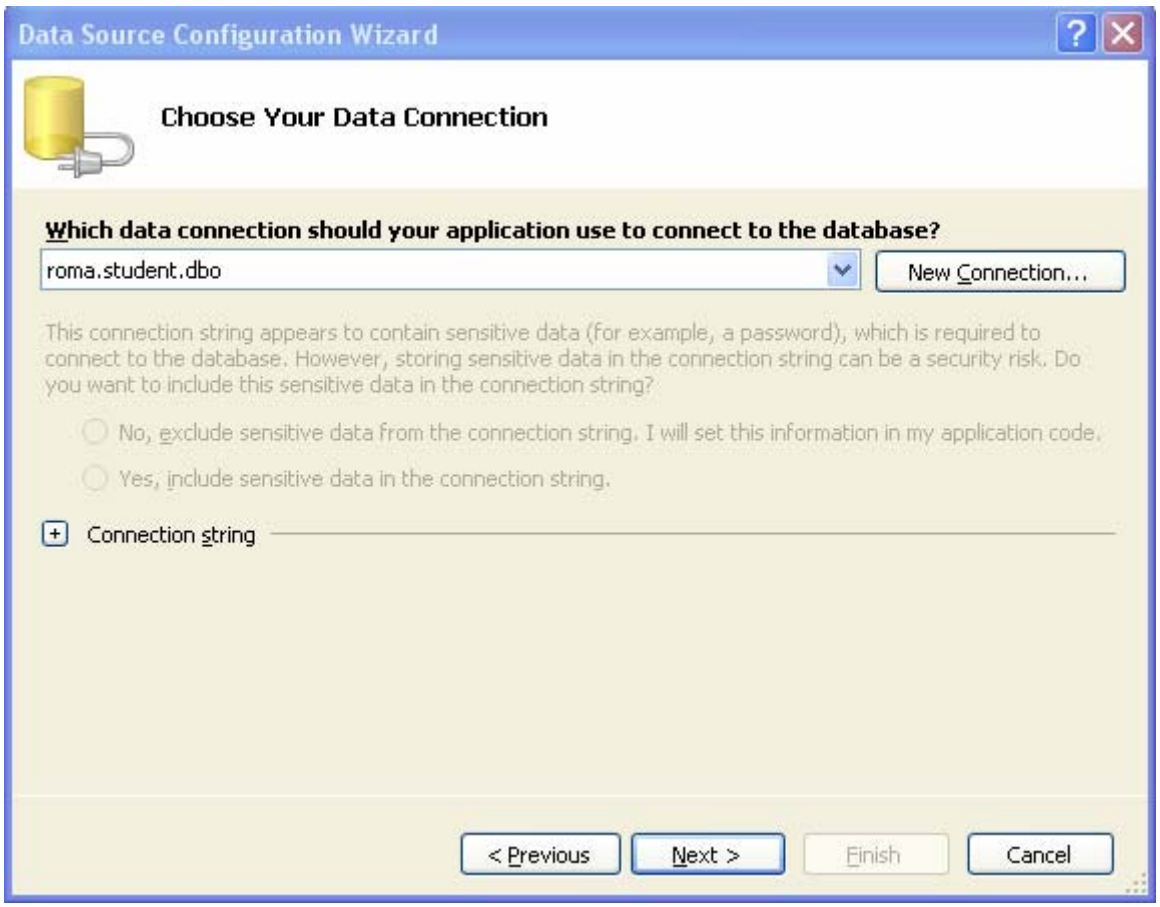
ნახ. 19.1

გაიხსნება ფანჯარა (ნახ. 19.1), რომელშიც ვირჩევთ Database მონაცემთა წყაროს და ვაჭერთ Next კლავიშს.

მომდევნო ფანჯარაში (ნახ. 19.2) ვხსნით Which data connection ... სიას და ვირჩევთ მონაცემთა ბაზას. თუ ამ სიაში არ არის ჩვენთვის საჭირო ბაზა, მაშინ ვაჭერთ New Connection კლავიშს. გაიხსნება ფანჯარა(ნახ. 19.3). Server name ველში შევიტანოთ სერვერის (კომპიუტერის) სახელი. ჩავრთოთ Use SQL Server Authentication გადამრთველი. User name და Password ველებში შევიტანოთ მომხმარებლის სახელი და პაროლი. Select or enter a database name სიიდან ავირჩიოთ საჭირო მონაცემთა ბაზის სახელი და დავაჭიროთ OK კლავიშს.

ვუბრუნდებით წინა ფანჯარას (ნახ. 19.4). დავაჭიროთ Connection string სტრიქონის მარცხნივ მოთავსებულ + ნიშანს. გამოჩნდება მონაცემთა ბაზასთან შეერთების სტრიქონი, რომელშიც სამი პარამეტრი ჩანს: სერვერის (კომპიუტერის) სახელი - Roma, მონაცემთა ბაზის სახელი - Student და მომხმარებლის სახელი - sa. ჩავრთოთ Yes, include sensitive data in the connection string გადამრთველი (ნახ. 19.5). შედეგად, შეერთების სტრიქონს დაემატება მეოთხე პარამტერი - პაროლი. ვაჭერთ Next კლავიშს.

გახსნილ ფანჯარაში (ნახ. 19.6) გამოჩნდება შეერთების სახელი. სურვილის შემთხვევაში შეგვიძლია მისი შეცვლა. თუ Yes, save the connection as გადამრთველს ჩართულს დავტოვებთ, მაშინ შეერთების სტრიქონი შეინახება ჩვენი პროგრამის საკონფიგურაციო ფაილში. ვაჭერთ Next კლავიშს.



ნახ. 19.2

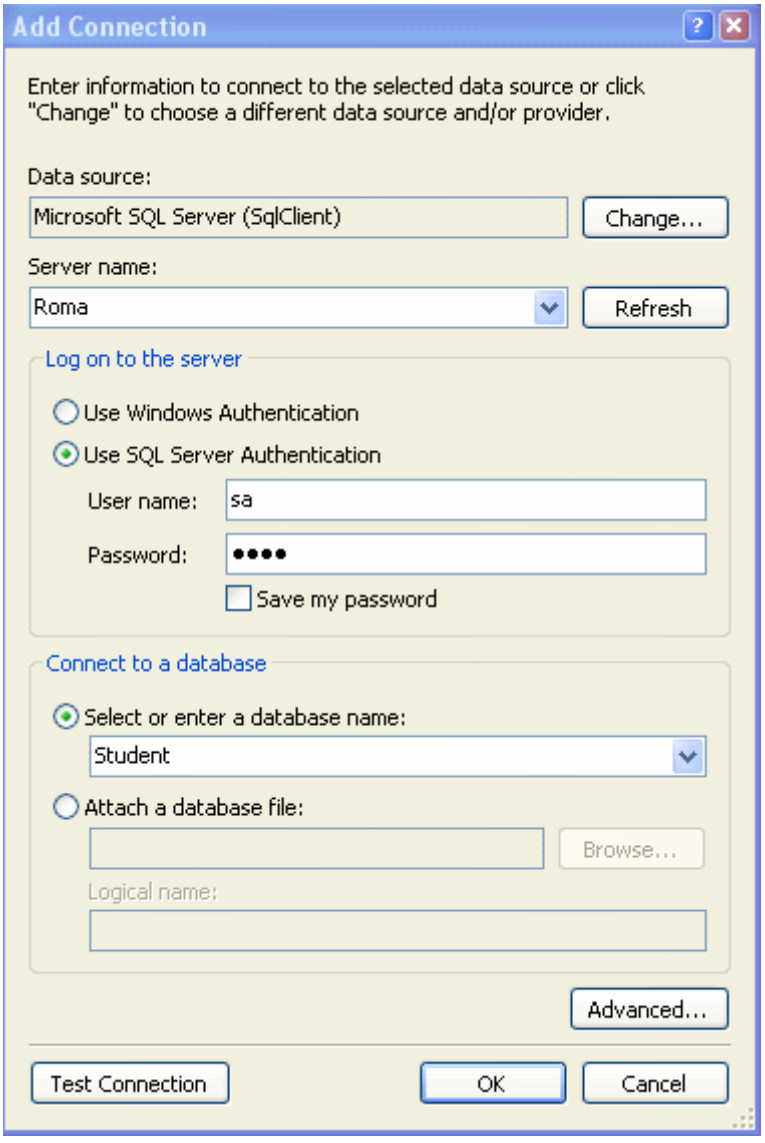
მომდევნო ფანჯარის (ნახ. 19.7) DataSet name ველში ჩანს მონაცემთა ნაკრების სახელი StudentDataSet. მასში ცხრილების მოსათავსებლად უნდა ჩავრთოთ Tables გადამრთველი. დავაჭიროთ Finish კლავიშს. შედეგად, ფორმის ქვეშ გამოჩნდება studentDataSet არავიზუალური კომპონენტი (ნახ. 19.8).

შემდეგ, ისევ მოვნიშნოთ bindingSource1 კომპონენტი. Properties ფანჯარაში გავხსნათ DataMember სია და ავირჩიოთ Salespeople ცხრილი. შედეგად, შეიქმნება salespeopleTable Adapter არავიზუალური კომპონენტი (ნახ 19. 9).

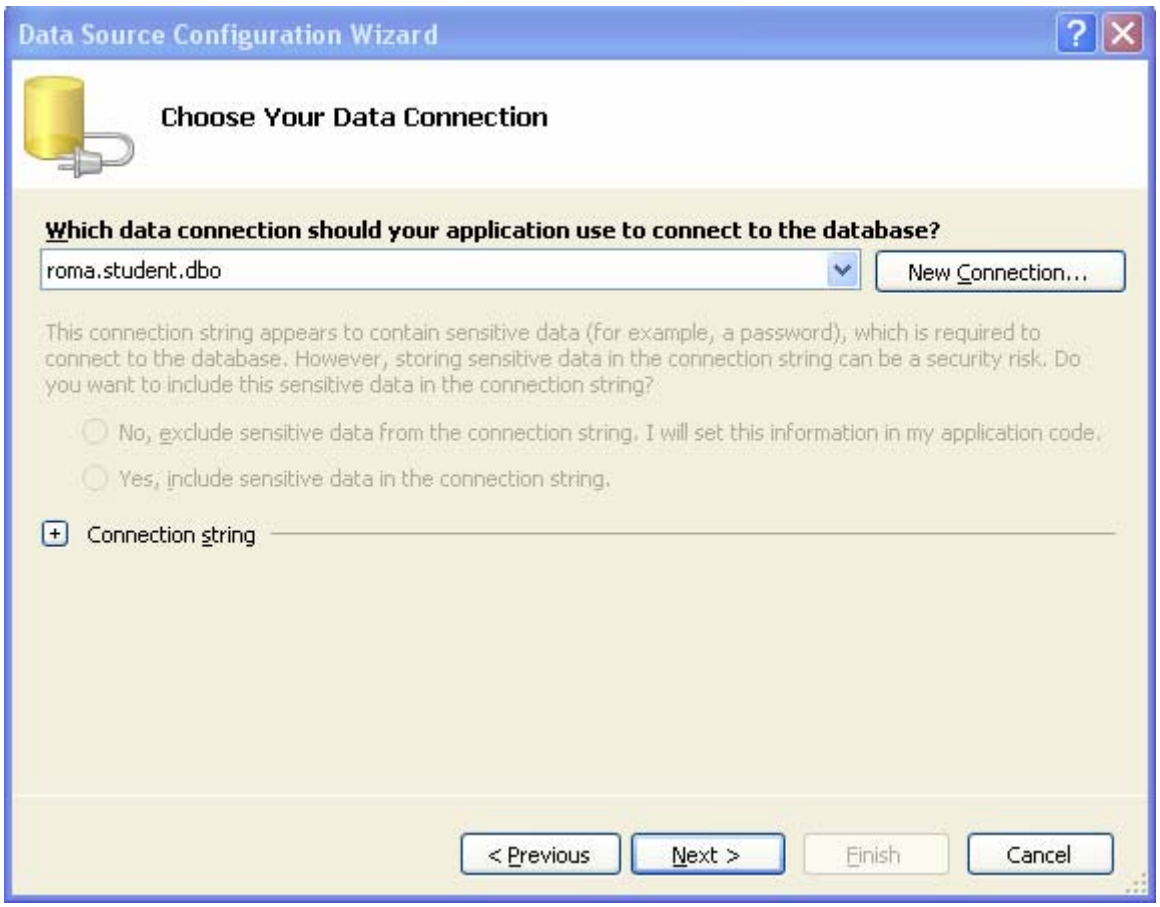
ახლა შევქმნათ ბმა (რელაცია) Salespeople და Orders ცხრილებს შორის. ამისათვის, მოვნიშნოთ studentDataSet კომპონენტი, გავხსნათ კონტექსტური მენიუ და შევასრულოთ Edit in DataSet Designer ბრძანება. გაიხსნება studentDataSet.xsd განყოფილება (ნახ 19. 10). ცარიელ ადგილზე მოვათავსოთ კურსორი, გავხსნათ კონტექსტური მენიუ, Add ქვემენიუ და შევასრულოთ Relation ბრძანება. გაიხსნება ფანჯარა (ნახ. 19.11), რომლის Name ველში ჩანს ბმის სახელი - Salespeople_Orders. სურვილის შემთხვევაში შეგვიძლია მისი შეცვლა. Parent Table სიიდან ვირჩევთ მთავარი (მშობელი) ცხრილის სახელს - Salespeople, ხოლო Childe Table სიიდან კი - დეტალების (შვილობილი) ცხრილის სახელს - Orders. Key Columns სიიდან ვირჩევთ მშობელი ცხრილის პირველად გასაღებს. ესაა Salespeople ცხრილის Snum ველი. Foreign Key Columns სიიდან ვირჩევთ შვილობილი ცხრილის გარე გასაღებს. ესაა Orders ცხრილის Snum ველი. ჩავრთოთ Relation Only გადამრთველი და დავაჭიროთ OK კლავიშს. studentDataSet.xsd განყოფილებაში გამოჩნდება ცხრილებს შორის კავშირი.

(ნახ 19. 12).

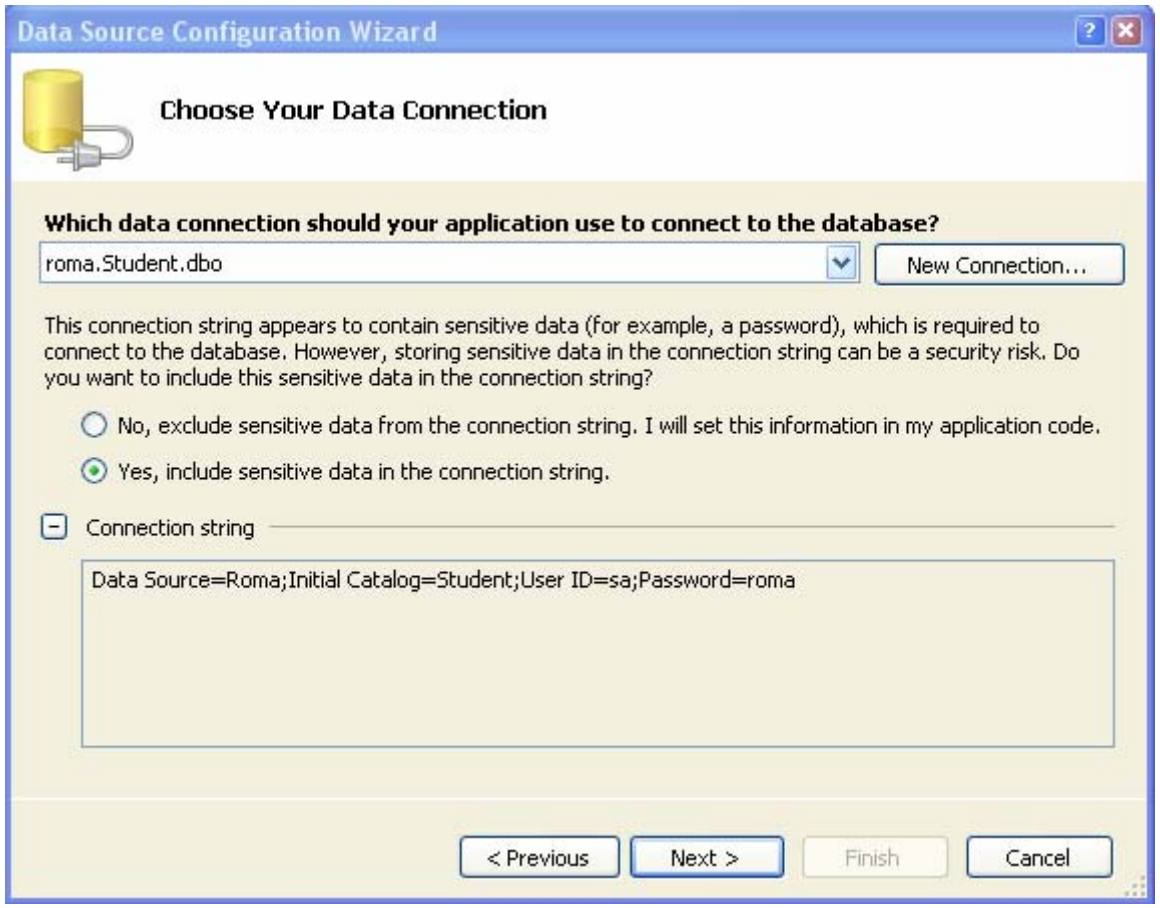
ამრიგად, ჩვენ სამუშაოდ მოვამზადეთ bindingSource1 კომპონენტი. იგი დავუკავშირეთ კონკრეტულ მონაცემთა ბაზასა და კონკრეტულ ცხრილს, და შევქმენით ბმა ცხრილებს შორის. ახლა ფორმაზე მოვათავსოთ bindingSource2 კომპონენტი. Properties ფანჯრის DataSource სიიდან ავირჩიოთ bindingSource1, ხოლო DataMember სიიდან კი - ბმის სახელი - Salespeople_Orders.



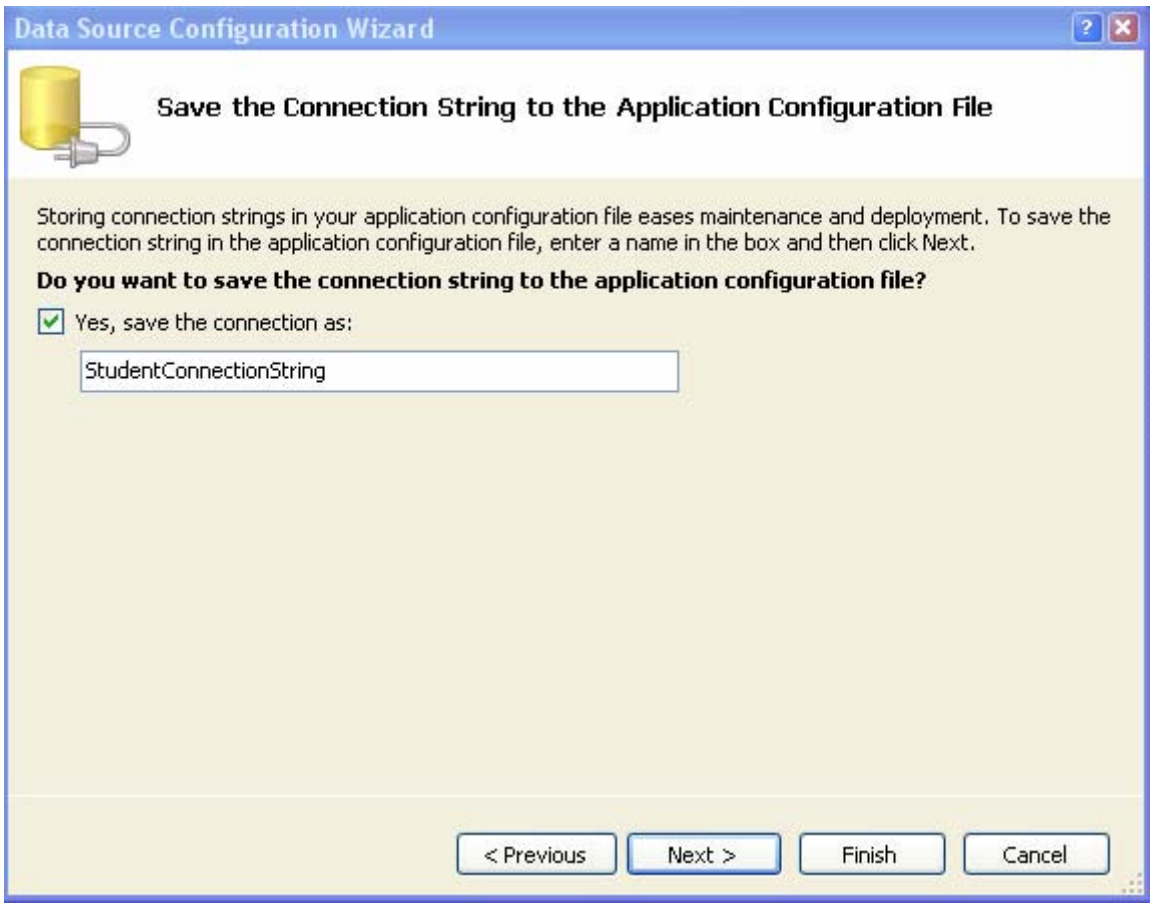
ნახ. 19.3



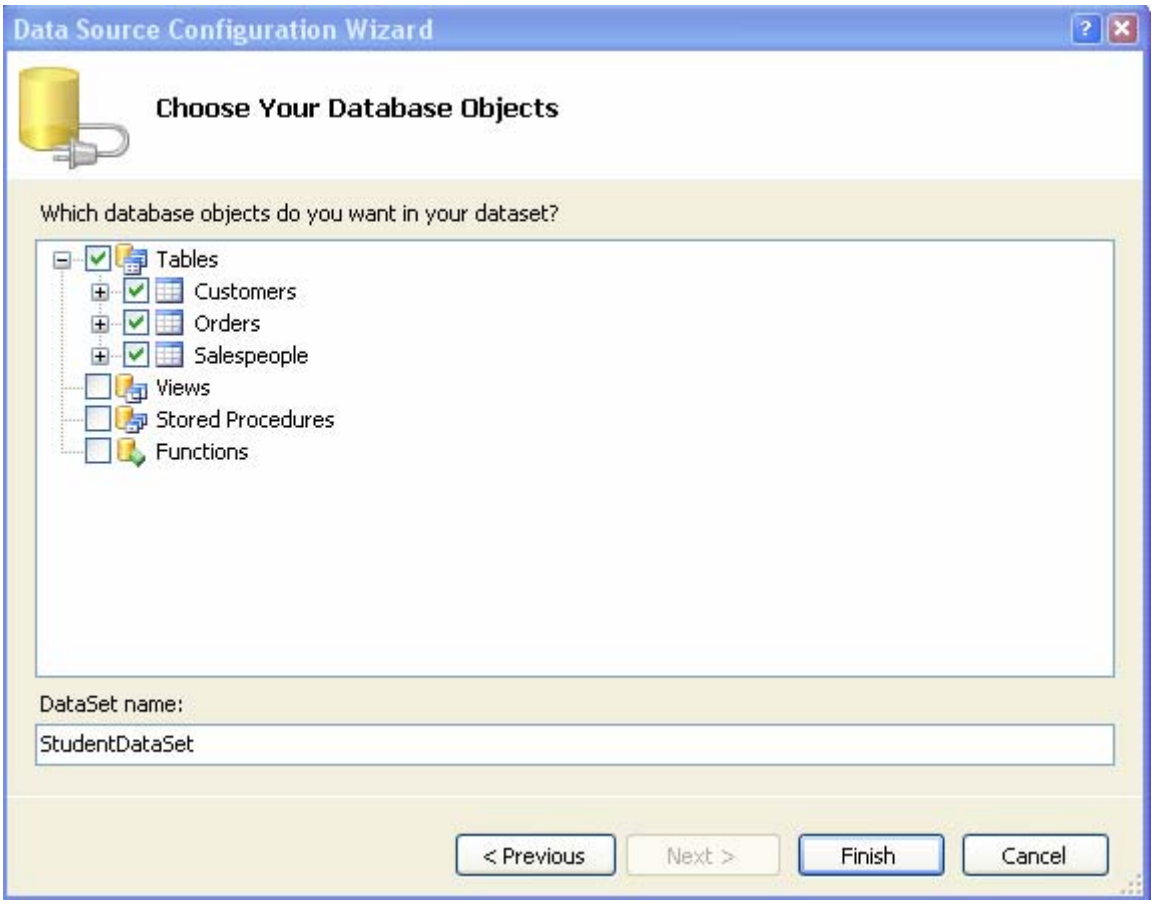
ნახ. 19.4



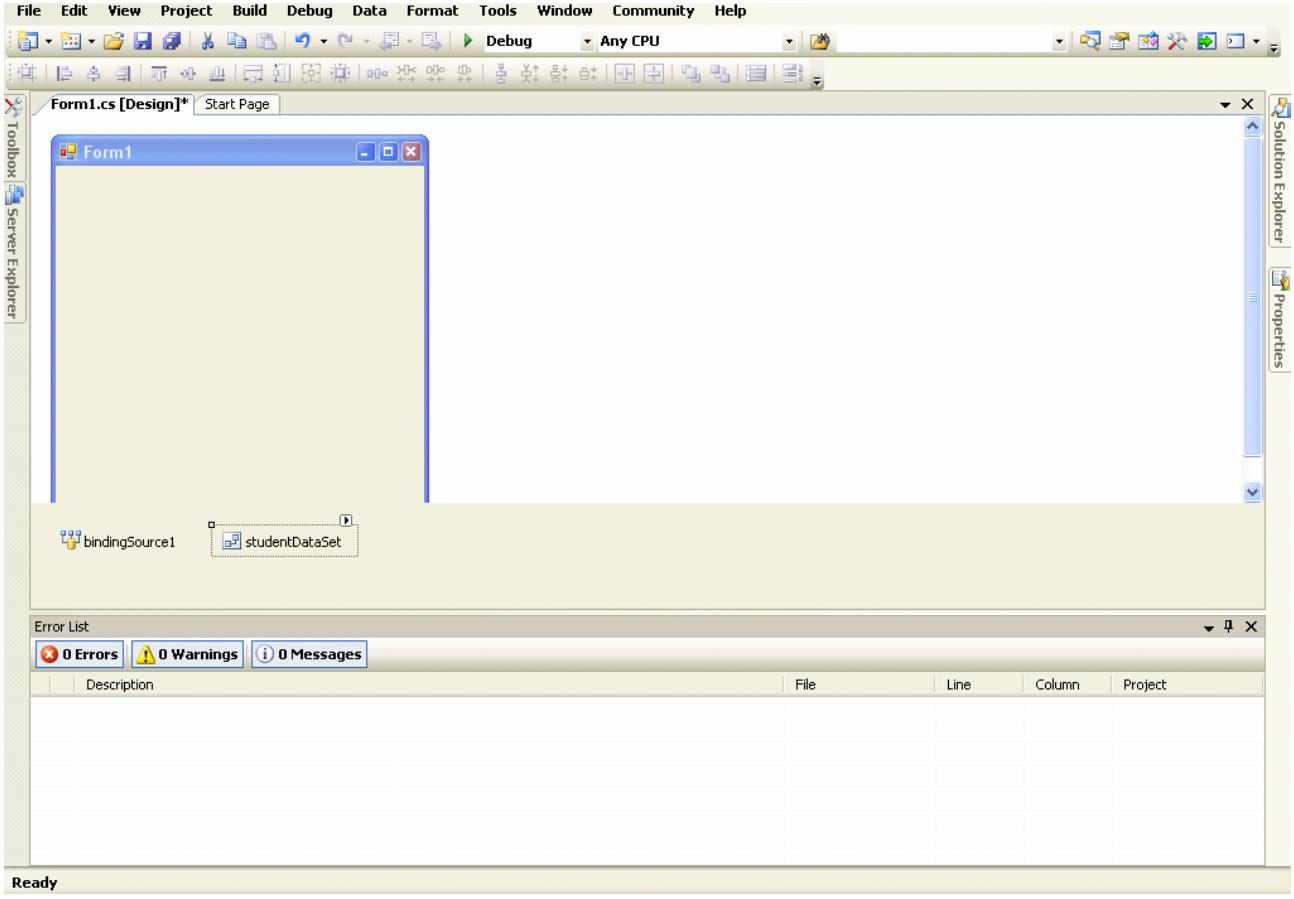
ნახ. 19.5



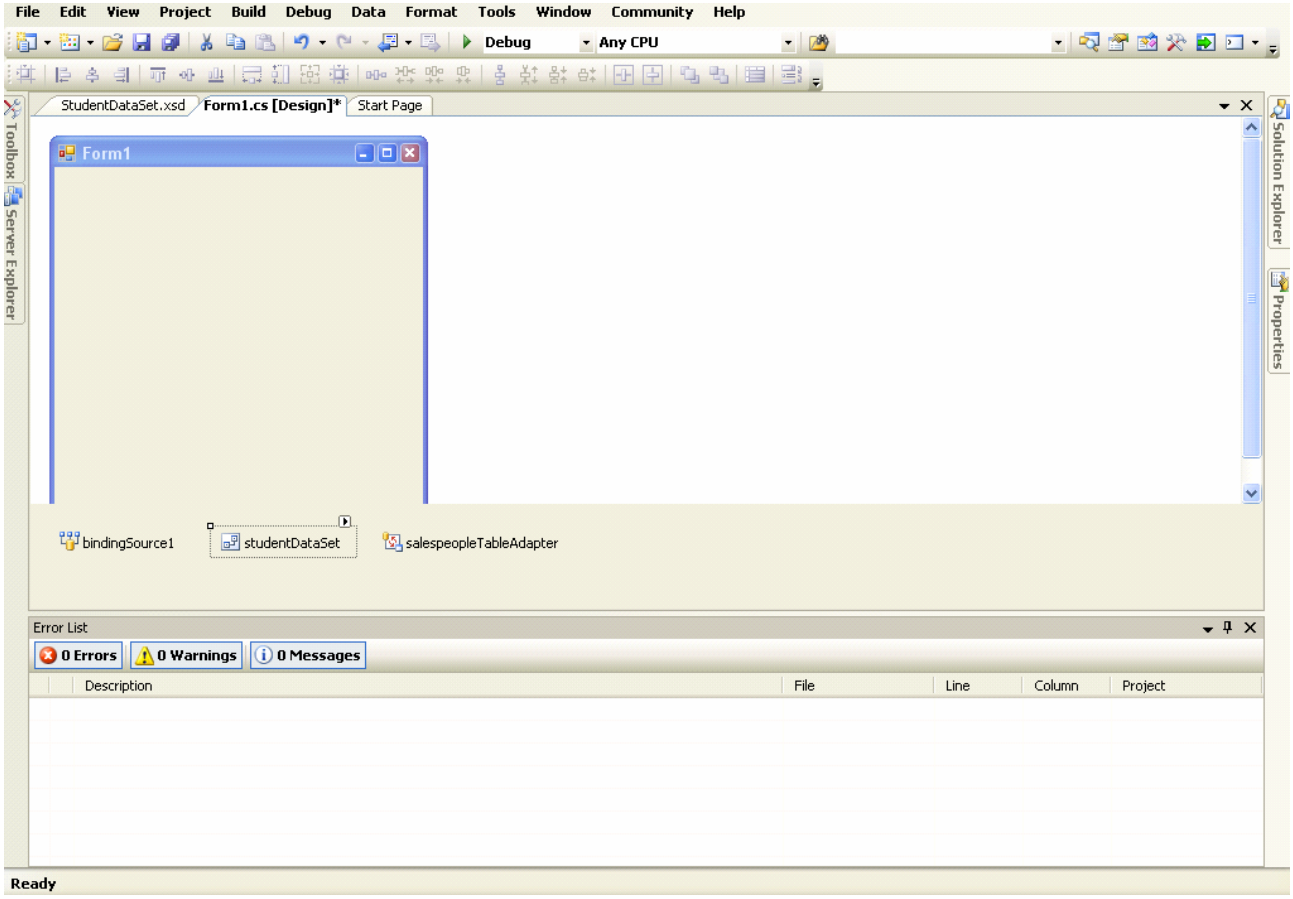
Бзб. 19.6



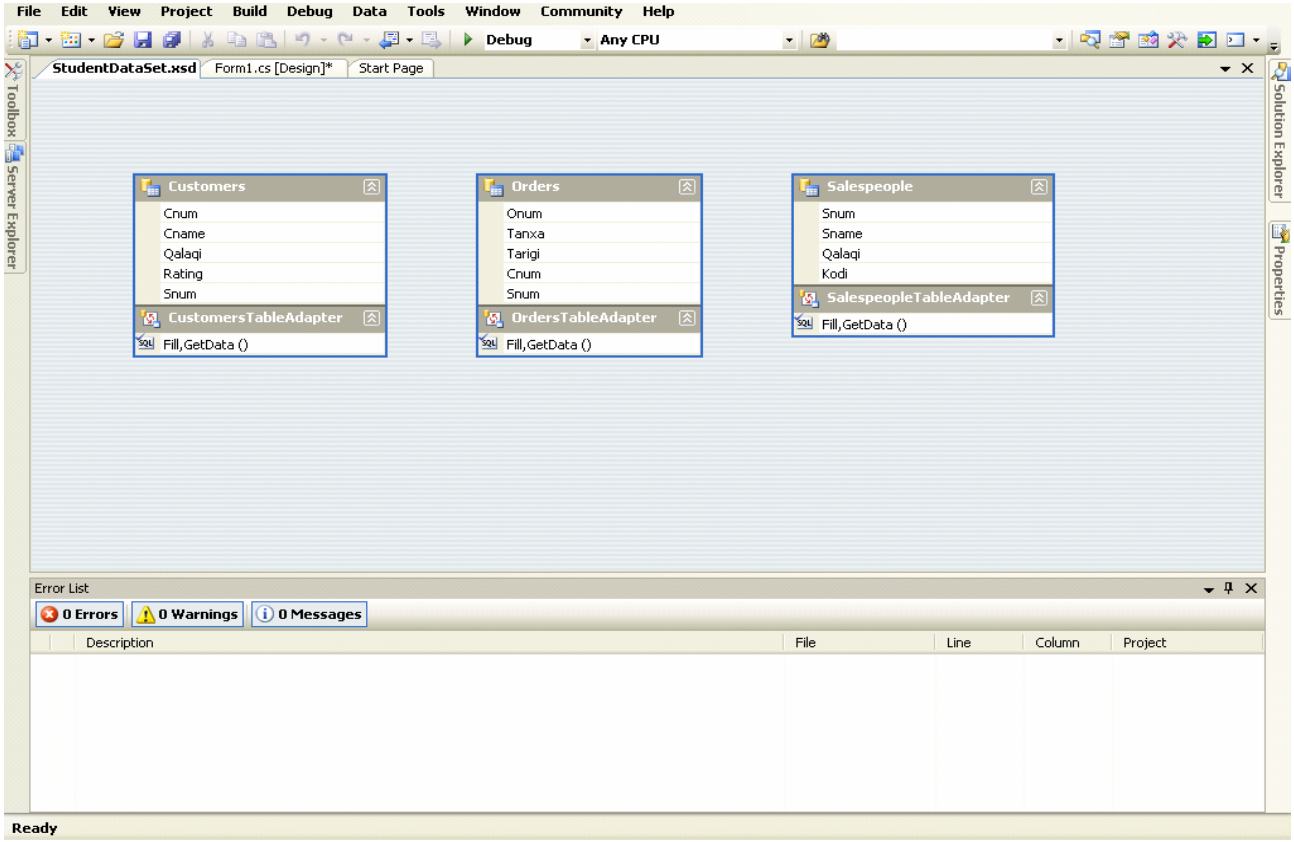
Бзб. 19.7



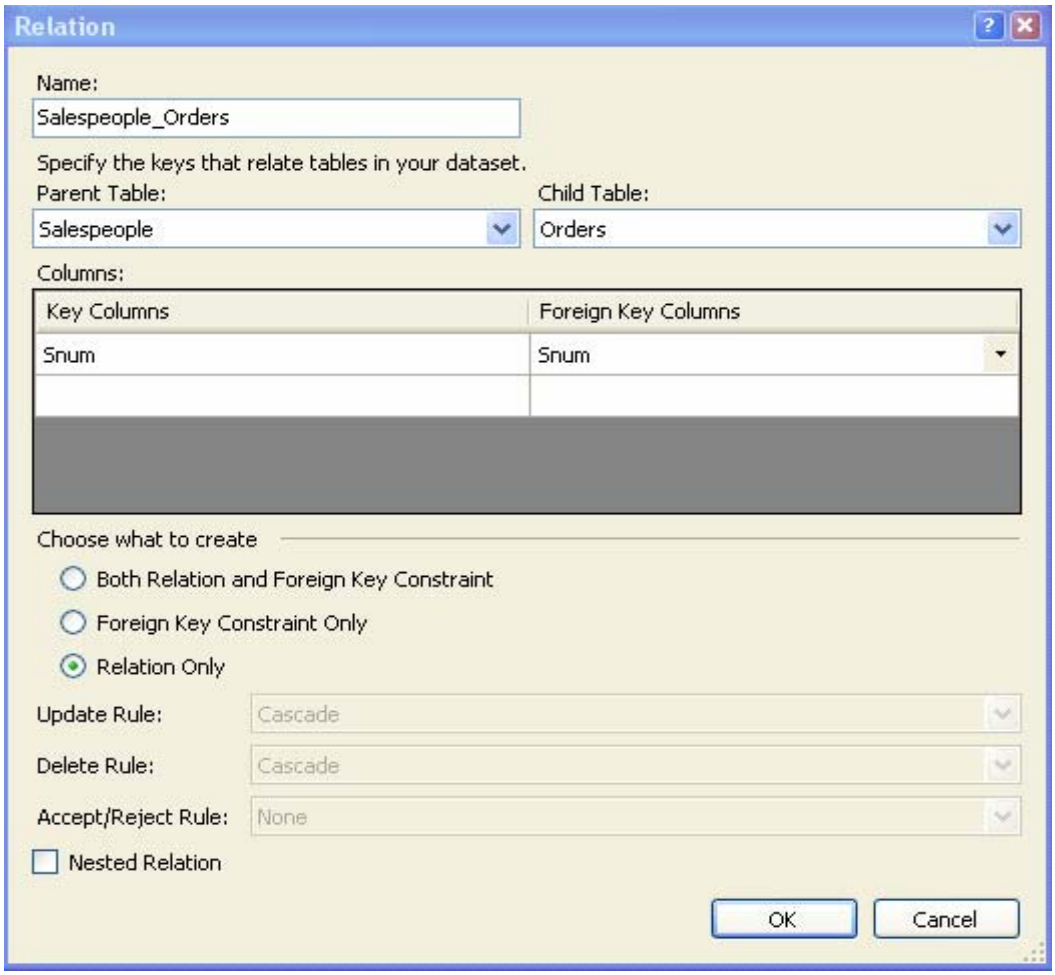
ნახ. 19.8



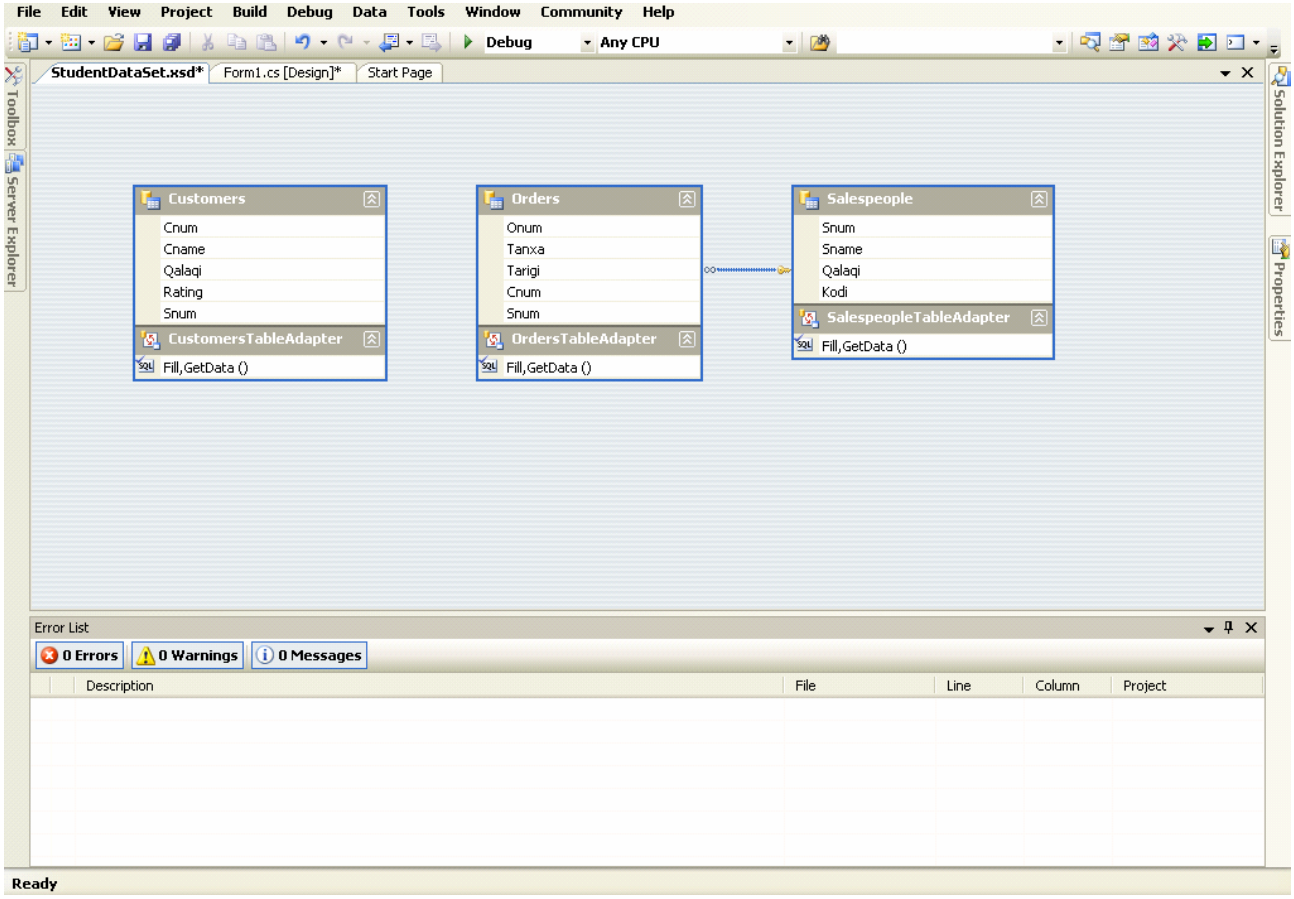
ნახ. 19.9



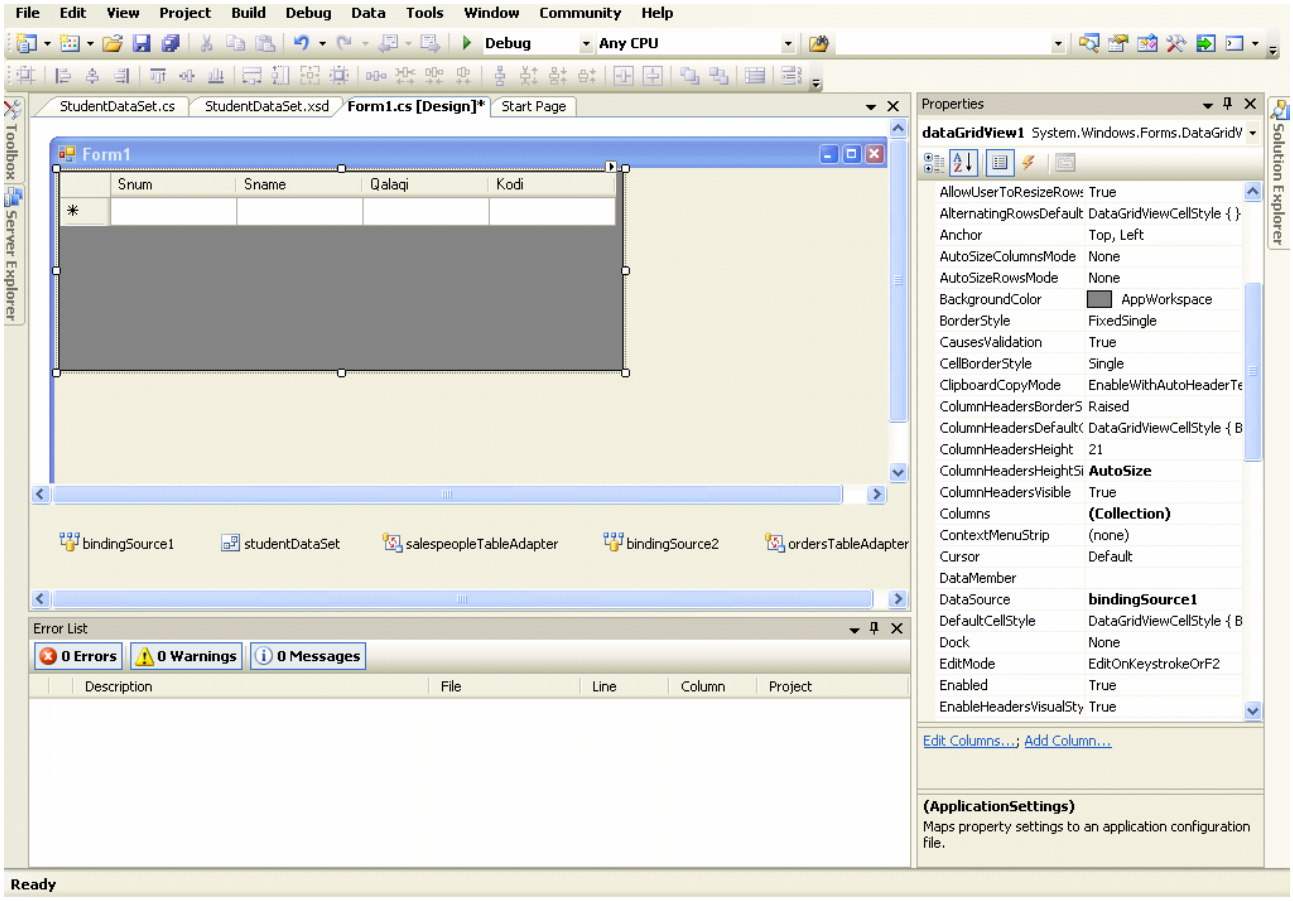
ნახ. 19.10



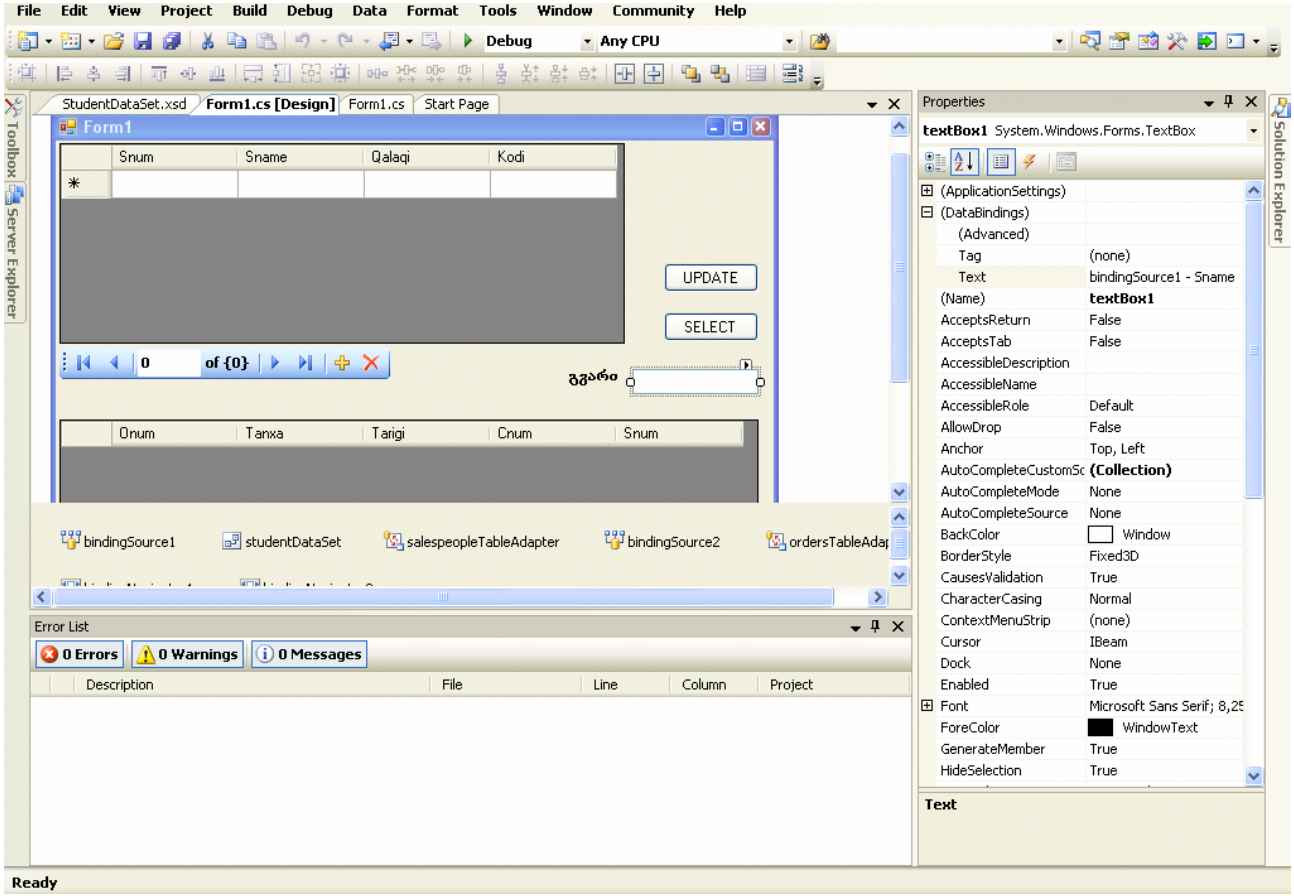
ნახ. 19.11



ნახ. 19.12



ნახ. 19.13



ნახ. 19.14

ფორმაზე მოვათავსოთ dataGridView1 კომპონენტი (ნახ. 19.13). მის მარჯვნივ გაიხსნება ფანჯარა. Choose Data Source სიიდან ავირჩიოთ bindingSource1. ცხრილის სტრიქონების სანახავად ჯერ დავაჭიროთ ამავე ფანჯრის Preview Data მიმართვას, შემდეგ კი გახსნილი ფანჯრის Preview კლავიშს. დავხუროთ ეს ფანჯარა და dataGridView1 კომპონენტი მოვათავსოთ ფორმის საჭირო ადგილზე და მივცეთ მას საჭირო ზომა. dataGridView1 კომპონენტისთვის bindingSource1 მონაცემების წყარო შეგვიძლია, აგრეთვე, ავირჩიოთ Properties ფანჯრის DataSource სიიდანაც.

ახლა ფორმაზე მოვათავსოთ dataGridView2 კომპონენტი და მივაბათ იგი bindingSource2 მონაცემთა წყაროს. თუ კომპონენტი ფორმაზე არ ეტევა, მაშინ ფორმის ზომები შესაბამისად გავზარდოთ. ეს შესაძლებელია ბუქსირის ოპერაციის გამოყენებით ან Properties ფანჯრის Size თვისებაში სიმაღლისა და სიგანის შეცვლის გზით.

ფორმაზე მოვათავსოთ bindingNavigator1 კომპონენტი (ნახ. 19.14). იგი ავტომატურად მოთავსთდება ფორმის ზედა ნაწილში. გავხსნათ Properties ფანჯრის Dock თვისება და ავირჩიოთ None. შედეგად, შევძლებთ ნავიგატორის მოთავსებას ფორმის ნებისმიერ ადგილზე. ახლა ნავიგატორი დავუკავშიროთ Salespeople ცხრილს. ამისათვის, DataSource სიიდან ავირჩიოთ bindingSource1. ფორმაზე მოვათავსოთ bindingNavigator2 კომპონენტი და დავუკავშიროთ იგი bindingSource2 მონაცემთა წყაროს.

იმისათვის, რომ შევძლოთ ცხრილების სტრიქონების ეკრანზე გამოტანა და მათი ცვლილება, ფორმაზე მოვათავსოთ button1 და button2 კომპონენტები. button1 კომპონენტის Text თვისებაში შევიტანოთ - Update, button2 კომპონენტის Text თვისებაში კი - Select. Update კლავიშს მივაბათ კოდი, რომელიც შეასრულებს salespeople ცხრილში

შეტანილი ცვლილებების შენახვას. ამისათვის, ორჯერ სწრაფად დავაწკაპუნით ამ კლავიშზე და შევიტანოთ პროგრამის კოდი:

```
this.Validate();
this.bindingSource1.EndEdit();
this. salespeopleTableAdapter.Update(this.studentDataSet.Salespeople);
```

Select კლავიშს მივაბათ კოდი, რომელიც შეასრულებს salespeople ცხრილის სტრიქონების dataGridView კომპონენტებში გამოტანას:

```
this. salespeopleTableAdapter.Fill(this.studentDataSet.Salespeople);
```

ანალოგიურად ვიქცევით Orders ცხრილისათვის. პროგრამის დასაწყისში ავტომატურად შეიქმნება კოდი, რომელიც შეასრულებს dataGridView კომპონენტებში ამ ცხრილების გამოტანას ფორმის გახსნისთანავე.

ახლა შევასრულოთ ეს პროგრამა F5 კლავიშზე დაჭერის გზით.

Salespeople ცხრილის გასწვრივ ნავიგაციის დროს გამოჩნდება Orders ცხრილის შესაბამისი სტრიქონები. დავხუროთ ეს ფანჯარა.

ახლა ფორმაზე მოვათავსოთ textBox1 კომპონენტი (ნახ. 19.14) და მივაბათ ის Salespeople ცხრილის Sname ველს. ამისათვის, Properties ფამჯარაში DataBindings თვისების მარცხნივ დავაჭიროთ + ნიშანს, გავხსნათ Text სია, დავაჭიროთ bindingSource1 სახელის მარცხნივ მოთავსებულ + ნიშანს და ავირჩიოთ Sname ველი. ისევ შევასრულოთ ჩვენი პროგრამა. ნავიგაციის დროს textBox1 კომპონენტში გამოჩნდება Sname ველის მნიშვნელობები. ამავე კომპონენტში შეგვიძლია შევიტანოთ, აგრეთვე, Sname ველის ახალი მნიშვნელობებიც.

SELECT-მოთხოვნის შესრულება ADO.NET კლასების საშუალებით

ამ განყოფილებაში მოყვანილ მაგალითში ნაჩვენებია თუ როგორ უნდა დავამყაროთ კავშირი Student მონაცემთა ბაზსთან SQL-სერვერზე და შევასრულოთ SELECT-მოთხოვნა, რომელიც გასცემს Customers ცხრილის Cnum, Cname, Qalaqi, Rating და Snum ველების მნიშვნელობებს. ამის გასაკეთებლად შემდეგ მოქმედებებს ვასრულებთ.

1. სტრიქონის ფორმირება, რომელიც შეიცავს მონაცემთა ბაზასთან შეერთების პარამეტრებს. SQL-სერვერზე მონაცემთა ბაზასთან დასაკავშირებლად შეერთების სტრიქონში უნდა მიეთითოს შემდეგი მნიშვნელობები:

- კომპიუტერის სახელი, რომელზეც მუშაობს Microsoft SQL-სერვერი. ეს სახელი ეთითება შეერთების სტრიქონის server ველში. თუ Microsoft SQL-სერვერი ჩვენს კომპიუტერზე მუშაობს, მაშინ კომპიუტერის სახელად შეგვიძლია გამოვიყენოთ localhost სიტყვა. მაგალითად, server = localhost.
- მონაცემთა ბაზის სახელი. იგი ეთითება database ველში. მაგალითად, database = Student.
- მომხმარებლის სახელი მონაცემთა ბაზასთან შესაერთებლად. იგი ეთითება შეერთების სტრიქონის uid ველში. მაგალითად, uid = sa.
- პაროლი მონაცემთა ბაზის მოცემული მომხმარებლისათვის. იგი ეთითება pwd ველში. მაგალითად, pwd = paroli.

ვქმნით string ტიპის connectionString ობიექტს, რომელსაც ენიჭება ლოკალურ კომპიუტერზე მოთავსებულ Student მონაცემთა ბაზასთან შეერთების სტრიქონი:

string connectionString = “server=localhost;database= Student;uid=sa;pwd=paroli”;

2. mySqlConnection ობიექტის შექმნა მონაცემთა ბაზასთან შეერთების მიზნით:
SqlConnection mySqlConnection = new SqlConnection(connectionString);

3. SELECT-მოთხოვნის შემცველი სტრიქონის ფორმირება, რომელიც Customers ცხრილიდან ამოიღებს Cnum, Cname, Qalaqi, Rating და Snum ველების მნიშვნელობებს:
string selectString = “SELECT Cnum, Cname, Qalaqi, Rating, Snum “ +
“FROM Customers WHERE Rating < 300”;

4. mySqlCommand ობიექტის შექმნა SELECT-მოთხოვნის შესანახად, რისთვისაც ვასრულებთ mySqlConnection ობიექტის CreateCommand() მეთოდს:
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();

ამის შემდეგ, mySqlCommand ობიექტი შეგვიძლია გამოვიყენოთ SELECT-მოთხოვნის შესასრულებლად.

5. mySqlCommand ობიექტის CommandText თვისებისათვის SELECT-მოთხოვნის შემცველი სტრიქონის მინიჭება:
mySqlCommand.CommandText = selectString;

6. SqlDataAdapter ობიექტის შექმნა. ვქმნით mySqlDataAdapter ობიექტს, რომელიც გამოიყენება ინფორმაციის გადასაადგილებლად DataSet ობიექტსა და მონაცემთა ბაზას შორის:
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();

7. mySqlDataAdapter ობიექტის SelectCommand თვისებისათვის mySqlCommand ობიექტზე მიმართვის მინიჭება:
mySqlDataAdapter.SelectCommand = mySqlCommand;

მინიჭების შემდეგ შეგვეძლება mySqlCommand ობიექტის CommandText თვისებაში განსაზღვრული SELECT-მოთხოვნის შესრულება.

8. myDataSet ობიექტის შექმნა SELECT-მოთხოვნის შესრულების შედეგების შესანახად:
DataSet myDataSet = new DataSet();

9. მონაცემთა ბაზასთან შეერთების დამყარება mySqlConnection ობიექტის Open() მეთოდის საშუალებით:
mySqlConnection.Open();

მონაცემთა ბაზასთან შეერთების დამყარების შემდეგ შეგვეძლება მასთან მუშაობა.

10. mySqlDataAdapter ობიექტის Fill() მეთოდის შესრულება ცხრილიდან ჩანაწერების მისაღებად და მათ შესანახად DataTable ობიექტში, რომელიც მოთავსებულია DataSet ობიექტში. მაგალითში მოყვანილ Fill() მეთოდს ორი პარამეტრი აქვს: პირველია - DataSet ობიექტი, მეორე კი - სტრიქონი, რომელიც შეიცავს იმ DataTable ობიექტის სახელს, რომელიც უნდა შეიქმნას. Fill() მეთოდი myDataSet ობიექტში ქმნის DataTable ობიექტს Customers სახელით და ავსებს მას ჩანაწერებით, რომელსაც გასცემს SELECT-მოთხოვნა.

მოყვანილ მაგალითში იქმნება dataTableName სტრიქონი, რომელსაც ენიჭება “Customers” მნიშვნელობა. შემდეგ გამოიძახება mySqlDataAdapter ობიექტის Fill() მეთოდი, რომელსაც პარამეტრებად გადაეცემა myDataSet და dataTableName ობიექტები:
string dataTableName = “Customers”;
mySqlDataAdapter.Fill(myDataSet, dataTableName);

11. myDataSet ობიექტიდან myDataTable ობიექტის მიღება. myDataSet ობიექტის

Tables თვისება გვიბრუნებს კვადრატულ ფრჩხილებში მითითებულ DataTable ობიექტს, რომელიც მოთავსთდება myDataTable ობიექტში:

```
DataTable myDataTable = myDataSet.Tables[dataTableName];
```

DataTable ობიექტის მიღება შეიძლება, აგრეთვე, Tables თვისებისათვის ნომრის გადაცემით. მაგალითად, myDataSet.Tables[0] თვისება აბრუნებს DataTable ობიექტს Customers სახელით.

12. myDataTable ცხრილიდან თითოეული ჩანაწერის ველების გამოტანა. DataTable კლასში განსაზღვრულია Rows თვისება, რომელიც გვიბრუნებს DataRow ობიექტების კოლექციას. თითოეული DataRow ობიექტი შეიცავს DataColumn ობიექტებს, რომლებშიც ინახება ცხრილის ველების მონაცემთა ბაზიდან მიღებული მნიშვნელობები. ამ ველებთან მიმართვისათვის DataRow ობიექტში ველის სახელი უნდა მივუთითოთ კვადრატულ ფრჩხილებში. მაგალითად, myDataRow[“Cnum”] თვისება გვიბრუნებს Cnum ველის მნიშვნელობას.

მოყვანილ მაგალითში foreach ციკლში ხდება myDataTable ცხრილის DataRow ობიექტთან მიმართვა და თითოეული ველის მნიშვნელობის გამოტანა label კომპონენტში:

```
foreach ( DataRow myDataRow in myDataTable.Rows )
{
label2.Text += myDataRow[“Cnum”] + “   “;
label2.Text += myDataRow[“Cname”] + “   “;
label2.Text += myDataRow[“Qalaqi”] + “   “;
label2.Text += myDataRow[“Rating”] + “   “;
label2.Text += myDataRow[“Snum”] + ‘\n’;
}
```

ველთან მიმართვა შეიძლება, აგრეთვე, DataRow ობიექტისათვის მისი ნომრის გადაცემით. მაგალითად, myDataRow[0] თვისება აბრუნებს Cnum ველის მნიშვნელობას.

13. მონაცემთა ბაზასთან შეერთების დახურვა SqlConnection ობიექტის Close() მეთოდის გამოყენებით:

```
mySqlConnection.Close();
```

ქვემოთ მოყვანილია პროგრამა, რომელშიც სრულდება ყველა აღწერილი მოქმედება. პროგრამის დასაწყისში, იქ სადაც მითითებულია using დირექტივები, უნდა მივუთითოთ

```
using System.Data.SqlClient;
{
//      პროგრამა 19.1
//      პროგრამაში ხდება  მოთხოვნის შესრულების დემონსტრირება
label2.Text = "";
//      სტრიქონის ფორმირება, რომელიც შეიცავს მონაცემთა ბაზასთან შეერთების
პარამეტრებს
string connectionString =
                                "server=localhost;database=Studenti;uid=sa;pwd=paroli";
//      SqlConnection ობიექტის შექმნა მონაცემთა ბაზასთან შეერთების
დამყარებისათვის
SqlConnection mySqlConnection = new SqlConnection(connectionString);
//      SELECT მოთხოვნის შემცველი სტრიქონის ფორმირება
```

```
string selectString = " SELECT Cnum, Cname, Qalaqi, Rating, Snum " +
                      " FROM Customers WHERE Rating < 300 " ;
//      SqlCommand ობიექტის შექმნა შექმნა SELECT მოთხოვნისათვის
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//      SqlCommand ობიექტის CommandText თვისებისათვის ფორმირებული SELECT-
//      მოთხოვნის შემცველი სტრიქონის მინიჭება
mySqlCommand.CommandText = selectString;
//      SqlDataAdapter ობიექტის შექმნა
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
//      SqlDataAdapter ობიექტის SelectCommand თვისებისათვის SqlCommand
//      ობიექტზე მიმთითებლის მინიჭება
mySqlDataAdapter.SelectCommand = mySqlCommand;
//      DataSet ობიექტის შექმნა SELECT-მოთხოვნის შედეგების შესანახად
DataSet myDataSet = new DataSet();
//      მონაცემთა ბაზასთან შეერთების გახსნა SqlConnection ობიექტის Open()
//      მეთოდის საშუალებით
mySqlConnection.Open();
//      SqlDataAdapter ობიექტის Fill() მეთოდის გამოძახება ცხრილიდან ჩანაწერების
//      მისაღებად და DataTable ობიექტში ლოკალურად შესანახად
string dataTableName = "Customers";
mySqlDataAdapter.Fill(myDataSet, dataTableName);
//      DataSet ობიექტიდან DataTable ობიექტის მიღება
DataTable myDataTable = myDataSet.Tables[dataTableName];
//      DataTable ობიექტის თითოეული ჩანაწერის ველების გამოტანა DataRow
ობიექტის გამოყენებით
foreach ( DataRow myDataRow in myDataTable.Rows )
{
        label2.Text += "Cnum   = " + myDataRow["Cnum"]   + "    ";
        label2.Text += "Cname  = " + myDataRow["Cname"]  + "    ";
        label2.Text += "Qalaqi = " + myDataRow["Qalaqi"] + "    ";
        label2.Text += "Rating = " + myDataRow["Rating"] + "    ";
        label2.Text += "Snum   = " + myDataRow["Snum"]   + '\n';
}
//      მონაცემთა ბაზასთან შეერთების დახურვა SqlConnection ობიექტის Close()
მეთოდის
//      გამოყენებით
mySqlConnection.Close();
}
```

ამ პროგრამის შესაბამის ფორმაზე უნდა მოვათავსოთ ერთი button კომპონენტი, რომელზეც დაჭერა ასრულებს ზემოთ მოყვანილ პროგრამას და ერთი label კომპონენტი, რომელშიც გამოგვაქვს ველების მნიშვნელობები.

INSERT, UPDATE და DELETE მოთხოვნების შესრულება ADO.NET კლასების საშუალებით

ამ განყოფილებაში განვიხილავთ INSERT, UPDATE და DELETE მოთხოვნების შესასრულებლად საჭირო ნაბიჯებს მონაცემთა ბაზის ცხრილებში ჩანაწერების დამატების, მოდიფიცირებისა და წაშლისათვის. ამ მოთხოვნების შესასრულებლად საჭირო მოქმედებები ერთნაირია. დაწვრილებით განვიხილოთ ისინი.

1. SQL-მოთხოვნის შემცველი სტრიქონის ფორმირება. SQL-მოთხოვნა შეიძლება შეიცავდეს INSERT, UPDATE ან DELETE მოთხოვნებს. მოყვანილი სტრიქონი შეიცავს INSERT მოთხოვნას, რომელიც Customers ცხრილს ჩანაწერებს ჩაუმატებს:

```
string insertString = “INSERT INTO Customers “ +
    “ ( Cnum, Cname, Qalaqi, Rating, Snum ) ” +
    “ (VALUES ( @Cnum, @Cname, @Qalaqi, @Rating,  @Snum) ”;
```

ეს სტრიქონი შეიცავს ქვესტრიქონებს, რომლებიც @ ნიშნით იწყება. ეს ქვესტრიქონები წარმოადგენენ ადგილს პარამეტრების ტიპებისა და მნიშვნელობებისათვის. პარამეტრების ტიპებსა და მნიშვნელობებს მეოთხე და მეხუთე ნაბიჯზე განვიხილავთ. პარამეტრებისათვის გამოყოფილი ადგილები შეიძლება გამოვიყენოთ ველების მნიშვნელობების განსაზღვრისათვის პროგრამის შესრულების დროს.

2. mySqlCommand ობიექტის შექმნა SQL-მოთხოვნის შესანახად:

```
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
```

3. mySqlCommand ობიექტის CommandText თვისებისათვის INSERT-მოთხოვნის შემცველი სტრიქონის მინიჭება:

```
mySqlCommand.CommandText = insertString;
```

4. Add() მეთოდის გამოძახება mySqlCommand ობიექტისთვის პარამეტრების დასამატებლად. Add() მეთოდის აქ გამოყენებულ ვერსიას სამი პარამეტრი აქვს:

- სტრიქონი-შემვსები, რომელიც ახდენს ადგილის დარეზერვებას SQL-მოთხოვნის ტექსტში. მაგალითად, “@Cnum” - ესაა პირველი სტრიქონი-შემვსები insertString სტრიქონში.
- ველის ტიპი მონაცემთა ბაზაში. ეს ტიპები მოყვანილია ცხრილში 19.1.
- პარამეტრის მნიშვნელობის მაქსიმალური სიგრძე. ეს პარამეტრი არაა აუცილებელი.

ცხრილი 19.1. SqlDbType ჩამოთვლადი ტიპის ელემენტები.

ელემენტი	აღწერა
BigInt	64-ბიტიანი ნიშნიანი მთელი რიცხვი, რომელიც იღებს მნიშვნელობას -263÷263-1 (-9223372036854775808÷9223372036854775807)
Binary	ბაიტების მასივი სიგრძით 8000 ელემენტამდე
Bit	უნიშნო მთელი რიცხვი, რომელიც იღებს მნიშვნელობებს 0, 1 ან null
Char	სტრიქონი სიგრძით 8000 სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს
DateTime	დრო და თარიღი 1753 წლის 1 იანვრიდან 9999 წლის 31 დეკემბრამდე
Decimal	რიცხვი მითითებული სიზუსტით და მასშტაბით მნიშვნელობით -

	$10^{38}-1 \div 10^{38}-1$ (-79228162514264337593543950335÷ 79228162514264337593543950335)
Float	მცურავწერტილიანი რიცხვი დიაპაზონში -1.79E+308 1.79E+308
Image	ბაიტების მასივი სიგრძით $2^{31}-1$ (2147483647) ელემენტამდე
Int	32-თანრიგა მთელი რიცხვი ნიშნით, დიაპაზონში $-2^{31} \div 2^{31}-1$ (-2147483648÷2147483647)
Money	ფული დიაპაზონში -922337203685477.5808÷922337203685477.5807
NChar	Unicode სიმბოლოების სტრიქონი მაქსიმალური სიგრძით 4000 სიმბოლომდე
NText	Unicode სიმბოლოების სტრიქონი მაქსიმალური სიგრძით $2^{30}-1$ (1073741823) სიმბოლომდე
NVarChar	Unicode სიმბოლოების სტრიქონი მაქსიმალური სიგრძით 4000 სიმბოლომდე
Real	მცურავწერტილიანი რიცხვი დიაპაზონში -3.40E+38÷3.40E+38
SmallDateTime	დრო და თარიღი 1900 წლის 1 იანვრიდან 2079 წლის 6 ივნისამდე
SmallInt	16-ბიტიანი მთელი რიცხვი ნიშნით
SmallMoney	ფული დიაპაზონში -214748.3648÷214748.3647
Text	სტრიქონი მაქსიმალური სიგრძით $2^{31}-1$ (2147483647) სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს
Timestamp	თარიღი და დრო ფორმატში yyyymmddhhmmss (წელი/თვე/დღე/საათი/წუთები/წამები)
TinyInt	8-ბიტიანი უნიშნო მთელი რიცხვი დიაპაზონში $0 \div 2^{8}-1$ (255)
UniqueIdentifier	GUID (Globally Unique IDentifier - გლობალური უნიკალური იდენტიფიკატორი)
VarBinary	ბაიტების მასივი მაქსიმალური სიგრძით 8000 ელემენტამდე
VarChar	სტრიქონი მაქსიმალური სიგრძით 8000 სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს
Variant	მონაცემების ტიპი, რომელიც შეიძლება შეიცავდეს რიცხვებს, სტრიქონებს, ბაიტებსა და თარიღებს

ჩვენი INSERT მოთხოვნა შეიცავს დარეზერვებულ ადგილებს ხუთი პარამეტრისათვის. ქვემოთ ნაჩვენებია Add() მეთოდის გამოყენება mySqlCommand ობიექტში პარამეტრების დასამატებლად:

```
mySqlCommand.Parameters.Add(“@Cnum”,  SqlDbType.Int, 4);
mySqlCommand.Parameters.Add(“@Cname”, SqlDbType.NVarChar, 15);
mySqlCommand.Parameters.Add(“@Qalaqi”,SqlDbType.NVarChar, 15);
mySqlCommand.Parameters.Add(“@Rating”,SqlDbType.Int, 4);
mySqlCommand.Parameters.Add(“@Snum”,  SqlDbType.Int, 4);
```

5. პარამეტრებისათვის მნიშვნელობების მინიჭება Value თვისების საშუალებით. მოყვანილ მაგალითში Customers ცხრილის cnum, cname, qalaqi, rating და snum ველებს ენიჭებათ კონკრეტული მნიშვნელობები:

```
mySqlCommand.Parameters[“@Cnum”].Value = 208;
mySqlCommand.Parameters[“@Cname”].Value = “ავერსი”;
mySqlCommand.Parameters[“@Qalaqi”].Value = “ზესტაფონი”;
```

```
mySqlCommand.Parameters[“@Rating”].Value = 250;
mySqlCommand.Parameters[“@Snum”].Value = 103;
```

ამ მაგალითში “@Cnum” პარამეტრს ენიჭება მნიშვნელობა 208. შემდეგ, ეს მნიშვნელობა მიენიჭება Cnum ველს ახალ ჩანაწერში INSERT მოთხოვნის შესრულების დროს. ზემოთ მოყვანილი მინიჭებების შედეგად, INSERT მოთხოვნა შემდეგ სახეს მიიღებს:

```
INSERT INTO Customers (Cnum, Cname, Qalaqi, Rating, Snum)
                                Values (208, ‘ავერსი’, ‘ზესტაფონი’, 250, 103);
```

6. ExecuteNonQuery() მეთოდის გამოძახება SQL-მოთხოვნის შესასრულებლად:

```
mySQLCommand.ExcuteNonQuery();
```

ეს ბრძანება ასრულებს INSERT მოთხოვნას და უმატებს ახალ ჩანაწერს Customers ცხრილს. ამ მეთოდის გამოყენება იმითაა განპირობებული, რომ INSERT მოთხოვნა არ გასცემს მნიშვნელობებს.

ქვემოთ მოყვანილი პროგრამა ასრულებს INSERT, UPDATE და DELETE მოთხოვნებს Customer ცხრილში ჩანაწერების დასამატებლად, შესაცვლელად და წასაშლელად. ეს პროგრამა ოთხ მეთოდს მოიცავს, რომლებიც Main() მეთოდის შემდეგ უნდა მოვათავსოთ. ეს მეთოდებია:

- DisplayDataTable(). მეთოდს ეკრანზე გამოაქვს DataTable ობიექტში შემავალი ჩანაწერების Cnum, Cname, Qalaqi, Rating და Snum ველები. DataTable ობიექტი DisplayDataTable() მეთოდს გადაეცემა პარამეტრის სახით.
- AddRow(). მეთოდი ასრულებს INSERT მოთხოვნას, რომელიც Customers ცხრილში ახალ ჩანაწერს უმატებს.
- ModifyRow(). მეთოდი ასრულებს UPDATE მოთხოვნას, რომელიც Customers ცხრილში ახალი ჩანაწერის შეცვლას ახდენს.
- RemoveRow(). მეთოდი ასრულებს DELETE მოთხოვნას, რომელიც შლის ახალ ჩანაწერს Customers ცხრილიდან.

თითოეული მეთოდის გამოძახების შემდეგ დამატებით გამოიძახება DisplayDataTable() მეთოდი Customers ცხრილში შეტანილი ცვლილებების ასახვისათვის. პროგრამის შესაბამის ფორმაზე შეგვიძლია მოვათავსოთ ერთი dataGrid კომპონენტი, რომელშიც გამოჩნდება ცხრილის ჩანაწერები და სამი button კომპონენტი. მათზე დაჭერის შედეგად შესაბამისად შესრულდება INSERT, UPDATE და DELETE მოთხოვნები. ქვემოთ მოყვანილია ეს პროგრამა.

```
//      პროგრამა 19.2
//      პროგრამაში ხდება INSERT, UPDATE და DELETE მოთხოვნების შესრულების
დემონსტრირება
static void Main()
{
        Application.Run(new Form1());
}
//      მეთოდი ახდენს ჩანაწერების ასახვას
public static void DisplayDataTable(SqlDataAdapter mySqlDataAdapter,
                                        DataSet myDataSet, DataTable myDataTable)
{
//      Clear() შლის ყველა ჩანაწერს DataSet ობიექტიდან
myDataSet.Clear();
```

```
// Fill() მეთოდი myDataSet ობიექტში ქმნის Customers ობიექტს და ავსებს მას
ჩანაწერებით
mySqlDataAdapter.Fill(myDataSet, "Customers");
}
public static void AddRow(SqlConnection mySqlConnection,
                          SqlDataAdapter mySqlDataAdapter, DataSet myDataSet)
{
//      SQL-მოთხოვნის შემცველი სტრიქონის ფორმირება
string insertString =
                    "INSERT INTO Customers (Cnum, Cname, Qalaqi, Rating, Snum)" +
                    "VALUES (@Cnum, @Cname, @Qalaqi, @Rating, @Snum)";
//      SqlCommand ობიექტის შექმნა SQL-მოთხოვნის შესანახად
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//      SqlCommand ობიექტის CommandText თვისებას მოვანიჭოთ SQL-მოთხოვნის
შემცველი სტრიქონი
mySqlCommand.CommandText = insertString;
//      mySqlCommand ობიექტისათვის პარამეტრების დამატება
mySqlCommand.Parameters.Add("@Cnum",   SqlDbType.Int,   4);
mySqlCommand.Parameters.Add("@Cname",  SqlDbType.Char, 15);
mySqlCommand.Parameters.Add("@Qalaqi", SqlDbType.Char, 15);
mySqlCommand.Parameters.Add("@Rating", SqlDbType.Int,   4);
mySqlCommand.Parameters.Add("@Snum",   SqlDbType.Int,   4);
//      პარამეტრებისათვის საჭირო მნიშვნელობების მინიჭება Value თვისების
საშუალებით
mySqlCommand.Parameters["@Cnum"].Value   = 208;
mySqlCommand.Parameters["@Cname"].Value  = "'aversi'";
mySqlCommand.Parameters["@Qalaqi"].Value = "'zestafoni'";
mySqlCommand.Parameters["@Rating"].Value = 240;
mySqlCommand.Parameters["@Snum"].Value   = 103;
//      SQL-მოთხოვნის შესრულება ExecuteNonQuery() მეთოდის გამოყენებით
mySqlCommand.ExecuteNonQuery();
}
//      მეთოდი ახდენს ცხრილის ჩანაწერის ველების მნიშვნელობების შეცვლას
public static void ModifyRow(SqlConnection mySqlConnection,
                             SqlDataAdapter mySqlDataAdapter, DataSet myDataSet)
{
//      SQL-მოთხოვნის ფორმირება
string updateString = "UPDATE Customers SET Cname = @Cname, Qalaqi = @Qalaqi,
                       Rating = @Rating, Snum = @Snum WHERE Cnum = @Cnum";
//      SqlCommand ობიექტის შექმნა SQL-მოთხოვნის შესანახად
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//      SqlCommand ობიექტის CommandText თვისებისათვის SQL-მოთხოვნის შემცველი
//      სტრიქონის მინიჭება
mySqlCommand.CommandText = updateString;
//      SqlCommand ობიექტის Parameters კოლექციის Add() მეთოდის გამოყენებით
```

```
//        პარამეტრები დავუმატოთ SqlCommand ობიექტს
mySqlCommand.Parameters.Add("@Cnum",   SqlDbType.Int,   4);
mySqlCommand.Parameters.Add("@Cname",  SqlDbType.Char, 15);
mySqlCommand.Parameters.Add("@Qalaqi", SqlDbType.Char, 15);
mySqlCommand.Parameters.Add("@Rating", SqlDbType.Int,   4);
mySqlCommand.Parameters.Add("@Snum",   SqlDbType.Int,   4);
//        პარამეტრებისათვის საჭირო მნიშვნელობების მინიჭება Value თვისების
საშუალებით
mySqlCommand.Parameters["@Cnum"].Value = 208;
mySqlCommand.Parameters["@Cname"].Value = "aversi";
mySqlCommand.Parameters["@Qalaqi"].Value = "borjomi";
mySqlCommand.Parameters["@Rating"].Value = 330;
mySqlCommand.Parameters["@Snum"].Value = 104;
//        SQL-მოთხოვნის შესრულება ExecuteNonQuery() მეთოდის გამოყენებით
mySqlCommand.ExecuteNonQuery();
}
//        მეთოდი ახდენს ცხრილიდან ჩანაწერის წაშლას
public static void RemoveRow(SqlConnection mySqlConnection,
                                    SqlDataAdapter mySqlDataAdapter, DataSet myDataSet)
{
//        SQL-მოთხოვნის ფორმირება
string deleteString = "DELETE FROM Customers WHERE Cnum = @Cnum";
//        SqlCommand ობიექტის შექმნა SQL-მოთხოვნის შესანახად
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//        SqlCommand ობიექტის CommandText თვისებისათვის SQL-მოთხოვნიანი  სტრიქონის
//        მინიჭება
mySqlCommand.CommandText = deleteString;
//        SqlCommand ობიექტის Parameters კოლექციის Add() მეთოდის
//        გამოყენებით პარამეტრების დამატება SqlCommand ობიექტისათვის
mySqlCommand.Parameters.Add("@Cnum", SqlDbType.Int, 4);
//        პარამეტრებისათვის საჭირო მნიშვნელობების მინიჭება Value თვისების საშუალებით
mySqlCommand.Parameters["@Cnum"].Value = 208;
//        SQL-მოთხოვნის შესრულება ExecuteNonQuery() მეთოდის გამოყენებით
mySqlCommand.ExecuteNonQuery();
}

private void button1_Click(object sender, System.EventArgs e)
{
label2.Text = "";
//        სტრიქონის ფორმირება, რომელიც შეიცავს მონაცემთა ბაზასთან შეერთების
//        პარამეტრებს
string connectionString = "server=localhost;database=Student;uid=sa;pwd=paroli";
//        SqlConnection ობიექტის შექმნა მონაცემთა ბაზასთან შეერთების დამყარებისათვის
SqlConnection mySqlConnection = new SqlConnection(connectionString);
//        SELECT მოთხოვნის ფორმირება
```

```
string selectString = "SELECT Cnum, Cname, Qalaqi, Rating, Snum FROM Customers";
//      SqlCommand ობიექტის შექმნა SELECT მოთხოვნისათვის
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//      SqlCommand ობიექტის CommandText თვისებისათვის SELECT-მოთხოვნის მინიჭება
mySqlCommand.CommandText = selectString;
//      SqlDataAdapter ობიექტის შექმნა
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();

//      SqlDataAdapter ობიექტის SelectCommand თვისებისათვის SqlCommand ობიექტზე
//      მიმთითებლის მინიჭება
mySqlDataAdapter.SelectCommand = mySqlCommand;
//      DataSet ობიექტის შექმნა SELECT-მოთხოვნის შედეგების შესანახად
DataSet myDataSet = new DataSet();
//      მონაცემთა ბაზასთან შეერთების გახსნა
mySqlConnection.Open();
//      SqlDataAdapter ობიექტის Fill() მეთოდის გამოძახება ცხრილიდან ჩანაწერების მისაღებად
//      და DataTable ობიექტში ლოკალურად შესანახად
string dataTableName = "Customers";
mySqlDataAdapter.Fill(myDataSet, dataTableName);

dataGrid1.DataSource = myDataSet;
dataGrid1.DataMember = "Customers";
//      DataSet ობიექტიდან DataTable ობიექტის მიღება
DataTable myDataTable = myDataSet.Tables[dataTableName];
//      DataTable ცხრილიდან ჩანაწერების ასახვა
DisplayDataTable(mySqlDataAdapter, myDataSet, myDataTable);
//      ახალი ჩანაწერის ჩამატება
AddRow(mySqlConnection, mySqlDataAdapter, myDataSet);
DisplayDataTable(mySqlDataAdapter, myDataSet, myDataTable);
}

private void button2_Click(object sender, System.EventArgs e)
{
label2.Text = "";
//      სტრიქონის ფორმირება, რომელიც შეიცავს მონაცემთა ბაზასთან შეერთების
//      პარამეტრებს
string connectionString = "server=localhost;database=Student;uid=sa;pwd=paroli";
//      SqlConnection ობიექტის შექმნა მონაცემთა ბაზასთან შეერთების დამყარებისათვის
SqlConnection mySqlConnection = new SqlConnection(connectionString);
//      SELECT მოთხოვნის შემცველი სტრიქონის ფორმირება
string selectString = "SELECT Cnum, Cname, Qalaqi, Rating, Snum FROM Customer ";
//      SqlCommand ობიექტის შექმნა SELECT მოთხოვნისათვის
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//      SqlCommand ობიექტის CommandText თვისებისათვის SELECT-მოთხოვნის მინიჭება
mySqlCommand.CommandText = selectString;
```

```
//        SqlDataAdapter ობიექტის შექმნა
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
//        SqlDataAdapter ობიექტის SelectCommand თვისებისათვის SqlCommand ობიექტზე
//        მიმთითებლის მინიჭება
mySqlDataAdapter.SelectCommand = mySqlCommand;
//        DataSet ობიექტის შექმნა SELECT-მოთხოვნის შედეგების შესანახად
DataSet myDataSet = new DataSet();
//        მონაცემთა ბაზასთან შეერთების გახსნა
mySqlConnection.Open();
//        SqlDataAdapter ობიექტის Fill() მეთოდის გამოძახება ცხრილიდან ჩანაწერების
//        მისაღებად და DataTable ობიექტში ლოკალურად შესანახად
string dataTableName = "Customers";
mySqlDataAdapter.Fill(myDataSet, dataTableName);

dataGrid1.DataSource = myDataSet;
dataGrid1.DataMember = "Customers";
//        DataSet ობიექტიდან DataTable ობიექტის მიღება
DataTable myDataTable = myDataSet.Tables[dataTableName];
//        DataTable ცხრილიდან ჩანაწერების ასახვა
DisplayDataTable(mySqlDataAdapter, myDataSet, myDataTable);
//        ჩანაწერის შეცვლა
ModifyRow(mySqlConnection, mySqlDataAdapter, myDataSet);
DisplayDataTable(mySqlDataAdapter, myDataSet, myDataTable);
}
private void button3_Click(object sender, System.EventArgs e)
{
label2.Text = "";
//        სტრიქონის ფორმირება, რომელიც შეიცავს მონაცემთა ბაზასთან შეერთების
//        პარამეტრებს
string connectionString = "server=localhost;database=Student;uid=sa;pwd=paroli";
//        SqlConnection ობიექტის შექმნა ბაზასთან შეერთების დამყარებისათვის
SqlConnection mySqlConnection = new SqlConnection(connectionString);
//        SELECT მოთხოვნის ფორმირება
string selectString = "SELECT Cnum, Cname, Qalaqi, Rating, Snum FROM Customers";
//        SqlCommand ობიექტის შექმნა SELECT მოთხოვნისათვის
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//        SqlCommand ობიექტის CommandText თვისებისათვის SELECT-მოთხოვნის მინიჭება
mySqlCommand.CommandText = selectString;
//        SqlDataAdapter ობიექტის შექმნა
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
//        SqlDataAdapter ობიექტის SelectCommand თვისებისათვის SqlCommand ობიექტზე
//        მიმთითებლის მინიჭება
mySqlDataAdapter.SelectCommand = mySqlCommand;
//        DataSet ობიექტის შექმნა SELECT-მოთხოვნის შედეგების შესანახად
DataSet myDataSet = new DataSet();
```

```
//      მონაცემთა ბაზასთან შეერთების გახსნა
mySqlConnection.Open();
//      SqlDataAdapter ობიექტის Fill() მეთოდის გამოძახება ცხრილიდან ჩანაწერების
//      მისაღებად და DataTable ობიექტში ლოკალურად შესანახად
string dataTableName = "Customers";
mySqlDataAdapter.Fill(myDataSet, dataTableName);

dataGrid1.DataSource = myDataSet;
dataGrid1.DataMember = "Customers";
//      DataSet ობიექტიდან DataTable ობიექტის მიღება
DataTable myDataTable = myDataSet.Tables[dataTableName];
//      DataTable ცხრილიდან ჩანაწერების მიღება
DisplayDataTable(mySqlDataAdapter, myDataSet, myDataTable);
//      ჩანაწერის ჭაშლა
RemoveRow(mySqlConnection, mySqlDataAdapter, myDataSet);
DisplayDataTable(mySqlDataAdapter, myDataSet, myDataTable);
}
```

ტრანზაქციებთან მუშაობა ADO.NET საშუალებებით

ADO.NET გარემოში ტრანზაქციებთან სამუშაოდ გამოიყენება SqlTransaction ობიექტი. დავუშვათ, გვინდა ორი INSERT მოთხოვნის შესრულება. ამისათვის, აუცილებელია შემდეგი მოქმედებების შესრულება:

1. SqlTransaction ობიექტის შექმნა და ტრანზაქციის დაწყება SqlConnection ობიექტის BeginTransaction() მეთოდის გამოძახების გზით.
2. SqlCommand ობიექტის შექმნა SQL-მოთხოვნის შესანახად.
3. SqlCommand ობიექტის Transaction თვისების დაყენება.
4. პირველი INSERT მოთხოვნის შემცველი სტრიქონის ფორმირება.
5. SqlCommand ობიექტის CommandText თვისებისათვის პირველი INSERT მოთხოვნის შემცველი სტრიქონის მინიჭება.
6. პირველი INSERT მოთხოვნის შესრულება SqlCommand ობიექტის ExecuteNonQuery() მეთოდის საშუალებით.
7. მეორე INSERT მოთხოვნის შემცველი სტრიქონის ფორმირება.
8. SqlCommand ობიექტის CommandText თვისებისათვის მეორე INSERT მოთხოვნის შემცველი სტრიქონის მინიჭება.
9. მეორე INSERT მოთხოვნის შესრულება SqlCommand ობიექტის ExecuteNonQuery() მეთოდის საშუალებით.
10. ტრანზაქციის დაფიქსირება SqlTransaction ობიექტის Commit() მეთოდის გამოყენებით. შედეგად, INSERT მოთხოვნებით ჩამატებული ორი ჩანაწერი დაემატება მონაცემთა ბაზას.

მოყვანილ პროგრამაში ორი INSERT მოთხოვნა სრულდება ტრანზაქციაში.

```
{
//      პროგრამა 19.3
//      პროგრამაში ხდება ტრანზაქციასთან მუშაობის დემონსტრირება
//      სტრიქონის ფორმირება, რომელიც შეიცავს მონაცემთა ბაზასთან შეერთების
```

პარამეტრებს
```
string connectionString = "server=localhost;database=Student;uid=sa;pwd=paroli";
//      SqlConnection ობიექტის შექმნა მონაცემთა ბაზასთან შესაერთებლად
SqlConnection mySqlConnection = new SqlConnection(connectionString);
//      მონაცემთა ბაზასთან შეერთების გახსნა
mySqlConnection.Open();
//      SqlTransaction ობიექტის შექმნა და ტრანზაქციის დაწყება mySqlConnection
//      ობიექტის BeginTransaction() მეთოდის გამოძახების გზით
SqlTransaction mySqlTransaction = mySqlConnection.BeginTransaction();
//      mySqlCommand ობიექტის შექმნა SQL-მოთხოვნის შესასრულებლად
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//      mySqlCommand ობიექტის Transaction თვისების დაყენება
mySqlCommand.Transaction = mySqlTransaction;
//      სტრიქონის ფორმირება, რომელიც შეიცავს პირველ INSERT მოთხოვნას
string insertString = "INSERT INTO Salespeople (Snum, Sname, Qalaqi, Comm)
                                VALUES (106, 'qevxiSvili', 'bakuriani', 16)";
//      mySqlCommand ობიექტის CommandText თვისებისათვის INSERT-მოთხოვნის
//      შემცველი სტრიქონის მინიჭება
mySqlCommand.CommandText = insertString;
//      პირველი INSERT მოთხოვნის შესრულება
mySqlCommand.ExecuteNonQuery();
//      მეორე INSERT მოთხოვნის ფორმირება
insertString = "INSERT INTO Customers (Cnum, Cname, Qalaqi, Rating, Snum)
                                VALUES (209, 'Roma&Co', 'bakuriani', 220, 106)";
//      mySqlCommand ობიექტის CommandText თვისებისათვის
//      INSERT-მოთხოვნიანი სტრიქონის მინიჭება
mySqlCommand.CommandText = insertString;
//      მეორე INSERT მოთხოვნის მინიჭება
mySqlCommand.ExecuteNonQuery();
//      ტრანზაქციის დაფიქსირება
mySqlTransaction.Commit();
//      ბაზასთან შეერთების დახურვა
mySqlConnection.Close();
}
```

ჩანაწერების ფილტრაცია და დახარისხება

დახარისხების ქვეშ იგულისხმება ცხრილის ჩანაწერების დალაგება ერთი ან მეტი ველის მნიშვნელობების ზრდადობის ან კლებადობის მიხედვით. ფილტრაციის ქვეშ იგულისხმება ცხრილიდან იმ ჩანაწერების ამორჩევა, რომლებიც აკმაყოფილებენ ფილტრში მითითებულ პირობებს. DataTable ცხრილში მოთავსებული ჩანაწერების ფილტრაციისა და დახარისხებისათვის შეგვიძლია გამოვიყენოთ DataView ობიექტი. ფილტრი ეთითება DataView ობიექტის RowFilter თვისების საშუალებით. ჩანაწერების

დახარისხებისათვის DataView ობიექტის Sort თვისებაში უნდა მივუთითოთ დასახარისხებელი ველების სახელები.

ჩანაწერების ფილტრაციისა და დახარისხებისათვის შეგვიძლია, აგრეთვე, DataTable ობიექტის Select() მეთოდის გამოყენება.

დავუშვათ გვაქვს myDataSet სახელის მქონე DataSet ობიექტი, რომელშიც შექმნილი იყო DataTable ობიექტი, რომელიც შეიცავს Customers ცხრილის Cnum, Cname, Qalaqi, Rating და Snum ველებს. დავუშვათ, გვინდა ამ ჩანაწერების გაფილტვრა ისე, რომ მივიღოთ მონაცემები იმ შემკვეთების შესახებ, რომელთა რეიტინგი აღემატება 150-ს. ამასთან, ეს ჩანაწერები უნდა იყოს დახარისხებული ზრდადობით Cnum ველის მნიშვნელობების მიხედვით. DataView ტიპის myDataView ობიექტი, რომელიც შეიცავს DataTable ცხრილის ჩანაწერებს, იქმნება შემდეგი გამოსახულების საშუალებით:

```
DataView myDataView = new DataView(myDataSet.Tables[“Customers”]);
```

შემდეგ, შეგვიძლია გავფილტროთ ჩანაწერები DataView ობიექტში ისე, რომ დარჩეს მხოლოდ ის ჩანაწერები, რომლებსაც Rating ველში აქვთ 150-ზე მეტი მნიშვნელობა. ასეთი ფილტრაცია სრულდება myDataView ობიექტის RowFilter თვისებისათვის “Rating > 150” მნიშვნელობის მინიჭებით:

```
myDataView.RowFilter = “Rating > 150”;
```

გაფილტრული ჩანაწერები შეგვიძლია დავახარისხოთ ზრდადობით Cnum ველის მნიშვნელობების მიხედვით myDataView ობიექტის Sort თვისებისათვის “Cnum ASC” მნიშვნელობის მინიჭების გზით:

```
myDataView.Sort = “Cnum ASC”;
```

შეიძლება, აგრეთვე, ჩანაწერების დახარისხება კლებადობით ASC სიტყვის ნაცვლად DESC სიტყვის გამოყენებით:

```
myDataView.Sort = “Cnum DESC”;
```

შეიძლება, ჩანაწერების დახარისხება რამდენიმე ველის მიხედვით, მაგალითად, Snum ველის მიხედვით ზრდადობით და Rating ველის მიხედვით კლებადობით:

```
myDataView.Sort = “Snum ASC, Rating DESC”;
```

ქვემოთ მოყვანილ პროგრამაში DataView ობიექტი გამოიყენება Customers ცხრილიდან ამოღებული ჩანაწერების ფილტრაციისათვის. DataView ობიექტში მოთავსდება მხოლოდ ის ჩანაწერები, რომლებსაც Rating ველში აქვთ 150-ზე მეტი მნიშვნელობა. სრულდება ჩანაწერების დახარისხება ზრდადობით ან კლებადობით Cnum ველის მნიშვნელობების მიხედვით. DataView ობიექტის შემცველობის გამოტანა ხდება dataGrid და label კომპონენტებში.

```
//      პროგრამა 19.4
//      პროგრამაში ხდება Customers  ცხრილის ფილტრაცია და დახარისხება
private void button1_Click(object sender, System.EventArgs e)
{
string connectionString = "server=localhost;database=Student;uid=sa;pwd=paroli";
SqlConnection mySqlConnection = new SqlConnection(connectionString);
string selectString = "SELECT * FROM Customers ";
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText = selectString;
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
DataSet myDataSet = new DataSet();
```

```
mySqlConnection.Open();
mySqlDataAdapter.Fill(myDataSet, "Customers");
DataView myDataView = new DataView(myDataSet.Tables["Customers"]);
myDataView.RowFilter = "Rating > 150";
label1.Text = "";
foreach ( DataRowView myDataRowView in myDataView )
{
for ( int count = 0; count < myDataView.Table.Columns.Count; count++ )
        label1.Text += myDataRowView[count].ToString();
        label1.Text += '\n';
}
//  dataGrid კომპონენტში მონაცემების გამოტანა
myDataSet.Clear();
mySqlDataAdapter.Fill(myDataSet, "Customers");
dataGrid1.DataSource = myDataView;
mySqlConnection.Close();
}
private void button2_Click(object sender, System.EventArgs e)
{
string connectionString = "server=localhost;database=Student;uid=sa;pwd=paroli";
SqlConnection mySqlConnection = new SqlConnection(connectionString);
string selectString = "SELECT * FROM Customers ";
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText = selectString;
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
DataSet myDataSet = new DataSet();
mySqlConnection.Open();
mySqlDataAdapter.Fill(myDataSet, "Customers");
DataView myDataView = new DataView(myDataSet.Tables["Customers"]);
//        ASC - ახარისხებს ზრდადობით, DESC - კლებადობით
myDataView.Sort = "Rating ASC";
label1.Text = "";
foreach ( DataRowView myDataRowView in myDataView )
{
        for ( int count = 0; count < myDataView.Table.Columns.Count; count++ )
                label1.Text += myDataRowView[count].ToString();
                label1.Text += '\n';
}
//  dataGrid კომპონენტში მონაცემების გამოტანა
myDataSet.Clear();
mySqlDataAdapter.Fill(myDataSet, "Customers");
dataGrid1.DataSource = myDataView;
mySqlConnection.Close();
}
```

```
private void button4_Click(object sender, System.EventArgs e)
{
string connectionString = "server=localhost;database=Student;uid=sa;pwd=paroli";
SqlConnection mySqlConnection = new SqlConnection(connectionString);
string selectString = "SELECT * FROM Customers ";
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText = selectString;
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
DataSet myDataSet = new DataSet();
mySqlConnection.Open();
mySqlDataAdapter.Fill(myDataSet, "Customers");
DataView myDataView = new DataView(myDataSet.Tables["Customers"]);
//        ASC - ახარისხებს ზრდადობით, DESC - კლებადობით
myDataView.Sort = "Rating DESC";
label1.Text = "";
foreach ( DataRowView myDataRowView in myDataView )
{
        for ( int count = 0; count < myDataView.Table.Columns.Count; count++ )
                label1.Text += myDataRowView[count].ToString();
                label1.Text += '\n';
}
//  dataGrid კომპონენტში მონაცემების გამოტანა
myDataSet.Clear();
mySqlDataAdapter.Fill(myDataSet, "Customers");
dataGrid1.DataSource = myDataView;
mySqlConnection.Close();
}
```

DataTable ობიექტებს შორის ბმების განსაზღვრა

DataSet ობიექტში შეიძლება განისაზღვროს ბმა (კავშირი) ორ ან მეტ DataTable ობიექტს შორის. მაგალითად, Customers და Orders ცხრილები შეიძლება იმყოფებოდნენ “მშობელი-შვილი” (“მთავარი-დეტალი”) მიმართებაში. Orders ცხრილის Cnum ველი არის გარე გასაღები, რომელიც მიმართავს Customers ცხრილის Cnum ველს. ამ მიმართების მოდელირება შესაძლებელია პროგრამაში. შემდეგ, შეგვიძლია გამოვიყენოთ მშობელი DataTable ობიექტის GetChildRows() მეთოდი შვილობილი DataTable ცხრილიდან ჩანაწერების მისაღებად.

დავუშვათ, გვაქვს DataSet ტიპის myDataSet ობიექტი, რომელიც შეიცავს ორ DataTable ობიექტს: “Customers” და “Orders”. როგორც აღვნიშნეთ Customers და Orders ცხრილები იმყოფებიან “მშობელი-შვილი” კავშირში. DataSet ობიექტში ამის მოდელირება ხდება Customers და Orders ობიექტებს შორის ბმის შექმნით. ბმებთან მიმართვა DataSet ობიექტში შეგვიძლია Relationships კოლექციის გამოყენებით, ხოლო ახალი ბმა იქმნება ამ კოლექციის Add() მეთოდის გამოძახებით.

DataSet ობიექტის Relationships თვისება აბრუნებს DataRelationCollection კლასის ობიექტს. მასში მოთავსებულია DataRelation ობიექტები. Add() მეთოდი განსაზღვრულია DataRelationCollection კლასში და იძლევა DataSet ობიექტის DataRelationCollection ობიექტში DataRelation ობიექტის დამატების საშუალებას.

მოყვანილ მაგალითში ნაჩვენებია Add() მეთოდის გამოყენება myDataSet ობიექტში Customers და Orders ობიექტებს შორის ბმის დასამატებლად:

```
myDataSet.Relations.Add("Orders",  myDataSet.Tables["Customers"].Columns["Cnum"],
                                   myDataSet.Tables["Orders"].Columns["Cnum"]);
```

Add() მეთოდის პირველი პარამეტრია სტრიქონი, რომელიც შეიცავს შესაქმნელი ბმის სახელს. მეორე და მესამე პარამეტრებია DataTable ობიექტში მოთავსებული DataColumn ობიექტები. პირველი DataColumn ობიექტი ესაა მშობელი ცხრილის პირველადი გასაღები, ხოლო მეორე DataColumn ობიექტი - შვილობილი ცხრილის გარე გასაღები.

როგორც მაგალითიდან ჩანს ბმის სახელია "Orders". DataColumn ობიექტი, რომელიც პირველადი გასაღებია, წარმოადგენს Customers ცხრილის (ობიექტის) Cnum ველს. DataColumn ობიექტი, რომელიც გარე გასაღებია, წარმოადგენს Orders ცხრილის (ობიექტის) Cnum ველს.

მოყვანილ პროგრამაში ნაჩვენებია ბმის შექმნა და გამოყენება. აქ GetChildRows() მეთოდი გამოიყენება შვილობილი Orders ცხრილიდან ჩანაწერების მისაღებად.

```
{
//      პროგრამა 19.5
//      პროგრამაში ხდება ცხრილებს შორის ბმის შექმნის დემონსტრირება
string connectionString = "server=localhost;database=Student;uid=sa;pwd=paroli";
SqlConnection mySqlConnection = new SqlConnection(connectionString);
string selectString = "SELECT * FROM Salespeople";
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText = selectString;
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
DataSet myDataSet = new DataSet();
mySqlConnection.Open();
mySqlDataAdapter.Fill(myDataSet, "Salespeople");
dataGrid1.DataSource = myDataSet;
dataGrid1.DataMember = "Salespeople";
selectString = "SELECT * FROM Orders WHERE Onum = 301";
mySqlCommand.CommandText = selectString;
mySqlDataAdapter.Fill(myDataSet, "Orders");
//      Relations კოლექციის Add() მეთოდის გამოყენებით ერთმანეთთან ვაკავშირებთ
//      DataTable ობიექტის Customers და Orders ცხრილებს
myDataSet.Relations.Add("Orders",
                                myDataSet.Tables["Salespeople"].Columns["Snum"],
                                myDataSet.Tables["Orders"].Columns["Snum"]);
label1.Text = "";
label2.Text = "";
DataTable Customers = myDataSet.Tables["Customers"];
```

```
foreach ( DataRow Customers in Customers.Rows )
{
        label1.Text += Customers["Snum"]   + “   “;
        label1.Text += Customers["Sname"]  + “   “;
        label1.Text += Customers["Qalaqi"] + “   “;
        label1.Text += Customers["Comm"]   + “   “;
        label1.Text += '\n';
        DataRow[] Orders = Customers.GetChildRows("Orders");
        foreach ( DataRow order in Orders )
        {
                label2.Text += order["Onum"]   + “   “;
                label2.Text += order["Tanxa"]  + “   “;
                label2.Text += order["Tarigi"] + “   “;
                label2.Text += order["Cnum"]   + “   “;
                label2.Text += order["Snum"];
                 label2.Text += '\n';
        }
}
mySqlConnection.Close();
}
```

დანართი 1. ამოცანები თავების მიხედვით

თავი 2. C# ენის საფუძვლები

არითმეტიკული გამოსახულებები

შეადგინეთ პროგრამა, რომელიც გამოთვლის ქვემოთ მოყვანილი არითმეტიკული გამოსახულებების მნიშვნელობებს:

2.1.1. $$y = 1 + x + \frac{x^2}{2}$$

2.1.2. $$y = \frac{x^2 + z^2}{1 - \frac{x^2 - z^2}{2}}$$

2.1.3. $$y = \sqrt{1 + \sqrt{|x|}}$$

2.1.4. $$y = (1 + x)\frac{x - \frac{\sqrt{x - 1}}{4}}{e^x + \frac{1}{x^2 + 4}}$$

2.1.5. $$y = \sin x^3 + \cos^3 x^4 - e^{\sqrt{x}}$$

შეადგინეთ პროგრამა, რომელიც:

2.1.6. გამოთვლის დენის ძალას ($J = \frac{U}{R}$).

2.1.7. გამოთვლის ხუთი რიცხვის საშუალო არითმეტიკულს.

2.1.8. გამოთვლის ოთხი რიცხვის საშუალო გეომეტრიულს.

2.1.9. გამოთვლის სამკუთხედის ფართობს ($S = \frac{ah}{2}$).

2.1.10. გამოთვლის სამკუთხედის ფართობს ჰერონის ფორმულის გამოყენებით ($S = \sqrt{p(p-a)(p-b)(p-c)}$).

2.1.11. გამოთვლის სამკუთხედის პერიმეტრს ($p = a + b + c$).

2.1.12. გამოთვლის კვადრატის ფართობს.

2.1.13. გამოთვლის მართკუთხედის პერიმეტრს.

2.1.14. გამოთვლის წრეწირის სიგრძეს ($C = 2\pi\pi$).

2.1.15. გამოთვლის წრეწირის ფართობს ($S = \pi r^2$).

$$S = \frac{a+b}{2}h$$

2.1.16. გამოთვლის ტრაპეციის ფართობს ( ).
2.1.17. გამოთვლის ტრაპეციის პერიმეტრს.

ლოგიკური გამოსახულებები

შეადგინეთ პროგრამა, რომელიც გამოთვლის ქვემოთ მოყვანილი ლოგიკური გამოსახულებების მნიშვნელობებს:

2.2.1. $y = x1 \&\& x2 \,||\, x3 \&\& !x4$, სადაც x1=true, x2=false, x3=true, x4=false

2.2.2. $y = x1 \,||\, !x2 \&\& x3 \,||\, x4$, სადაც x1=false, x2=false, x3=true, x4=true

2.2.3. $y = !x1 \&\& !x2 \&\& x3 \,||\, x4$, სადაც x1=true, x2=true, x3=false, x4=false

2.2.4. $y = !x1 \,||\, x2 \,||\, !x3 \&\& x4$, სადაც x1=false, x2=true, x3=true, x4=false

თავი 3. მმართველი ოპერატორები

if ოპერატორი

შეადგინეთ პროგრამა, რომელიც:

3.1.1. დაადგენს რიცხვი დადებითია თუ უარყოფითი. რიცხვი textBox კომპონენტიდან შეიტანეთ. label კომპონენტში გამოიტანეთ შესაბამისი შეტყობინება.
3.1.2. დაადგენს რიცხვი კენტია თუ ლუწი.
3.1.3. დაადგენს რიცხვი არის თუ არა 5-ის ჯერადი.
3.1.4. დაადგენს რიცხვი არის თუ არა 30-ის ტოლი.
3.1.5. დაადგენს რიცხვი მეტია თუ არა 10-ზე.
3.1.6. დაადგენს რიცხვი არის თუ არა მეტი ან ტოლი 15-ზე.
3.1.7. დაადგენს რიცხვი არის თუ არა ნაკლები ან ტოლი 23-ზე.
3.1.8. ორ რიცხვს შორის იპოვის მაქსიმალურს.
3.1.9. ორ რიცხვს შორის იპოვის მინიმალურს.

ჩადგმული if ოპერატორი

შეადგინეთ პროგრამა, რომელიც:

3.2.1. სამ რიცხვს შორის იპოვის მაქსიმალურს.
3.2.2. სამ რიცხვს შორის იპოვის მინიმალურს.
3.2.3. დაადგენს ორ რიცხვს შორის არის თუ არა დადებითი.
3.2.4. დაადგენს ორ რიცხვს შორის არის თუ არა უარყოფითი.
3.2.5. დაადგენს ორ რიცხვს შორის არის თუ არა 7-ის ჯერადი.
3.2.6. დაადგენს სამ რიცხვს შორის არის თუ არა ლუწი.
3.2.7. დაადგენს სამ რიცხვს შორის არის თუ არა დადებითი.
3.2.8. დაადგენს სამ რიცხვს შორის არის თუ არა უარყოფითი.
3.2.9. დათვლის სამ რიცხვს შორის რამდენია კენტი.
3.2.10. დათვლის სამ რიცხვს შორის რამდენია 5-ის ტოლი.
3.2.11. დათვლის სამ რიცხვს შორის რამდენია 10-ზე მეტი.

ლოგიკა

შეადგინეთ პროგრამა, რომელიც:

3.3.1. დაადგენს x რიცხვი არის თუ არა $0 < x \leq 17$ დიაპაზონში მოთავსებული.

3.3.2. დაადგენს x რიცხვი არის თუ არა $10 \leq x < 19$ დიაპაზონში მოთავსებული.

3.3.3. დაადგენს x რიცხვი არის თუ არა 25-ზე ნაკლები ან 100-ზე მეტი.

3.3.4. დაადგენს x რიცხვი არის თუ არა 30-ის ტოლი ან 5-ზე ნაკლები.

for, while, do-while ოპერატორები

შეადგინეთ პროგრამა, რომელიც გამოთვლის შემდეგი გამოსახულებების მნიშვნელობებს:

3.4.1. $$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5}$$

3.4.2. $$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9}$$

3.4.3. $$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2}$$

3.4.4. $$\frac{\pi^2}{8} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2}$$

3.4.5. $$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5}$$

3.4.6. $$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \frac{x^5}{5}$$

3.4.7. $$\text{arctg}x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9}$$

3.4.8. $$\frac{1}{1+x} = 1 - x + x^2 - x^3 + x^4 - x^5$$

3.4.9. $$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + x^5$$

3.4.10. $$\sqrt{1+x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3$$

3.4.11. $$\frac{1}{\sqrt{1+x}} = 1 - \frac{1}{2}x + \frac{3}{8}x^2 - \frac{5}{16}x^3$$

3.4.12. $$\arcsin x = x + \frac{1}{2 \cdot 3}x^3 + \frac{1 \cdot 3}{3 \cdot 4 \cdot 5}x^5$$

3.4.13. $$\ln\frac{1+x}{1-x} = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7}\right)$$

3.4.14. $y = \frac{1 \cdot 2}{3 \cdot 4} + \frac{3 \cdot 4}{5 \cdot 6} + \frac{5 \cdot 6}{7 \cdot 8}$

შეადგინეთ პროგრამა, რომელიც:

3.4.15. დათვლის ლუწი რიცხვების რაოდენობას 5-დან 27-მდე დიაპაზონში.

3.4.16. შეკრებს კენტ რიცხვებს 10-დან 20-მდე დიაპაზონში.

3.4.17. დათვლის 8-ის ჯერადი რიცხვების რაოდენობას 20-დან 50-მდე დიაპაზონში და გამოიტანს მათ label კომპონენტში.

3.4.18. იპოვის 10-დან 70-მდე იმ რიცხვების ჯამს, რომლებიც არ არიან 8-ს ჯერადი.

continue, break ოპერატორები

შეადგინეთ პროგრამა, რომელიც:

3.5.1. შეკრებს 1-დან 20-მდე რიცხვებს მანამ, სანამ ჯამი არ გახდება 24-ზე მეტი.

3.5.2. დათვლის 4-ის ჯერადი რიცხვებს 6-დან 79-მდე დიაპაზონში მანამ, სანამ 4-ის ჯერადი რიცხვი არ გახდება 50-ზე მეტი.

3.5.3. შეკრებს 9-ის ჯერად რიცხვებს 2-დან 47-მდე დიაპაზონში მანამ, სანამ 9-ის ჯერადი რიცხვი იქნება 30-ზე ნაკლები.

თავი 4. მასივები, სტრიქონები და ბიტობრივი ოპერაციები

ერთგანზომილებიანი მასივები

შეადგინეთ პროგრამა, რომელიც:

4.1.1. იპოვის M_{10} მასივის ელემენტების ჯამს.

4.1.2. იპოვის M_{10} მასივის ელემენტების ნამრავლს.

4.1.3. იპოვის M_{10} მასივის ელემენტების კვადრატების ჯამს.

4.1.4. იპოვის M_{10} მასივის მინიმალურ ელემენტს.

4.1.5. იპოვის M_{10} მასივის მაქსიმალური ელემენტის ინდექსს.

4.1.6. იპოვის M_{10} მასივის ლუწი ელემენტების ჯამს.

4.1.7. იპოვის M_{10} მასივის კენტი ელემენტების რაოდენობას.

4.1.8. იპოვის M_{10} მასივის დადებითი ელემენტების ჯამს.

4.1.9. იპოვის M_{10} მასივის უარყოფითი ელემენტების რაოდენობას.

4.1.10. იპოვის M_{10} მასივის არანულოვანი ელემენტების ჯამს.

4.1.11. იპოვის M_{10} მასივის ნულოვანი ელემენტების რაოდენობას.

4.1.12. იპოვის M_{10} მასივის ლუწი ინდექსის მქონე ელემენტების ჯამს.

4.1.13. იპოვის M_{10} მასივის კენტი ინდექსის მქონე ელემენტების რაოდენობას.

4.1.14. M_{10} მასივის დადებით ელემენტებს გადაწერს N_{10} მასივში.

4.1.15. M_{10} მასივის 5-ის ჯერად ელემენტებს გადაწერს N_{10} მასივში.

4.1.16. M_{10} მასივის არანულოვან ელემენტებს გადაწერს N_{10} მასივში.

4.1.17. M_{10} მასივის კენტი ინდექსის მქონე ელემენტებს გადაწერს N_5 მასივში.

4.1.18. M_{10} მასივის ლუწი ინდექსის მქონე ელემენტებს გადაწერს N_5 მასივში.

4.1.19. M_{10} მასივის უარყოფით ელემენტებს შეცვლის 0-ებით.

4.1.20. M_{10} მასივის კენტ ელემენტებს შეცვლის მათი კვადრატებით.

4.1.21. M_{10} მასივის დადებითი ელემენტების ინდექსებს გადაწერს N_{10} მასივში.

4.1.22. M_{10} მასივის ლუწ ელემენტებს გადაწერს N_{10} მასივში.

4.1.23. M_{10} მასივის კენტი ინდექსის მქონე ელემენტებს გადაწერს N_5 მასივში.

4.1.24. M_{10} მასივის 3-ის ჯერადი ელემენტების ინდექსებს გადაწერს N_{10} მასივში.
4.1.25. M_{10} მასივის ელემენტებს უკუ (შებრუნებული) თანმიმდევრობით დაალაგებს.
4.1.26. M_{20} მასივის ლუწინდექსიან ელემენტებს გადაწერს A_{10} მასივში, ხოლო კენტინდექსიან ელემენტებს კი - B_{10} მასივში.
4.1.27. იპოვის მაქსიმალურ სხვაობას M_{10} და N_{10} მასივების ელემენტებს შორის.
4.1.28. შექმნის A_{10} მასივს, რომელშიც ჯერ მოთავსებული იქნება M_{10} მასივის არანულოვანი ელემენტები, შემდეგ კი - ნულოვანი.
4.1.29. M_{10} მასივში ელემენტებს დაალაგებს ისე, რომ ჯერ მოთავსთდეს არანულოვანი ელემენტები, შემდეგ კი - ნულოვანი.
4.1.30. იპოვის M_{10} მასივის პირველ უარყოფით ელემენტს.
4.1.31. იპოვის M_{10} მასივის პირველი დადებითი ელემენტის ინდექსს.
4.1.32. M_{10} მასივის ელემენტებს ციკლურად დაძრავს მარჯვნივ n ელემენტით.
4.1.33. M_{10} მასივის ელემენტებს ციკლურად დაძრავს მარცხნივ n ელემენტით.
4.1.34. მოახდენს ზრდადობის მიხედვით დალაგებული A_{10} და B_{10} მასივების შერწყმას და შექმნის ახალ ზრდადობის მიხედვით დალაგებულ M_{20} მასივს.
4.1.35. შეამოწმებს ემთხვევა თუ არა M_{10} მასივის პირველი ნახევარი მის მეორე ნახევარს.
4.1.36. შეამოწმებს განსხვავდებიან თუ არა M_{10} და N_{10} მასივები.
4.1.37. განსაზღვრავს შეიცავს თუ არა M_{10} მასივი N_{10} მასივს.
4.1.38. განსაზღვრავს M_{10} მასივის ერთმანეთის მიყოლებით მდებარე ნულის ტოლი ელემენტების ყველაზე გრძელ მიმდევრობას.
4.1.39. დათვლის M_{10} მასივის განსხვავებული ელემენტების რაოდენობას.
4.1.40. იპოვის M_{10} მასივის უარყოფითი ელემენტებიდან უდიდესის ინდექსს.
4.1.41. გამოთვლის M_{10} მასივის ელემენტების საშუალო არითმეტიკულს.
4.1.42. გამოთვლის M_{10} მასივის ელემენტების საშუალო გეომეტრიულს.
4.1.43. M_{10} მასივში ადგილებს შეუცვლის კენტი და ლუწი ინდექსის მქონე ელემენტებს.
4.1.44. M_{10} მასივში განსაზღვრავს გაორმაგებული კენტი რიცხვების რაოდენობას და ჩაწერს მათ N მასივში.
4.1.45. დაადგენს არის თუ არა M_{10} მასივში ერთნაირი ელემენტები.
4.1.46. N მასივში გადაწერს M_{10} მასივის იმ ელემენტებს, რომლებიც 5-ზე გაყოფისას იძლევიან 1-დან 4-ის ტოლ ნაშთს.
4.1.47. შეკუმშავს M_{10} მასივს მისი გამეორებადი ელემენტების წაშლის გზით.
4.1.48. შეკუმშავს M_{10} მასივს მისი იმ ელემენტების წაშლის გზით, რომლებიც მასივში ერთხელ გვხვდება.
4.1.49. შეკუმშავს M_{10} მასივს მისი ნულოვანი ელემენტების წაშლის გზით.
4.1.50. M_{10} მასივში იპოვის ზრდადობით დალაგებულ ყველაზე გრძელ მიმდევრობას.
4.1.51. M_{10} მასივში იპოვის კლებადობით დალაგებულ ყველაზე მოკლე მიმდევრობას.
4.1.52. M_{10} მასივის ელემენტებს დაალაგებს ისე, რომ დასაწყისში განლაგდეს ყველა უარყოფითი რიცხვი რიგითობის შენარჩუნებით, შემდეგ კი დადებითი რიცხვები რიგითობის შენარჩუნებით. დამხმარე მასივი არ გამოიყენოთ.
4.1.53. დათვლის M_{10} მასივის იმ ელემენტების რაოდენობას, რომლებიც მეტია b რიცხვზე.
4.1.54. იპოვის M_{10} მასივის იმ ელემენტების ჯამს, რომლებიც ნაკლებია b რიცხვზე.
4.1.55. დათვლის M_{10} მასივის მეზობელ ელემენტებს შორის ნიშანცვლათა რაოდენობას.
4.1.56. განსაზღვრავს თუ რამდენი ელემენტია მოთავსებული M_{10} მასივის ორ მოცემულ ელემენტს შორის.

4.1.57. იპოვის M_{10} მასივის იმ ელემენტების ინდექსებს, რომელთა შორის სხვაობა უდიდესია.
4.1.58. იპოვის M_{10} მასივის იმ ელემენტების ინდექსებს, რომელთა შორის სხვაობა უმცირესია.
4.1.59. იპოვის M_{10} მასივის დადებითი ელემენტებიდან უმცირესს.
4.1.60. იპოვის M_{10} მასივის უარყოფითი ელემენტებიდან უდიდესს.
4.1.61. იპოვის იმ ელემენტების ჯამს, რომლებიც მოთავსებულია M_{10} მასივის ორ მოცემულ ელემენტს შორის.
4.1.62. იპოვის იმ ელემენტებს შორის მაქსიმალურს, რომლებიც მოთავსებულია M_{10} მასივის ორ მოცემულ ელემენტს შორის.
4.1.63. დათვლის M_{10} მასივის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები 20-ზე მეტია.
4.1.64. M_{10} მასივიდან N_{10} მასივში გადაწერს იმ ელემენტებს, რომელთა მნიშვნელობები 20-ზე ნაკლებია.
4.1.65. M_{10} მასივში იპოვის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები აღემატება საკუთარ ინდექსს.
4.1.66. M_{10} მასივში იპოვის იმ 5-ელემენტიანი მიმდევრობის პირველ წევრს, რომლის ელემენტების ჯამი მინიმალურია.

ორგანზომილებიანი მასივები

შეადგინეთ პროგრამა, რომელიც:
4.2.1. იპოვის $A_{5,5}$ მასივის მაქსიმალურ ელემენტს.
4.2.2. იპოვის $A_{5,5}$ მასივის მინიმალური ელემენტის ინდექსებს.
4.2.3. იპოვის $A_{5,5}$ მასივის ელემენტების ჯამს.
4.2.4. იპოვის $A_{5,5}$ მასივის ელემენტების ნამრავლს.
4.2.5. იპოვის $A_{5,5}$ მასივის დადებითი ელემენტების ჯამს.
4.2.6. იპოვის $A_{5,5}$ მასივის უარყოფითი ელემენტების რაოდენობას.
4.2.7. იპოვის $A_{5,5}$ მასივის კენტი ელემენტების ჯამს.
4.2.8. იპოვის $A_{5,5}$ მასივის ლუწი ელემენტების რაოდენობას.
4.2.9. იპოვის $A_{5,5}$ მასივის არანულოვანი ელემენტების ნამრავლს.
4.2.10. იპოვის $A_{5,5}$ მასივის ნულოვანი ელემენტების რაოდენობას.
4.2.11. იპოვის $A_{5,5}$ მასივის მთავარი დიაგონალის ელემენტების ჯამს.
4.2.12. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის ელემენტების ნამრავლს.
4.2.13. იპოვის $A_{5,5}$ მასივის მთავარი დიაგონალის მინიმალურ ელემენტს.
4.2.14. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის მაქსიმალური ელემენტის ინდექსებს.
4.2.15. იპოვის $A_{5,5}$ მასივის მთავარი დიაგონალის 6-ის ჯერადი ელემენტების რაოდენობას.
4.2.16. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის დადებითი ელემენტების ჯამს.
4.2.17. $A_{5,5}$ მასივის ნულოვან ელემენტებს შეცვლის 1-ებით.
4.2.18. იპოვის $A_{5,5}$ მასივის 4-ის ჯერადი ელემენტების რაოდენობას.
4.2.19. $A_{5,5}$ მასივის ლუწ ელემენტებს გადაწერს B_{25} მასივში.
4.2.20. $A_{5,5}$ მასივის არანულოვან ელემენტებს გადაწერს B_{25} მასივში.
4.2.21. $A_{5,5}$ მასივის თითოეული სვეტის ელემენტების ჯამს ჩაწერს B_5 მასივში.

4.2.22. $A_{5,5}$ მასივის თითოეული სტრიქონის ელემენტების ნამრავლს ჩაწერს B_5 მასივში.
4.2.23. დათვლის $A_{5,5}$ მასივის თითოეული სვეტის დადებითი ელემენტების რაოდენობას და ჩაწერს მათ B_5 მასივში.
4.2.24. იპოვის $A_{5,5}$ მასივის თითოეული სტრიქონის კენტი ელემენტების ჯამს და ჩაწერს მათ B_5 მასივში.
4.2.25. იპოვის $A_{5,5}$ მასივის თითოეული სვეტის მაქსიმალურ ელემენტს და ჩაწერს მათ B_5 მასივში.
4.2.26. იპოვის $A_{5,5}$ მასივის თითოეული სტრიქონის მინიმალურ ელემენტს და ჩაწერს მათ B_5 მასივში.
4.2.27. იპოვის $A_{5,5}$ მასივის იმ სტრიქონის ინდექსს, რომელიც შეიცავს 8-ის ჯერადი ელემენტების მაქსიმალურ რაოდენობას.
4.2.28. იპოვის $A_{5,5}$ მასივის იმ სვეტის ინდექსს, რომელიც შეიცავს დადებითი ელემენტების მინიმალურ რაოდენობას.
4.2.29. იპოვის $A_{5,5}$ მასივის მთავარი დიაგონალის მაქსიმალურ ელემენტს და ამ ელემენტის შემცველ სტრიქონს გადაწერს B_5 მასივში.
4.2.30. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის მინიმალურ ელემენტს და ამ ელემენტის შემცველ სვეტს გადაწერს B_5 მასივში.
4.2.31. იპოვის $A_{5,5}$ მასივის იმ ელემენტების ჯამს, რომლებიც მთავარი დიაგონალის ზემოთაა მოთავსებული.
4.2.32. იპოვის $A_{5,5}$ მასივის იმ ელემენტების ნამრავლს, რომლებიც მთავარი დიაგონალის ქვემოთაა მოთავსებული.
4.2.33. იპოვის $A_{5,5}$ მასივის ორი ტოლი ელემენტის ინდექსებს.
4.2.34. იპოვის $A_{5,5}$ მასივის იმ ელემენტს, რომელიც ყველა სტრიქონში შედის.
4.2.35. იპოვის $A_{5,5}$ მასივის იმ ელემენტს, რომელიც ყველა სვეტში შედის.
4.2.36. იპოვის $A_{5,5}$ მასივის იმ სტრიქონს, რომელიც შეიცავს ლუწი ელემენტების მაქსიმალურ რაოდენობას და ამ სტრიქონს გადაწერს B5,5 მასივში.
4.2.37. იპოვის $A_{5,5}$ მასივის იმ სვეტს, რომელიც შეიცავს კენტი ელემენტების მინიმალურ რაოდენობას და ამ სვეტს გადაწერს B_5 მასივში.
4.2.38. იპოვის $A_{5,5}$ მასივის მაქსიმალური ელემენტის შემცველ სტრიქონს და გადაწერს მას B_5 მასივში.
4.2.39. იპოვის $A_{5,5}$ მასივის მინიმალური ელემენტის შემცველ სვეტს და გადაწერს მას B_5 მასივში.
4.2.40. $A_{5,5}$ მასივში ადგილებს შეუცვლის მითითებული ნომრის მქონე სვეტსა და უკანასკნელ სვეტს.
4.2.41. $A_{5,5}$ მასივში ადგილებს შეუცვლის მითითებული ნომრის მქონე სტრიქონსა და პირველ სტრიქონს.
4.2.42. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის ზემოთ მოთავსებული ელემენტების ჯამს.
4.2.43. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის ქვემოთ მოთავსებული ელემენტების ჯამს.
4.2.44. იპოვის $A_{5,5}$ მასივის იმ სტრიქონის ნომერს, რომლის ელემენტების ჯამი აღემატება დანარჩენი სტრიქონების ელემენტების ჯამს.
4.2.45. იპოვის $A_{5,5}$ მასივის იმ სვეტის ნომერს, რომლის ელემენტების ჯამი აღემატება დანარჩენი სვეტების ელემენტების ჯამს.
4.2.46. განსაზღვრავს არის თუ არა $A_{5,5}$ მასივში ერთნაირი სვეტები.

4.2.47. განსაზღვრავს არის თუ არა $A_{5,5}$ მასივში ერთნაირი სტრიქონები.
4.2.48. $A_{5,5}$ მასივიდან წაშლის იმ სვეტსა და სტრიქონს, რომელთა გადაკვეთაზე მდებარეობს მთავარი დიაგონალის მაქსიმალური ელემენტი.
4.2.49. $A_{5,5}$ მასივის რომელიმე სვეტსა და სტრიქონს შეავსებს ნულებით მათ გადაკვეთაზე მდებარე ელემენტის გარდა.
4.2.50. $A_{5,5}$ მასივის მთავარი დიაგონალის ელემენტებს B_5 მასივში გადაწერს.
4.2.51. $A_{5,5}$ მასივის არამთავარი დიაგონალის ელემენტებს B_5 მასივში გადაწერს.
4.2.52. იანგარიშებს $A_{5,5}$ მასივის ლუწი ნომრების მქონე სვეტების ელემენტების საშუალო არითმეტიკულს და ჩაწერს მათ B_3 მასივში.
4.2.53. იანგარიშებს $A_{5,5}$ მასივის კენტი ნომრების მქონე სტრიქონების ელემენტების საშუალო გეომეტრიულს და ჩაწერს მათ B_2 მასივში.
4.2.54. იპოვის $A_{5,5}$ მასივის იმ სტრიქონების ინდექსებს, რომელთა ყველა ელემენტი ნულის ტოლია. ინდექსები ჩაწერეთ B_5 მასივში.
4.2.55. იპოვის $A_{5,5}$ მასივის იმ სვეტების ინდექსებს, რომელთა ყველა ელემენტი კენტია. ინდექსები ჩაწერეთ B_5 მასივში.
4.2.56. იპოვის $A_{5,5}$ მასივის იმ სტრიქონების ინდექსებს, რომელთა ელემენტები ქმნიან მონოტონურად ზრდად მიმდევრობას. ინდექსები ჩაწერეთ B_5 მასივში.
4.2.57. იპოვის $A_{5,5}$ მასივის იმ სვეტების ინდექსებს, რომელთა ელემენტები ქმნიან მონოტონურად კლებად მიმდევრობას. ინდექსები ჩაწერეთ B_5 მასივში.
4.2.58. შეასრულებს $A_{5,5}$ მასივის სტრიქონების ძვრას ზევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ მეორე სტრიქონი დაიკავებს პირველის ადგილს, მესამე სტრიქონი დაიკავებს მეორის ადგილს და ა.შ. უკანასკნელი სტრიქონი შეივსება ნულებით.
4.2.59. შეასრულებს $A_{5,5}$ მასივის სტრიქონების ძვრას ზევით ორი სტრიქონით.
4.2.60. შეასრულებს $A_{5,5}$ მასივის სტრიქონების ძვრას ქვევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ პირველი სტრიქონი დაიკავებს მეორის ადგილს, მეორე სტრიქონი დაიკავებს მესამის ადგილს და ა.შ. პირველი სტრიქონი შეივსება ნულებით.
4.2.61. შეასრულებს $A_{5,5}$ მასივის სტრიქონების ძვრას ქვევით ორი სტრიქონით.
4.2.62. შეასრულებს $A_{5,5}$ მასივის სტრიქონების ციკლისებურ ძვრას ზევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ მეორე სტრიქონი დაიკავებს პირველის ადგილს, მესამე სტრიქონი დაიკავებს მეორის ადგილს და ა.შ. პირველი სტრიქონი დაიკავებს უკანასკნელის ადგილს.
4.2.63. შეასრულებს $A_{5,5}$ მასივის სტრიქონების ციკლისებურ ძვრას ზევით ორი სტრიქონით.
4.2.64. შეასრულებს $A_{5,5}$ მასივის სტრიქონების ციკლისებურ ძვრას ქვევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ პირველი სტრიქონი დაიკავებს მეორის ადგილს, მეორე სტრიქონი დაიკავებს მესამის ადგილს და ა.შ. უკანასკნელი სტრიქონი დაიკავებს პირველის ადგილს.
4.2.65. შეასრულებს $A_{5,5}$ მასივის სტრიქონების ციკლისებურ ძვრას ქვევით ორი სტრიქონით.
4.2.66. შეასრულებს $A_{5,5}$ მასივის სვეტების ძვრას მარჯვნივ ერთი სვეტით.
4.2.67. შეასრულებს $A_{5,5}$ მასივის სვეტების ძვრას მარჯვნივ ორი სვეტით.
4.2.68. შეასრულებს $A_{5,5}$ მასივის სვეტების ძვრას მარცხნივ ერთი სვეტით.
4.2.69. შეასრულებს $A_{5,5}$ მასივის სვეტების ძვრას მარცხნივ ორი სვეტით.
4.2.70. შეასრულებს $A_{5,5}$ მასივის სვეტების ციკლისებურ ძვრას მარცხნივ ერთი სვეტით.
4.2.71. შეასრულებს $A_{5,5}$ მასივის სვეტების ციკლისებურ ძვრას მარცხნივ ორი სვეტით.

4.2.72. შეასრულებს $A_{5,5}$ მასივის სვეტების ციკლისებურ ძვრას მარჯვნივ ერთი სვეტით.
4.2.73. შეასრულებს $A_{5,5}$ მასივის სვეტების ციკლისებურ ძვრას მარჯვნივ ორი სვეტით.
4.2.74. $A_{5,5}$ მასივიდან წაშლის იმ სვეტსა და სტრიქონს, რომელთა გადაკვეთაზე მდებარეობს მატრიცის მინიმალური ელემენტი.
4.2.75. $A_{5,5}$ მასივიდან წაშლის ნულების შემცველ სტრიქონს.
4.2.76. $A_{5,5}$ მასივიდან წაშლის ნულების შემცველ სვეტს.
4.2.77. იპოვის $A_{5,5}$ მასივის უნაგირა წერტილს. უნაგირა წერტილი არის მასივის ის ელემენტი, რომელიც უმცირესია თავის სტრიქონში და უდიდესია თავის სვეტში.
4.2.78. იპოვის $A_{5,5}$ მასივის იმ ელემენტების ჯამს, რომლებიც არამთავარი დიაგონალის ქვემოთაა მოთავსებული.
4.2.79. იპოვის $A_{5,5}$ მასივის იმ ელემენტების ჯამს, რომლებიც არამთავარი დიაგონალის ზემოთაა მოთავსებული.
4.2.80. დათვლის $A_{5,5}$ მასივის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები 20-ზე მეტია.
4.2.81. $A_{5,5}$ მასივიდან M_{25} მასივში გადაწერს იმ ელემენტებს, რომელთა მნიშვნელობები 20-ზე ნაკლებია.

თავი 5. კლასები, ინკაფსულაცია, მეთოდები

5.1.1. შექმენით ტელევიზორის კლასი, რომელიც შეიცავს:
- პრივატულ ცვლადებს:
  - 1 საათში დახარჯულ ელექტროენერგიას ვატებში და მუშაობის დროს საათებში;
- საერთოწვდომის მეთოდებს:
  - პირველი მეთოდი პრივატულ ცვლადებს მნიშვნელობას ანიჭებს;
  - მეორე მეთოდს ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები;
  - მესამე მეთოდი გამოთვლის და აბრუნებს ტელევიზორის მიერ მითითებული დროის განმავლობაში დახარჯული ელექტროენერგიის რაოდენობას.

5.1.2. შექმენით მატარებლის კლასი, რომელიც შეიცავს:
- პრივატულ ცვლადებს:
  - 1 კილომეტრის გავლისას დახარჯულ ელექტროენერგიას (კილოვატებში) და გავლილ მანძილს;
- საერთოწვდომის მეთოდებს:
  - პირველი მეთოდია კონსტრუქტორი, რომელიც პრივატულ ცვლადებს მნიშვნელობებეს ანიჭებს;
  - მეორე მეთოდს ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები;
  - მესამე მეთოდი გამოთვლის და აბრუნებს მატარებლის მიერ მითითებული მანძილის გავლისას დახარჯულ ელექტროენერგიას.

5.1.3. შექმენით ავტობუსის კლასი, რომელიც შეიცავს:
- პრივატულ ცვლადებს:
  - ბაკის ტევადობა (ლიტრებში) და მანძილი, რომელსაც გაივლის ავტობუსი

1 ლიტრი საწვავით;

- პრივატულ მეთოდს, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს.
- საერთოწვდომის მეთოდებს:
    - პირველი მეთოდი იძახებს პრივატულ მეთოდს, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს;
    - მეორე მეთოდს ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები;
    - მესამე მეთოდი გამოთვლის და აბრუნებს ავტობუსის მიერ სავსე ბაკით გავლილ მანძილს.

5.1.4. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს:

- პრივატულ ცვლადებს:
    - ბაკის ტევადობას (ლიტრებში) და მანძილს, რომელსაც გაიფრენს თვითმფრინავი 1 ლიტრი საწვავით;
- საერთოწვდომის მეთოდებს:
    - პირველი მეთოდია კონსტრუქტორი, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს;
    - მეორე მეთოდს ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები;
    - მესამე მეთოდი გამოთვლის და აბრუნებს თვითმფრინავის მიერ სავსე ბაკით გავლილ მანძილს.

5.1.5. შექმენით სამკუთხედის კლასი, რომელიც შეიცავს:

- პრივატულ ცვლადებს:
    - სამკუთხედის სამივე გვერდის ზომებს;
- პრივატულ მეთოდს, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს.
- საერთოწვდომის მეთოდებს:
    - პირველი მეთოდი იძახებს პრივატულ მეთოდს, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს;
    - მეორე მეთოდს ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები;
    - მესამე მეთოდი გამოთვლის და აბრუნებს სამკუთხედის პერიმეტრს.

5.1.6. შექმენით მართკუთხედის კლასი, რომელიც შეიცავს:

- პრივატულ ცვლადებს:
    - მართკუთხედის ოთხივე გვერდის ზომებს;
- საერთოწვდომის მეთოდებს:
    - პირველი მეთოდი პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს;
    - მეორე მეთოდს ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები;
    - მესამე მეთოდი გამოთვლის და აბრუნებს მართკუთხედის ფართობს.

5.1.7. შეადგინეთ კლასი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი იპოვის და დააბრუნებს A_{10} მასივის მინიმალურ ელემენტს. მეთოდს გადაეცემა მასივი.

5.1.8. შეადგინეთ კლასი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი გამოთვლის და აბრუნებს $y = 4x^3 + x^2 - 3x + 2$ გამოსახულების მნიშვნელობას. მეთოდს გადაეცემა x მნიშვნელობა.

5.1.9. შეადგინეთ კლასი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი იპოვის და დააბრუნებს A_{10} მასივის ელემენტების ჯამს. მეთოდს გადაეცემა მასივი.

5.1.10. შეადგინეთ კლასი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი სტრიქონში დათვლის და დააბრუნებს 'ს' სიმბოლოს რაოდენობას. მეთოდს გადაეცემა სტრიქონი.
5.1.11. შეადგინეთ კლასი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი დათვლის და დააბრუნებს სტრიქონში ციფრების რაოდენობას 0-დან 9-მდე. მეთოდს გადაეცემა სტრიქონი.
5.1.12. შეადგინეთ კლასი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი A_{10} მასივის უარყოფით ელემენტებს გადაწერს B_{10} მასივში. მეთოდს გადაეცემა ორივე მასივი.
5.1.13. შეადგინეთ კლასი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი სტრიქონში იპოვის და დააბრუნებს 'დ' სიმბოლოს პოზიციას. მეთოდს გადაეცემა სტრიქონი.

შემთხვევითი რიცხვები

შეადგინეთ პროგრამა, რომელიც:
5.1.14. მოახდენს [1,100] ინტერვალში მოთავსებული 20 შემთხვევითი რიცხვის გენერირებას.
5.1.15. დათვლის თუ რამდენჯერ გვხვდება q რიცხვი [1,1000] ინტერვალში.
5.1.16. ჯერ მოახდენს 100 მთელი რიცხვის გენერირებას [1,100] ინტერვალში, შემდეგ კი დათვლის თუ რამდენჯერ გამოჩნდა თითოეული რიცხვი ამ ინტერვალში.
5.1.17. განსაზღვრავს თუ რამდენი მთელი რიცხვის გენერირება უნდა მოვახდინოთ [1,30] ინტერვალიდან, რომ მოცემული R რიცხვი მათ შორის შეგვხვდეს 7-ჯერ.
5.1.18. განსაზღვრავს თუ რამდენი ასანთი უნდა ამოვიღოთ ასანთის 5 ყუთიდან იმისათვის, რომ დაცარიელდეს ერთ-ერთი. თითო ყუთში 20 ასანთია.

თავი 6. პოლიმორფიზმი

6.1.1. შეადგინეთ კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 3 მეთოდი. პირველი მეთოდი უპარამეტრებოა და ეკრანზე გამოაქვს შეტყობინება: "უპარამეტრებო მეთოდი". მეორე მეთოდს 1 მთელი ტიპის პარამეტრი აქვს და აბრუნებს მის გაორმაგებულ მნიშვნელობას. მესამე მეთოდს 1 წილადი პარამეტრი აქვს და აბრუნებს მის კუბს.
6.1.2. შეადგინეთ კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 2 მეთოდი. პირველი მეთოდს 2 წილადი პარამეტრი აქვს და აბრუნებს მათ ჯამს. მეორე მეთოდს 3 წილადი პარამეტრი აქვს და აბრუნებს მათ ნამრავლს.
6.1.3. შეადგინეთ კვადრატის კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 2 მეთოდი. პირველ მეთოდს 1 მთელი ტიპის პარამეტრი აქვს - კვადრატის გვერდი და აბრუნებს კვადრატის ფართობს ($S = a^2$). მეორე მეთოდს 1 წილადი პარამეტრი აქვს - კვადრატის გვერდი და აბრუნებს კვადრატის პერიმეტრს ($P = 4a$).

თავი 8. მემკვიდრეობითობა

8.1.1. შექმენით პიროვნების საბაზო კლასი, რომელიც შეიცავს:
- პრივატულ ცვლადებს:

    - გვარს, სახელსა და ასაკს;
- საერთოწვდომის მეთოდებს:
    - პირველი მეთოდი არის კონსტრუქტორი, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს;
    - მეორე მეთოდს ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები;

შექმნენით სტუდენტის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს:

- პრივატულ ცვლადებს:
    - სტუდენტის დაბადების თარიღს;
    - კურსს;
    - ჯგუფის ნომერს;
    - ინსტიტუტის დასახელებას;
- საერთოწვდომის მეთოდს:

მეთოდს გადაეცემა სტუდენტის ნიშნები 15-ელემენტიანი მთელრიცხვა მასივის სახით. მეთოდი გამოთვლის და აბრუნებს ნიშნების საშუალო არითმეტიკულს.

8.1.2. შექმენით პიროვნების საბაზო კლასი, რომელიც შეიცავს:

- პრივატულ ცვლადებს:
    - გვარს, სახელსა და ასაკს;
- საერთოწვდომის მეთოდებს:
    - პირველი მეთოდი არის კონსტრუქტორი, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს;
    - მეორე მეთოდს ფანჯარაში გამოაქვს პრივატული ცვლადების მნიშვნელობები;

შექმენით ექიმის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს:

- პრივატულ ცვლადებს:
    - ექიმის დაბადების თარიღს;
    - განყოფილების დასახელებას;
    - ექიმის თანამდებობას;
    - საავადმყოფოს დასახელებას;
    - ერთი ავადმყოფის მიერ გადახდილ თანხას;
    - ავადმყოფების რაოდენობას;
- საერთოწვდომის მეთოდებს:

რომელიც გამოთვლის და აბრუნებს ექიმისათვის ავადმყოფების მიერ გადახდილ თანხას.

თავი 9. ინფორმაციის შეტანა-გამოტანა

შეადგინეთ პროგრამა, რომელიც:

9.1.1. ფაილში ჯერ ჩაწერს ჩასაწერი რიცხვების რაოდენობას, შემდეგ კი - თვით რიცხვებს.

9.1.2. ფაილიდან წაიკითხავს მთელ რიცხვებს და დათვლის მათ შორის უარყოფითი, ნულოვანი და დადებითი რიცხვების რაოდენობას. ფაილში ჯერ ჩაწერილია რიცხვების რაოდენობა, შემდეგ კი თვით რიცხვები.

9.1.3. ფაილიდან წაიკითხავს წილად რიცხვებს, დათვლის მათ რაოდენობას და

გამოთვლის მათ ჯამს.
9.1.4. ფაილში ჩაწერილი წილადი რიცხვების შუაში მოთავსებულ რიცხვს. ფაილში ჩაწერილია კენტი რაოდენობის რიცხვი.
9.1.5. ფაილიდან წაიკითხავს მთელ რიცხვებს, დაალაგებს მათ ზრდადობის მიხედვით და ჩაწერს ამავე ფაილში.
9.1.6. ფაილიდან წაიკითხავს რიცხვებს და მათ შორის პირველსა და უკანასკნელს მეორე ფაილში გადაწერს.
9.1.7. ფაილიდან წაიკითხავს რიცხვებს და მეორე ფაილში გადაწერს:
ა. მათ შორის მინიმალურსა და მაქსიმალურს;
ბ. კენტი ნომრის მქონე რიცხვებს შორის უდიდესს;
გ. ლუწი ნომრის მქონე რიცხვებს შორის უმცირესს.
დ. ლუწი და კენტი რიცხვების რაოდენობას;
9.1.8. სიმბოლური ფაილიდან ორ მონაცემს კითხულობს და განსაზღვრავს ისინი ციფრებია თუ არა. თუ ეს მონაცემები ციფრებია, მაშინ მათი ჯამი, ნამრავლი და სხვაობა ამავე ფაილს დაუმატეთ.
9.1.9. ერთ რიცხვით ფაილს მეორეში გადაწერს.
9.1.10. ფაილიდან წაიკითხავს რიცხვებს და მეორე ფაილში გადაწერს:
ა. კენტ რიცხვებს;
ბ. ლუწ რიცხვებს;
გ. 7-ის ჯერად რიცხვებს;
დ. რიცხვებს, რომლებიც წარმოადგენენ კვადრატებს.
9.1.11. განსაზღვრავს ორი ფაილი შეიცავს თუ არა ერთნაირ რიცხვებს.
9.1.12. ერთი სიმბოლური ფაილიდან მეორეში მონაცემებს ისე გადაწერს, რომ მიმდევრობით მოთავსებული ინტერვალის ნიშნები შეიცვალოს ერთით.
9.1.13. ფაილის ყოველი სტრიქონის წინ ინტერვალის ნიშანს ჩასვამს და გადაწერს მეორე ფაილში.
9.1.14. ფაილის ყოველი სტრიქონის წინ და ბოლოში ინტერვალის ნიშნება წაშლის და მიღებულ სტრიქონებს ამავე ფაილში ჩაწერს.
9.1.15. სიმბოლურ ფაილში:
ა. დათვლის ‘ა’ სიმბოლოს რაოდენობას;
ბ. დათვლის ‘ს’, ‘რ’ და ‘ე’ სიმბოლოების რაოდენობას;
გ. დაადგენს შეიცავს თუ არა ფაილი “კომპიუტერი” სიმბოლოების მიმდევრობას;
დ. დათვლის “კომპიუტერი” სიტყვის რაოდენობას.
9.1.16. სიმბოლური ფაილიდან წაშლის ყველა სამსიმბოლოიან სიტყვას და შედეგს მეორე ფაილში გადაწერს.
9.1.17. სიმბოლურ ფაილში:
ა. იპოვის ყველაზე გრძელ სიტყვას;
ბ. იპოვის ყველაზე მოკლე სიტყვას;
გ. იპოვის ყველაზე გრძელ სიტყვას, რომლის მესამე სიმბოლოა ‘ფ’;
დ. დათვლის ორი, სამი და ოთხი სიმბოლოსაგან შემდგარი სიტყვების რაოდენობას.
9.1.18. ერთ სიმბოლურ ფაილს მეორეში გადაწერს.
9.1.20. ერთი სიმბოლური ფაილიდან წაიკითხავს ტექსტს, პატარა ასოებს დიდ ასოებად გარდაქმნის და შედეგს მეორე ფაილში ჩაწერს.
9.1.21. ერთი სიმბოლური ფაილიდან მონაცემებს უკუ მიმდევრობით გადაწერს მეორე ფაილში.

9.1.22. ტექსტურ ფალში მოძებნის 10 სიტყვას, რომლებიც ყველაზე ხშირად გვხვდება.
9.1.23. ტექსტურ ფალში დათვლის სტრიქონების რაოდენობას.
9.1.24. ტექსტური ფალის თითოეულ სტრიქონში დათვლის სიმბოლოების რაოდენობას.
9.1.25. ეკრანზე გამოიტანს ტექსტური ფაილის ყველაზე გრძელ და მოკლე სტრიქონებს.
9.1.26. ტექსტურ ფალში დათვლის იმ სტრიქონების რაოდენობას, რომლებშიც სიმბოლოების რაოდენობა 60-ს აღემატება.
9.1.27. ფაილში მოთავსებულ ტექსტს ეკრანზე გამოიტანს.
9.1.28. ფაილში ჩაწერს 10 სტუდენტის გვარს, დაბადების თარიღსა და სიმაღლეს.
9.1.29. ფაილის მეხუთე პოზიციიდან 3 ძახილის ნიშანს ჩაწერს.
9.1.30. ერთი ფაილის კენტი ნომრის მქონე სიმბოლოებს მეორე ფაილის ლუწი ნომრის პოზიციებში გადაწერს.
9.1.31. ერთი ფაილიდან სიმბოლოებს დაწყებული პირველი პოზიციიდან გადაწერს მეორე ფაილში დაწყებული უკანასკნელი პოზიციიდნ.
9.1.32. მოცემულია ორი ფაილი, რომლებიც შეიცავენ ზრდადობით დალაგებულ რიცხვებს. შეადგინეთ პროგრამა, რომელიც მესამე ფაილს შექმნის და მასში ზრდადობის მიხედვით ჩაწერს ორივე ფაილში მოთავსებულ რიცხვებს.
9.1.33. ფაილში ჩაწერილია სხვადასხვა თარიღები. თარიღს აქვს სახე: რიცხვი, თვე და წელი. შეადგინეთ პროგრამა, რომელიც იპოვის და ეკრანზე გამოიტანს:
ა. უდიდესი წლის მქონე თარიღს;
ბ. უმცირესი თვის მქონე თარიღს;
გ. ყველა საზაფხულო თარიღს;
9.1.34. მოცემულია ორი ტექსტური ფაილი. შეადგინეთ პროგრამა, რომელიც მესამე ფაილში ჯერ ჩაწერს პირველ ფაილში მოთავსებულ ტექსტს, შემდეგ კი - მეორე ფაილში მოთავსებულ ტექსტს.
9.1.35. ტექსტური ფაილი შეიცავს C# ენაზე შედგენილ საწყის კოდს (პროგრამას). კოდის თითოეული ოპერატორი თითო სტრიქონს იკავებს. შეადგინეთ პროგრამა, რომელიც:
ა. შეამოწმებს გამხსნელი და დამხურავი მრგვალი ფრჩხილების გამოყენების სისწორეს;
ბ. შეამოწმებს გამხსნელი და დამხურავი ფიგურული ფრჩხილების გამოყენების სისწორეს;
9.1.36. შეადგინეთ პირველი პროგრამა, რომელიც ტექსტურ ფაილში ჩაწერს ერთი ჯგუფის ბაკალავრების გვარებს, სახელებს, დაბადების თარიღსა და ერთ საგანში შეფასების შედეგებს. ამასთან, ფაილის თითო სტრიქონი უნდა შეიცავდეს ინფორმაციას თითო ბაკალავრის შესახებ. შეადგინეთ მეორე პროგრამა, რომელიც:
ა. მეორე ტექსტურ ფაილში გადაწერს იმ ბაკალავრების გვარებს, რომლებიც ‘თ’ სიმბოლოთი იწყება;
ბ. მეორე ტექსტურ ფაილში გადაწერს იმ ბაკალავრების გვარებს, რომელთა ასაკი 20 წელს აღემატება;
გ. დაადგენს არიან თუ არა ჯგუფში თანამოგვარეები და მათ გვარებს მეორე ფაილში გადაწერს;
დ. მეორე ფაილში გადაწერს იმ სტუდენტების გვარებს, რომლებმაც მიიღეს შეფასება 5;
ე. მეორე ფაილში გადაწერს იმ სტუდენტების გვარებს, რომლებმაც მიიღეს შეფასება 3 და რომელთა გვარები ‘ლ’ სიმბოლოთი იწყება;
ვ. დათვლის თუ რამდენმა სტუდენტმა მიიღო შეფასება 2, 3, 4 და 5:
ზ. გამოთვლის საშუალო შეფასებას:
თ. მეორე ფაილში გადაწერს იმ სტუდენტების გვარებს, რომლების სახელებია “გიორგი”

და “ნინო”;
ი. მეორე ფაილში გადაწერს იმ სტუდენტების გვარებს, რომლებიც მარტში და ოქტომბერში დაიბადნენ.
9.1.37. შეადგინეთ პირველი პროგრამა, რომელიც ტექსტურ ფაილში ჩაწერს ერთი ჯგუფის ბაკალავრების სახელებს, გვარებს და ხუთ საგანში გამოცდის შედეგებს. ამასთან, ფაილის თითო სტრიქონი უნდა შეიცავდეს ინფორმაციას თითო ბაკალავრის შესახებ. შეადგინეთ მეორე პროგრამა, რომელიც:
ა. თითოეული ბაკალავრისათვის გამოთვლის საშუალო ქულას და მას ჩაწერს ამავე ფაილში შესაბამისი ბაკალავრისათვის გამოყოფილი სტრიქონის ბოლოში;
ბ. ბაკალავრების გვარებს დაალაგებს საშუალო ქულის კლების მიხედვით და მათ მეორე ფაილში ჩაწერს.
9.1.38. შეადგინეთ პირველი პროგრამა, რომელიც ტექსტურ ფაილში პროფესორების გვარს, სახელს, დაბადების წელს, სქესს და ხელფასს ჩაწერს. შეადგინეთ მეორე პროგრამა, რომელიც:
ა. ‘X’ სიმბოლოთი მონიშნავს იმ პროფესორების გვარებს, რომელთა ხელფასი 150 ლარს აღემატება;
ბ. მეორე ფაილში იმ პროფესორების გვარებს გადაწერს, რომელთა წლიური შემოსავალი აღემატება 1500 ლარს;
გ. დათვლის კაცებისა და ქალების რაოდენობას;
დ. დათვლის იმ კაცების რაოდენობას, რომელთა ასაკი 65 წელს აღემატება;
ე. დათვლის იმ ქალების რაოდენობას, რომელთა ასაკი 65 წელს არ აღემატება;
ვ. დათვლის იმ კაცების რაოდენობას, რომლებიც 1950 წელზე ადრე დაიბადნენ.
9.1.39. შეადგინეთ პირველი პროგრამა, რომელიც ტექსტურ ფაილში რეგბის გუნდის წევრების გვარს, სახელს, სიმაღლეს (სანტიმეტრებში) და წონას ჩაწერს. შეადგინეთ მეორე პროგრამა, რომელიც:
ა. გამოთვლის და ამავე ფაილს დაუმატებს რეგბისტების საშუალო სიმაღლესა და წონას;
ბ. მეორე ფაილში გადაწერს იმ რეგბისტების გვარებს, რომელთა სიმაღლე აღემატება 195 სანტიმეტრს;
9.1.40. ტექსტური ფაილის თითოეულ სტრიქონში დათვლის სიტყვების რაოდენობას.

თავი 12. სტრიქონები

სტრიქონები

ქვემოთ მოყვანილ ამოცანებში სტრიქონების შესატანად textBox კომპონენტი გამოიყენეთ. შეადგინეთ პროგრამა, რომელიც:
12.3.1. სტრიქონში დათვლის სიმბოლოების რაოდენობას. სტრიქონი წერტილით მთავრდება.
12.3.2. სტრიქონში დათვლის “ბ” სიმბოლოს რაოდენობას.
12.3.3. სტრიქონში დათვლის ციფრების რაოდენობას 0-დან 9-მდე.
12.3.4. სტრიქონში დათვლის სასვენი ნიშნების რაოდენობას (, ; ! ? :).
12.3.5. სტრიქონში დათვლის ხმოვნების რაოდენობას (a, o, u, i, y, e).
12.3.6. სტრიქონში იპოვის “ა” სიმბოლოს პოზიციას.
12.3.7. სტრიქონში ყველა “ი” სიმბოლოს “ს” სიმბოლოთი შეცვლის.

12.3.8. სტრიქონში ყველა ციფრს “რ” სიმბოლოთი შეცვლის.
12.3.9. სტრიქონში “ეს არის ტესტი” ჩაუმატებს “კომპიუტერი” სიტყვას დაწყებული მე-3 პოზიციიდან.
12.3.10. სტრიქონიდან “ეს არის ტესტი” წაშლის “არის” სიტყვას.
12.3.11. სტრიქონიდან “ეს არის ტესტი” წაშლის 6 სიმბოლოს დაწყებული მე-2 პოზიციიდან.
12.3.12. სტრიქონში “ეს არის ტესტი” “ეს” სიტყვას შეცვლის “კომპიუტერი” სიტყვით.
12.3.13. სტრიქონში “ეს არის ტესტური პროგრამა” “ეს არის” სიტყვებს შეცვლის “ახლა შევასრულოთ” სიტყვებით.
12.3.14. სიმბოლოების $M_{5,5}$ მასივში მოძებნის იმ სტრიქონს, რომელიც შეიცავს “ა” სიმბოლოს მაქსიმალურ რაოდენობას.
12.3.15. სიმბოლოების $M_{5,5}$ მასივში მოძებნის იმ სტრიქონს, რომელიც შეიცავს განსხვავებული სიმბოლოების მაქსიმალურ რაოდენობას.
12.3.16. სიმბოლოების $M_{5,5}$ მასივში მოძებნის იმ სვეტს, რომელიც შეიცავს “ა” სიმბოლოს მინიმალურ რაოდენობას.
12.3.17. სიმბოლოების $M_{5,5}$ მასივში მოძებნის იმ სვეტს, რომელიც შეიცავს განსხვავებული სიმბოლოების მინიმალურ რაოდენობას.
12.3.18. სტრიქონის მე-10 პოზიციიდან 30-ე პოზიციამდე დათვლის “+” სიმბოლოს რაოდენობას.
12.3.19. სტრიქონში დათვლის “რო” სიმბოლოების რაოდენობას.
12.3.20. სტრიქონში დათვლის ინტერვალების რაოდენობას.
12.3.21. განსაზღვრავს შეიცავს თუ არა სტრიქონი “დ” სიმბოლოს.
12.3.22. განსაზღვრავს შეიცავს თუ არა სტრიქონი ერთნაირი სიმბოლოების წყვილებს.
12.3.23. განსაზღვრავს შეიცავს თუ არა სტრიქონი 4 მეზობელ “კ” სიმბოლოს.
12.3.24. სტრიქონში განსაზღვრავს მეზობელი ინტერვალების ყველაზე გრძელი მიმდევრობის ზომას.
12.3.25. სტრიქონიდან წაშლის “ათი” სიტყვებს.
12.3.26. სტრიქონში “ს” სიმბოლოს შემდეგ მოთავსებულ ყველა სიმბოლოს “თ” სიმბოლოთი შეცვლის.
12.3.27. label კომპონენტში ინგლისური ანბანის ყველა ასოს გამოიტანს.
12.3.28. label კომპონენტში კლავიატურის ყველა სიმბოლოს გამოიტანს.
12.3.29. სტრიქონში დათვლის სიტყვების რაოდენობას.
12.3.30. სტრიქონის უკანასკნელ სიტყვაში “ლ” სიმბოლოს რაოდენობას დათვლის.
12.3.31. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომლებიც “ო” სიმბოლოთი იწყებიან.
12.3.32. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომლებიც “ი” სიმბოლოთი მთავრდებიან.
12.3.33. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომელთა პირველი და უკანასკნელი ასოები ერთმანეთს ემთხვევა.
12.3.34. სტრიქონში ყველაზე გრძელ სიტყვას იპოვის.
12.3.35. სტრიქონში ყველაზე მოკლე სიტყვას იპოვის.
12.3.36. სტრიქონიდან წაშლის ციფრებს.
12.3.37. სტრიქონში პატარა ასოებს დიდი ასოებით შეცვლის.
12.3.38. სტრიქონში დიდ ასოებს პატარა ასოებით შეცვლის.
12.3.39. სტრიქონში წაშლის წერტილის წინ მოთავსებულ რიცხვებს.

12.3.40. სტრიქონში წაშლის წერტილის შემდეგ მოთავსებულ რიცხვებს.
12.3.41. სტრიქონის სიმბოლოებს უკუ მიმდევრობით დაალაგებს.
12.3.42. ტექსტში მრგვალი ფრჩხილების ბალანსს შეამოწმებს. მრგვალი ფრჩხილები დაბალანსებულია თუ გამხსნელ ფრჩხილს მოსდევს შესაბამისი დამხურავი ფრჩხილი და მათი რაოდენობა ტოლია.
12.3.43. შეამოწმებს არის თუ არა ერთი წინადადება მეორის ნაწილი და პირიქით.
12.3.44. შეამოწმებს ერთნაირია თუ არა ორი წინადადება.
12.3.45. ორ წინადადებაში მოძებნის იმ სიტყვებს, რომლებიც ორივე წინადადებაში შედიან და ამ სიტყვებს ჩაწერს სტრიქონების მასივში.
12.3.46. მოცემულ რიცხვს (1-დან 10-მდე) ჩაწერს სიტყვების საშუალებით.
12.3.47. სტრიქონში მოძებნის იმ სიტყვებს, რომლებიც “ძე” სიმბოლოებით მთავრდება და ამ სიტყვებს ჩაწერს სტრიქონების მასივში.
12.3.48. განსაზღვრავს ტექსტში რამდენი სიტყვა შეიცავს 1 ხმოვანს, 2 ხმოვანს, 3 ხმოვანს და ა.შ.
12.3.49. განსაზღვრავს ტექსტში სიტყვების რამდენი პროცენტი იწყება “დ” სიმბოლოთი.
12.3.50. განსაზღვრავს შეიცავს თუ არა ტექსტი ასოებისაგან განსხვავებულ სიმბოლოებს.
12.3.51. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს ხმოვნების მაქსიმალურ რაოდენობას.
12.3.52. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს თანხმოვნების მინიმალურ რაოდენობას.
12.3.53. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს “გ” სიმბოლოს მაქსიმალურ რაოდენობას.
12.3.54. ტექსტში იპოვის სიტყვას, რომელიც ყველაზე ხშირად გვხვდება.
12.3.55. ტექსტში იპოვის 10 სიტყვას, რომელიც ყველაზე ხშირად გვხვდება და დათვლის თითოეული მათგანის რაოდენობას.
12.3.56. ტექსტში განსაზღვრავს თუ რომელი ასოთი იწყება სიტყვების უმრავლესობა.
12.3.57. ტექსტში განსაზღვრავს თუ რამდენჯერ გვხვდება ანბანის თითოეული ასო.
12.3.58. ტექსტში განსაზღვრავს სიტყვების პირველი ასოების მინიმალურ რაოდენობას, რომელთა მიხედვით შეიძლება განვასხვავოთ სიტყვები.
12.3.59. ტექსტში იპოვის იმ სიტყვებს, რომლებიც შეიცავენ გაორმაგებულ ხმოვნებს.
12.3.60. ტექსტში იპოვის იმ სიტყვებს, რომლებიც შეიცავენ გაორმაგებულ თანხმოვნებს.
12.3.61. ტექსტიდან წაშლის ზედმეტ ინტერვალის ნიშნებს და სიტყვებს შორის დატოვებს ინტერვალის თითო ნიშანს.
12.3.62. მოცემულია ორი სიტყვა. ისინი გააერთიანეთ ისე, რომ წინ მოთავსთდეს მოკლე სიტყვა, შემდეგ კი - გრძელი.
12.3.63. მოცემულია ორი სიტყვა. მეორე სიტყვა მოათავსეთ პირველი სიტყვის შუაში.
12.3.64. დაადგინეთ რამდენი პალინდრომია მოცემულ ტექსტში. პალინდრომია, მაგალითად, “აბგგბა”, 123321 და ა.შ.

სხვადასხვა

კომბინატორიკა

1. შეადგინეთ პროგრამა, რომელიც დააფორმირებს ხუთკაციანი გუნდის ყველა შესაძლო ვარიანტს არსებული ცხრა კანდიდატურისაგან.
2. შეადგინეთ პროგრამა, რომელიც მოძებნის ისეთ ოთხნიშნა რიცხვებს, რომელთა

ციფრების ჯამი წინასწარ მოცემული N რიცხვის ტოლია.
4. შეადგინეთ პროგრამა, რომელიც დააფორმირებს 1,2,3,4 ციფრების ყველა შესაძლო გადაადგილებას.
5. შეადგინეთ პროგრამა, რომელიც 1,2,3,4,5,6,7 ციფრებისაგან დააფორმირებს ყველა შესაძლო ოთხთანრიგა რიცხვს.

დახარისხება

1. შეადგინეთ პროგრამა, რომელიც ერთგანზომილებიანი მასივის ელემენტებს ზრდადობის მიხედვით დაალაგებს.
2. შეადგინეთ პროგრამა, რომელიც ერთგანზომილებიანი მასივის ელემენტებს კლებადობის მიხედვით დაალაგებს.
3. შეადგინეთ პროგრამა, რომელიც ორგანზომილებიანი მასივის თითოეული სტრიქონის ელემენტებს კლებადობის მიხედვით დაალაგებს.
4. შეადგინეთ პროგრამა, რომელიც ორგანზომილებიანი მასივის თითოეული სვეტის ელემენტებს ზრდადობის მიხედვით დაალაგებს.
5. შეადგინეთ პროგრამა, რომელიც შეამოწმებს ერთგანზომილებიანი მასივი არის თუ არა ზრდადობის მიხედვით დალაგებული.
6. შეადგინეთ პროგრამა, რომელიც შეამოწმებს ერთგანზომილებიანი მასივი არის თუ არა კლებადობის მიხედვით დალაგებული.
7. მასივი დალაგებულია ზრდადობის მიხედვით. შეადგინეთ პროგრამა, რომელიც ამ მასივში ჩასვამს R რიცხვს მასივის ზრდადობის მიხედვით მოწესრიგების დაურღვევლად.
8. მასივი დალაგებულია კლებადობის მიხედვით. შეადგინეთ პროგრამა, რომელიც ამ მასივში ჩასვამს R რიცხვს მასივის კლებადობის მიხედვით მოწესრიგების დაურღვევლად.
9. მოცემულია ზრდადობის მიხედვით დალაგებული ორი მასივი. შეადგინეთ პროგრამა, რომელიც ამ ორი მასივისაგან შექმნის მესამე მასივს, რომლის ელემენტებიც, ასევე, ზრდადობის მიხედვით იქნება დალაგებული.
10. შეადგინეთ პროგრამა, რომელიც მაგისტრების გვარებს ანბანის მიხედვით დაალაგებს.

კალენდარი

შეადგინეთ პროგრამა, რომელიც:
1. კვირის დღის ნომრის მიხედვით კვირის შესაბამისი დღის დასახელებას ქართულ ენაზე გამოიტანს ეკრანზე.
2. რიცხვის, თვისა და წლის მიხედვით კვირის დღეს განსაზღვრავს.
3. ეკრანზე ნაკიან წლებს გამოიტანს. ნაკიანი წელი თუ მისი ნომერი უნაშთოდ იყოფა 4-ზე.
4. ეკრანზე გამოიტანს მოცემული წლის კალენდარს.
5. განსაზღვრავს თუ რამდენჯერ მოხვდა თქვენი დაბადების დღე კვირის თითოეულ დღეზე.
6. განსაზღვრავს თვის იმ რიცხვებს, როცა მოუწევს ყოველი თვის უკანასკნელი კვირა დღე.
7. შეადგინეთ პროგრამა, რომელსაც თარიღი შემდეგი სახით მიეწოდება - 19.03.05. პროგრამამ ეკრანზე უნდა გამოიტანოს 2005 წლის 19 მარტი.

8. შეადგინეთ პროგრამა, რომელსაც თარიღი მიეწოდება. პროგრამამ უნდა განსაზღვროს შესაბამისი დღის ნომერი წლის დასაწყისიდან.
9. შეადგინეთ პროგრამა, რომელსაც ორი თარიღი მიეწოდება. პროგრამამ უნდა გამოთვალოს:
ა. დღეების რაოდენობა, რომელიც ამ ორ თარიღს შორის გავიდა;
ბ. თვეების რაოდენობა, რომელიც ამ ორ თარიღს შორის გავიდა;
გ. წლების რაოდენობა, რომელიც ამ ორ თარიღს შორის გავიდა;
10. შეადგინეთ პროგრამა, რომელსაც თარიღი მიეწოდება. პროგრამამ უნდა:
ა. შეამოწმოს თარიღის კორექტულობა. მაგალითად, არაკორექტულია 2005 წლის 30 თებერვალი;
ბ. განსაზღვროს თუ რამდენი დღე დარჩა წლის ბოლომდე;
გ. განსაზღვროს თუ რამდენი დღე გავიდა წლის დასაწყისიდან.
11. მოცემულია თვე და რიცხვი. შეადინეთ პროგრამა, რომელიც განსაზღვრავს თუ რომელ დღეზე მოდის ეს თარიღი თუ წელი არანაკიანია, 1 იანვარი კი - ხუთშაბათი.

პედაგოგიკა

შეადგინეთ პროგრამა, რომელიც:
1. შეამოწმებს გამრავლების ტაბულის ცოდნას.
2. შეამოწმებს საისტორიო თარიღების ცოდნას. პროგრამამ ეკრანზე უნდა გამოიტანოს ისტორიული მოვლენის დასახელება, თქვენ კი უნდა შეიტანოთ შესაბამისი თარიღი.
3. შეამოწმებს:
ა. გამრავლების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;
ბ. შეკრების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;
გ. გამოკლების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;
დ. გაყოფის ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;
ე. ახარისხების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;
ვ. კვადრატული ფესვის ამოღების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში.

დანართი 2. C# ენის ტესტები

თავი 2. C# ენის საფუძვლები

საწყისი ცნებები

2.1.1. C# ენაზე შედგენილი პროგრამა იწყება და მთავრდება:
ა. კვადრატული ფრჩხილებით
ბ. ფიგურული ფრჩხილებით
გ. მრგვალი ფრჩხილებით

2.1.2. // სიმბოლოებით იწყება:
ა. მრავალსტრიქონიანი კომენტარი
ბ. კომენტარი ამ სიმბოლოებით არ იწყება
გ. ერთსტრიქონიანი კომენტარი

2.1.3. /* და */ სიმბოლოებს შორის მოთავსებულია:
ა. მრავალსტრიქონიანი კომენტარი
ბ. ერთსტრიქონიანი კომენტარი
გ. კომენტარი ამ სიმბოლოებით არ იწყება

2.1.4. პროგრამის შესრულების დროს ხდება:
ა. კომენტარების გამოტოვება
ბ. კომენტარების შესრულება
გ. კომენტარების გარდაქმნა მთელ ტიპად

2.1.5. ცვლადების გამოცხადებას ამთავრებს:
ა. წერტილ-მძიმე “;”
ბ. მძიმე “,”
გ. ორი წერტილი “:”

2.1.6. ოპერატორები ერთმანეთისაგან გამოიყოფა:
ა. მძიმით “,”
ბ. წერტილ-მძიმით “;”
გ. ორი წერტილით “:”

2.1.7. int სიტყვა იწყებს:
ა. წილადის გამოცხადებას
ბ. ლოგიკური ცვლადის გამოცხადებას
გ. მთელი რიცხვის გამოცხადებას

2.1.8. double სიტყვა იწყებს:
ა. მთელი რიცხვის გამოცხადებას
ბ. ორმაგი სიზუსტის წილადის გამოცხადებას

გ. ლოგიკური ცვლადის გამოცხადებას

2.1.9. bool სიტყვა იწყებს:
ა. ლოგიკური ცვლადის გამოცხადებას
ბ. მთელი რიცხვის გამოცხადებას
გ. წილადის გამოცხადებას

2.1.10. ცვლადების სახელები აუცილებლად უნდა იწყებოდეს:
ა. რიცხვით
ბ. სიმბოლოთი
გ. სპეციალური სიმბოლოთი

2.1.11. ცვლადი გამოცხადებული უნდა იყოს:
ა. მის გამოყენებამდე
ბ. მისი გამოყენების შემდეგ
გ. არასოდეს არ უნდა გამოცხადებული

2.1.12. შეიძლება თუ არა პროგრამის ერთ სტრიქონში ორი ან მეტი ოპერატორის მოთავსება:
ა. ნაწილობრივ შეიძლება
ბ. არ შეიძლება
გ. შეიძლება

2.1.13. მონაცემების შესატანად გამოიყენება:
ა. label კომპონენტი
ბ. textBox კომპონენტი
გ. button კომპონენტი

2.1.14. მონაცემების გამოსატანად გამოიყენება:
ა. label კომპონენტი
ბ. textBox კომპონენტი
გ. button კომპონენტი

2.1.15. მეთოდის შესასრულებლად გამოიყენება:
ა. button კომპონენტი
ბ. label კომპონენტი
გ. textBox კომპონენტი

2.1.16. textBox კომპონენტის Text თვისებას აქვს:
ა. მთელი ტიპი
ბ. წილადი ტიპი
გ. სტრიქონული ტიპი

2.1.17. textBox კომპონენტში შეტანილი მონაცემები ენიჭება ამავე კომპონენტის:
ა. Text თვისებას

ბ. Font თვისებას
გ. Enabled თვისებას

2.1.18. Convert.ToInt32 მეთოდი თავის არგუმენტს გარდაქმნის:
ა. წილად რიცხვად
ბ. სტრიქონად
გ. მთელ რიცხვად

2.1.19. მინიჭების ოპერატორის მარცხნივ მოთავსებულ ცვლადს და მარჯვნივ მოთავსებულ გამოსახულებას უნდა ჰქონდეს:
ა. ერთნაირი ტიპი
ბ. სხვადასხვა ტიპი
გ. მხოლოდ მთელი ტიპი

2.1.20. ToString() მეთოდი ნებისმიერი ტიპის ცვლადს გარდაქმნის:
ა. მთელ რიცხვად
ბ. სტრიქონად
გ. სიმბოლოდ

2.1.21. C# ენაში ზედა და ქვედა რეგისტრის სიმბოლოები:
ა. არ არის განსხვავებული
ბ. წილადი ტიპისაა
გ. განსხვავებულია

2.1.22. კოდის რომელი ფრაგმენტი შეიცავს შეცდომას:
ა.
```
{
    int Ricxvebi;
    ricxvebi = 51;
}
```
ბ.
```
{
    int ricxvebi;
    ricxvebi = 51;
}
```
გ.
```
{
    int Ricxvebi;
    Ricxvebi = 51;
}
```

2.1.23. C# პროგრამის ფაილებს აქვთ:
ა. .exe გაფართოება
ბ. .cs გაფართოება
გ. .cpp გაფართოება

C# ენის საბაზო ტიპები

2.2.1. C# ენაში არსებობს:
ა. ჩვეულებრივი და მიმართვითი ტიპები
ბ. მხოლოდ ჩვეულებრივი ტიპები
გ. მხოლოდ მიმართვითი ტიპები

2.2.2. ჩვეულებრივი ტიპის ცვლადები შეიცავენ:
ა. ობიექტების მისამართებს
ბ. კონკრეტულ მნიშვნელობებს
გ. მხოლოდ სტრიქონებს

2.2.3. მიმართვითი ტიპის ცვლადები შეიცავენ:
ა. ობიექტების მისამართებს
ბ. კონკრეტულ მნიშვნელობებს
გ. მხოლოდ სტრიქონებს

2.2.4. byte ტიპი აღწერს:
ა. წილადს
ბ. ლოგიკურ ცვლადს
გ. 8 ბიტიანი უნიშნო მთელი რიცხვს

2.2.5. char ტიპი აღწერს:
ა. მთელ რიცხვს
ბ. სიმბოლოს
გ. ლოგიკურ ცვლადს

2.2.6. decimal ტიპი აღწერს:
ა. ციფრული ტიპის საფინანსო ცვლადს
ბ. მთელ რიცხვს
გ. წილადს

2.2.7. float ტიპი აღწერს:
ა. ორმაგი სიზუსტის წილადს
ბ. ლოგიკურ ცვლადს
გ. ერთმაგი სიზუსტის წილადს

2.2.8. long ტიპი აღწერს:
ა. გრძელ მთელ რიცხვს
ბ. წილადს
გ. ლოგიკურ ცვლადს

2.2.9. sbyte ტიპი აღწერს:
ა. 8 ბიტიან ნიშნიან მთელ რიცხვს
ბ. მთელ რიცხვს
გ. წილადს

2.2.10. short ტიპი აღწერს:
ა. წილადს
ბ. ლოგიკურ ცვლადს
გ. მოკლე მთელ რიცხვს

2.2.11. uint ტიპი აღწერს:
ა. მთელ რიცხვს
ბ. უნიშნო მთელ რიცხვს
გ. ლოგიკურ ცვლადს

2.2.12. ulong ტიპი აღწერს:
ა. უნიშნო გრძელ მთელ რიცხვს
ბ. მთელ რიცხვს
გ. წილადს

2.2.13. ushort ტიპი აღწერს:
ა. წილადს
ბ. ლოგიკურ ცვლადს
გ. უნიშნო მოკლე მთელ რიცხვს

2.2.14. byte ტიპის ცვლადი მნიშვნელობებს იღებს დიაპაზონში:
ა. 0÷65537
ბ. -32768÷32767
გ. 0÷255

2.2.15. bool ტიპის ცვლადი იღებს:
ა. მხოლოდ true მნიშვნელობებს
ბ. მხოლოდ false მნიშვნელობებს
გ. true და false მნიშვნელობებს

2.2.16. int ტიპის ცვლადი იკავებს:
ა. 8 ბაიტს
ბ. 4 ბაიტს
გ. 1 ბაიტს

2.2.17. long ტიპის ცვლადი იკავებს:
ა. 1 ბაიტს
ბ. 8 ბაიტს
გ. 4 ბაიტს

2.2.18. byte ტიპის ცვლადი იკავებს:
ა. 1 ბაიტს
ბ. 8 ბაიტს
გ. 4 ბაიტს

2.2.19. char ტიპის ცვლადი იკავებს:
ა. 2 ბაიტს
ბ. 1 ბაიტს
გ. 4 ბაიტს

2.2.20. decimal ტიპის ცვლადი იკავებს:
ა. 8 ბაიტს
ბ. 16 ბაიტს
გ. 4 ბაიტს

2.2.21. double ტიპის ცვლადი იკავებს:
ა. 4 ბაიტს
ბ. 16 ბაიტს
გ. 8 ბაიტს

2.2.22. float ტიპის ცვლადი იკავებს:
ა. 4 ბაიტს
ბ. 16 ბაიტს
გ. 8 ბაიტს

2.2.23. sbyte ტიპის ცვლადი იკავებს:
ა. 1 ბაიტს
ბ. 2 ბაიტს
გ. 8 ბაიტს

2.2.24. short ტიპის ცვლადი იკავებს:
ა. 4 ბაიტს
ბ. 1 ბაიტს
გ. 2 ბაიტს

2.2.25. uint ტიპის ცვლადი იკავებს:
ა. 4 ბაიტს
ბ. 1 ბაიტს
გ. 8 ბაიტს

2.2.26. ulong ტიპის ცვლადი იკავებს:
ა. 4 ბაიტს
ბ. 1 ბაიტს
გ. 8 ბაიტს

2.2.27. ushort ტიპის ცვლადი იკავებს:
ა. 4 ბაიტს
ბ. 2 ბაიტს
გ. 8 ბაიტს

2.2.28. მთელრიცხვა ტიპებია:
ა. char, byte, short, int
ბ. char, string, ulong, uint
გ. sbyte, short, double, long

2.2.29. ნიშნიან რიცხვებში უფროსი ბიტი:
ა. შედის რიცხვის შემადგენლობაში
ბ. გამოიყენება ნიშნის მისათითებლად
გ. არ გამოიყენება ნიშნის მისათითებლად

2.2.30. უნიშნო რიცხვებში უფროსი ბიტი:
ა. გამოიყენება ნიშნის მისათითებლად
ბ. შედის რიცხვის შემადგენლობაში
გ. არ შედის რიცხვის შემადგენლობაში

2.2.31. უნიშნო რიცხვების აბსოლუტური მნიშვნელობა ნიშნიანი რიცხვების მნიშვნელობას აღემატება:
ა. ოთხჯერ
ბ. სამჯერ
გ. ორჯერ

2.2.32. მცურავწერტილიანი ტიპებია:
ა. float და double
ბ. int და char
გ. char და string

2.2.33. C# ენაში სიმბოლოების წარმოსადგენად გამოიყენება:
ა. სიმბოლოების არც ერთი ნაკრები არ გამოიყენება
ბ. ASCII სიმბოლოების ნაკრები
გ. Unicode სიმბოლოების ნაკრები

2.2.34. Unicode სიმბოლო იკავებს:
ა. 16 ბიტს
ბ. 8 ბიტს
გ. 4 ბიტს

2.2.35. ლიტერალი არის:
ა. ცვლადი
ბ. მუდმივა
გ. ცვლადი და მუდმივა

2.2.36. ლიტერალი შეიძლება იყოს:
ა. მხოლოდ სტრიქონი
ბ. მხოლოდ წილადი
გ. ნებისმიერი ტიპის მნიშვნელობა

2.2.37. u ან U სუფიქსი გამოიყენება:
ა. უნიშნო მთელრიცხვა ლიტერალის აღსანიშნავად
ბ. უნიშნო წილადი ლიტერალის აღსანიშნავად
გ. სტრიქონის აღსანიშნავად

2.2.38. f ან F სუფიქსი გამოიყენება:
ა. მთელრიცხვა ლიტერალის აღსანიშნავად
ბ. წილადი ლიტერალის აღსანიშნავად
გ. სტრიქონის აღსანიშნავად

2.2.39. m ან M სუფიქსი გამოიყენება:
ა. მთელრიცხვა ლიტერალის აღსანიშნავად
ბ. decimal ტიპის ლიტერალის აღსანიშნავად
გ. სიმბოლოს აღსანიშნავად

2.2.40. ul ან UL სუფიქსი გამოიყენება:
ა. სტრიქონის აღსანიშნავად
ბ. გრძელი უნიშნო წილადი ლიტერალის აღსანიშნავად
გ. გრძელი უნიშნო მთელრიცხვა ლიტერალის აღსანიშნავად

2.2.41. ToDecimal() მეთოდი მონაცემებს გარდაქმნის:
ა. მთელი ტიპის მონაცემად
ბ. სიმბოლოდ
გ. საფინანსო ტიპის მონაცემად

2.2.42. ToBoolean() მეთოდი მონაცემებს გარდაქმნის:
ა. ლოგიკური ტიპის მონაცემად
ბ. სიმბოლოდ
გ. საფინანსო ტიპის მონაცემად

2.2.43. ToDouble() მეთოდი მონაცემებს გარდაქმნის:
ა. წილადად
ბ. სიმბოლოდ
გ. საფინანსო ტიპის მონაცემად

2.2.44. ToChar() მეთოდი მონაცემებს გარდაქმნის:
ა. ლოგიკური ტიპის მონაცემად
ბ. სიმბოლოდ
გ. საფინანსო ტიპის მონაცემად

2.2.45. რა მნიშვნელობებს იღებს bool ტიპის მნიშვნელობა:
ა. -1 და 0
ბ. -1 და 1
გ. true და false

2.2.46. მთელრიცხვა ტიპებია:
ა. char, byte, sbyte, short, ushort, int, uint, long, ulong
ბ. string, double
გ. bool, char, int, float

2.2.47. ნიშნიანი მთელრიცხვა ტიპებია:
ა. uint, bool
ბ. sbyte, short, int, long
გ. ushort, char

2.2.48. უნიშნო მთელრიცხვა ტიპებია:
ა. byte, uint, ulong, ushort
ბ. sbyte, short
გ. uint, bool

2.2.49. რაში მდგომარეობს განსხვავება ნიშნიან და უნიშნო მთელ რიცხვებს შორის:
ა. ნიშნიან რიცხვებში უფროსი თანრიგი ეთმობა რიცხვის ნიშანს
ბ. უნიშნო რიცხვებში უფროსი თანრიგი ეთმობა რიცხვის ნიშანს
გ. ნიშნიან რიცხვებში უმცროსი თანრიგი ეთმობა რიცხვის ნიშანს

ცვლადები და მათი ინიციალიზაცია

2.3.1. ცვლადი არის:
ა. მეხსიერების სახელდებული უბანი, რომელსაც მნიშვნელობა ენიჭება
ბ. მუდმივა, რომელსაც სახელი აქვს
გ. მხოლოდ წილადი ტიპის მონაცემი

2.3.2. ცვლადის ინიციალიზაცია არის:
ა. ცვლადისათვის მნიშვნელობის მინიჭება მისი გამოცხადების გარეშე
ბ. ცვლადისათვის მნიშვნელობის მინიჭება მისი გამოცხადების დროს
გ. ცვლადისათვის მნიშვნელობის მინიჭება მისი გამოცხადების შემდეგ

2.3.3. ცვლადის დინამიური ინიციალიზაცია სრულდება:
ა. პროგრამის ნებისმიერ ადგილში
ბ. პროგრამის დასაწყისში
გ. მხოლოდ მასივის გამოცხადების წინ

2.3.4. როდის სრულდება ცვლადის დინამიური ინიციალიზაცია:
ა. პროგრამის დამთავრების შემდეგ
ბ. პროგრამის შესრულების დროს
გ. პროგრამის დაწყებამდე

2.3.5. როგორია ცვლადის გამოცხადების სინტაქსი:
ა. ტიპი

ბ. ცვლადის_სახელი
გ. ტიპი ცვლადის_სახელი

2.3.6. როგორია ცვლადის ინიციალიზაციის ოპერატორის სინტაქსი:
ა. ტიპი ცვლადის_სახელი = მნიშვნელობა;
ბ. ტიპი = მნიშვნელობა;
გ. ტიპი = ცვლადის_სახელი;

ოპერატორები

2.4.1. ოპერატორების რამდენი ძირითადი კლასი არსებობს C# ენაში:
ა. 4
ბ. 3
გ. 5

2.4.2. ოპერატორების რომელი ძირითადი კლასები არსებობს:
ა. მხოლოდ არითმეტიკის და შედარების
ბ. არითმეტიკის, ლოგიკის, შედარებისა და ბიტობრივი
გ. მხოლოდ ლოგიკის და ბიტობრივი

2.4.3. C# ენაში არსებობს შემდეგი ოპერატორები:
ა. მხოლოდ ლოგიკის და ბიტობრივი
ბ. მხოლოდ არითმეტიკის და შედარების
გ. არითმეტიკის, ლოგიკის, შედარებისა და ბიტობრივი

2.4.4. რა არის ოპერატორი:
ა. ოპერატორი არის მასივი, რომელიც კომპილატორს ატყობინებს თუ რომელი ოპერაცია უნდა შესრულდეს
ბ. ოპერატორი არის სიმბოლო, რომელიც კომპილატორს ატყობინებს თუ რომელი ოპერაცია უნდა შესრულდეს
გ. ოპერატორი არის ობიექტი, რომელიც კომპილატორს ატყობინებს თუ რომელი ოპერაცია უნდა შესრულდეს

2.4.5. არითმეტიკის ოპერატორები გამოიყენება:
ა. სიმბოლოების მიმართ
ბ. სტრიქონების მიმართ
გ. მთელი და წილადი რიცხვების მიმართ

2.4.6. როცა გაყოფის ოპერატორს ვიყენებთ მთელი რიცხვების მიმართ, მაშინ შედეგი იქნება:
ა. წილადი
ბ. მთელი რიცხვი
გ. სიმბოლო

2.4.7. როცა გაყოფის ოპერატორს ვიყენებთ წილადი რიცხვების მიმართ, მაშინ

შედეგი იქნება:
ა. წილადი
ბ. მთელი რიცხვი
გ. სტრიქონი

2.4.8. როცა წილადი იყოფა მთელ რიცხვზე, მაშინ შედეგი იქნება:
ა. მთელი რიცხვი
ბ. წილადი
გ. სიმბოლო

2.4.9. როცა მთელი რიცხვი იყოფა წილადზე, მაშინ შედეგი იქნება:
ა. მთელი რიცხვი
ბ. ლოგიკური მნიშვნელობა
გ. წილადი

2.4.10. რომელია არითმეტიკის ოპერატორები:
ა. <, >, !=
ბ. +, -, *, /, %, ++, --
გ. =, >, !=

2.4.11. რომელი ოპერატორი გამოიყენება ნაშთის მისაღებად:
ა. %
ბ. /
გ. *

2.4.12. (2 + 3) % 2 გამოსახულების გამოთვლის შედეგი იქნება:
ა. 0
ბ. 1
გ. 2

2.4.13. ინკრემენტის ოპერატორი:
ა. ორით ზრდის თავისი ოპერანდის მნიშვნელობას
ბ. ერთით ამცირებს თავისი ოპერანდის მნიშვნელობას
გ. ერთით ზრდის თავისი ოპერანდის მნიშვნელობას

2.4.14. დეკრემენტის ოპერატორი:
ა. ორით ზრდის თავისი ოპერანდის მნიშვნელობას
ბ. ერთით ამცირებს თავისი ოპერანდის მნიშვნელობას
გ. ერთით ზრდის თავისი ოპერანდის მნიშვნელობას

2.4.15. რა ფუნქციას ასრულებს ++ ოპერატორი:
ა. ერთით ზრდის თავისი ოპერანდის მნიშვნელობას
ბ. ერთით ამცირებს თავისი ოპერანდის მნიშვნელობას
გ. ორით ზრდის თავისი ოპერანდის მნიშვნელობას

2.4.16. რა ფუნქციას ასრულებს -- ოპერატორი:
ა. ერთით ზრდის თავისი ოპერანდის მნიშვნელობას
ბ. ერთით ამცირებს თავისი ოპერანდის მნიშვნელობას
გ. ორით ზრდის თავისი ოპერანდის მნიშვნელობას

2.4.17. როცა ინკრემენტის ოპერატორი მითითებულია ოპერანდის წინ, მაშინ გვაქვს:
ა. პოსტფიქსური ინკრემენტი
ბ. პრეფიქსური დეკრემენტი
გ. პრეფიქსური ინკრემენტი

2.4.18. როცა ინკრემენტის ოპერატორი მითითებულია ოპერანდის შემდეგ, მაშინ გვაქვს:
ა. პოსტფიქსური ინკრემენტი
ბ. პოსტფიქსური დეკრემენტი
გ. პრეფიქსური ინკრემენტი

2.4.19. როცა დეკრემენტის ოპერატორი მითითებულია ოპერანდის წინ, მაშინ გვაქვს:
ა. პოსტფიქსური ინკრემენტი
ბ. პრეფიქსური დეკრემენტი
გ. პრეფიქსური ინკრემენტი

2.4.20. როცა დეკრემენტის ოპერატორი მითითებულია ოპერანდის შემდეგ, მაშინ გვაქვს:
ა. პოსტფიქსური ინკრემენტი
ბ. პოსტფიქსური დეკრემენტი
გ. პრეფიქსური ინკრემენტი

2.4.21. პრეფიქსური ინკრემენტის შემთხვევაში:
ა. ჯერ შესრულდება ოპერანდის მნიშვნელობის გაზრდა, შემდეგ კი მისი გამოყენება გამოსახულებაში
ბ. ჯერ შესრულდება ოპერანდის გამოყენება გამოსახულებაში, შემდეგ კი მისი მნიშვნელობის გაზრდა
გ. ჯერ შესრულდება ოპერანდის მნიშვნელობის შემცირება, შემდეგ კი მისი გამოყენება გამოსახულებაში

2.4.22. პოსტფიქსური ინკრემენტის შემთხვევაში:
ა. ჯერ შესრულდება ოპერანდის მნიშვნელობის გაზრდა, შემდეგ კი მისი გამოყენება გამოსახულებაში
ბ. ჯერ შესრულდება ოპერანდის გამოყენება გამოსახულებაში, შემდეგ კი მისი მნიშვნელობის გაზრდა
გ. ჯერ შესრულდება ოპერანდის მნიშვნელობის შემცირება, შემდეგ კი მისი გამოყენება გამოსახულებაში

2.4.23. პრეფიქსური დეკრემენტის შემთხვევაში:
ა. ჯერ შესრულდება ოპერანდის მნიშვნელობის გაზრდა, შემდეგ კი მისი გამოყენება გამოსახულებაში
ბ. ჯერ შესრულდება ოპერანდის გამოყენება გამოსახულებაში, შემდეგ კი მისი

მნიშვნელობის გაზრდა
გ. ჯერ შესრულდება ოპერანდის მნიშვნელობის შემცირება, შემდეგ კი მისი გამოყენება გამოსახულებაში

2.4.24. პოსტფიქსური დეკრემენტის შემთხვევაში:
ა. ჯერ შესრულდება ოპერანდის მნიშვნელობის გაზრდა, შემდეგ კი მისი გამოყენება გამოსახულებაში
ბ. ჯერ შესრულდება ოპერანდის გამოყენება გამოსახულებაში, შემდეგ კი მისი მნიშვნელობის შემცირება
გ. ჯერ შესრულდება ოპერანდის მნიშვნელობის შემცირება, შემდეგ კი მისი გამოყენება გამოსახულებაში

2.4.25. {
int ricxvi = 5, shedegi;
shedegi = ++ricxvi;
}
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:
ა. 4
ბ. 5
გ. 6

2.4.26. {
int ricxvi = 5, shedegi;
shedegi = ricxvi++;
}
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:
ა. 4
ბ. 5
გ. 6

2.4.27. {
int ricxvi = 5, shedegi;
shedegi = --ricxvi;
}
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:
ა. 5
ბ. 4
გ. 6

2.4.28. {
int ricxvi = 5, shedegi;
shedegi = ricxvi--;
}
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:
ა. 4

ბ. 5
გ. 6

2.4.29. {
int ricxvi = 5, shedegi;
shedegi = ++ricxvi;
}
ამ კოდის შესრულების შედეგად ricxvi ცვლადი მიიღებს მნიშვნელობას:
ა. 4
ბ. 5
გ. 6

2.4.30. {
int ricxvi = 5, shedegi;
shedegi = ricxvi++;
}
ამ კოდის შესრულების შედეგად ricxvi ცვლადი მიიღებს მნიშვნელობას:
ა. 4
ბ. 5
გ. 6

2.4.31. {
int ricxvi = 5, shedegi;
shedegi = --ricxvi;
}
ამ კოდის შესრულების შედეგად ricxvi ცვლადი მიიღებს მნიშვნელობას:
ა. 5
ბ. 4
გ. 6

2.4.32. {
int ricxvi = 5, shedegi;
shedegi = ricxvi--;
}
ამ კოდის შესრულების შედეგად ricxvi ცვლადი მიიღებს მნიშვნელობას:
ა. 4
ბ. 5
გ. 6

შედარებისა და ლოგიკის ოპერატორები

2.5.1. შედარების ოპერატორები გასცემენ:
ა. char ტიპის მნიშვნელობებს
ბ. bool ტიპის მნიშვნელობებს
გ. int ტიპის მნიშვნელობებს

2.5.2. bool ტიპის მნიშვნელობების შედარება შეიძლება:

ა. მეტობაზე ან ნაკლებობაზე

ბ. ტოლობაზე ან მეტობაზე

გ. ტოლობაზე ან უტოლობაზე

2.5.3. მონაცემთა როგორი ტიპების მიმართ გამოიყენება შედარების ოპერატორები >, <, >=, <= :

ა. ჩამოთვლადი ტიპების მიმართ

ბ. არაჩამოთვლადი ტიპების მიმართ

გ. მხოლოდ სტრიქონების მიმართ

2.5.4.
```
{
    int ricxvi1 = 5, ricxvi2 = 10;
    bool shedegi;
    shedegi = ricxvi1 > ricxvi2;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. false

ბ. true

გ. 5

2.5.5.
```
{
    int ricxvi1 = 5, ricxvi2 = 10;
    bool shedegi;
    shedegi = ricxvi1 < ricxvi2;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. false

ბ. true

გ. 5

2.5.6.
```
{
    int ricxvi1 = 5, ricxvi2 = 10;
    bool shedegi;
    shedegi = ricxvi1 == ricxvi2;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. 5

ბ. true

გ. false

2.5.7.
```
{
    int ricxvi1 = 5, ricxvi2 = 10;
    bool shedegi;
```

```
        shedegi = ricxvi1 != ricxvi2;
    }
```

ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. 5
ბ. true
გ. false

2.5.8.
```
    {
        int ricxvi1 = 5, ricxvi2 = 10;
        bool shedegi;
        shedegi = ricxvi1 >= ricxvi2;
    }
```

ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. 5
ბ. true
გ. false

2.5.9.
```
    {
        int ricxvi1 = 5, ricxvi2 = 10;
        bool shedegi;
        shedegi = ricxvi1 <= ricxvi2;
    }
```

ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. 5
ბ. true
გ. false

2.5.10. რა არის შედარების ოპერატორების შესრულების შედეგი:

ა. true და false
ბ. -1 და 0
გ. -1 და 1

2.5.11. როგორი ტიპი უნდა ჰქონდეს ლოგიკის ოპერატორებს:

ა. int
ბ. bool
გ. short

2.5.12. რომელია შედარების ოპერატორები:

ა. +, -, *
ბ. ==, !=, >, <, <=, >=
გ. /, %, ++

2.5.13. რომელია ლოგიკის ოპერატორები:

ა. ==, !=, >
ბ. -, /, *

გ. &, |, ^, &&, ||, !

2.5.14. & ოპერატორი გასცემს true მნიშვნელობას მაშინ, როცა:
ა. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა false
ბ. მისი ორივე ოპერანდის მნიშვნელობაა false
გ. მისი ორივე ოპერანდის მნიშვნელობაა true

2.5.15. | ოპერატორი გასცემს true მნიშვნელობას მაშინ, როცა:
ა. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა true
ბ. მისი ორივე ოპერანდის მნიშვნელობაა false
გ. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა 0

2.5.16. ! ოპერატორი გასცემს true მნიშვნელობას მაშინ, როცა:
ა. მისი ოპერანდის მნიშვნელობაა false
ბ. მისი ორივე ოპერანდის მნიშვნელობაა false
გ. მისი ოპერანდის მნიშვნელობაა true

2.5.17. ^ ოპერატორი გასცემს true მნიშვნელობას მაშინ, როცა:
ა. მის ორივე ოპერანდს ერთნაირი მნიშვნელობები აქვს
ბ. მის ორივე ოპერანდს განსხვავებული მნიშვნელობები აქვს
გ. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა true

2.5.18. & ოპერატორი გასცემს false მნიშვნელობას მაშინ, როცა:
ა. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა false
ბ. მისი ორივე ოპერანდის მნიშვნელობაა true
გ. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა true

2.5.19. | ოპერატორი გასცემს false მნიშვნელობას მაშინ, როცა:
ა. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა true
ბ. მისი ორივე ოპერანდის მნიშვნელობაა false
გ. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა false

2.5.20. ! ოპერატორი გასცემს false მნიშვნელობას მაშინ, როცა:
ა. მისი ოპერანდის მნიშვნელობაა false
ბ. მისი ორივე ოპერანდის მნიშვნელობაა false
გ. მისი ოპერანდის მნიშვნელობაა true

2.5.21. ^ ოპერატორი გასცემს false მნიშვნელობას მაშინ, როცა:
ა. მის ორივე ოპერანდს ერთნაირი მნიშვნელობები აქვს
ბ. მის ორივე ოპერანდს განსხვავებული მნიშვნელობები აქვს
გ. მისი ერთ-ერთი ოპერანდის მნიშვნელობაა true

2.5.22. {

```
bool b1 = true, b2 = false, shedegi;
shedegi = b1 & b2;
```

```
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. true

ბ. false

გ. 5

2.5.23.
```
{
    bool b1 = true, b2 = false, shedegi;
    shedegi = b1 | b2;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. true

ბ. false

გ. 5

2.5.24.
```
{
    bool b1 = true, b2 = false, shedegi;
    shedegi = b1 ^ b2;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. true

ბ. false

გ. 5

2.5.25.
```
{
    bool b1 = true, shedegi;
    shedegi = !b1;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. true

ბ. 5

გ. false

2.5.26.
```
{
    bool b1 = true, b2 = false, b3 = true, b4 = false, shedegi;
    shedegi = !b1 || b2 && !b3 || b4;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. true

ბ. 5

გ. false

2.5.27.
```
{
    bool b1 = true, b2 = false, b3 = true, shedegi;
    shedegi = b1 || b2 && b3;
```

}

ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:

ა. true
ბ. 5
გ. false

2.5.28. რომელია ლოგიკის სწრაფი ოპერატორი:

ა. &&
ბ. &
გ. ||

2.5.29. ლოგიკის სწრაფი ოპერატორია:

ა. &
ბ. !
გ. ||

2.5.30. თუ && ოპერატორის შესრულებისას პირველმა ოპერანდმა მიიღო false მნიშვნელობა, მაშინ შედეგი იქნება:

ა. true
ბ. false
გ. 5

2.5.31. თუ || ოპერატორის შესრულებისას პირველმა ოპერანდმა მიიღო true მნიშვნელობა, მაშინ შედეგი იქნება:

ა. true
ბ. false
გ. 5

მინიჭების ოპერატორი. შედგენილი ოპერატორები

2.6.1. როგორია მინიჭების ოპერატორის სინტაქსი:

ა. ცვლადი = გამოსახულება;
ბ. გამოსახულება = ცვლადი;
გ. ცვლადი + გამოსახულება;

2.6.2. შეიძლება თუ არა, რომ ცვლადსა და გამოსახულებას სხვადასხვა ტიპი ჰქონდეს:

ა. კი
ბ. არა
გ. ნაწილობრივ

2.6.3. ricxvi1 = ricxvi2 = ricxvi3 = 25; ოპერატორის შესრულების შედეგად როგორი იქნება ricxvi2 ცვლადის მნიშვნელობა:

ა. 25
ბ. 52

გ. 0

2.6.4. შეიძლება თუ არა, რომ მინიჭების ოპერატორის საშუალებით რამდენიმე ცვლადს ერთდროულად მივანიჭოთ მნიშვნელობა:
ა. ნაწილობრივ
ბ. არა
გ. კი

2.6.5. რომელია შედგენილი ოპერატორები:
ა. +=, -+, *=, /=, %=, &=, |=, ^=
ბ. !==, <==
გ. ==, >==

2.6.6. როგორია შედგენილი ოპერატორის სინტაქსი:
ა. ცვლადი ოპერაცია გამოსახულება;
ბ. ცვლადი =ოპერაცია გამოსახულება;
გ. ცვლადი ოპერაცია= გამოსახულება;

2.6.7. რა უპირატესობა აქვს მინიჭების შედგენილი ოპერატორის გამოყენებას მინიჭების ჩვეულებრივი ოპერატორის გამოყენებასთან შედარებით:
ა. მინიჭების შედგენილი ოპერატორი უფრო კომპაქტურია და მისი გამოყენებისას ჩქარდება კოდი კომპილირება
ბ. მინიჭების შედგენილი ოპერატორი ნაკლებად კომპაქტურია და მისი გამოყენებისას არ ჩქარდება კოდი კომპილირება
გ. არავითარი უპირატესობა არ აქვს

2.6.8. jami += 25; მინიჭების ოპერატორი ქვემოთ მოყვანილი რომელი ოპერატორის ანალოგიურია:
ა. jami = jami + jami + 25;
ბ. jami = jami - 25;
გ. jami = jami + 25;

2.6.9. jami -= 25; მინიჭების ოპერატორი ქვემოთ მოყვანილი რომელი ოპერატორის ანალოგიურია:
ა. jami = jami - jami - 25;
ბ. jami = jami - 25;
გ. jami = 25 - jami;

2.6.10. jami *= 25; მინიჭების ოპერატორი ქვემოთ მოყვანილი რომელი ოპერატორის ანალოგიურია:
ა. jami = jami * jami * 25;
ბ. jami = jami - 25;
გ. jami = jami * 25;

2.6.11. jami /= 25; მინიჭების ოპერატორი ქვემოთ მოყვანილი რომელი ოპერატორის

ანალოგიურია:
ა. jami = jami / jami / 25;
ბ. jami = jami / 25;
გ. jami = jami % 25;

2.6.12. jami %= 25; მინიჭების ოპერატორი ქვემოთ მოყვანილი რომელი ოპერატორის ანალოგიურია:
ა. jami = jami % 25;
ბ. jami = jami % jami % 25;
გ. jami = jami / 25;

ოპერატორების პრიორიტეტები

2.7.1. * და + ოპერატორებს შორის რომელს აქვს მაღალი პრიორიტეტი:
ა. *
ბ. +
გ. მათ თანაბარი პრიორიტეტი აქვთ

2.7.2. - და / ოპერატორებს შორის რომელს აქვს მაღალი პრიორიტეტი:
ა. -
ბ. /
გ. მათ თანაბარი პრიორიტეტი აქვთ

2.7.3. {
int ricxvi1 = 5, ricxvi2 = 3, ricxvi3 = 2, ricxvi4 = 4, ricxvi5 = 1,shedegi;
shedegi = ricxvi1 - ricxvi2 / ricxvi3 + ricxvi4 * ricxvi5;
}
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:
ა. 7.5
ბ. 5.7
გ. 3

2.7.4. {
int ricxvi1 = 1, ricxvi2 = 3, ricxvi3 = 2, ricxvi4 = 5, ricxvi5 = 4, ricxvi6 = 6;
double shedegi = (ricxvi1 + ricxvi2) / ricxvi3 +
(ricxvi4 - ricxvi5) * ricxvi6;
}
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:
ა. 5
ბ. 8
გ. 3

2.7.5. რომელი მსჯელობაა სწორი:
ა. / ოპერატორს აქვს - ოპერატორზე მაღალი პრიორიტეტი
ბ. + ოპერატორს აქვს % ოპერატორზე მაღალი პრიორიტეტი

გ. - ოპერატორს აქვს + ოპერატორზე მაღალი პრიორიტეტი

გამოსახულებები. მათემატიკური მეთოდები

2.8.1. რისთვის გამოიყენება ტაბულაციის სიმბოლოები და ინტერვალები:
ა. პროგრამის კომპილირების დასაჩქარებლად
ბ. პროგრამის შესრულების დასაჩქარებლად
გ. პროგრამის კითხვადობის გასაუმჯობესებლად

2.8.2. რომელ სტანდარტულ კლასშია აღწერილი მათემატიკური ფუნქციები:
ა. System.Math
ბ. System.Object
გ. System.Exception

2.8.3. რომელ ტიპს იყენებს მათემატიკური ფუნქციების უმრავლესობა:
ა. long
ბ. double
გ. uint

2.8.4. Floor(x) მეთოდი თავის არგუმენტს ამრგვალებს:
ა. ნაკლებობით და მეტობით
ბ. მეტობით
გ. ნაკლებობით

2.8.5. Ceiling(x) მეთოდი თავის არგუმენტს ამრგვალებს:
ა. ნაკლებობით და მეტობით
ბ. მეტობით
გ. ნაკლებობით

2.8.6. Sqrt(x) მეთოდი:
ა. იღებს კვადრატულ ფესვს თავისი არგუმენტიდან
ბ. აყავს კვადრატში თავისი არგუმენტი
გ. ამრგვალებს თავის არგუმენტს

2.8.7. Round(x,y) მეთოდი:
ა. გასცემს პირველ არგუმენტს ისეთი სახით, რომ მას წილად ნაწილში ჰქონდეს მეორე
არგუმენტით მითითებული თანრიგების რაოდენობა
ბ. გასცემს მის ორ არგუმენტს შორის მაქსიმალურს
გ. გასცემს მის ორ არგუმენტს შორის მინიმალურს

2.8.8. Max(x,y) მეთოდი:
ა. გასცემს მის ორ არგუმენტს შორის მინიმალურს
ბ. გასცემს მისი პირველი არგუმენტის კვადრატს
გ. გასცემს მის ორ არგუმენტს შორის მაქსიმალურს

2.8.9. Min(x,y) მეთოდი:

ა. გასცემს მის ორ არგუმენტს შორის მაქსიმალურს

ბ. გასცემს მის ორ არგუმენტს შორის მინიმალურს

გ. გასცემს მისი მეორე არგუმენტის კვადრატს

2.8.10. Abs(x) მეთოდი:

ა. გასცემს მისი არგუმენტის უარყოფით მნიშვნელობას

ბ. გასცემს მისი არგუმენტის აბსოლუტურ მნიშვნელობას

გ. გასცემს მისი არგუმენტის კვადრატს

2.8.11. Pow(x,y) მეთოდი:

ა. პირველი არგუმენტი აჰყავს მეორე არგუმენტით მითითებულ ხარისხში

ბ. მეორე არგუმენტი აჰყავს პირველი არგუმენტით მითითებულ ხარისხში

გ. გასცემს მისი არგუმენტის კვადრატს

2.8.12. Log(x) მეთოდი:

ა. გასცემს მისი არგუმენტის კვადრატს

ბ. გასცემს მისი არგუმენტის უარყოფით მნიშვნელობას

გ. გასცემს მისი არგუმენტის ლოგარითმს

2.8.13. Sin(x) მეთოდი:

ა. გასცემს მისი არგუმენტის ლოგარითმს

ბ. გასცემს მისი არგუმენტის კოსინუსს

გ. გასცემს მისი არგუმენტის სინუსს

2.8.14. Cos(x) მეთოდი:

ა. გასცემს მისი არგუმენტის სინუსს

ბ. გასცემს მისი არგუმენტის კოსინუსს

გ. გასცემს მისი არგუმენტის ლოგარითმს

2.8.15. Tan(x) მეთოდი:

ა. გასცემს მისი არგუმენტის ტანგენსს

ბ. გასცემს მისი არგუმენტის სინუსს

გ. გასცემს მისი არგუმენტის კოსინუსს

2.8.16. Exp(x) მეთოდი:

ა. გასცემს მისი არგუმენტის კვადრატს

ბ. e რიცხვი აჰყავს მითითებულ ხარისხში

გ. გასცემს მისი არგუმენტის კოსინუსს

2.8.17. რომელი მინიჭების ოპერატორი გამოთვლის მოცემული გამოსახულების მნიშვნელობას:

$$y = \frac{\sqrt{x^3 + e^x}}{\sin x}$$

ა. y = Math.Sqrt(Math.Pow(x,3) + Math.Exp(x)) / Math.Sin(x);
ბ. y = Math.Sqrt((Math.Pow(x,3) + Math.Exp(x)) / Math.Sin(x));
გ. y = (Math.Pow(x,3) + Math.Exp(x)) / Math.Sin(x);

ტიპების გარდაქმნა

2.9.1. ჩვეულებრივ შემთხვევაში ერთი ტიპის მქონე ცვლადს:
ა. ნაწილობრივ უნდა მივანიჭოთ ამავე ტიპის მნიშვნელობა
ბ. არ უნდა მივანიჭოთ ამავე ტიპის მნიშვნელობა
გ. უნდა მივანიჭოთ ამავე ტიპის მნიშვნელობა

2.9.2. ჩვეულებრივ შემთხვევაში მინიჭების ოპერატორის მარცხნივ მოთავსებულ ცვლადს და მარჯვნივ მოთავსებულ გამოსახულებას:
ა. ერთნაირი ტიპები არ უნდა ჰქონდეს
ბ. ერთნაირი ტიპები უნდა ჰქონდეს
გ. ერთნაირი ტიპები ნაწილობრივ უნდა ჰქონდეს

2.9.3. შეიძლება თუ არა, რომ double ტიპის ცვლადს მივანიჭოთ int ტიპის მნიშვნელობა
ა. კი
ბ. არა
გ. ნაწილობრივ

2.9.4. თუ მინიჭების ოპერატორში გამოიყენება თავსებადი ტიპები, მაშინ:
ა. მინიჭების ოპერატორის მარჯვნივ მოთავსებული გამოსახულების ტიპი ავტომატურად გარდაიქმნება მინიჭების ოპერატორის მარცხნივ მოთავსებული ცვლადის ტიპად
ბ. მინიჭების ოპერატორის მარჯვნივ მოთავსებული გამოსახულების ტიპი ავტომატურად არ გარდაიქმნება მინიჭების ოპერატორის მარცხნივ მოთავსებული ცვლადის ტიპად
გ. მინიჭების ოპერატორის მარცხნივ მოთავსებული გამოსახულების ტიპი ავტომატურად გარდაიქმნება მინიჭების ოპერატორის მარჯვნივ მოთავსებული ცვლადის ტიპად

2.9.5. თავსებადია თუ არა int და bool ტიპები:
ა. ნაწილობრივ
ბ. კი
გ. არა

2.9.6. შესრულდება თუ არა ტიპების ავტომატური გარდაქმნა decimal და double ტიპებს შორის:
ა. არა

ბ. კი
გ. ნაწილობრივ

2.9.7. შესრულდება თუ არა ტიპების ავტომატური გარდაქმნა decimal და float ტიპებს შორის:
ა. არა
ბ. კი
გ. ნაწილობრივ

2.9.8. ტიპების ავტომატური გარდაქმნა არ სრულდება:
ა. double და float ტიპებს შორის
ბ. long და int ტიპებს შორის
გ. decimal და float ტიპებს შორის

2.9.9. ტიპების ავტომატური გარდაქმნა არ სრულდება:
ა. long და int ტიპებს შორის
ბ. short და int ტიპებს შორის
გ. decimal და double ტიპებს შორის

2.9.10. ტიპების ავტომატური გარდაქმნა არ სრულდება:
ა. char და bool ტიპებს შორის
ბ. long და int ტიპებს შორის
გ. double და float ტიპებს შორის

2.9.11. ტიპების ავტომატური გარდაქმნა არ სრულდება:
ა. short და int ტიპებს შორის
ბ. int და char ტიპებს შორის
გ. double და float ტიპებს შორის

2.9.12. ტიპების ავტომატური გარდაქმნა არ სრულდება:
ა. int და bool ტიპებს შორის
ბ. double და float ტიპებს შორის
გ. short და int ტიპებს შორის

2.9.13. როდის სრულდება ტიპების ავტომატური გარდაქმნა:
ა. როცა ორივე ტიპი თავსებადია და საბოლოო ტიპი მეტია საწყის ტიპზე
ბ. როცა ორივე ტიპი თავსებადია და საბოლოო ტიპი ნაკლებია საწყის ტიპზე
გ. როცა ორივე ტიპი არათავსებადია და საბოლოო ტიპი მეტია საწყის ტიპზე

2.9.14. როდის სრულდება ტიპების გაფართოებადი გარდაქმნა:
ა. როცა ორივე ტიპი არათავსებადია და საბოლოო ტიპი მეტია საწყის ტიპზე
ბ. როცა ორივე ტიპი თავსებადია და საბოლოო ტიპი ნაკლებია საწყის ტიპზე
გ. როცა ორივე ტიპი თავსებადია და საბოლოო ტიპი მეტია საწყის ტიპზე

2.9.15. საბოლოო ტიპი მეტია საწყის ტიპზე - ეს იმას ნიშნავს, რომ:

ა. საბოლოო ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობა ნაკლებია საწყისი ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობაზე
ბ. საბოლოო ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობა მეტია საწყისი ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობაზე
გ. საბოლოო ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობა ტოლია საწყისი ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობაზე

2.9.16. ტიპების გაფართოებადი გარდაქმნის შესასრულებლად:
ა. ყველა რიცხვითი ტიპი მთელრიცხვა და მცურავწერტილიანი ტიპების ჩათვლით თავსებადი უნდა იყოს
ბ. ყველა რიცხვითი ტიპი მთელრიცხვა და მცურავწერტილიანი ტიპების ჩათვლით თავსებადი არ უნდა იყოს
გ. ყველა რიცხვითი ტიპი მთელრიცხვა და ლოგიკური ტიპების ჩათვლით თავსებადი უნდა იყოს

2.9.17. შეიძლება თუ არა long ტიპის ცვლადი გარდაიქმნას double ტიპის ცვლადად:
ა. ნაწილობრივ
ბ. არა
გ. კი

2.9.18. შეიძლება თუ არა double ტიპის ცვლადის ავტომატური გარდაქმნა long ტიპის ცვლადად:
ა. კი
ბ. არა
გ. ნაწილობრივ

2.9.19. რატომ არ შეიძლება double ტიპის ცვლადის ავტომატური გარდაქმნა long ტიპის ცვლადად:
ა. იმიტომ, რომ ასეთი გარდაქმნა არის გაფართოებადი
ბ. იმიტომ, რომ ასეთი გარდაქმნა არ არის გაფართოებადი
გ. იმიტომ, რომ ასეთი გარდაქმნა ნაწილობრივ არის გაფართოებადი

2.9.20. რატომ არ შეიძლება long ტიპის ცვლადის ავტომატური გარდაქმნა double ტიპის ცვლადად:
ა. იმიტომ, რომ ასეთი გარდაქმნა არის გაფართოებადი
ბ. იმიტომ, რომ ასეთი გარდაქმნა არ არის გაფართოებადი
გ. იმიტომ, რომ ასეთი გარდაქმნა ნაწილობრივ არის გაფართოებადი

2.9.21. რა არის ტიპის დაყვანის ოპერაცია:
ა. ეს არის bool ტიპის string ტიპად გარდაქმნის ოპერაცია
ბ. ეს არ არის ერთი ტიპის მეორე ტიპად გარდაქმნის ოპერაცია
გ. ეს არის ერთი ტიპის მეორე ტიპად გარდაქმნის ოპერაცია

2.9.22. როგორია ტიპის დაყვანის ოპერაციის სინტაქსი:
ა. გამოსახულება (საბოლოო_ტიპი);

ბ. (საბოლოო_ტიპი) გამოსახულება;
გ. (საწყისი_ტიპი) გამოსახულება;

2.9.23. ტიპის დაყვანას სხვანაირად ეწოდება:
ა. ცხადი გარდაქმნა
ბ. არაცხადი გარდაქმნა
გ. ნაწილობრივი გარდაქმნა

2.9.24. {
int ricxvi1 = 60;
byte ricxvi2;
ricxvi2 = (byte) ricxvi1;
}
კოდის შესრულების შედეგად:
ა. არ მოხდება ინფორმაციის დაკარგვა
ბ. მოხდება ინფორმაციის დაკარგვა
გ. ნაწილობრივ მოხდება ინფორმაციის დაკარგვა

2.9.25. {
char simbolo;
byte baiti = 10;
simbolo = (char) baiti;
}
კოდის შესრულების შედეგად:
ა. ტიპების დაყვანის ოპერაცია არ სრულდება საერთოდ
ბ. ტიპების დაყვანის ოპერაცია სრულდება თავსებად ტიპებს შორის
გ. ტიპების დაყვანის ოპერაცია სრულდება შეუთავსებ ტიპებს შორის

2.9.26. {
double wiladi1 = 10.93, wiladi2 = 3.21;
int mteli;
mteli = (int) (wiladi1 / wiladi2);
}
კოდის შესრულების შედეგად:
ა. int ტიპის გამოსახულება გარდაიქმნება double ტიპად
ბ. double ტიპის გამოსახულება არ გარდაიქმნება int ტიპად
გ. double ტიპის გამოსახულება გარდაიქმნება int ტიპად

2.9.27. ტიპების დაყვანის ოპერაცია სრულდება მაშინ, როცა:
ა. ერთი ტიპი ნაწილობრივ გარდაიქმნება მეორე ტიპად
ბ. ერთი ტიპი ავტომატურად გარდაიქმნება მეორე ტიპად
გ. ერთი ტიპი ავტომატურად არ გარდაიქმნება მეორე ტიპად

2.9.28. როცა ტიპებს შორის სრულდება კუმშვადი გარდაქმნა, მაშინ:
ა. ინფორმაცია შეიძლება დაიკარგოს

ბ. ინფორმაცია შეიძლება არ დაიკარგოს
გ. ინფორმაცია შეიძლება ნაწილობრივ დაიკარგოს

2.9.29. კუმშვადი გარდაქმნის დროს შეიძლება თუ არა ინფორმაციის დაკარგვა:
ა. კი
ბ. არა
გ. არასოდეს

2.9.30. მცურავწერტილიანი მნიშვნელობის გარდაქმნისას მთელ მნიშვნელობად ხდება თუ არა წილადი ნაწილის დაკარგვა:
ა. არა
ბ. კი
გ. არასოდეს

2.9.31. თუ long ტიპის დაყვანისას int ტიპზე, long ტიპის მქონე ცვლადის მნიშვნელობა არ ეტევა int ტიპის დიაპაზონის ფარგლებში, მაშინ:
ა. უმცროსი თანრიგის ბიტები იკარგება
ბ. უფროსი თანრიგის ბიტები იკარგება
გ. უფროსი თანრიგის ბიტები არ იკარგება

2.9.32. თუ long ტიპის დაყვანისას int ტიპზე, long ტიპის მქონე ცვლადის მნიშვნელობა ეტევა int ტიპის დიაპაზონის ფარგლებში, მაშინ:
ა. უფროსი თანრიგის ბიტები არ იკარგება
ბ. უფროსი თანრიგის ბიტები იკარგება
გ. უმცროსი თანრიგის ბიტები იკარგება

2.9.33. როცა ხდება მცურავწერტილიანი მნიშვნელობის დაყვანა მთელ მნიშვნელობაზე, მაშინ:
ა. წილადი ნაწილი არ იკარგება
ბ. წილადი ნაწილი იკარგება
გ. წილადი ნაწილი ნაწილობრივ იკარგება

2.9.34. {

```
        double wiladi = 5.09;
        int mteli;
        mteli = ( int ) wiladi1;
    }
```

კოდის შესრულების შედეგად mteli ცვლადი მიიღებს მნიშვნელობას:
ა. 5.09
ბ. 09
გ. 5

2.9.35. byte ტიპის გარდაქმნისას char ტიპად ინფორმაცია:
ა. ნაწილობრივ იკარგება
ბ. იკარგება

გ. არ იკარგება

2.9.36. არითმეტიკულ გამოსახულებაში თუ ერთ ოპერანდს აქვს decimal ტიპი, მაშინ მეორე ოპერანდი:
ა. ავტომატურად გარდაიქმნება decimal ტიპად, თუ მას არ აქვს double და float ტიპი
ბ. ავტომატურად გარდაიქმნება decimal ტიპად, თუ მას აქვს double და float ტიპი
გ. ავტომატურად არ გარდაიქმნება decimal ტიპად, თუ მას არ აქვს double და float ტიპი

2.9.37. არითმეტიკულ გამოსახულებაში თუ ერთ ოპერანდს აქვს double ტიპი, მაშინ მეორე ოპერანდი:
ა. ავტომატურად არ გარდაიქმნება ამ ტიპად
ბ. ავტომატურად გარდაიქმნება ამ ტიპად
გ. ნაწილობრივ გარდაიქმნება ამ ტიპად

2.9.38. არითმეტიკულ გამოსახულებაში თუ ერთ ოპერანდს აქვს float ტიპი, მაშინ მეორე ოპერანდი:
ა. ავტომატურად გარდაიქმნება ამ ტიპად
ბ. ავტომატურად არ გარდაიქმნება ამ ტიპად
გ. ნაწილობრივ გარდაიქმნება ამ ტიპად

2.9.39. არითმეტიკულ გამოსახულებაში თუ ერთ ოპერანდს აქვს ulong ტიპი, მაშინ პირველი ოპერანდი:
ა. ავტომატურად გარდაიქმნება ამ ტიპად თუ მეორე ოპერანდს არ აქვს sbyte, short, int და long ტიპი
ბ. ავტომატურად არ გარდაიქმნება ამ ტიპად თუ მეორე ოპერანდს არ აქვს sbyte, short, int და long ტიპი
გ. ნაწილობრივ გარდაიქმნება ამ ტიპად თუ მეორე ოპერანდს არ აქვს sbyte, short, int და long ტიპი

2.9.40. არითმეტიკულ გამოსახულებაში თუ ერთ ოპერანდს აქვს uint ტიპი და მეორეს sbyte, short ან int ტიპი, მაშინ:
ა. ორივე ოპერანდი ავტომატურად გარდაიქმნება uint ტიპად
ბ. ორივე ოპერანდი ავტომატურად გარდაიქმნება long ტიპად
გ. ორივე ოპერანდი ავტომატურად გარდაიქმნება int ტიპად

2.9.41. არითმეტიკულ გამოსახულებაში თუ ერთ ოპერანდს აქვს uint ტიპი, მაშინ მეორე ოპერანდი:
ა. ნაწილობრივ გარდაიქმნება ამ ტიპად
ბ. ავტომატურად არ გარდაიქმნება ამ ტიპად
გ. ავტომატურად გარდაიქმნება ამ ტიპად

2.9.42. არითმეტიკულ გამოსახულებაში თუ არ იყო გამოყენებული ტიპების ავტომატური გარდაქმნის არც ერთი წესი, მაშინ:
ა. ორივე ოპერანდი double ტიპად გარდაიქმნება

ბ. ორივე ოპერანდი long ტიპად გარდაიქმნება
გ. ორივე ოპერანდი int ტიპად გარდაიქმნება

2.9.43. არითმეტიკულ გამოსახულებაში:
ა. შეუძლებელია ყველა ტიპის ავტომატურად შეთავსება
ბ. შესაძლებელია ყველა ტიპის ავტომატურად შეთავსება
გ. შესაძლებელია ყველა ტიპის ნაწილობრივ შეთავსება

2.9.44. არითმეტიკულ გამოსახულებაში float ან double ტიპის ავტომატურად გარდაქმნა decimal ტიპად:
ა. შესაძლებელია
ბ. შეუძლებელია
გ. შესაძლებელია ნაწილობრივ

2.9.45. არითმეტიკულ გამოსახულებაში ulong ტიპის ავტომატურად გარდაქმნა სხვა ნიშნიან მთელრიცხვა ტიპთან:
ა. შეუძლებელია
ბ. შესაძლებელია
გ. შესაძლებელია ნაწილობრივ

2.9.46. არითმეტიკულ გამოსახულებაში float ან double ტიპის ავტომატურად გარდაქმნა decimal ტიპად:
ა. ნაწილობრივ შესაძლებელია ტიპის აშკარა გარდაქმნის ოპერაციის გამოყენებით
ბ. შეუძლებელია ტიპის აშკარა გარდაქმნის ოპერაციის გამოყენებით
გ. შესაძლებელია ტიპის აშკარა გარდაქმნის ოპერაციის გამოყენებით

2.9.47. არითმეტიკულ გამოსახულებაში ulong ტიპის ავტომატურად გარდაქმნა სხვა ნიშნიან მთელრიცხვა ტიპთან:
ა. შეუძლებელია ტიპის აშკარა გარდაქმნის ოპერაციის გამოყენებით
ბ. შესაძლებელია ტიპის აშკარა გარდაქმნის ოპერაციის გამოყენებით
გ. ნაწილობრივ შესაძლებელია ტიპის აშკარა გარდაქმნის ოპერაციის გამოყენებით

2.9.48. თუ არითმეტიკულ გამოსახულების გამოთვლისას არ იყო გამოყენებული ტიპების ავტომატური გარდაქმნის წესები, მაშინ:
ა. char, sbyte, byte, ushort და short ტიპები ნაწილობრივ გარდაიქმნება int ტიპად
ბ. char, sbyte, byte, ushort და short ტიპები არ გარდაიქმნება int ტიპად
გ. char, sbyte, byte, ushort და short ტიპები გარდაიქმნება int ტიპად

2.9.49. როცა არითმეტიკულ ოპერაციას ვასრულებთ byte ტიპის ცვლადებზე, მაშინ:
ა. შედეგს ექნება double ტიპი
ბ. შედეგს ექნება int ტიპი
გ. შედეგს ექნება byte ტიპი

2.9.50. როცა არითმეტიკულ ოპერაციას ვასრულებთ char ტიპის ცვლადებზე, მაშინ:
ა. შედეგს ექნება int ტიპი

ბ. შედეგს ექნება byte ტიპი
გ. შედეგს ექნება char ტიპი

თავი 3. მმართველი ოპერატორები

ამორჩევის ოპერატორები. if ოპერატორი

3.1.1. რომელი კატეგორიის მმართველი ოპერატორები არსებობს:
ა. შედარების, არითმეტიკული
ბ. არითმეტიკული, ლოგიკური
გ. ამორჩევის, იტერაციული, გადასვლის

3.1.2. რა მნიშვნელობებს იღებს პირობა if გამოსახულებაში:
ა. true და false
ბ. bool და 0
გ. -1 და 1

3.1.3. როგორია if ოპერატორის სინტაქსი:
ა. if (პირობა) ოპერატორი_1;
else ოპერატორი_2;
ბ. if else ოპერატორი_1;
(პირობა) ოპერატორი_2;
გ. else (პირობა) ოპერატორი_1;
if ოპერატორი_2;

3.1.4. როდის გამოიყენება ჩადგმული if ოპერატორი:
ა. ჩადგმული if ოპერატორი გამოიყენება მასივების დამატებით შესამოწმებლად
ბ. ჩადგმული if ოპერატორი გამოიყენება კლასების დამატებით შესამოწმებლად
გ. ჩადგმული if ოპერატორი გამოიყენება მონაცემების დამატებით შესამოწმებლად
მას შემდეგ, რაც წინა if ოპერატორის პირობამ მიიღო true ან false მნიშვნელობა

3.1.5.
```
{
    int ricxvi = 5, shedegi;
    if ( ricxvi == 5 ) shedegi = 10;
        else shedegi = 25;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:
ა. 25
ბ. 10
გ. 15

3.1.6. if ოპერატორის პირობა არის გამოსახულება, რომელსაც აქვს:
ა. სიმბოლური ტიპი
ბ. მთელი ტიპი
გ. ლოგიკური ტიპი

3.1.7. ჩადგმულია if ოპერატორი, რომელიც:
ა. მოთავსებულია სხვა if ოპერატორში
ბ. არაა მოთავსებული სხვა if ოპერატორში
გ. მოთავსებულია სხვა for ოპერატორში

3.1.8. ჩადგმულ if ოპერატორში else ნაწილი:
ა. ეკუთვნის უახლოეს for ოპერატორს
ბ. არ ეკუთვნის უახლოეს if ოპერატორს
გ. ეკუთვნის უახლოეს if ოპერატორს

3.1.9.
```
{
        int ricxvi1 = 5, ricxvi2 = 10, shedegi;
        if ( ricxvi1 > 5 ) shedegi = 50;
                else if ( ricxvi2 <= 10 ) shedegi = 100;
}
```
ამ კოდის შესრულების შედეგად shedegi ცვლადი მიიღებს მნიშვნელობას:
ა. 50
ბ. 100
გ. 150

ხილვადობის უბანი

3.2.1. ლოკალურია ცვლადი, რომელიც გამოცხადებულია:
ა. მეთოდის შიგნით
ბ. მეთოდის გარეთ
გ. მხოლოდ if ოპერატორში

3.2.2. გლობალურია ცვლადი, რომელიც გამოცხადებულია:
ა. მეთოდის შიგნით
ბ. მეთოდის გარეთ
გ. for ოპერატორში

3.2.3. ხილვადობის უბანში გამოცხადებული ცვლადი:
ა. მიუწვდომელია ამ უბნის გარეთ განსაზღვრული კოდისთვის
ბ. მისაწვდომია ამ უბნის გარეთ განსაზღვრული კოდისთვის
გ. მისაწვდომია მხოლოდ if ოპერატორში

3.2.4. პროგრამის ბლოკი მოთავსებულია:
ა. ფიგურულ ფრჩხილებში
ბ. მრგვალ ფრჩხილებში
გ. კვადრატულ ფრჩხილებში

3.2.5. ერთ ბლოკში რამდენიმე ოპერატორი იმ შემთხვევაში უნდა მოვათავსოთ, როცა:

ა. საჭიროა მათ შორის ციფრული კავშირის დამყარება
ბ. საჭიროა მათ შორის ფიზიკური კავშირის დამყარება
გ. საჭიროა მათ შორის ლოგიკური კავშირის დამყარება

3.2.6. თუ ცვლადი გამოცხადებულია მეთოდის დასაწყისში, მაშინ:
ა. მასთან მიმართვა შესაძლებელი იქნება მის გამოცხადებამდე
ბ. მასთან მიმართვა შესაძლებელი იქნება მხოლოდ if ოპერატორში
გ. მასთან მიმართვა შესაძლებელი იქნება მთელი კოდის ფარგლებში

3.2.7. ხილვადობის უბანში განსაზღვრული ცვლადი ხილვადობის უბნიდან გამოსვლისას:
ა. კარგავს თავის მნიშვნელობას
ბ. ინახავს თავის მნიშვნელობას
გ. ორჯერ ზრდის თავის მნიშვნელობას

3.2.8. მეთოდის შიგნით გამოცხადებული ცვლადი:
ა. ინახავს თავის მნიშვნელობას ამ მეთოდის გამოძახებებს შორის
ბ. არ ინახავს თავის მნიშვნელობას ამ მეთოდის გამოძახებებს შორის
გ. ორჯერ ზრდის თავის მნიშვნელობას

3.2.9. ცვლადის სიცოცხლის დრო:
ა. შეზღუდულია მასივის საზღვრებით
ბ. არ არის შეზღუდული მისიხილვადობის უბნით
გ. შეზღუდულია მისიხილვადობის უბნით

3.2.10. ხილვადობის უბანი:
ა. არ შეიძლება იყოს ჩადგმული
ბ. შეიძლება იყოს ჩადგმული
გ. შეზღუდულია მასივის საზღვრებით

3.2.11. გარე ხილვადობის უბანში გამოცხადებული ობიექტები და ცვლადები:
ა. ხილული იქნება შიდა ხილვადობის უბანში
ბ. ხილული არ იქნება შიდა ხილვადობის უბანში
გ. შეზღუდულია მასივის საზღვრებით

3.2.12. შიდა ხილვადობის უბანში გამოცხადებული ობიექტები და ცვლადები:
ა. ხილული არ იქნება გარე ხილვადობის უბანში
ბ. ხილული იქნება გარე ხილვადობის უბანში
გ. შეზღუდულია მასივის საზღვრებით

3.2.13. გარე და ჩადგმულ ბლოკებში გამოცხადებულ ცვლადებს:
ა. შეიძლება ჰქონდეს ერთნაირი სახელები
ბ. არ შეიძლება ჰქონდეს ერთნაირი სახელები
გ. აქვთ შეზღუდული ზომები

3.2.14. გლობალური ცვლადები:
ა. შეზღუდული ზომების მქონეა
ბ. არ მოქმედებენ ყველა მეთოდის შიგნით
გ. მოქმედებენ ყველა მეთოდის შიგნით

3.2.15. გლობალური ცვლადები:
ა. არ კარგავენ თავიანთ მნიშვნელობებს მეთოდიდან გამოსვლისას
ბ. კარგავენ თავიანთ მნიშვნელობებს მეთოდიდან გამოსვლისას
გ. ფორმდება როგორც მასივი

3.2.16. მეთოდის პარამეტრები ჩართული არიან თუ არა ამავე მეთოდის ხილვადობის უბანში:
ა. არა
ბ. კი
გ. ნაწილობრივ

3.2.17. ხილვადობის უბანში გამოცხადებული ცვლადი ხილულია თუ უხილავი ამ უბნის გარეთ განსაზღვრული კოდისთვის:
ა. უხილავია
ბ. ხილულია
გ. ნაწილობრივ არის ხილული

3.2.18. პროგრამული კოდის ბლოკის შექმნისას იქმნება თუ არა ხილვადობის ახალი უბანი:
ა. კი
ბ. არა
გ. ნაწილობრივ

3.2.19. შეიძლება თუ არა პროგრამული ბლოკის შიგნით ცვლადების გამოცხადება ნებისმიერ ადგილას:
ა. კი
ბ. არა
გ. ხანდახან

3.2.20. შეიძლება თუ არა ცვლადთან მიმართვა მის გამოცხადებამდე:
ა. კი
ბ. არა
გ. ზოგჯერ

3.2.21. ინახავს თუ არა ცვლადი თავის მნიშვნელობას ხილვადობის უბნიდან გამოსვლისას:
ა. არა
ბ. კი
გ. ზოგჯერ

3.2.22. ინახავს თუ არა მეთოდის შიგნით გამოცხადებული ცვლადი თავის მნიშვნელობას ამ მეთოდის გამოძახებებს შორის:
ა. ნაწილობრივ
ბ. კი
გ. არა

3.2.23. არის თუ არა ცვლადის სიცოცხლის დრო შეზღუდული მისი ხილვადობის უბნით:
ა. ნაწილობრივ
ბ. არა
გ. კი

3.2.24. შეიძლება თუ არა ჰქონდეთ ერთნაირი სახელები გარე და ჩადგმულ ბლოკებში გამოცხადებულ ცვლადებს:
ა. არა
ბ. კი
გ. ზოგჯერ

ამორჩევის ოპერატორები. switch ოპერატორი

3.3.1. შიდა და გარე switch ოპერატორების case განშტოებების მუდმივებს:
ა. შეიძლება ერთნაირი მნიშვნელობები ჰქონდეთ
ბ. არ შეიძლება ერთნაირი მნიშვნელობები ჰქონდეთ
გ. მხოლოდ ლოგიკური მნიშვნელობები უნდა ჰქონდეთ

3.3.2. როგორი ტიპი შეიძლება ჰქონდეს switch ოპერატორის გამოსახულებას:
ა. char, short, double
ბ. char, byte, int
გ. float, byte, int

3.3.3. switch ოპერატორის case განშტოებების მუდმივებს:
ა. უნდა ჰქონდეს გამოსახულების ტიპთან თავსებადი ტიპი
ბ. არ უნდა ჰქონდეს გამოსახულების ტიპთან თავსებადი ტიპი
გ. ხან უნდა ჰქონდეს გამოსახულების ტიპთან თავსებადი ტიპი, ხან არა

3.3.4. switch ოპერატორის case განშტოებების მუდმივებს შეიძლება თუ არა, რომ ერთნაირი მნიშვნელობები ჰქონდეს :
ა. ხან შეიძლება, ხან არა
ბ. შეიძლება
გ. არ შეიძლება

3.3.5. როდის შესრულდება switch ოპერატორის default განშტოების ოპერატორები:
ა. როცა case განშტოების არც ერთი მუდმივა არ შეესაბამება გამოსახულების მნიშვნელობას

ბ. როცა case განშტოების ერთ-ერთი მუდმივა შეესაბამება გამოსახულების მნიშვნელობას
გ. როცა case განშტოების ყველა მუდმივა შეესაბამება გამოსახულების მნიშვნელობას

3.3.6. break ოპერატორი მართვას გადასცემს:
ა. მომდევნო case განშტოებას
ბ. switch ოპერატორის შემდეგ მოთავსებულ ოპერატორს
გ. default განშტოებას

3.3.7. switch ოპერატორში აუცილებელია თუ არა default განშტოების მითითება:
ა. არა
ბ. კი
გ. ნაწილობრივ

3.3.8. switch ოპერატორში რა ფუნქციას ასრულებს break ოპერატორი:
ა. იგი მართვას გადასცემს switch ოპერატორის წინ მოთავსებულ ოპერატორს
ბ. იგი მართვას გადასცემს switch ოპერატორის შუაში მოთავსებულ ოპერატორს
გ. იგი მართვას გადასცემს switch ოპერატორის შემდეგ მოთავსებულ ოპერატორს

3.3.9. შეიძლება თუ არა მართვა გადაეცეს ერთი case განშტოებიდან მეორე case განშტოებას:
ა. არა
ბ. კი
გ. ნაწილობრივ

3.3.10. ერთ case განშტოებასთან დაკავშირებულმა ოპერატორების მიმდევრობამ:
ა. მართვა უნდა გადასცეს მომდევნო case განშტოების ოპერატორებს
ბ. მართვა შეიძლება გადასცეს მომდევნო case განშტოების ოპერატორებს
გ. მართვა არ უნდა გადასცეს მომდევნო case განშტოების ოპერატორებს

3.3.11. შეიძლება თუ არა, რომ switch ოპერატორის ერთი ან მეტი განშტოება დაკავშირებული იყოს ოპერატორების ერთსა და იმავე მიმდევრობასთან:
ა. არა
ბ. კი
გ. ნაწილობრივ

3.3.12. switch ოპერატორში ჩავარდნა არის შემთხვევა, როცა:
ა. switch ოპერატორს არც ერთი განშტოება არაა დაკავშირებული ოპერატორების ერთსა და იმავე მიმდევრობასთან
ბ. switch ოპერატორს მხოლოდ ერთი განშტოებაა დაკავშირებული ოპერატორების ერთსა და იმავე მიმდევრობასთან
გ. switch ოპერატორს ორი ან მეტი განშტოება დაკავშირებული ოპერატორების ერთსა და იმავე მიმდევრობასთან

3.3.13. შეიძლება თუ არა, რომ switch ოპერატორი იყოს გარე switch ოპერატორის ნაწილი:
ა. არა
ბ. კი
გ. ნაწილობრივ

იტერაციის ოპერატორები. for ოპერატორი

3.4.1. რისთვის გამოიყენება for ოპერატორი:
ა. პროგრამის შესრულების შესაწყვეტად
ბ. განშტოების შესასრულებლად
გ. ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად

3.4.2. როგორია for ოპერატორის სინტაქსი:
ა. for (ციკლის ტანი; პირობა; იტერაცია) { ინიციალიზაცია }
ბ. for (ინიციალიზაცია; პირობა; იტერაცია) { ციკლის ტანი }
გ. for (იტერაცია; პირობა; ციკლის ტანი) { ინიციალიზაცია }

3.4.3. როგორია for ოპერატორის “ინიციალიზაცია” შემადგენლის დანიშნულება:
ა. მასში მმართველ ცვლადს საწყისი მნიშვნელობა ენიჭება
ბ. მასში მმართველ ცვლადს საწყისი მნიშვნელობა არ ენიჭება
გ. მოწმდება “პირობა” შემადგენელი

3.4.4. როგორია for ოპერატორის “პირობა” შემადგენლის დანიშნულება:
ა. მოწმდება მასში მითითებული გამოსახულება. თუ იგი იღებს false მნიშვნელობას, მაშინ ციკლის ტანი მეორდება, წინააღმდეგ შემთხვევაში ციკლი მთავრდება
ბ. მოწმდება მასში მითითებული გამოსახულება. თუ იგი იღებს true მნიშვნელობას, მაშინ ციკლის ტანი მეორდება, წინააღმდეგ შემთხვევაში ციკლი მთავრდება
გ. მასში მმართველ ცვლადს საწყისი მნიშვნელობა ენიჭება

3.4.5. როგორია for ოპერატორის “იტერაცია” შემადგენლის დანიშნულება:
ა. მასში მმართველ ცვლადს საწყისი მნიშვნელობა ენიჭება
ბ. მოწმდება მასში მითითებული გამოსახულება. თუ იგი იღებს true მნიშვნელობას, მაშინ ციკლის ტანი მეორდება, წინააღმდეგ შემთხვევაში ციკლი მთავრდება
გ. ესაა გამოსახულება, რომელიც განსაზღვრავს სიდიდეს (ბიჯს), რომლითაც იცვლება მმართველი ცვლადის მნიშვნელობა ციკლის ყოველი გამეორების წინ

3.4.6. for ოპერატორის ინიციალიზატორი არის გამოსახულება, რომელიც:
ა. ციკლის დაწყების წინ გამოითვლება
ბ. ციკლის დამთავრებისას გამოითვლება
გ. არასოდეს არ გამოითვლება

3.4.7. for ოპერატორის ინიციალიზატორში სრულდება:
ა. გლობალური ცვლადის გამოცხადება
ბ. ციკლის მმართველი ცვლადის ინიციალიზაცია

გ. სტატიკური ცვლადი ინიციალიზაცია

3.4.8. for ოპერატორის პირობა არის ლოგიკური გამოსახულება, რომელიც იღებს:
ა. true ან int მნიშვნელობას
ბ. true ან false მნიშვნელობას
გ. double ან false მნიშვნელობას

3.4.9. for ოპერატორის პირობის პირობა გამოითვლება:
ა. პროგრამის დასაწყისში
ბ. ციკლის ყოველი იტერაციის შემდეგ
გ. ციკლის ყოველი იტერაციის წინ

3.4.10. for ციკლის გამეორება ხდება მანამ, სანამ პირობა იღებს:
ა. false მნიშვნელობას
ბ. int მნიშვნელობას
გ. true მნიშვნელობას

3.4.11. for ციკლის დამთავრება ხდება მაშინ, როცა პირობა იღებს:
ა. false მნიშვნელობას
ბ. int მნიშვნელობას
გ. true მნიშვნელობას

3.4.12. for ციკლის იტერატორი არის გამოსახულება, რომელიც:
ა. არ ცვლის მმართველი ცვლადის მნიშვნელობას
ბ. ზრდის ან ამცირებს მმართველი ცვლადის მნიშვნელობას
გ. მხოლოდ ამცირებს მმართველი ცვლადის მნიშვნელობას

3.4.13. ციკლის მმართველი ცვლადი შეიძლება თუ არა ამცირებდეს თავის მნიშვნელობას:
ა. კი
ბ. არა
გ. ნაწილობრივ

3.4.14. for ციკლის ტანი:
ა. აუცილებლად უნდა შესრულდეს
ბ. შეიძლება არ შესრულდეს
გ. აუცილებლად უნდა შესრულდეს ორჯერ

3.4.15. for ოპერატორში ინიციალიზატორის მითითება:
ა. აუცილებელია მასივის გამოცხადებისას
ბ. აუცილებელია
გ. არ არის აუცილებელი

3.4.16. for ოპერატორში იტერატორის მითითება:
ა. აუცილებელია მასივის გამოცხადებისას

ბ. აუცილებელია
გ. არ არის აუცილებელი

3.4.17. for ოპერატორში პირობის მითითება:
ა. აუცილებელია მასივის გამოცხადებისას
ბ. აუცილებელია
გ. არ არის აუცილებელი

3.4.18. for ოპერატორში ტანის მითითება:
ა. აუცილებელია მასივის გამოცხადებისას
ბ. აუცილებელია
გ. არ არის აუცილებელი

3.4.19. for ოპერატორში დასაშვებია:
ა. რამდენიმე მმართველი ცვლადის მითითება
ბ. მხოლოდ ერთი მმართველი ცვლადის მითითება
გ. მხოლოდ ორი მმართველი ცვლადის მითითება

3.4.20. რომელი მსჯელობაა სწორი:
ა. for ციკლის შესაწყვეტად დაუშვებელია break ოპერატორის გამოყენება
ბ. for ციკლის შესაწყვეტად შეგვიძლია break ოპერატორის გამოყენება
გ. for ციკლის შესაწყვეტად შეგვიძლია case ოპერატორის გამოყენება

3.4.21. დასაშვებია თუ არა for ოპერატორის საინიციალიზაციო ნაწილში მმართველი ცვლადის გამოცხადება:
ა. ნაწილობრივ
ბ. არა
გ. კი

3.4.22. for ოპერატორის ინიციალიზატორში გამოცხადებული ცვლადის გამოყენება შეიძლება:
ა. მხოლოდ ამ ციკლის ტანში
ბ. ამ ციკლის ტანის გარეთ
გ. ნებისმიერი მეთოდის შიგნით

3.4.23. თუ ციკლის ტანში ერთი ოპერატორია მითითებული, მაშინ:
ა. მრგვალი ფრჩხილების მითითება აუცილებელია
ბ. ფიგურული ფრჩხილების მითითება აუცილებელია
გ. ფიგურული ფრჩხილების მითითება არ არის აუცილებელი

იტერაციის ოპერატორები. while ოპერატორი

3.5.1. როგორია while ოპერატორის სინტაქსი:
ა. while (პირობა) { ციკლის კოდი }
ბ. while (ციკლის კოდი) { პირობა }

გ. while ( ) { ციკლის კოდი }

3.5.2. while ოპერატორის პირობა არის ლოგიკური გამოსახულება, რომელიც იღებს:
ა. true ან int მნიშვნელობას
ბ. true ან false მნიშვნელობას
გ. double ან false მნიშვნელობას

3.5.3. while ციკლის გამეორება ხდება მანამ, სანამ პირობა იღებს:
ა. false მნიშვნელობას
ბ. int მნიშვნელობას
გ. true მნიშვნელობას

3.5.4. while ციკლის დამთავრება ხდება მაშინ, როცა პირობა იღებს:
ა. false მნიშვნელობას
ბ. int მნიშვნელობას
გ. true მნიშვნელობას

3.5.5. while ციკლის ტანი:
ა. აუცილებლად უნდა შესრულდეს
ბ. შეიძლება არ შესრულდეს
გ. აუცილებლად უნდა შესრულდეს ერთხელ

იტერაციის ოპერატორები. do-while ოპერატორი

3.6.1. როგორია do-while ოპერატორის სინტაქსი:
ა. do { ციკლის კოდი } while (პირობა)
ბ. do { პირობა } while (ციკლის კოდი)
გ. do { } while { ციკლის კოდი }

3.6.2. do-while ოპერატორის პირობა არის ლოგიკური გამოსახულება, რომელიც იღებს:
ა. true ან int მნიშვნელობას
ბ. true ან false მნიშვნელობას
გ. double ან false მნიშვნელობას

3.6.3. do-while ციკლის გამეორება ხდება მანამ, სანამ პირობა იღებს:
ა. false მნიშვნელობას
ბ. int მნიშვნელობას
გ. true მნიშვნელობას

3.6.4. do-while ციკლის დამთავრება ხდება მაშინ, როცა პირობა იღებს:
ა. false მნიშვნელობას
ბ. int მნიშვნელობას
გ. true მნიშვნელობას

3.6.5. do-while ოპერატორის ტანი:
ა. შეიძლება არ შესრულდეს
ბ. აუცილებლად უნდა შესრულდეს
გ. არასოდეს არ შესრულდება

გადასვლის ოპერატორები. break ოპერატორი

3.7.1. რისთვის გამოიყენება break ოპერატორი:
ა. იგი გვაძლევს ციკლის იძულებით შეწყვეტისა და ციკლიდან გამოსვლის საშუალებას
ბ. იგი გვაძლევს ციკლის შესრულების გაგრძელებისა და ციკლიდან გამოსვლის საშუალებას
გ. იგი არ გვაძლევს ციკლის იძულებით შეწყვეტისა და ციკლიდან გამოსვლის საშუალებას

3.7.2. შეიძლება თუ არა break ოპერატორის გამოყენება while ციკლის შიგნით:
ა. არა
ბ. კი
გ. ნაწილობრივ

3.7.3. შეიძლება თუ არა break ოპერატორის გამოყენება do-while ციკლის შიგნით:
ა. ნაწილობრივ
ბ. არა
გ. კი

3.7.4. break ოპერატორის გამოყენება შეიძლება:
ა. მხოლოდ for ციკლის შიგნით
ბ. ნებისმიერი ციკლის შიგნით
გ. მხოლოდ while ციკლის შიგნით

3.7.5. break ოპერატორი გამოიყენება:
ა. პროგრამის დასაწყისში
ბ. ციკლის შესრულების გასაგრძელებლად
გ. ციკლის შესრულების შესაწყვეტად

3.7.6. break ოპერატორის შესრულების შედეგად:
ა. ციკლის დანარჩენი ოპერატორები არ შესრულდება
ბ. ციკლის დანარჩენი ოპერატორები მაინც შესრულდება
გ. ციკლის დანარჩენი ოპერატორები ხან შესრულდება, ხან არა

3.7.7. თუ break ოპერატორი გამოიყენება ჩადგმული ციკლის შიგნით, მაშინ იგი წყვეტს:
ა. ორივე ციკლის შესრულებას
ბ. გარე ციკლის შესრულებას

გ. შიდა ციკლის შესრულებას

გადასვლის ოპერატორები. continue ოპერატორი

3.8.1. რისთვის გამოიყენება continue ოპერატორი:
ა. იგი გამოიყენება ციკლის მომდევნო იტერაციაზე იძულებით გადასასვლელად
ბ. იგი გამოიყენება ციკლიდან გამოსასვლელად
გ. იგი გამოიყენება ციკლის შესაწყვეტად

3.8.2. შეიძლება თუ არა continue ოპერატორის გამოყენება while ოპერატორში:
ა. არა
ბ. კი
გ. ნაწილობრივ

3.8.3. შეიძლება თუ არა continue ოპერატორის გამოყენება do-while ოპერატორში:
ა. ნაწილობრივ
ბ. არა
გ. კი

3.8.4. continue ოპერატორის შესრულების შედეგად:
ა. ციკლის დანარჩენი ოპერატორები მაინც შესრულდება
ბ. ციკლის დანარჩენი ოპერატორები არ შესრულდება
გ. ციკლის დანარჩენი ოპერატორები ხან შესრულდება, ხან არა

3.8.5. continue ოპერატორი ახდენს:
ა. ციკლის იძულებით შეწყვეტას
ბ. მომდევნო იტერაციაზე იძულებით გადასვლას
გ. მომდევნო ოპერატორზე გადასვლას

3.8.6. continue ოპერატორის გამოყენება შეიძლება:
ა. მხოლოდ while ციკლის შიგნით
ბ. მხოლოდ for ციკლის შიგნით
გ. ნებისმიერი ციკლის შიგნით

გადასვლის ოპერატორები. goto ოპერატორი

3.9.1. goto ოპერატორი არის:
ა. უპირობო გადასვლის ოპერატორი
ბ. პირობითი გადასვლის ოპერატორი
გ. მრავალგანშტოებიანი ოპერატორი

3.9.2. ჭდე არის:
ა. წილადი რიცხვი
ბ. იდენტიფიკატორი, რომელსაც ორი წერტილი მოსდევს
გ. ლოგიკური გამოსახულება

3.9.3. როგორია goto ოპერატორის სინტაქსი:
ა. goto კლასი;
ბ. goto მასივი;
გ. goto ჭდე;

ტერნარული ოპერატორი

3.10.1. რამდენი ოპერანდი აქვს ტერნარულ ოპერატორს:
ა. ერთი
ბ. სამი
გ. ორი

3.10.2. ტერნარული ოპერატორის პირველი ოპერანდი არის:
ა. ლოგიკური გამოსახულება
ბ. მთელი ტიპის გამოსახულება
გ. სტრიქონი

3.10.3. თუ ტერნარული ოპერატორის პირობის გამოსახულების მნიშვნელობაა true, მაშინ:
ა. გაიცემა მესამე ოპერანდის მნიშვნელობა
ბ. გაიცემა მეორე ოპერანდის მნიშვნელობა
გ. არაფერი არ გაიცემა

3.10.4. ტერნარული ოპერატორის მეორე და მესამე ოპერანდების ტიპები:
ა. სულერთია როგორი იქნება
ბ. სხვადასხვა უნდა იყოს
გ. ერთნაირი უნდა იყოს

3.10.5. თუ ტერნარული ოპერატორის პირობის გამოსახულების მნიშვნელობაა false, მაშინ:
ა. გაიცემა მესამე ოპერანდის მნიშვნელობა
ბ. გაიცემა მეორე ოპერანდის მნიშვნელობა
გ. არაფერი არ გაიცემა

3.10.6. როგორია ტერნარული ოპრატორის სინტაქსი:
ა. გამოსახულება1 : გამოსახულება2 ? გამოსახულება3;
ბ. გამოსახულება1 ? გამოსახულება2 : გამოსახულება3;
გ. გამოსახულება1 ! გამოსახულება2 ? გამოსახულება3;

თავი 4. მასივები, სტრიქონები და ბიტობრივი ოპერაციები

ერთგანზომილებიანი მასივები

4.1.1. მასივი არის:

ა. ობიექტების ერთობლიობა
ბ. მუდმივა
გ. სიმბოლო

4.1.2. მასივი გამოიყენება:
ა. ერთი ცვლადის ან ობიექტის შესანახად
ბ. რამდენიმე ცვლადის ან ობიექტის შესანახად
გ. არაფრის შესანახად არ გამოიყენება

4.1.3. მასივის ელემენტებთან მიმართვისათვის უნდა მივუთითოთ:
ა. მასივის სახელი
ბ. ელემენტის ინდექსი
გ. მასივის სახელი და ელემენტის ინდექსი

4.1.4. რა არის მასივი:
ა. მასივი არის სხვადასხვა ტიპის მქონე ობიექტების ერთობლიობა
ბ. მასივი არის ერთი ტიპის მქონე ობიექტების ერთობლიობა
გ. მასივი არის მხოლოდ სტრიქონების ერთობლიობა

4.1.5. როგორია ერთგანზომილებიანი მასივის სინტაქსი:
ა. ტიპი[ ] მასივის_სახელი = new ტიპი [ზომა];
ბ. [ ] მასივის_სახელი = new ტიპი [ზომა];
გ. ტიპი[ზომა] მასივის_სახელი = new ტიპი [];

4.1.6. რომელი ეტაპებისგან შედგება მასივის შექმნის პროცესი:
ა. მასივის სახელის გამოცხადება
ბ. პირველ ეტაპზე ხდება მასივის სახელის გამოცხადება. მეორე ეტაპზე მასივისათვის მეხსიერება გამოიყოფა და მასივის ცვლადს მიენიჭება მიმართვა მეხსიერების ამ უბანზე
გ. მასივისათვის მეხსიერების გამოყოფა და მასივის ცვლადისთვის მიმართვის მიენიჭება მეხსიერების ამ უბანზე

4.1.7. როგორია ერთგანზომილებიანი მასივის ინიციალიზაციის სინტაქსი:
ა. მასივის_სახელი = new ტიპი [] { სიდიდე1, სიდიდე2, ..., სიდიდეn };
ბ. ტიპი[n] მასივის_სახელი = new ტიპი [] { სიდიდე1, სიდიდე2, ..., სიდიდეn };
გ. ტიპი[ ] მასივის_სახელი = new ტიპი [] { სიდიდე1, სიდიდე2, ..., სიდიდეn };

4.1.8. ერთგანზომილებიან მასივში კონკრეტული ელემენტის მდებარეობა განისაზღვრება:
ა. ერთი ინდექსით
ბ. ორი ინდექსით
გ. სამი ინდექსით

4.1.9. მასივს მეხსიერება new ოპერატორის საშუალებით გამოეყოფა:
ა. დინამიურად

ბ. სტატიკურად
გ. არ გამოეყოფა

4.1.10. რამდენ ბაიტს იკავებს მასივი int [ ] masivi = new int[15]; :
ა. 15
ბ. 60
გ. 30

4.1.11. რამდენ ბაიტს იკავებს მასივი double [ ] masivi = new double[10]; :
ა. 10
ბ. 80
გ. 8

4.1.12. მასივის პირველი ელემენტის ინდექსია:
ა. -1
ბ. 1
გ. 0

4.1.13. მასივის მესამე ელემენტის ინდექსია:
ა. 2
ბ. 3
გ. -2

4.1.14. თუ მასივი 5 ელემენტისაგან შედგება, მაშინ მისი ინდექსები მოთავსებული იქნება დიაპაზონში:
ა. 1÷5
ბ. 0÷4
გ. 0÷5

4.1.15. masivi[1] აღნიშნავს მასივის რიგით:
ა. მესამე ელემენტს
ბ. პირველ ელემენტს
გ. მეორე ელემენტს

4.1.16. double [ ] masivi = new double[3]; მასივის ელემენტებს აქვთ:
ა. ლოგიკური ტიპი
ბ. მთელი ტიპი
გ. წილადი ტიპი

4.1.17. თუ მასივის ინდექსის მნიშვნელობა გასცდა დასაშვებ საზღვრებს, მაშინ:
ა. პროგრამა ავარიულად დაამთავრებს მუშაობას
ბ. პროგრამა გააგრძელებს მუშაობას
გ. პროგრამა არც ავარიულად დაამთავრდება და არ გააგრძელებს მუშაობას

ორგანზომილებიანი მასივები

4.2.1. რამდენ ბაიტს იკავებს მასივი int[,] cxrili = new int[2,3]; :

ა. 6

ბ. 24

გ. 4

4.2.2. რამდენ ბაიტს იკავებს მასივი double[,] cxrili = new double[2,3]; :

ა. 6

ბ. 48

გ. 8

4.2.3. ორგანზომილებიან მასივში პირველი ინდექსი მიუთითებს:

ა. სტრიქონების რაოდენობას

ბ. სვეტების რაოდენობას

გ. სტრიქონებისა და სვეტების საერთო რაოდენობას

4.2.4. ორგანზომილებიან მასივში მეორე ინდექსი მიუთითებს:

ა. სტრიქონების რაოდენობას

ბ. სვეტების რაოდენობას

გ. სტრიქონებისა და სვეტების საერთო რაოდენობას

4.2.5. ორგანზომილებიანი cxrili მასივის მეორე სტრიქონისა და მესამე სვეტის გადაკვეთაზე მყოფი ელემენტი შემდეგნაირად აღინიშნება:

ა. cxrili[2,3]

ბ. cxrili[3,2]

გ. cxrili[1,2]

4.2.6. x = cxrili[2,4]; მინიჭების ოპერატორის შესრულების შედეგად x ცვლადს მიენიჭება:

ა. cxrili მასივის მეორე სვეტისა და მეოთხე სტრიქონის გადაკვეთაზე მყოფი ელემენტი

ბ. cxrili მასივის მესამე სვეტისა და მეხუთე სტრიქონის გადაკვეთაზე მყოფი ელემენტი

გ. cxrili მასივის მეოთხე სვეტისა და მეორე სტრიქონის გადაკვეთაზე მყოფი ელემენტი

4.2.7. ორგანზომილებიან მასივში სტრიქონის პირველი ელემენტის ინდექსია:

ა. -1

ბ. 1

გ. 0

4.2.8. ორგანზომილებიან მასივში სვეტის პირველი ელემენტის ინდექსია:

ა. 0

ბ. 1

გ. -1

4.2.9. ორგანზომილებიან მასივში კონკრეტული ელემენტის მდებარეობა განისაზღვრება:
ა. სამი ინდექსით
ბ. ერთი ინდექსით
გ. ორი ინდექსით

გაუსწორებელი მასივები

4.3.1. სწორკუთხაა მასივი, რომელშიც:
ა. სტრიქონების სიგრძე ერთნაირია
ბ. სტრიქონების სიგრძე სხვადასხვაა
გ. სტრიქონების სიგრძე ნულის ტოლია

4.3.2. გაუსწორებელია მასივი, რომელშიც:
ა. სტრიქონების სიგრძე ნულის ტოლია
ბ. სტრიქონების სიგრძე ერთნაირია
გ. სტრიქონების სიგრძე სხვადასხვაა

4.3.3. რა არის გაუსწორებელი მასივი:
ა. გაუსწორებელი მასივი არის შიდა მასივი, რომელიც შედგება სხვადასხვა სიგრძის გარე მასივებისაგან
ბ. გაუსწორებელი მასივი არის გარე მასივი, რომელიც შედგება სხვადასხვა სიგრძის შიდა მასივებისაგან
გ. გაუსწორებელი მასივი არის მხოლოდ გარე მასივი

Length თვისება

4.4.1. Length თვისება აქვს:
ა. მხოლოდ მთელრიცხვა მასივს
ბ. ნებისმიერ მასივს
გ. მხოლოდ სტრიქონებს

4.4.2. Length თვისება ინახავს ინფორმაციას:
ა. მასივში ელემენტების საერთო რაოდენობის შესახებ
ბ. მასივის თითოეული ელემენტის სიგრძის შესახებ
გ. მასივში მთელრიცხვა ელემენტების რაოდენობის შესახებ

4.4.3. გვაქვს 7-ელემენტიანი მასივი და შევსებულია მისი პირველი 3 ელემენტი. Length თვისების მნიშვნელობა იქნება:
ა. 3
ბ. 7
გ. 10

4.4.4. {

```
                int[ ][ ] masivi = new int[3][ ];
                masivi[0] = new int[4];
                masivi[1] = new int[1];
                masivi[2] = new int[2];
                int raodenoba = masivi.Length;
        }
```

ამ კოდის შესრულების შედეგად raodenoba ცვლადში ჩაიწერება:

ა. 7
ბ. 4
გ. 3

4.4.5.
```
        {
                int[ ][ ] masivi = new int[3][ ];
                masivi[0] = new int[4];
                masivi[1] = new int[1];
                masivi[2] = new int[2];
                int raodenoba = masivi.Length;
        }
```

ამ კოდის შესრულების შედეგად raodenoba ცვლადში ჩაიწერება:

ა. შიდა მასივების რაოდენობა
ბ. სვეტების რაოდენობა
გ. პირველ სტრიქონში ელემენტების რაოდენობა

4.4.6.
```
        {
                int[ ][ ] masivi = new int[3][ ];
                masivi[0] = new int[4];
                masivi[1] = new int[1];
                masivi[2] = new int[2];
                int raodenoba = masivi[0].Length;
        }
```

ამ კოდის შესრულების შედეგად raodenoba ცვლადში ჩაიწერება:

ა. masivi მასივის სვეტების რაოდენობა
ბ. masivi მასივის სტრიქონების რაოდენობა
გ. masivi მასივის პირველი სტრიქონის ელემენტების რაოდენობა

4.4.7.
```
        {
                int[ ][ ] masivi = new int[3][ ];
                masivi[0] = new int[4];
                masivi[1] = new int[1];
                masivi[2] = new int[2];
                int raodenoba = masivi[0].Length;
        }
```

ამ კოდის შესრულების შედეგად raodenoba ცვლადში ჩაიწერება:

ა. 4
ბ. 7

გ. 3

foreach ოპერატორი

4.5.1. როგორია foreach ოპერატორის სინტაქსი:
ა. foreach (ტიპი ცვლადის_სახელი in კოლექცია) { ციკლის ტანი }
ბ. foreach (ციკლის ტანი ცვლადის_სახელი in კოლექცია) { ტიპი }
გ. foreach (ტიპი ციკლის ტანი) { ცვლადის_სახელი in კოლექცია }

4.5.2. foreach ოპერატორში იტერაციული ცვლადის ტიპი:
ა. ხანდახან უნდა ემთხვეოდეს კოლექციის ტიპს
ბ. არ უნდა ემთხვეოდეს კოლექციის ტიპს
გ. უნდა ემთხვეოდეს კოლექციის ტიპს

4.5.3. foreach ოპერატორში იტერაციულ ცვლადს:
ა. კოლექციის მნიშვნელობები ენიჭება უკუ მიმდევრობით
ბ. მიმდევრობით ენიჭება კოლექციის მნიშვნელობები დაწყებული პირველი ელემენტიდან
გ. არ ენიჭება კოლექციის მნიშვნელობები

4.5.4. foreach ციკლში:
ა. დაუშვებელია იტერაციული ცვლადის შეცვლა
ბ. დასაშვებია იტერაციული ცვლადის შეცვლა
გ. იტერაციულ ცვლადს ენიჭება მხოლოდ მუდმივა

ბიტობრივი ოპერატორები

4.6.1. ბიტობრივი ოპერატორების ოპერანდებია:
ა. სტრიქონები
ბ. მთელი რიცხვები
გ. წილადი რიცხვები

4.6.2. როგორ ოპერაციებს ეწოდება ბიტობრივი:
ა. ბიტობრივი ეწოდება ისეთ ოპერაციებს, რომლებიც გამოიყენება მთელი რიცხვების ბიტების შესამოწმებლად, დასაყენებლად და ძვრისათვის
ბ. ბიტობრივი ეწოდება ისეთ ოპერაციებს, რომლებიც გამოიყენება მთელი რიცხვების შესაკრებად
გ. ბიტობრივი ეწოდება ისეთ ოპერაციებს, რომლებიც გამოიყენება მთელი რიცხვების გამოსაკლებად

4.6.3. რომელია ბიტობრივი ოპერატორები:
ა. +, -, *
ბ. &, |, ^, ~
გ. &, |, %, /

4.6.4. {

```
int b1 = 10, b2 = 12, shedegi;
shedegi = b1 & b2;
```

}

ამ კოდის შესრულების შედეგად shedegi ცვლადს მიენიჭება:

ა. -2
ბ. 22
გ. 8

4.6.5. თუ & ბიტობრივი ოპერატორის ორივე ოპერანდის მნიშვნელობაა 1, მაშინ შედეგი იქნება:

ა. -1
ბ. 0
გ. 1

4.6.6. თუ | ბიტობრივი ოპერატორის ორივე ოპერანდის მნიშვნელობაა 1, მაშინ შედეგი იქნება:

ა. 0
ბ. 1
გ. -1

4.6.7. თუ & ბიტობრივი ოპერატორის ერთ-ერთი ოპერანდია 0, მაშინ შედეგი იქნება:

ა. -1
ბ. 0
გ. 1

4.6.8. თუ | ბიტობრივი ოპერატორის ერთ-ერთი ოპერანდია 1, მაშინ შედეგი იქნება:

ა. -1
ბ. 0
გ. 1

4.6.9. თუ ~ ბიტობრივი ოპერატორის ოპერანდია 0, მაშინ შედეგი იქნება:

ა. -1
ბ. 0
გ. 1

4.6.10. თუ ~ ბიტობრივი ოპერატორის ოპერანდია 1, მაშინ შედეგი იქნება:

ა. -1
ბ. 0
გ. 1

4.6.11. თუ ^ ოპერატორის ორივე ოპერანდი ერთნაირია, მაშინ შედეგი იქნება:

ა. 0

ბ 1
გ. -1

4.6.12. თუ ^ ოპერატორის ოპერანდები განსხვავებულია, მაშინ შედეგი იქნება:
ა. 0
ბ 1
გ. -1

4.6.13. რიცხვის მარცხნივ ძვრის დროს:
ა. უფროსი თანრიგები იკარგება
ბ. უმცროსი თანრიგები იკარგება
გ. არც ერთი თანრიგი არ იკარგება

4.6.14. რიცხვის მარჯვნივ ძვრის დროს:
ა. უფროსი თანრიგები იკარგება
ბ. უმცროსი თანრიგები იკარგება
გ. არც ერთი თანრიგი არ იკარგება

4.6.15. რიცხვის მარჯვნივ ძვრის დროს:
ა. უმცროსი თანრიგები ნულებით ივსება
ბ. უფროსი თანრიგები ერთებით ივსება
გ. უფროსი თანრიგები ნულებით ივსება

4.6.16. რიცხვის მარცხნივ ძვრის დროს:
ა. უმცროსი თანრიგები ნულებით ივსება
ბ. უმცროსი თანრიგები ერთებით ივსება
გ. უფროსი თანრიგები ნულებით ივსება

4.6.17. ნიშნიანი რიცხვის მარჯვნივ ძვრის დროს:
ა. ნიშნის თანრიგი ინახება
ბ. ნიშნის თანრიგი ერთით ივსება
გ. ნიშნის თანრიგი ნულით ივსება

ჩამოთვლები

4.7.1. ჩამოთვლადი ტიპის განსაზღვრისათვის გამოიყენება:
ა. enum სიტყვა
ბ. public სიტყვა
გ. static სიტყვა

4.7.2. რომელი მსჯელობაა სწორი:
ა. ჩამოთვლები წარმოადგენენ სახელდებული წილადი მუდმივების ნაკრებს, რომლებიც განსაზღვრავენ ცვლადის ყველა დასაშვებ მნიშვნელობას მოცემული ტიპისათვის
ბ. ჩამოთვლები წარმოადგენენ სახელდებული მთელრიცხვა მუდმივების ნაკრებს,

რომლებიც განსაზღვრავენ ცვლადის ყველა დასაშვებ მნიშვნელობას მოცემული ტიპისათვის

გ. ჩამოთვლები წარმოადგენენ სახელდებული სტრიქონების მუდმივების ნაკრებს, რომლებიც განსაზღვრავენ ცვლადის ყველა დასაშვებ მნიშვნელობას მოცემული ტიპისათვის

4.7.3. ჩამოთვლაში თითოეულ მუდმივას შეესაბამება:

ა. სტრიქონული მნიშვნელობა

ბ. წილადი მნიშვნელობა

გ. მთელრიცხვა მნიშვნელობა

4.7.4. ჩამოთვლაში მთელრიცხვა მნიშვნელობა თითოეული მუდმივასათვის:

ა. მეტია წინა მუდმივას მნიშვნელობაზე

ბ. ნაკლებია წინა მუდმივას მნიშვნელობაზე

გ. ტოლია წინა მუდმივას მნიშვნელობის

4.7.5. ჩამოთვლაში მთელრიცხვა მნიშვნელობა თითოეული მუდმივასათვის:

ა. ტოლია უკანა მუდმივას მნიშვნელობის

ბ. მეტია უკანა მუდმივას მნიშვნელობაზე

გ. ნაკლებია უკანა მუდმივას მნიშვნელობაზე

4.7.6. ჩამოთვლაში პირველი მუდმივას მნიშვნელობა არის:

ა. -1

ბ. 0

გ. 1

4.7.7. ჩამოთვლის ელემენტებთან მიმართვისათვის:

ა. უნდა მივუთითოთ ჩამოთვლის სახელი, წერტილი და ელემენტის სახელი

ბ. მხოლოდ ჩამოთვლის სახელი

გ. მხოლოდ ჩამოთვლის ელემენტის სახელი

4.7.8. ჩამოთვლებში ინიციალიზატორის საშუალებით:

ა. ნაწილობრივ შეიძლება მივუთითოთ მუდმივას მნიშვნელობა

ბ. არ შეიძლება მივუთითოთ მუდმივას მნიშვნელობა

გ. შეიძლება მივუთითოთ მუდმივას მნიშვნელობა

4.7.9. ჩამოთვლებში ინიციალიზატორის მიერ განსაზღვრული მნიშვნელობა:

ა. ნაკლები უნდა იყოს წინა მუდმივას მნიშვნელობაზე

ბ. მეტი უნდა იყოს წინა მუდმივას მნიშვნელობაზე

გ. ტოლი უნდა იყოს წინა მუდმივას მნიშვნელობის

4.7.10. ჩამოთვლებში ინიციალიზატორის მიერ განსაზღვრული მნიშვნელობა:

ა. ნაკლები უნდა იყოს უკანა მუდმივას მნიშვნელობაზე

ბ. მეტი უნდა იყოს უკანა მუდმივას მნიშვნელობაზე

გ. ტოლი უნდა იყოს უკანა მუდმივას მნიშვნელობის

თავი 5. კლასები, ინკაფსულაცია, მეთოდები

კლასების გამოცხადება

5.1.1. რა არის კლასი:
ა. კლასი არის ობიექტების მასივი
ბ. კლასი არის ობიექტზე ორიენტირებული პროგრამირების ძირითადი სტრუქტურა, რომელიც გამოიყენება მასივების შესაქმნელად
გ. კლასი არის ობიექტზე ორიენტირებული პროგრამირების ძირითადი სტრუქტურა, რომელიც გამოიყენება ობიექტების შესაქმნელად

5.1.2. ობიექტზე ორიენტირებული პროგრამირების პრინციპებია:
ა. ინკაფსულაცია, პოლიმორფიზმი, მემკვიდრეობითობა
ბ. მხოლოდ ინკაფსულაცია
გ. მხოლოდ პოლიმორფიზმი

5.1.3. ინკაფსულაცია არის:
ა. კლასი, რომელიც ობიექტს აღწერს
ბ. დაპროგრამების მექანიზმი, რომელიც აერთიანებს პროგრამის კოდსა და მონაცემებს, აგრეთვე, გამიჯნავს მათ სხვა პროგრამების მხრიდან მიმართვებისაგან
გ. ობიექტი, რომელიც წარმოადგენს კლასის ეგზემპლარს

5.1.4. ობიექტი არის:
ა. მასივი
ბ. მუდმივა
გ. კლასის კონკრეტული რეალიზაცია კომპიუტერის მეხსიერებაში

5.1.5. ობიექტი წარმოადგენს:
ა. სტრიქონების მასივს
ბ. კლასის ეგზემპლარს
გ. სიმბოლოების მასივს

5.1.6. ინკაფსულაცია არის დაპროგრამების მექანიზმი, რომელიც:
ა. მუშაობს მხოლოდ მთელ რიცხვებთან
ბ. კლასის წევრებს იცავს არასწორი გამოყენებისაგან
გ. მუშაობს მხოლოდ სტრიქონებთან

5.1.7. დახურული ცვლადი აღიწერება სიტყვით:
ა. private
ბ. public
გ. static

5.1.8. ღია ცვლადი აღიწერება სიტყვით:
ა. public

ბ. private
გ. static

5.1.9. რომელი მოდიფიკატორები გამოიყენება კლასის წევრებთან მიმართვისათვის:
ა. params და static
ბ. out და ref
გ. public, private და protected

5.1.10. როგორია private მოდიფიკატორის დანიშნულება:
ა. იგი მიუთითებს, რომ კლასის წევრებთან მიმართვა შეუძლიათ მხოლოდ ამავე კლასის წევრებს
ბ. იგი მიუთითებს, რომ კლასის წევრებთან მიმართვა შეუძლიათ მხოლოდ სხვა კლასის წევრებს
გ. იგი მიუთითებს, რომ კლასის წევრებთან მიმართვა არ შეუძლიათ არც ერთი კლასის წევრებს

5.1.11. როგორია public მოდიფიკატორის დანიშნულება:
ა. იგი მიუთითებს, რომ კლასის წევრებთან მიმართვა შეუძლიათ მხოლოდ ამავე კლასის წევრებს
ბ. იგი მიუთითებს, რომ კლასის წევრებთან მიმართვა შეუძლიათ სხვა კლასის წევრებს
გ. იგი მიუთითებს, რომ კლასის წევრებთან მიმართვა არ შეუძლიათ არც ერთი კლასის წევრებს

5.1.12. კლასის პრივატულ წევრთან მიმართვა შეუძლიათ:
ა. სხვა კლასში განსაზღვრულ მეთოდებს
ბ. ამავე კლასში განსაზღვრულ მეთოდებს
გ. არც ერთ კლასში განსაზღვრულ მეთოდებს

5.1.13. კლასის საერთოწვდომის წევრთან მიმართვა შეუძლიათ:
ა. ნებისმიერ კლასში განსაზღვრულ მეთოდებს
ბ. ამავე კლასში განსაზღვრულ მეთოდებს
გ. არც ერთ კლასში განსაზღვრულ მეთოდებს

5.1.14. კლასი გამოიყენება:
ა. სტრიქონების აღსაწერად
ბ. ლოგიკური მონაცემების შესაქმნელად
გ. ობიექტების აღსაწერად და შესაქმნელად

5.1.15. კლასში შემავალ მეთოდებსა და ცვლადებს:
ა. მთელრიცხვა მასივი ეწოდება
ბ. სტრიქონები ეწოდებათ
გ. კლასის წევრები ეწოდებათ

5.1.16. კლასის გამოცხადება იწყება სიტყვით:
ა. static
ბ. private
გ. class

5.1.17. იქმნება თუ არა ობიექტი class სიტყვის გამოყენებით კლასის გამოცხადებისას:
ა. კი
ბ. არა
გ. ხანდახან

ინკაფსულაცია

5.2.1. კლასი ორ ძირითად ფუნქციას ასრულებს:
ა. მონაცემებს აკავშირებს კოდთან და განსაზღვრავს მხოლოდ სტრიქონებს
ბ. მონაცემებს აკავშირებს კოდთან და უზრუნველყოფს კლასის წევრებთან მიმართვის საშუალებებს
გ. უზრუნველყოფს მხოლოდ კლასის წევრებთან მიმართვის საშუალებებს

5.2.2. კლასის წევრებთან მიმართვა სრულდება შემდეგი მოდიფიკატორის გამოყენებით:
ა. private, public string
ბ. public, private, protected, internal
გ. protected, static, public

5.2.3. თუ კლასის წევრთან მიმართვის მოდიფიკატორი არ არის მითითებული, მაშინ იგულისხმება:
ა. private
ბ. public
გ. protected

5.2.4. class სიტყვის გამოყენებით კლასის გამოცხადებისას ხდება:
ა. სტრიქონის შექმნა
ბ. ობიექტის შექმნა
გ. კლასის აღწერა

5.2.5. ობიექტის შესაქმნელად გამოიყენება:
ა. class სიტყვა
ბ. new სიტყვა
გ. protected მოდიფიკატორი

5.2.6. ობიექტის ცვლადებთან მიმართვისათვის გამოიყენება:
ა. private მოდიფიკატორი
ბ. new ოპერატორი
გ. . (წერტილი) ოპერატორი

5.2.7. ობიექტის მეთოდთან მიმართვისათვის გამოიყენება:
ა. . (წერტილი) ოპერატორი
ბ. protected მოდიფიკატორი
გ. private მოდიფიკატორი

5.2.8. რომელი ოპერატორი გამოიყენება ობიექტის ცვლადებთან მიმართვისათვის:
ა. წერტილი (.)
ბ. გამრავლება (*)
გ. შეკრება (+)

5.2.9. შეიძლება თუ არა წერტილის გამოყენება ობიექტის მეთოდებთან მიმართვისათვის:
ა. არა
ბ. კი
გ. ხანდახან

5.2.10. რა არის მიმართვა:
ა. მიმართვა არის მეხსიერების იმ უბნის მისამართი, რომელიც ობიექტს არ გამოეყო new ოპერატორის მიერ
ბ. მიმართვა არ არის მეხსიერების იმ უბნის მისამართი, რომელიც ობიექტს გამოეყო new ოპერატორის მიერ
გ. მიმართვა არის მეხსიერების იმ უბნის მისამართი, რომელიც ობიექტს გამოეყო new ოპერატორის მიერ

5.2.11. თუ ორ ობიექტს ერთნაირი ტიპი აქვს, მაშინ მათი ცვლადები:
ა. ერთმანეთზე დამოკიდებულია
ბ. ერთმანეთისაგან დამოუკიდებელია
გ. მხოლოდ int ტიპისაა

5.2.12. ChemiKlasi obieqti = new ChemiKlasi();
ოპერატორი როგორ შეიძლება ჩავწეროთ ორი ოპერატორის სახით:
ა. ChemiKlasi obieqti;
obieqti = new ChemiKlasi();
ბ. obieqti = new ChemiKlasi();
ChemiKlasi obieqti;
გ. ChemiKlasi obieqti;
ChemiKlasi = new ChemiKlasi();

5.2.13. რა მოქმედებები შესრულდება
ChemiKlasi obieqti = new ChemiKlasi();
ოპერატორის შესრულების დროს:
ა. შესრულდება ერთი მოქმედება - ხდება ChemiKlasi ტიპის obieqti მიმართვითი ცვლადის გამოცხადება
ბ. შესრულდება ორი მოქმედება: 1. ხდება ChemiKlasi ტიპის obieqti მიმართვითი

ცვლადის გამოცხადება; 2. new ოპერატორი ქმნის ობიექტს. მასზე მიმართვა მიენიჭება obieqti ცვლადს

გ. შესრულდება ერთი მოქმედება - new ოპერატორი ქმნის ობიექტს. მასზე მიმართვა მიენიჭება obieqti ცვლადს

5.2.14. მიმართვითი ტიპის ცვლადი ინახავს:

ა. ცვლადის მისამართს

ბ. ცვლადის მნიშვნელობას

გ. ობიექტის მისამართს

5.2.15. ჩვეულებრივი ტიპის ცვლადი ინახავს:

ა. ობიექტის მისამართს

ბ. ცვლადის მნიშვნელობას

გ. ცვლადის მისამართს

5.2.16. როგორ სრულდება ობიექტთან მიმართვა:

ა. მიმართვის მიხედვით

ბ. მნიშვნელობის მიხედვით

გ. მასივის საშუალებით

5.2.17. როგორ ტიპებს ეკუთვნის კლასები:

ა. ჩვეულებრივ

ბ. მიმართვით

გ. ლოგიკურ

5.2.18. კლასები გამოიყენება:

ა. მიმართვითი ტიპის მქონე ობიექტების გამოცხადებისათვის

ბ. ჩვეულებრივი ტიპის მქონე ცვლადების გამოცხადებისათვის

გ. მთელრიცხვა მასივების გამოცხადებისათვის

5.2.19. {

```
klasi1 obieqti1 = new klasi1();
klasi2 obieqti2 = new klasi2();
obieqti1 = obieqti2;
}
```

ამ კოდის შესრულების შედეგად:

ა. obieqti1 ცვლადს ენიჭება obieqti2 კლასზე მიმართვა

ბ. obieqti2 ცვლადს ენიჭება იმ ობიექტზე მიმართვა, რომელსაც obieqti1 ცვლადი მიმართავს

გ. obieqti1 ცვლადს ენიჭება იმ ობიექტზე მიმართვა, რომელსაც obieqti2 ცვლადი მიმართავს

მეთოდები

5.3.1. მეთოდი არის:

ა. პროგრამა
ბ. სტრიქონი
გ. კლასის ობიექტი

5.3.2. ერთი მეთოდის გამოძახება შეიძლება:
ა. ერთხელ
ბ. ბევრჯერ
გ. ორჯერ

5.3.3. შეიძლება თუ არა გამოვიყენოთ დარეზერვებული სიტყვები მეთოდების სახელებად:
ა. ზოგჯერ
ბ. კი
გ. არა

5.3.4. რა შემთხვევაში ეთითება მეთოდისათვის void ტიპი:
ა. როცა მეთოდი მნიშვნელობას გასცემს
ბ. როცა მეთოდი მნიშვნელობას არ გასცემს
გ. როცა მეთოდი მთელ რიცხვს გასცემს

5.3.5. როგორია მეთოდის სინტაქსი:
ა. მიმართვის_მოდიფიკატორი ტიპი მეთოდის_სახელი (მეთოდის კოდი) { პარამეტრების_სია }
ბ. მიმართვის_მოდიფიკატორი ტიპი მეთოდის_სახელი (პარამეტრების_სია) { მეთოდის კოდი }
გ. მეთოდის_სახელი (პარამეტრების_სია) { მეთოდის კოდი }

5.3.6. რომელი მსჯელობაა არასწორი:
ა. მეთოდმა შეიძლება დააბრუნოს მხოლოდ სტრიქონი
ბ. მეთოდმა შეიძლება არ დააბრუნოს მნიშვნელობა
გ. მეთოდმა შეიძლება დააბრუნოს მნიშვნელობა

5.3.7. რომელი მსჯელობაა არასწორი:
ა. მეთოდს შეიძლება ჰქონდეს რამდენიმე პარამეტრი
ბ. მეთოდს შეიძლება არ ჰქონდეს პარამეტრები
გ. მეთოდს შეიძლება ჰქონდეს მხოლოდ მთელი ტიპის პარამეტრები

5.3.8. void სიტყვა მიუთითებს იმას, რომ მეთოდი:
ა. აბრუნებს მთელ რიცხვს
ბ. აბრუნებს წილადს
გ. არ აბრუნებს შედეგს

5.3.9. როცა მეთოდი მუშაობას ამთავრებს, მაშინ მართვა გადაეცემა გამომძახებელ პროგრამის:
ა. იმ სტრიქონს, საიდანაც მოხდა მეთოდის გამოძახება

ბ. დასაწყისს
გ. დასასრულს

5.3.10. როცა მეთოდი მუშაობას ამთავრებს, მაშინ გამომძახებელი პროგრამის შესრულება გაგრძელდება:
ა. პროგრამის დასაწყისიდან
ბ. იმ სტრიქონიდან, რომელიც მოსდევს მეთოდის გამომძახებელ სტრიქონს
გ. პროგრამის დასასრულიდან

5.3.11. მეთოდიდან მართვის დაბრუნება ხდება მაშინ, როცა გვხვდება:
ა. static სიტყვა
ბ. მეთოდის დამხურავი ფიგურული ფრჩხილი ან return ოპერატორი
გ. public მოდიფიკატორი

5.3.12. რომელი მსჯელობაა მცდარი:
ა. როცა მეთოდი მნიშვნელობას არ აბრუნებს, მაშინ ეთითება return 7;
ბ. როცა მეთოდი მნიშვნელობას არ აბრუნებს, მაშინ ეთითება return ოპერატორი მნიშვნელობის გარეშე
გ. როცა მეთოდი მნიშვნელობას არ აბრუნებს, მაშინ არ ეთითება return ოპერატორი

5.3.13. მეთოდისათვის გადასაცემ მნიშვნელობას ეწოდება:
ა. წილადი
ბ. პარამეტრი
გ. არგუმენტი

5.3.14. ცვლადს მეთოდის შიგნით, რომელიც იღებს არგუმენტის მნიშვნელობას ეწოდება:
ა. სიმბოლო
ბ. არგუმენტი
გ. პარამეტრი

5.3.15. პარამეტრი მოქმედებს:
ა. მხოლოდ თავისი მეთოდის ხილვადობის უბანში
ბ. სხვა მეთოდის მეთოდის ხილვადობის უბანში
გ. მთელს პროგრამაში

5.3.16. თუ მეთოდი არ აბრუნებს შედეგს, მაშინ რომელი ტიპი უნდა მივუთითოთ:
ა. int
ბ. void
გ. bool

5.3.17. შეიძლება თუ არა მეთოდის გამოცხადება პარამეტრების გარეშე:
ა. ნაწილობრივ
ბ. არა
გ. კი

5.3.18. რომელი მსჯელობაა სწორი:
ა. მეთოდიდან მართვის დაბრუნება ხდება მაშინ, როცა არ გვხვდება მეთოდის დამხურავი ფიგურული ფრჩხილი ან return ოპერატორი
ბ. მეთოდიდან მართვის დაბრუნება ხდება მაშინ, როცა გვხვდება მეთოდის დამხურავი ფიგურული ფრჩხილი ან return ოპერატორი
გ. მეთოდიდან მართვის დაბრუნება ხდება მაშინ, როცა გვხვდება მეთოდის გამხსნელი ფიგურული ფრჩხილი ან continue ოპერატორი

5.3.19. return ოპერატორის რა ფორმები არსებობს:
ა. არსებობს ორი ფორმა: პირველი ფორმა გამოიყენება მეთოდებში, რომლებიც მნიშვნელობებს არ აბრუნებენ, მეორე კი მეთოდებში, რომლებიც მნიშვნელობებს აბრუნებენ
ბ. არსებობს ერთი ფორმა, რომელიც გამოიყენება მეთოდებში, რომლებიც მნიშვნელობებს არ აბრუნებენ
გ. არსებობს ერთი ფორმა, რომელიც გამოიყენება მეთოდებში, რომლებიც მნიშვნელობებს აბრუნებენ

5.3.20. შესრულდება თუ არა return ოპერატორის შემდეგ მოთავსებული ოპერატორები:
ა. კი
ბ. არა
გ. ხანდახან

5.3.21. იმყოფება თუ არა პარამეტრი თავისი მეთოდის ხილვადობის უბანში:
ა. კი
ბ. არა
გ. ზოგჯერ

კონსტრუქტორი

5.4.1. კონსტრუქტორი არის მეთოდი, რომლის დანიშნულებაა:
ა. ობიექტის ცვლადების ინიციალიზება
ბ. მასივის შევსება რიცხვებით
გ. სტრიქონების ფორმირება

5.4.2. რა არის კონსტრუქტორი:
ა. კონსტრუქტორი არის მეთოდი, რომელიც ახდენს ობიექტის ინიციალიზებას მისი შექმნისას. მისი სახელი ემთხვევა კლასის სახელს
ბ. კონსტრუქტორი არის მეთოდი, რომელიც ახდენს ობიექტის ინიციალიზებას მისი შექმნისას. მისი სახელი არ ემთხვევა კლასის სახელს
გ. კონსტრუქტორი არის მეთოდი, რომელიც არ ახდენს ობიექტის ინიციალიზებას მისი შექმნისას. მისი სახელი არ ემთხვევა კლასის სახელს

5.4.3. როგორია კონსტრუქტორის სინტაქსი:
ა. კლასის_სახელი {კონსტრუქტორის კოდი}

ბ. კონსტრუქტორის კოდი () { კლასის_სახელი }
გ. კლასის_სახელი() {კონსტრუქტორის კოდი}

5.4.4. აბრუნებს თუ არა კონსტრუქტორი შედეგს:
ა. არა
ბ. კი
გ. ხანდახან

5.4.5. აქვს თუ არა ყველა კლასს კონსტრუქტორი:
ა. არა
ბ. კი
გ. ხანდახან

5.4.6. რისთვის გამოიყენება კონსტრუქტორი:
ა. ობიექტის ცვლადების შესაცვლელად
ბ. ობიექტის ცვლადების წასაშლელად
გ. ობიექტის ცვლადებისათვის საწყისი მნიშვნელობების მისანიჭებლად

5.4.7. როდის გამოიძახება ავტომატური კონსტრუქტორი:
ა. როცა კლასში კონსტრუქტორი არ არის განსაზღვრული
ბ. როცა კლასში კონსტრუქტორი არის განსაზღვრული
გ. როცა კლასში მეთოდები არ არის განსაზღვრული

5.4.8. კონსტრუქტორის მიერ ობიექტის ინიციალიზება სრულდება:
ა. მეხსიერებიდან ობიექტის წაშლისას
ბ. ობიექტის შექმნისას
გ. ობიექტის შეცვლისას სხვა ობიექტით

5.4.9. კონსტრუქტორის სახელი ემთხვევა:
ა. ცვლადის სახელს
ბ. მასივის სახელს
გ. მისი კლასის სახელს

5.4.10. კონსტრუქტორში დასაბრუნებელი მნიშვნელობის ტიპი:
ა. არ ეთითება
ბ. ეთითება
გ. ეთითება ლოგიკური ტიპი

5.4.11. ყველა კლასს:
ა. აქვს კონსტრუქტორი იმ შემთხვევაში თუ იგი განსაზღვრულია კლასში
ბ. არ აქვს კონსტრუქიტორი
გ. აქვს კონსტრუქტორი

5.4.12. ავტომატური კონსტრუქტორი:
ა. ყველა ცვლადს ერთეულოვან მნიშვნელობას ანიჭებს

ბ. ყველა ცვლადს ნულოვან მნიშვნელობას ანიჭებს
გ. ცვლადებს არანაირ მნიშვნელობას არ ანიჭებს

5.4.13. ავტომატური კონსტრუქტორი იმ შემთხვევაში გამოიძახება, როცა:
ა. კლასში კონსტრუქტორი არის განსაზღვრული
ბ. კლასში კონსტრუქტორი არ არის განსაზღვრული
გ. კლასში ორი კონსტრუქტორია განსაზღვრული

5.4.14. თუ კლასში კონსტრუქტორი აშკარადაა განსაზღვრული, მაშინ შესრულდება:
ა. ორივე კონსტრუქიტორის გამოძახება
ბ. ავტომატური კონსტრუქტორის გამოძახება
გ. მისი გამოძახება

დესტრუქტორი

5.5.1. "ნაგვის შეგროვება" არის ოპერაცია, როცა ხდება:
ა. ობიექტის მიერ დაკავებული მეხსიერების გათავისუფლება
ბ. მეხსიერების დაკავება ობიექტის მიერ
გ. მეხსიერების დაკავებს სტრიქონის მიერ

5.5.2. "ნაგვის შეგროვება" სრულდება:
ა. პროგრამის მუშაობის დასაწყისში
ბ. პროგრამის მუშაობის დროს
გ. მაშინ, როცა არ არსებობს მეხსიერებიდან წასაშლელი ობიექტები

5.5.3. დესტრუქტორი არის მეთოდი, რომელიც გამოიყენება:
ა. ობიექტის ცვლადების ინიციალიზებისათვის
ბ. მეხსიერებაში ობიექტის მისათავსებლად
გ. მეხსიერებიდან ობიექტის წასაშლელად

5.5.4. ~ (ტილდა) სიმბოლო ეთითება:
ა. დესტრუქტორის სახელის წინ
ბ. კონსტრუქტორის სახელის წინ
გ. კონსტრუქტორის სახელის შემდეგ

5.5.5. როგორია დესტრუქტორის სინტაქსი:
ა. ~კლასის_სახელი {დესტრუქტორის კოდი}
ბ. ~კლასის_სახელი() {დესტრუქტორის კოდი}
გ. ~ (დესტრუქტორის კოდი) { კლასის_სახელი }

5.5.6. აბრუნებს თუ არა დესტრუქტორი შედეგს:
ა. კი
ბ. არა
გ. ხანდახან

5.5.7. როდის გამოიძახება დესტრუქტორი:
ა. “ნაგვის შეგროვების” შემდეგ
ბ. “ნაგვის შეგროვების” შუაში
გ. უშუალოდ “ნაგვის შეგროვების” წინ

this სიტყვა

5.6.1. მეთოდს გამოძახებისას გადაეცემა:
ა. ობიექტის სახელი
ბ. გამომძახებელ ობიექტზე მიმართვა
გ. ლოგიკური მნიშვნელობა

5.6.2. this მიმართვა მიუთითებს:
ა. ობიექტის ცვლადზე
ბ. გამომძახებელ ობიექტზე
გ. სტრიქონზე

5.6.3. როცა პარამეტრის სახელი ემთხვევა ლოგიკური ცვლადის სახელს, მაშინ:
ა. ლოკალური ცვლადი მალავს ლოგიკურ ცვლადს
ბ. ლოკალური ცვლადი მალავს პარამეტრს
გ. პარამეტრი მალავს ლოკალურ ცვლადს

5.6.4. როცა პარამეტრი მალავს ლოკალურ ცვლადს, მაშინ:
ა. ლოკალურ ცვლადთან მიმართვა შესაძლებელია this მიმართვის გარეშე
ბ. პარამეტრთან მიმართვა შესაძლებელია this მიმართვის გამოყენებით
გ. ლოკალურ ცვლადთან მიმართვა შესაძლებელია this მიმართვის გამოყენებით

შემთხვევითი რიცხვების გენერატორი

5.7.1. შემთხვევით რიცხვებთან სამუშაოდ გამოიყენება:
ა. System.Random კლასი
ბ. System.IO კლასი
გ. System.Data კლასი

5.7.2. int Random.Next(int ცვლადი) მეთოდი:
ა. გასცემს უარყოფით შემთხვევით მთელ რიცხვს, რომელიც ნაკლებია მითითებული ცვლადის მნიშვნელობაზე
ბ. გასცემს არაუარყოფით შემთხვევით მთელ რიცხვს, რომელიც ნაკლებია მითითებული ცვლადის მნიშვნელობაზე
გ. გასცემს არაუარყოფით შემთხვევით წილად რიცხვს, რომელიც ნაკლებია მითითებული ცვლადის მნიშვნელობაზე

5.7.3. double Random.NextDouble() მეთოდი:
ა. გასცემს შემთხვევით წილად რიცხვს, რომლის მნიშვნელობა მოთავსებულია -1.0-სა და 1.0-ს შორის

ბ. გასცემს შემთხვევით მთელ რიცხვს, რომლის მნიშვნელობა მოთავსებულია 0.0-სა და 1.0-ს შორის
გ. გასცემს შემთხვევით წილად რიცხვს, რომლის მნიშვნელობა მოთავსებულია 0.0-სა და 1.0-ს შორის

5.7.4. void Random.NextBytes(byte[ ] მასივი) მეთოდი:
ა. მითითებულ მასივს შეავსებს შემთხვევითი რიცხვებით, რომელთა ტიპია int
ბ. მითითებულ მასივს შეავსებს შემთხვევითი რიცხვებით, რომელთა ტიპია byte
გ. მითითებულ მასივს შეავსებს შემთხვევითი რიცხვებით, რომელთა ტიპია float

5.7.5. ricxvi = obieqti.Next(5); ოპერატორის შესრულების შედეგად ricxvi ცვლადს მიენიჭება:
ა. 5-ზე მეტი მნიშვნელობა
ბ. 5-ზე ნაკლები მნიშვნელობა
გ. 5-ის ტოლი მნიშვნელობა

თავი 6. მეთოდები უფრო დაწვრილებით. პოლიმორფიზმი

მეთოდისთვის ობიექტის გადაცემა

6.1.1. მეთოდს პარამეტრებად შეგვიძლია გადავცეთ:
ა. მხოლოდ ობიექტები
ბ. ჩვეულებრივი ტიპის მნიშვნელობები და ობიექტები
გ. მხოლოდ ჩვეულებრივი ტიპის მნიშვნელობები

6.1.2. obieqti1.Metodi(obieqti2); გამოსახულებაში:
ა. გამომძახებელი ობიექტია obieqti1
ბ. გამომძახებელი ობიექტია obieqti2
გ. გამომძახებელი ობიექტია Metodi

მეთოდისთვის არგუმენტების გადაცემის ხერხები

6.2.1. მეთოდს არგუმენტები შეგვიძლია გადავცეთ:
ა. მნიშვნელობის მიხედვით ან მიმართვის მიხედვით
ბ. მხოლოდ მნიშვნელობის მიხედვით
გ. მხოლოდ მიმართვის მიხედვით

6.2.2. როცა მეთოდს არგუმენტი გადაეცემა მნიშვნელობის მიხედვით, მაშინ:
ა. არგუმენტის მისამართი ენიჭება პარამეტრს
ბ. არგუმენტის მნიშვნელობის ასლი ენიჭება მეთოდის პარამეტრს
გ. პარამეტრს ენიჭება არგუმენტის მნიშვნელობაც და მისამართიც

6.2.3. როცა მეთოდს არგუმენტი გადაეცემა მიმართვის მიხედვით, მაშინ:
ა. არგუმენტის მნიშვნელობა ენიჭება მეთოდის პარამეტრს
ბ. მეთოდის პარამეტრს გადაეცემა არგუმენტის მისამართი

გ. მეთოდის პარამეტრს გადაეცემა არგუმენტის მნიშვნელობაც და მისამართიც

6.2.4. როცა მეთოდს არგუმენტი გადაეცემა მიმართვის მიხედვით, მაშინ:
ა. პარამეტრის მნიშვნელობის შეცვლისას იცვლება არგუმენტის მისამართი
ბ. პარამეტრის მნიშვნელობის შეცვლისას არ იცვლება არგუმენტის მნიშვნელობა
გ. პარამეტრის მნიშვნელობის შეცვლისას იცვლება არგუმენტის მნიშვნელობა

6.2.5. მეთოდის პარამეტრის მნიშვნელობის შეცვლისას იცვლება არგუმენტის მნიშვნელობა იმიტომ, რომ:
ა. პარამეტრიც და არგუმენტიც მეხსიერების ერთსა და იმავე უბანს მიმართავენ
ბ. პარამეტრიც და არგუმენტიც მეხსიერების სხვადასხვა უბანს მიმართავენ
გ. პარამეტრიც და არგუმენტიც ლოგიკურ მონაცემებს მიმართავენ

6.2.6. მეთოდისთვის ჩვეულებრივი ტიპის მქონე მნიშვნელობის გადაცემისას გამოიყენება:
ა. გადაცემა მნიშვნელობის მიხედვით
ბ. გადაცემა მიმართვის მიხედვით
გ. ლოგიკური მონაცემების გადაცემა

6.2.7. როცა მეთოდს არგუმენტი გადაეცემა მნიშვნელობის მიხედვით, მაშინ:
ა. პარამეტრის მნიშვნელობის შეცვლისას იცვლება არგუმენტის მნიშვნელობა
ბ. პარამეტრის მნიშვნელობის შეცვლისას არ იცვლება არგუმენტის მნიშვნელობა
გ. პარამეტრის მნიშვნელობის შეცვლისას იცვლება მხოლოდ სტრიქონები

ref და out მოდიფიკატორები

6.3.1. ref მოდიფიკატორის გამოყენება იწვევს:
ა. არგუმენტისათვის პარამეტრის მინიჭებას
ბ. პარამეტრისათვის არგუმენტის მნიშვნელობის მინიჭებას
გ. პარამეტრისათვის არგუმენტის მისამართის მინიჭებას

6.3.2. ref მოდიფიკატორი ეთითება:
ა. მეთოდის გამოცხადებისას და მეთოდის გამოძახებისას
ბ. მხოლოდ მეთოდის გამოცხადებისას
გ. მხოლოდ მეთოდის გამოძახებისას

6.3.3. out მოდიფიკატორის გამოყენებით შესაძლებელია:
ა. მხოლოდ ერთი მნიშვნელობის დაბრუნება
ბ. რამდენიმე მნიშვნელობის დაბრუნება
გ. მხოლოდ ორი მნიშვნელობის დაბრუნება

6.3.4. out მოდიფიკატორი ეთითება:
ა. იმ პარამეტრის წინ, რომლის დაბრუნებაც გვინდა
ბ. იმ პარამეტრის წინ, რომლის დაბრუნებაც არ გვინდა
გ. პროგრამის პირველ სტრიქონში გამოცხადებული ცვლადის წინ

6.3.5. out მოდიფიკატორიანი პარამეტრი გამოიყენება:
ა. მეთოდისთვის ინფორმაციის გადასაცემად
ბ. მეთოდიდან ინფორმაციის დასაბრუნებლად
გ. მეთოდისთვის როგორც ინფორმაციის გადასაცემად, ისე მეთოდიდან ინფორმაციის დასაბრუნებლად

6.3.6. მეთოდმა out მოდიფიკატორიან პარამეტრს:
ა. შეიძლება არ მიანიჭოს მნიშვნელობა
ბ. არ უნდა მიანიჭოს მნიშვნელობა
გ. აუცილებლად უნდა მიანიჭოს მნიშვნელობა

params მოდიფიკატორი

6.4.1. მეთოდისთვის ცვლადი რაოდენობის არგუმენტების გადასაცემად უნდა გამოვიყენოთ:
ა. out მოდიფიკატორი
ბ. public მოდიფიკატორი
გ. params მოდიფიკატორი

6.4.2. params მოდიფიკატორი საშუალებას გვაძლევს მეთოდს გადავცეთ:
ა. არც ერთი არგუმენტი
ბ. მხოლოდ ორი არგუმენტი
გ. არგუმენტების ნებისმიერი რაოდენობა

6.4.3. როცა მეთოდში მითითებულია ჩვეულებრივი და ცვლადი სიგრძის პარამეტრები, მაშინ:
ა. ცვლადი სიგრძის პარამეტრი პარამეტრების სიაში უნდა იყოს უკანასკნელი
ბ. ცვლადი სიგრძის პარამეტრი პარამეტრების სიაში უნდა იყოს პირველი
გ. ცვლადი სიგრძის პარამეტრი პარამეტრების სიაში უნდა იყოს რიგით მეორე

მეთოდის გადატვირთვა. პოლიმორფიზმი

6.5.1. როცა კლასში ორ ან მეტ მეთოდს ერთნაირი სახელი აქვს, მაშინ:
ა. ამ მეთოდების პარამეტრების რაოდენობა უნდა იყოს სხვადასხვა ან პარამეტრებს სხვადასხვა ტიპი უნდა ჰქონდეთ
ბ. ეს მეთოდები უნდა განსხვავდებოდეს მხოლოდ პარამეტრების რაოდენობის მიხედვით
გ. ეს მეთოდები უნდა განსხვავდებოდეს მხოლოდ პარამეტრების ტიპის მიხედვით

6.5.2. როცა კლასში რამდენიმე მეთოდს ერთნაირი სახელი აქვს, მაშინ
ა. ისინი არ არიან გადატვირთვადი მეთოდები
ბ. მათ გადატვირთვადი ეწოდებათ
გ. ადგილი აქვს ინკაფსულაციას

6.5.3. კლასში ერთნაირი სახელის მქონე მეთოდების განსაზღვრის პროცესს ეწოდება:
ა. მემკვიდრეობითობა
ბ. ინკაფსულაცია
გ. მეთოდის გადატვირთვა

6.5.4. პოლიმორფიზმი საშუალებას გვაძლევს:
ა. მეთოდის რამდენიმე სახელის ნაცვლად გამოვიყენოთ ერთი
ბ. მეთოდის ერთი სახელის ნაცვლად გამოვიყენოთ რამდენიმე
გ. მეთოდის სახელები მემკვიდრეობით გადავცეთ

6.5.5. პოლიმორფიზმის უზრუნველყოფა ხდება:
ა. ინკაფსულაციის საშუალებით
ბ. მეთოდების გადატვირთვის საშუალებით
გ. ერთი მეთოდისათვის რამდენიმე სახელის დარქმევის საშუალებით

6.5.6. გადატვირთვადი მეთოდის გამოძახებისას სრულდება ის მეთოდი, რომლის:
ა. პარამეტრები წილადი ტიპისაა
ბ. პარამეტრები ემთხვევა არ არგუმენტებს
გ. პარამეტრები ემთხვევა არგუმენტებს

6.5.7. გადატვირთვად მეთოდებში თუ პარამეტრების რაოდენობა თანაბარია, მაშინ
ა. პარამეტრს უნდა ჰქონდეს მთელი ტიპი
ბ. არ უნდა განსხვავდებოდეს მათი ტიპები
გ. უნდა განსხვავდებოდეს მათი ტიპები

6.5.8. მეთოდების გადატვირთვისას:
ა. შეიძლება ref და out მოდიფიკატორების გამოყენება
ბ. არ შეიძლება ref და out მოდიფიკატორების გამოყენება
გ. შეიძლება მხოლოდ out მოდიფიკატორების გამოყენება

6.5.9. თუ გადატვირთვად მეთოდებში პარამეტრების რაოდენობა თანაბარია და არგუმენტებსა და პარამეტრებს განსხვავებული ტიპები აქვს, მაშინ:
ა. არ სრულდება ტიპების ავტომატური გარდაქმნა
ბ. სრულდება ტიპების ავტომატური გარდაქმნა
გ. ყველა მონაცემი გარდაიქმნება მთელი ტიპის მონაცემად

კონსტრუქტორების გადატვირთვა

6.6.1. რომელი მსჯელობაა სწორი:
ა. შეიძლება კონსტრუქტორის გადატვირთვა
ბ. არ შეიძლება კონსტრუქტორის გადატვირთვა
გ. კონსტრუქტორის გადატვირთვა ხანდახან შეიძლება

6.6.2. ერთი მეთოდის ინიციალიზებისათვის კონსტრუქტორმა:
ა. შეიძლება გამოიყენოს მხოლოდ ლოგიკური ობიექტები
ბ. არ შეიძლება მეორე ობიექტი გამოიყენოს
გ. შეიძლება მეორე ობიექტი გამოიყენოს

6.6.3. კლასის შიგნით ერთი გადატვირთვადი კონსტრუქტორის მიერ მეორის გამოძახება შესაძლებელია:
ა. public მოდიფიკატორის გამოყენებით
ბ. this სიტყვის გამოყენებით
გ. static მოდიფიკატორის გამოყენებით

6.6.4. გამომძახებელი კონსტრუქტორის შესრულებისას:
ა. ჯერ გამოიძახება მითითებული გადატვირთვადი კონსტრუქტორი, შემდეგ კი შესრულდება გამომძახებელი კონსტრუქტორის დანარჩენი ოპერატორები
ბ. ჯერ შესრულდება გამომძახებელი კონსტრუქტორის ოპერატორები, შემდეგ კი გამოიძახება მითითებული გადატვირთვადი კონსტრუქტორი
გ. არ გამოიძახება გადატვირთვადი კონსტრუქტორი

რეკურსია

6.7.1. რომელი მსჯელობაა სწორი:
ა. მეთოდს შეუძლია თავისი თავის გამოძახება
ბ. მეთოდს არ შეუძლია თავისი თავის გამოძახება
გ. მეთოდს შეუძლია მხოლოდ სხვა მეთოდის გამოძახება

6.7.2. რეკურსია არის პროცესი, როცა მეთოდი:
ა. სხვა მეთოდს იძახებს
ბ. თავის თავს იძახებს
გ. სხვა პროგრამას იძახებს

6.7.3. რეკურსიული არის მეთოდი, რომელიც:
ა. სხვა მეთოდს იძახებს
ბ. თავის თავს იძახებს
გ. სხვა პროგრამას იძახებს

static მოდიფიკატორი

6.8.1. კლასის სტატიკური წევრის გამოცხადება იწყება სიტყვით:
ა. static
ბ. public
გ. private

6.8.2. კლასის სტატიკურ წევრთან მუშაობა შეიძლება:
ა. ამ კლასის ობიექტის შექმნის შემდეგ

ბ. ამ კლასის ობიექტის შექმნამდე
გ. მასივის საშუალებით

6.8.3. კლასის სტატიკურ წევრთან მიმართვისათვის უნდა მივუთითოთ:
ა. კლასის სახელი და წევრის სახელი წერტილის გარეშე
ბ. ობიექტის სახელი, წერტილი და წევრის სახელი
გ. კლასის სახელი, წერტილი და წევრის სახელი

6.8.4. თუ სტატიკური ცვლადი რიცხვითი ტიპისაა და არ მიენიჭა საწყისი მნიშვნელობა, მაშინ მას ავტომატურად მიენიჭება:
ა. null
ბ. 0
გ. false

6.8.5. თუ სტატიკური ცვლადი მიმართვითი ტიპისაა და არ მიენიჭა საწყისი მნიშვნელობა, მაშინ მას ავტომატურად მიენიჭება:
ა. null
ბ. 0
გ. true

6.8.6. თუ სტატიკური ცვლადი ლოგიკური ტიპისაა და არ მიენიჭა საწყისი მნიშვნელობა, მაშინ მას ავტომატურად მიენიჭება:
ა. null
ბ. 0
გ. false

6.8.7. სტატიკურ მეთოდს:
ა. აქვს this მიმართვა
ბ. არ აქვს this მიმართვა
გ. ხანდახან აქვს this მიმართვა

6.8.8. სტატიკურ მეთოდს შეუძლია:
ა. პირდაპირ მიმართოს მხოლოდ სტატიკურ მონაცემებს
ბ. პირდაპირ მიმართოს მხოლოდ არასტატიკურ მონაცემებს
გ. პირდაპირ მიმართოს მხოლოდ სტრიქონებს

6.8.9. სტატიკურ მეთოდს შეუძლია:
ა. უშუალოდ გამოიძახოს სტატიკური მეთოდი
ბ. უშუალოდ გამოიძახოს არასტატიკური მეთოდი
გ. ირიბად გამოიძახოს სტატიკური მეთოდი

6.8.10. სტატიკურ მეთოდს შეუძლია მიმართოს თავისი კლასის ობიექტის მეთოდებსა და ცვლადებს მხოლოდ:
ა. სხვა კლასის ობიექტის გამოყენებით
ბ. თავისი კლასის ობიექტის გამოყენებით

გ. ლოგიკური ცვლადების გამოყენებით

თავი 7. ოპერატორების გადატვირთვა, თვისებები და ინდექსატორები

ოპერატორების გადატვირთვა

7.1.1. ოპერატორის გადატვირთვა არის მოქმედებების ერთობლიობა:
ა. რომლებსაც ოპერატორი შეასრულებს ჩვენს მიერ შექმნილი კლასის მიმართ
ბ. როცა იცვლება ოპერატორის საწყისი დანიშნულება
გ. როცა ხდება კლასის გამოცხადება

7.1.2. ოპერატორის გადატვირთვისას მისი:
ა. გამოყენება ხდება მასივებში
ბ. საწყისი დანიშნულება იცვლება
გ. საწყისი დანიშნულება არ იცვლება

7.1.3. ოპერატორის გადატვირთვისას:
ა. მის არსენალს ახალი ფუნქციები არ ემატება
ბ. მის არსენალს ახალი ფუნქციები ემატება
გ. ხდება ინდექსატორის გამოცხადება

7.1.4. ოპერატორის გადატვირთვისათვის გამოიყენება:
ა. public სიტყვა
ბ. operator სიტყვა
გ. static სიტყვა

7.1.5. operator სიტყვა განსაზღვრავს:
ა. ოპერატორის მეთოდს
ბ. გადატვირთვად მეთოდს
გ. სტატიკურ მეთოდს

7.1.6. უნარულია ოპერატორი, რომელსაც აქვს: (ხომ არ მეორდება წინა თავებში)
ა. სამი ოპერანდი
ბ. ორი ოპერანდი
გ. ერთი ოპერანდი

7.1.7. ბინარულია ოპერატორი, რომელსაც აქვს: (ხომ არ მეორდება წინა თავებში)
ა. სამი ოპერანდი
ბ. ერთი ოპერანდი
გ. ორი ოპერანდი

7.1.8. უნარული ოპერატორის გადატვირთვისას:
ა. ოპერანდს უნდა ჰქონდეს მთელი ტიპი
ბ. ოპერანდი უნდა ეკუთვნოდეს იმ კლასის ტიპს, რომლისათვისაც განისაზღვრა

ოპერატორი
გ. ოპერანდს უნდა ჰქონდეს ლოგიკური ტიპი

7.1.9. ბინარული ოპერატორების გადატვირთვისას:
ა. ერთ-ერთი ოპერანდის ტიპი უნდა ემთხვეოდეს იმ კლასის ტიპს, რომლისთვისაც ხდება ოპერატორის ხელახალი განსაზღვრა
ბ. ერთ-ერთი ოპერანდი უნდა იყოს მთელი ტიპის
გ. ერთ-ერთი ოპერანდი უნდა იყოს წილადი ტიპის

7.1.10. არ შეიძლება + ოპერატორის გადატვირთვა:
ა. არ არსებული ტიპებისათვის
ბ. უკვე არსებული ტიპებისათვის
გ. ნებისმიერი ტიპისათვის

7.1.11. - ოპერატორის გადატვირთვისას მნიშვნელობა:
ა. აქვს მხოლოდ მთელი რიცხვების მიმდევრობას
ბ. არ აქვს ოპერანდების მიმდევრობას
გ. აქვს ოპერანდების მიმდევრობას

7.1.12. არაკომუტაციური ოპერატორების გადატვირთვისას:
ა. არ უნდა გავითვალისწინოთ ოპერანდების ურთიერთგანლაგება
ბ. უნდა გავითვალისწინოთ მხოლოდ მთელი რიცხვების ურთიერთგანლაგება
გ. უნდა გავითვალისწინოთ ოპერანდების ურთიერთგანლაგება

7.1.13. კომუტაციური ოპერატორების გადატვირთვისას:
ა. არ უნდა გავითვალისწინოთ ოპერანდების ურთიერთგანლაგება
ბ. უნდა გავითვალისწინოთ ოპერანდების ურთიერთგანლაგება
გ. უნდა გავითვალისწინოთ ლოგიკური მონაცემების ურთიერთგანლაგება

7.1.14. შედარების გადატვირთული ოპერატორები აბრუნებენ:
ა. true ან false მნიშვნელობას
ბ. true ან int ტიპის მნიშვნელობას
გ. false ან double ტიპის მნიშვნელობას

7.1.15. შედარების ოპერატორების გადატვირთვა:
ა. უნდა შესრულდეს წყვილობრივად
ბ. არ უნდა შესრულდეს წყვილობრივად
გ. უნდა შესრულდეს მხოლოდ მთელი რიცხვებისათვის

7.1.16. C# ენაში არსებობს ოპერატორების შემდეგი წყვილები:
ა. < და <=, > და >=, <= და >=
ბ. == და >=, <= და !=, < და >
გ. == და !=, > და <, <= და >=

7.1.17. ოპერატორების გადატვირთვისას:

ა. შეიძლება ოპერატორების პრიორიტეტების შეცვლა
ბ. შეუძლებელია ოპერატორების პრიორიტეტების შეცვლა
გ. შეიძლება ოპერატორების პრიორიტეტების შეცვლა მხოლოდ მთელი რიცხვებისათვის

7.1.18. ოპერატორების გადატვირთვისას:
ა. შეიძლება ოპერატორის ოპერანდების რაოდენობის შეცვლა
ბ. შეიძლება ოპერატორის ოპერანდების რაოდენობის შეცვლა მხოლოდ წილადებისათვის
გ. არ შეიძლება ოპერატორის ოპერანდების რაოდენობის შეცვლა

7.1.19. მინიჭების ოპერატორის გადატვირთვა:
ა. შეიძლება მხოლოდ მთელი რიცხვებისათვის
ბ. შეიძლება
გ. არ შეიძლება

7.1.20. რომელი ოპერატორების გადატვირთვა არ შეიძლება:
ა. + , <
ბ. && , ||
გ. * , >=

7.1.21. რომელი ოპერატორების გადატვირთვა შეიძლება:
ა. / , %
ბ. ? , =
გ. new , >

თვისებები

7.2.1. თვისება არის კლასის წევრი, რომელიც საშუალებას გვაძლევს:
ა. ვიმუშაოთ მხოლოდ მასივებთან
ბ. მეთოდების საშუალებით მივიღოთ ან შვცვალოთ ცვლადების მნიშვნელობები
გ. ვიმუშაოთ მხოლოდ წილადებთან

7.2.2. თვისება შედგება:
ა. მხოლოდ public მოდიფიკატორიანი მეთოდებისაგან
ბ. მხოლოდ სტატიკური ცვლადებისაგან
გ. მიმართვის get და set მეთოდებისაგან

7.2.3. თვისების get მეთოდი გამოიყენება:
ა. ცვლადის მნიშვნელობის მისაღებად
ბ. ცვლადის მნიშვნელობის შესაცვლელად
გ. ცვლადის მნიშვნელობისათვის წილადის მისანიჭებლად

7.2.4. თვისების set მეთოდი გამოიყენება:
ა. მხოლოდ წილადი მნიშვნელობის გასაცემად

ბ. ცვლადის მნიშვნელობის გადასაცემად
გ. ცვლადისათვის მნიშვნელობის მისანიჭებლად

7.2.5. თვისების set მეთოდს ავტომატურად გადაეცემა:
ა. value არაცხადი პარამეტრი
ბ. value ცხადი პარამეტრი
გ. კლასის მისამართი

7.2.6. თვისების set მეთოდისთვის ავტომატურად გადაცემული value პარამეტრი:
ა. არ შეიცავს ცვლადისათვის მისანიჭებელ მნიშვნელობას
ბ. შეიცავს ცვლადისათვის მისანიჭებელ მნიშვნელობას
გ. შეიცავს მხოლოდ მთელ რიცხვს

7.2.7. რომელი მსჯელობაა სწორი:
ა. თვისების სახელი არ შეიძლება გამოვიყენოთ გამოსახულებაში
ბ. თვისების სახელი შეიძლება გამოვიყენოთ გამოსახულებაში
გ. თვისების სახელი შეიძლება გამოვიყენოთ მხოლოდ მთელი რიცხვების მიმართ

7.2.8. რომელი მსჯელობაა სწორი:
ა. ჩვეულებრივ, თვისებაში არ ხდება ცვლადის გამოცხადება
ბ. თვისებაში აუცილებლად ხდება ცვლადის გამოცხადება
გ. თვისებაში ხდება მხოლოდ წილადი რიცხვების გამოცხადება

7.2.9. თუ თვისების სახელი მითითებულია მინიჭების ოპერატორის მარცხნივ, მაშინ:
ა. გამოიძახება get მეთოდი
ბ. არ გამოიძახება არც ერთი მეთოდი
გ. გამოიძახება set მეთოდი

7.2.10. თუ თვისების სახელი მითითებულია მინიჭების ოპერატორის მარჯვნივ, მაშინ:
ა. გამოიძახება get მეთოდი
ბ. გამოიძახება set მეთოდი
გ. გამოიძახება ორივე მეთოდი

7.2.11. რომელი მსჯელობაა სწორი:
ა. თვისება შეიძლება გადაეცეს მეთოდს, როგორც ref ან out პარამეტრი
ბ. თვისება შეიძლება გადაეცეს მეთოდს, მხოლოდ როგორც ref პარამეტრი
გ. თვისება არ შეიძლება გადაეცეს მეთოდს, როგორც ref ან out პარამეტრი

7.2.12. რომელი მსჯელობაა სწორი:
ა. თვისება შეიძლება იყოს გადატვირთული
ბ. თვისება არ შეიძლება იყოს გადატვირთული
გ. თვისება ნაწილობრივ შეიძლება იყოს გადატვირთული

7.2.13. რომელი მსჯელობაა სწორი:

ა. ჩვეულებრივ, თვისებამ არ უნდა შეცვალოს ცვლადების მნიშვნელობა get მეთოდის გამოყენებით
ბ. თვისებამ აუცილებლად უნდა შეცვალოს ცვლადების მნიშვნელობა get მეთოდის გამოყენებით
გ. თვისებამ აუცილებლად უნდა შეცვალოს მხოლოდ სტრიქონი get მეთოდის გამოყენებით

ინდექსატორი

7.3.1. ინდექსატორი საშუალებას გვაძლევს ობიექტის ცვლადებს მივმართოთ:
ა. ობიექტის სახელისა და ინდექსის საშუალებით
ბ. მხოლოდ ობიექტის სახელის საშუალებით
გ. მხოლოდ ინდექსის საშუალებით

7.3.2. ინდექსატორს შეიძლება ჰქონდეს:
ა. ერთი ან მეტი განზომილება
ბ. მხოლოდ ერთი განზომილება
გ. მხოლოდ ორი განზომილება

7.3.3. თუ ინდექსატორი მითითებულია მინიჭების ოპერატორის მარცხნივ, მაშინ:
ა. არც ერთი მეთოდი არ გამოიძახება
ბ. გამოიძახება get მეთოდი
გ. გამოიძახება set მეთოდი

7.3.4. თუ ინდექსატორი მითითებულია მინიჭების ოპერატორის მარჯვნივ, მაშინ:
ა. გამოიძახება set მეთოდი
ბ. გამოიძახება get მეთოდი
გ. ორივე მეთოდი გამოიძახება

7.3.5. რომელი მსჯელობაა მცდარი:
ა. ინდექსატორში შეიძლება განსაზღვრული იყოს set და get მეთოდები
ბ. ინდექსატორში შეიძლება არ იყოს განსაზღვრული set და get მეთოდები
გ. ინდექსატორში შეიძლება განსაზღვრული იყოს set ან get მეთოდი

7.3.6. რომელი მსჯელობაა სწორი:
ა. ინდექსატორისათვის არ შეიძლება ref ან out მოდიფიკატორის გამოყენება
ბ. ინდექსატორისათვის შეიძლება ref ან out მოდიფიკატორის გამოყენება
გ. ინდექსატორისათვის შეიძლება მხოლოდ ref მოდიფიკატორის გამოყენება

თავი 8. მემკვიდრეობითობა. ვირტუალური მეთოდები

მემკვიდრეობითობის საფუძვლები

8.1.1. წინაპარი არის კლასი, რომლის:
ა. ცვლადები და მეთოდები ავტომატურად არ ხდება სხვა ახალი კლასის წევრები
ბ. მხოლოდ ცვლადები ხდება სხვა ახალი კლასის წევრები
გ. ცვლადები და მეთოდები ავტომატურად ხდება სხვა ახალი კლასის წევრები

8.1.2. მემკვიდრე არის კლასი, რომელიც:
ა. წინაპარის კლასის წევრებს ახალ წევრებს არ უმატებს
ბ. წინაპარის კლასის წევრებს მხოლოდ მეთოდებს უმატებს
გ. წინაპარის კლასის წევრებს ახალ წევრებს უმატებს

8.1.3. მემკვიდრე კლასი წინაპარი კლასისაგან:
ა. მემკვიდრეობით იღებს ყველა წევრს
ბ. მემკვიდრეობით არ იღებს ყველა წევრს
გ. მემკვიდრეობით იღებს მხოლოდ ინდექსატორებს

8.1.4. მემკვიდრე კლასისათვის შეგვიძლია მივუთითოთ:
ა. მხოლოდ ორი წინაპარი კლასი
ბ. მხოლოდ ერთი წინაპარი კლასი
გ. რამდენიმე წინაპარი კლასი

8.1.5. მემკვიდრე კლასი:
ა. შეიძლება გახდეს წინაპარი სხვა კლასისათვის
ბ. არ შეიძლება გახდეს წინაპარი სხვა კლასისათვის
გ. შეიძლება გახდეს წინაპარი სტატიკური კლასისათვის

8.1.6. რომელი მსჯელობაა სწორი:
ა. კლასი შეიძლება იყოს თავისი თავის წინაპარი
ბ. მემკვიდრე კლასი შეიძლება გახდეს წინაპარი სტატიკური კლასისათვის
გ. კლასი არ შეიძლება იყოს თავისი თავის წინაპარი

protected მოდიფიკატორი

8.2.1. მემკვიდრეობითობის დროს:
ა. ძალაში არ არის დახურულ წევრებთან მიმართვისას არსებული შეზღუდვები
ბ. ძალაშია დახურულ წევრებთან მიმართვისას არსებული შეზღუდვები
გ. ძალაშია მხოლოდ დახურულ თვისებებთან მიმართვისას არსებული შეზღუდვები

8.2.2. მემკვიდრე კლასის მეთოდს:
ა. შეუძლია მიმართოს მემკვიდრე კლასის დახურულ წევრებს
ბ. შეუძლია მიმართოს წინაპარი კლასის დახურულ წევრებს
გ. არ შეუძლია მიმართოს წინაპარი კლასის დახურულ წევრებს

8.2.3. კლასის დაცული წევრი:
ა. გახსნილია მხოლოდ მემკვიდრე კლასებისათვის და დახურულია სხვა

კლასებისათვის
ბ. დახურულია მხოლოდ მემკვიდრე კლასებისათვის და გახსნილია სხვა კლასებისათვის
გ. დახურულია როგორც მემკვიდრე, ისე სხვა კლასებისათვის

8.2.4. კლასის დაცული წევრი იქმნება:
ა. protected მოდიფიკატორის საშუალებით
ბ. public მოდიფიკატორის საშუალებით
გ. private მოდიფიკატორის საშუალებით

კონსტრუქტორები და მემკვიდრეობითობა

8.3.1. მემკვიდრე კლასის ობიექტი:
ა. მთლიანად იქმნება
ბ. ნაწილ-ნაწილ იქმნება
გ. არ იქმნება

8.3.2. თუ კონსტრუქტორი არ არის განსაზღვრული წინაპარ კლასში და განსაზღვრულია მემკვიდრე კლასში, მაშინ:
ა. მემკვიდრე კლასის ობიექტი არ იქმნება
ბ. წინაპარი კლასის ობიექტის კონსტრუირება სრულდება მისი კონსტრუქტორის მიერ, ხოლო მემკვიდრე კლასში განსაზღვრული ობიექტის ნაწილი კი იქმნება მისი ავტომატური კონსტრუქტორის მიერ
გ. მემკვიდრე კლასის ობიექტის კონსტრუირება სრულდება მისი კონსტრუქტორის მიერ, ხოლო წინაპარ კლასში განსაზღვრული ობიექტის ნაწილი კი იქმნება მისი ავტომატური კონსტრუქტორის მიერ

8.3.3. თუ კონსტრუქტორები განსაზღვრულია როგორც წინაპარ, ისე მემკვიდრე კლასებში, მაშინ:
ა. წინაპარი კლასის კონსტრუქტორის გამოსაძახებლად არ გამოიყენება base სიტყვა
ბ. მემკვიდრე კლასის კონსტრუქტორის გამოსაძახებლად გამოიყენება base სიტყვა
გ. წინაპარი კლასის კონსტრუქტორის გამოსაძახებლად გამოიყენება base სიტყვა

8.3.4. base სიტყვის გამოყენებით გამოიძახება ის კონსტრუქტორი, რომლის:
ა. პარამეტრების სია არ შეესაბამება არგუმენტების სიას
ბ. პარამეტრების სია შეესაბამება არგუმენტების სიას
გ. პარამეტრები სტატიკურია

8.3.5. მემკვიდრეობითობის დროს base სიტყვა ყოველთვის მიმართავს:
ა. უახლოესი წინაპარი კლასის კონსტრუქტორს
ბ. ბოლოს წინა კლასის კონსტრუქტორს
გ. მომდევნო მემკვიდრე კლასის კონსტრუქტორს

8.3.6. თუ მემკვიდრე კლასის კონსტრუქტორში არ არის მითითებული base სიტყვა, მაშინ:

ა. არ გამოიძახება წინაპარი კლასის ავტომატური კონსტრუქტორი
ბ. გამოიძახება წინაპარი კლასის ავტომატური კონსტრუქტორი
გ. არ გამოიძახება არც ერთი კონსტრუქტორი

ცვლადების დამალვა მემკვიდრეობითობის დროს

8.4.1. რომელი მსჯელობაა სწორი:
ა. წინაპარი და მემკვიდრე კლასების წევრებს აუცილებლად უნდა ჰქონდეთ ერთი და იგივე სახელი
ბ. წინაპარი და მემკვიდრე კლასების წევრებს არ შეიძლება ერთი და იგივე სახელი ჰქონდეთ
გ. წინაპარი და მემკვიდრე კლასების წევრებს შეიძლება ერთი და იგივე სახელი ჰქონდეთ

8.4.2. თუ წინაპარი და მემკვიდრე კლასების წევრებს ერთნაირი სახელები აქვთ, მაშინ:
ა. წინაპარი კლასის წევრი დამალულია წინაპარი კლასის წევრებისათვის
ბ. წინაპარი კლასის წევრი არ არის დამალული მემკვიდრე კლასის წევრებისათვის
გ. წინაპარი კლასის წევრი დამალულია მემკვიდრე კლასის წევრებისათვის

8.4.3. წინაპარი კლასის წევრი დამალულია მემკვიდრე კლასის წევრებისათვის მაშინ, როცა:
ა. წინაპარი და მემკვიდრე კლასების წევრებს ერთნაირი სახელები აქვს
ბ. წინაპარი და მემკვიდრე კლასების წევრებს სხვადასხვა სახელები აქვს
გ. ყველა მემკვიდრე კლასის წევრებს ერთნაირი სახელები აქვს

8.4.4. თუ განზრახ ვმალავთ წინაპარი კლასის წევრს, მაშინ:
ა. მემკვიდრე კლასის შესაბამისი წევრის წინ არ უნდა მივუთითოთ new ოპერატორი
ბ. მემკვიდრე კლასის შესაბამისი წევრის წინ უნდა მივუთითოთ new ოპერატორი
გ. მემკვიდრე კლასის შესაბამისი წევრის წინ უნდა მივუთითოთ static ოპერატორი

8.4.5. როცა განზრახ ვმალავთ წინაპარი კლასის წევრს, მაშინ მემკვიდრე კლასის შესაბამისი წევრის წინ მოთავსებული new ოპერატორი:
ა. ახდენს ობიექტის შექმნას
ბ. არ ახდენს ობიექტის შექმნას
გ. ახდენს კლასის შექმნას

8.4.6. წინაპარი კლასის დამალულ წევრთან მიმართვისათვის გამოიყენება:
ა. protected სიტყვა
ბ. public სიტყვა
გ. base სიტყვა

კლასების იერარქია

8.5.1. რომელი მსჯელობაა სწორი:
ა. მემკვიდრე კლასი შეიძლება იყოს წინაპარი სხვა კლასებისათვის
ბ. მემკვიდრე კლასი არ შეიძლება იყოს წინაპარი სხვა კლასებისათვის
გ. წინაპარი კლასი არ შეიძლება იყოს მემკვიდრე სხვა კლასებისათვის

8.5.2. კლასების იერარქიაში კონსტრუქტორების გამოძახების რიგითობაა:
ა. წინაპარი კლასიდან მემკვიდრე კლასისაკენ
ბ. მემკვიდრე კლასიდან წინაპარი კლასისაკენ
გ. წინაპარი კლასიდან წინაპარი კლასისაკენ

8.5.3. კლასების იერარქიაში თუ არ გამოიყენება base სიტყვა, მაშინ:
ა. არ გამოიძახება თითოეული წინაპარი კლასის ავტომატური კონსტრუქტორი
ბ. გამოიძახება თითოეული წინაპარი კლასის ავტომატური კონსტრუქტორი
გ. გამოიძახება თითოეული მემკვიდრე კლასის ავტომატური კონსტრუქტორი

მიმართვები წინაპარი და მემკვიდრე კლასის ობიექტებზე

8.6.1. ერთი კლასის მიმართვით ცვლადს:
ა. შეიძლება მიენიჭოს მხოლოდ წინაპარი კლასის ობიექტზე მიმართვა
ბ. შეიძლება მიენიჭოს სხვა კლასის ობიექტზე მიმართვა
გ. არ შეიძლება მიენიჭოს სხვა კლასის ობიექტზე მიმართვა

8.6.2. წინაპარი კლასის მიმართვით ცვლადს:
ა. შეიძლება მიენიჭოს მემკვიდრე კლასის ობიექტზე მიმართვა
ბ. არ შეიძლება მიენიჭოს მემკვიდრე კლასის ობიექტზე მიმართვა
გ. შეიძლება მიენიჭოს წინაპარი კლასის ობიექტზე მიმართვა

8.6.3. კლასის წევრთან მიმართვის შესაძლებლობა დამოკიდებულია:
ა. მიმართვითი ცვლადის ტიპზე
ბ. ობიექტის ტიპზე
გ. მასივის ტიპზე

8.6.4. თუ მიმართვით ცვლადს, რომელსაც აქვს წინაპარი კლასის ტიპი, ენიჭება მემკვიდრე კლასის ობიექტზე მიმართვა, მაშინ:
ა. ამ ცვლადის საშუალებით შეგვეძლება მხოლოდ იმ წევრებთან მიმართვა, რომლებიც განსაზღვრული იყო მემკვიდრე კლასში
ბ. ამ ცვლადის საშუალებით შეგვეძლება მხოლოდ იმ წევრებთან მიმართვა, რომლებიც განსაზღვრული იყო წინაპარ კლასში
გ. ამ ცვლადის საშუალებით შეგვეძლება მიმართვა მხოლოდ მასივებთან

ვირტუალური მეთოდები

8.7.1. პროგრამის შესრულების დროს ვირტუალური მეთოდები უზრუნველყოფენ:
ა. მასივებთან მუშაობას

ბ. ინკაფსულაციას
გ. პოლიმორფიზმს

8.7.2. ვირტუალური მეთოდის კოდი მემკვიდრე კლასებში:
ა. შეიძლება შეიცვალოს
ბ. არ შეიძლება შეიცვალოს
გ. შეიძლება შეიცვალოს სტატიკური მეთოდებისათვის

8.7.3. virtual საკვანძო სიტყვა გამოიყენება:
ა. სტატიკური მეთოდის გამოცხადებისათვის
ბ. აბსტრაქტული მეთოდის გამოცხადებისათვის
გ. ვირტუალური მეთოდის გამოცხადებისათვის

8.7.4. ვირტუალური მეთოდის ვერსიის არჩევა ხორციელდება:
ა. მიმართვითი ცვლადის ტიპის მიხედვით
ბ. იმ ობიექტის ტიპის შესაბამისად, რომელსაც მიმართვითი ცვლადი მიმართავს
გ. სტატიკური მეთოდის ტიპის მიხედვით

8.7.5. ვირტუალური მეთოდის არჩევა ხორციელდება:
ა. პროგრამის შესრულების დროს
ბ. პროგრამის შესრულების შემდეგ
გ. პროგრამის შესრულებამდე

8.7.6. რომელი მსჯელობაა სწორი:
ა. ტიპი განსაზღვრავს გამოსაძახებელი სტატიკური მეთოდის ვერსიას
ბ. ობიექტის ტიპი განსაზღვრავს გამოსაძახებელი ვირტუალური მეთოდის ვერსიას
გ. მიმართვითი ცვლადის ტიპი განსაზღვრავს გამოსაძახებელი ვირტუალური მეთოდის ვერსიას

8.7.7. მემკვიდრე კლასში ვირტუალური მეთოდის ხელახალი განსაზღვრისას გამოიყენება:
ა. override სიტყვა
ბ. virtual სიტყვა
გ. public სიტყვა

8.7.8. წინაპარ კლასში ვირტუალური მეთოდის განსაზღვრისას გამოიყენება:
ა. override სიტყვა
ბ. public სიტყვა
გ. virtual სიტყვა

8.7.9. ვირტუალური მეთოდი:
ა. შეიძლება განისაზღვროს params მოდიფიკატორის გამოყენებით
ბ. არ შეიძლება განისაზღვროს static მოდიფიკატორის გამოყენებით
გ. შეიძლება განისაზღვროს static მოდიფიკატორის გამოყენებით

8.7.10. მეთოდის ხელახალი განსაზღვრა ეწოდება ვირტუალური მეთოდის განსაზღვრის პროცესს, როცა:
ა. ნაწილობრივ ან მთლიანად იცვლება მეთოდის ტანი, ხოლო მეთოდის სახელი, პარამეტრები და მათი ტიპები იგივე რჩება
ბ. იცვლება მეთოდის ტანი და სახელი
გ. იცვლება მეთოდის სახელი და პარამეტრები

8.7.11. მეთოდის ხელახალი განსაზღვრა საფუძვლად უდევს:
ა. გამოსაძახებელი მეთოდის სტატიკური არჩევის კონცეფციას
ბ. გამოსაძახებელი მეთოდის დინამიური არჩევის კონცეფციას
გ. მასივების განსაზღვრის კონცეფციას

8.7.12. ვირტუალური მეთოდის გადატვირთვა:
ა. აუცილებელია მხოლოდ მასივებისათვის
ბ. არ არის აუცილებელი
გ. აუცილებელია

8.7.13. თუ მემკვიდრე კლასს არ აქვს ვირტუალური მეთოდის საკუთარი ვერსია, მაშინ:
ა. არ გამოიყენება წინაპარი კლასის მეთოდი
ბ. გამოიყენება წინაპარი კლასის მეთოდი
გ. გამოიყენება მემკვიდრე კლასის მეთოდი

აბსტრაქტული კლასები

8.8.1. მეთოდის ხელმოწერა არის:
ა. მხოლოდ მეთოდის სახელი
ბ. დასაბრუნებელი მნიშვნელობის ტიპი, მეთოდის სახელი და პარამეტრების სია
გ. მხოლოდ პარამეტრების სია

8.8.2. აბსტრაქტული მეთოდების რეალიზება უნდა მოხდეს:
ა. წინაპარ კლასში
ბ. მემკვიდრე კლასებში
გ. მასივებში

8.8.3. აბსტრაქტულ მეთოდებს:
ა. ტანი არ აქვთბ. ტანი აქვთ
გ. ტანი აქვთ წინაპარ კლასში

8.8.4. აბსტრაქტული მეთოდების გამოცხადებისათვის გამოიყენება:
ა. params მოდიფიკატორი
ბ. public მოდიფიკატორი
გ. abstract მოდიფიკატორი

8.8.5. აბსტრაქტული მეთოდი:
ა. არ არის ვირტუალური მეთოდი

ბ. არის ვირტუალური მეთოდი
გ. არის სტატიკური მეთოდი

8.8.6. virtual და abstract მოდიფიკატორების ერთობლივი გამოყენება:
ა. დაუშვებელია
ბ. დასაშვებია
გ. დასაშვებია მასივებთან ერთად

8.8.7. abstract მოდიფიკატორი:
ა. შეიძლება გამოვიყენოთ მასივების მიმართ
ბ. შეიძლება გამოვიყენოთ სტატიკური მეთოდების მიმართ
გ. არ შეიძლება გამოვიყენოთ სტატიკური მეთოდების მიმართ

8.8.8. კლასი, რომელიც აბსტრაქტულ მეთოდებს შეიცავს, გამოცხადებული უნდა იყოს როგორც:
ა. აბსტრაქტული
ბ. ვირტუალური
გ. სტატიკური

8.8.9. აბსტრაქტული კლასის ობიექტის შექმნა:
ა. შესაძლებელი
ბ. შეუძლებელია
გ. შესაძლებელი ლოგიკურ მონაცემებთან ერთად

8.8.10. თუ მემკვიდრე კლასი არ ახდენს წინაპარი კლასის აბსტრაქტული მეთოდების რეალიზებას, მაშინ ეს მემკვიდრე კლასი გამოცხადებული უნდა იყოს როგორც:
ა. ვირტუალური
ბ. აბსტრაქტული
გ. სტატიკური

მემკვიდრეობითობის აკრძალვა

8.9.1. მემკვიდრეობითობის აკრძალვისათვის გამოიყენება:
ა. sealed მოდიფიკატორი
ბ. public მოდიფიკატორი
გ. protected მოდიფიკატორი

თავი 9. ინფორმაციის შეტანა-გამოტანა

შეტანა-გამოტანის ნაკადების კლასები

9.1.1. C#-პროგრამები მონაცემების შეტანა-გამოტანას ახდენენ:
ა. ნაკადების საშუალებით
ბ. ინდექსატორების საშუალებით
გ. მოვლენების საშუალებით

9.1.2. მონაცემების ნაკადი არის:
ა. ფაილების მასივი
ბ. თვისებების მასივი
გ. მონაცემების მიმდევრობა

9.1.3. ნაკადი ფიზიკურ მოწყობილობას უკავშირდება:
ა. ვირტუალური მეთოდების გამოყენებით
ბ. სტატიკური მეთოდების გამოყენებით
გ. შეტანა-გამოტანის სისტემის საშუალებით

9.1.4. შეტანა-გამოტანის ოპერაციები ყველაზე დაბალ დონეზე ოპერირებენ:
ა. სტრიქონებთან
ბ. ბაიტებთან
გ. წილადებთან

9.1.5. შეტანა-გამოტანის ოპერაციები ყველაზე დაბალ დონეზე ბაიტებთან ოპერირებენ, რადგან:
ა. შეტანა-გამოტანის მოწყობილობების უმრავლესობა არის ბაიტ-ორიენტირებული
ბ. შეტანა-გამოტანის მოწყობილობების უმრავლესობა არის სიმბოლოზე ორიენტირებული
გ. შეტანა-გამოტანის მოწყობილობების უმრავლესობა არის მთელ რიცხვებზე ორიენტირებული

9.1.6. ნაკადების კლასები განსაზღვრულია:
ა. System.Data კლასში
ბ. System.IO კლასში
გ. System.Drawing კლასში

9.1.7. რომელი მსჯელობაა არასწორი:
ა. არსებობს სიმბოლოების object ნაკრები
ბ. არსებობს სიმბოლოების ASCII ნაკრები
გ. არსებობს სიმბოლოების Unocode ნაკრები

9.1.8. ASCII სიმბოლოს ზომაა:
ა. 1 ბაიტი
ბ. 2 ბაიტი
გ. 4 ბაიტი

9.1.9. Unicode სიმბოლოს ზომაა:
ა. 1 ბაიტი
ბ. 2 ბაიტი
გ. 4 ბაიტი

9.1.10. C# ენაში ნაკადები იქმნება:

ა. System.Net კლასის საშუალებით
ბ. System კლასის საშუალებით
გ. Stream კლასის საშუალებით

9.1.11. Stream კლასი არის:
ა. სტატიკური
ბ. ვირტუალური
გ. აბსტრაქტული

9.1.12. Stream კლასის Close() მეთოდი:
ა. ნაკადს ხურავს
ბ. ნაკადს ხსნის
გ. ნაკადს ცვლის

9.1.13. Stream კლასის Flush() მეთოდი:
ა. ნაკადს ფაილში წერს
ბ. ნაკადს ფაილიდან კითხულობს
გ. ნაკადს ხსნის

9.1.14. Stream კლასის ReadByte() მეთოდი:
ა. შესასვლელი ნაკადიდან კითხულობს 2 ბაიტს და გასცემს მის მთელრიცხვა მნიშვნელობას
ბ. შესასვლელი ნაკადიდან კითხულობს 1 ბაიტს და გასცემს მის მთელრიცხვა მნიშვნელობას
გ. შესასვლელი ნაკადიდან კითხულობს 1 ბაიტს და გასცემს მის ლოგიკურ მნიშვნელობას

9.1.15. Stream კლასის Read(byte[] ბუფერი, int პოზიცია, int ბაიტების_რაოდენობა) მეთოდი:
ა. შესასვლელ ნაკადში წერს მითითებული რაოდენობის ბაიტს და მათ ათავსებს ბუფერში დაწყებული მითითებული პოზიციიდან
ბ. შესასვლელ ნაკადში ცვლის მითითებული რაოდენობის ბაიტს
გ. შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის ბაიტს და მათ ათავსებს ბუფერში დაწყებული მითითებული პოზიციიდან

9.1.16. Stream კლასის Seek() მეთოდი:
ა. გასცემს ნაკადის პირველი ბაიტის პოზიციას
ბ. გასცემს ნაკადის მიმდინარე ბაიტის პოზიციას
გ. გასცემს ნაკადის უკანასკნელი ბაიტის პოზიციას

9.1.17. Stream კლასის WriteByte(byte ცვლადი) მეთოდი:
ა. გამოსასვლელ ნაკადში 1 ბაიტს წერს
ბ. გამოსასვლელ ნაკადში 2 ბაიტს წერს
გ. შესასვლელი ნაკადიდან 1 ბაიტს კითხულობს

9.1.18. Stream კლასის Write(byte[ ] ბუფერი, int პოზიცია, int ბაიტების_რაოდენობა) მეთოდი:
ა. გამოსასვლელ ნაკადში წერს მითითებული რაოდენობის ბაიტს დაწყებული მითითებული პოზიციიდან
ბ. გამოსასვლელ ნაკადში წერს მხოლოდ წილად რიცხვებს
გ. შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის ბაიტს

9.1.19. Stream კლასის CanRead თვისება იღებს:
ა. true მნიშვნელობას თუ შეუძლებელია ნაკადიდან წაკითხვა
ბ. false მნიშვნელობას თუ შესაძლებელია ნაკადიდან წაკითხვა
გ. true მნიშვნელობას თუ შესაძლებელია ნაკადიდან წაკითხვა

9.1.20. Stream კლასის CanSeek თვისება იღებს:
ა. false მნიშვნელობას თუ შესაძლებელია ნაკადის ელემენტის მიმდინარე პოზიციის განსაზღვრა
ბ. true მნიშვნელობას თუ შესაძლებელია ნაკადის ელემენტის მიმდინარე პოზიციის განსაზღვრა
გ. true მნიშვნელობას თუ შეუძლებელია ნაკადის ელემენტის მიმდინარე პოზიციის განსაზღვრა

9.1.21. Stream კლასის CanWrite თვისება იღებს:
ა. true მნიშვნელობას თუ შესაძლებელია ნაკადში ჩაწერა
ბ. false მნიშვნელობას თუ შესაძლებელია ნაკადში ჩაწერა
გ. true მნიშვნელობას თუ შეუძლებელია ნაკადში ჩაწერა

9.1.22. Stream კლასის Length თვისება განსაზღვრავს:
ა. ნაკადის სიგრძეს
ბ. ნაკადის მიმდინარე ელემენტის პოზიციას
გ. ნაკადში შეიძლება თუ არა მონაცემების ჩაწერა

9.1.23. Stream კლასის Position თვისება განსაზღვრავს:
ა. ნაკადში შეიძლება თუ არა მონაცემების წაკითხვა
ბ. ნაკადის მიმდინარე ელემენტის პოზიციას
გ. ნაკადის სიგრძეს

9.1.24. Stream კლასის მემკვიდრე FileStream კლასი განკუთვნილია:
ა. ფაილში მონაცემების მხოლოდ წასაკითხად
ბ. ფაილში მონაცემების მხოლოდ ჩასაწერად
გ. ფაილში შეტანა-გამოტანის ოპერაციების შესასრულებლად

9.1.25. სიმბოლოების ნაკადების კლასებია:
ა. FileStream
ბ. TextReader და TextWriter
გ. BufferedStream

9.1.26. ბაიტების ნაკადების კლასებია:
ა. TextWriter და TextReader
ბ. FileStream და MemoryStream
გ. BufferedStream და TextWriter

9.1.27. TextReader კლასის Close() მეთოდი:
ა. ცვლის შესასვლელ ნაკადს
ბ. ხსნის შესასვლელ ნაკადს
გ. ხურავს შესასვლელ ნაკადს

9.1.28. TextReader კლასის Peek() მეთოდი:
ა. კითხულობს შესასვლელი ნაკადის მომდევნო სიმბოლოს და არ შლის მას
ბ. კითხულობს შესასვლელი ნაკადის მომდევნო სიმბოლოს და შლის მას
გ. წერს შესასვლელ ნაკადში მომდევნო სიმბოლოს და არ შლის მას

9.1.29. TextReader კლასის Peek() მეთოდი:
ა. გასცემს -2-ს თუ არ არის მომდევნო წასაკითხი სიმბოლო
ბ. გასცემს -1-ს თუ არ არის მომდევნო წასაკითხი სიმბოლო
გ. გასცემს -1-ს თუ არის მომდევნო წასაკითხი სიმბოლო

9.1.30. TextReader კლასის Read() მეთოდი:
ა. წერს შესასვლელი ნაკადის მომდევნო სიმბოლოს და გასცემს მის მთელრიცხვა წარმოდგენას
ბ. კითხულობს შესასვლელი ნაკადის მომდევნო სიმბოლოს და გასცემს მის მთელრიცხვა წარმოდგენას
გ. კითხულობს შესასვლელი ნაკადის მომდევნო სიმბოლოს და გასცემს მის წილად წარმოდგენას

9.1.31. TextReader კლასის Read(char[ ] ბუფერი, int პოზიცია,
int სიმბოლოების_რაოდენობა) მეთოდი:
ა. შესასვლელი ნაკადიდან კითხულობს 10 სიმბოლოს და ათავსებს მათ ბუფერში დაწყებული პირველი პოზიციიდან
ბ. შესასვლელ ნაკადში წერს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ ბუფერში დაწყებული მითითებული პოზიციიდან
გ. შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ ბუფერში დაწყებული მითითებული პოზიციიდან

9.1.32. TextReader კლასის ReadBlock(char[ ] ბუფერი, int პოზიცია,
int სიმბოლოების_რაოდენობა) მეთოდი:
ა. შესასვლელი ნაკადიდან კითხულობს 10 სიმბოლოს და ათავსებს მათ ბუფერში დაწყებული პირველი პოზიციიდან
ბ. შესასვლელ ნაკადში წერს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ ბუფერში დაწყებული მითითებული პოზიციიდან
გ. შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ ბუფერში დაწყებული მითითებული პოზიციიდან

9.1.33. TextReader კლასის ReadLine() მეთოდი:
ა. კითხულობს ტექსტის სტრიქონს და გასცემს მას C#-სტრიქონის სახით
ბ. წერს ტექსტის სტრიქონს და გასცემს მას C#-სტრიქონის სახით
გ. კითხულობს ტექსტის სტრიქონს და გასცემს მის მთელრიცხვა წარმოდგენას

9.1.34. TextReader კლასის ReadToEnd() მეთოდი:
ა. კითხულობს ნაკადის დარჩენილ სიმბოლოებს და გასცემს მათ C#-სტრიქონის სახით
ბ. წერს ნაკადის დარჩენილ სიმბოლოებს და გასცემს მათ C#-სტრიქონის სახით
გ. კითხულობს ნაკადის დარჩენილ სიმბოლოებს და გასცემს მათ წილადის სახით

9.1.35. TextWriter კლასის Write(int ცვლადი) მეთოდი ახდენს:
ა. int ტიპის მნიშვნელობის წაკითხვას
ბ. double ტიპის მნიშვნელობის ჩაწერას
გ. int ტიპის მნიშვნელობის ჩაწერას

9.1.36. TextWriter კლასის Write(double ცვლადი) მეთოდი ახდენს:
ა. int ტიპის მნიშვნელობის ჩაწერას
ბ. double ტიპის მნიშვნელობის ჩაწერას
გ. double ტიპის მნიშვნელობის წაკითხვას

9.1.37. TextWriter კლასის Write(bool ცვლადი) მეთოდი ახდენს:
ა. int ტიპის მნიშვნელობის ჩაწერას
ბ. bool ტიპის მნიშვნელობის ჩაწერას
გ. bool ტიპის მნიშვნელობის წაკითხვას

9.1.38. TextWriter კლასის WriteLine(string ცვლადი) მეთოდი ახდენს:
ა. სტრიქონის ჩაწერას, რომელიც არ მთავრდება მომდევნო სტრიქონზე გადასვლის სიმბოლოთი
ბ. სტრიქონის ჩაწერას, რომელიც მთავრდება მომდევნო სტრიქონზე გადასვლის სიმბოლოთი
გ. სტრიქონის წაკითხვას, რომელიც მთავრდება მომდევნო სტრიქონზე გადასვლის სიმბოლოთი

9.1.39. TextWriter კლასის WriteLine(uint ცვლადი) მეთოდი ახდენს:
ა. uint ტიპის მნიშვნელობისა და მომდევნო სტრიქონზე გადასვლის სიმბოლოს ჩაწერას
ბ. uint ტიპის მნიშვნელობისა და მომდევნო სტრიქონზე გადასვლის სიმბოლოს წაკითხვას
გ. uint ტიპის მნიშვნელობის ჩაწერას

9.1.40. TextWriter კლასის WriteLine(char ცვლადი) მეთოდი ახდენს:
ა. სტრიქონის ჩაწერას, რომელსაც მოსდევს მომდევნო სტრიქონზე გადასვლის სიმბოლო

ბ. სიმბოლოს წაკითხვას, რომელსაც მოსდევს მომდევნო სტრიქონზე გადასვლის სიმბოლო
გ. სიმბოლოს ჩაწერას, რომელსაც მოსდევს მომდევნო სტრიქონზე გადასვლის სიმბოლო

9.1.41. TextWriter კლასის Close() მეთოდი:
ა. ცვლის ნაკადს
ბ. ხსნის ნაკადს
გ. ხურავს ნაკადს

9.1.42. TextWriter კლასის Flush() მეთოდი ახდენს:
ა. ბუფერში დარჩენილი მონაცემების ჩაწერას გამოსასვლელ ნაკადში
ბ. ბუფერში დარჩენილი მონაცემების წაკითხვას გამოსასვლელ ნაკადში
გ. ნაკადის დახურვას

9.1.43. ორობითი ნაკადების კლასებია:
ა. FileStream
ბ. BinaryReader და BinaryWriter
გ. TextReader და TextWriter

FileStream კლასი

9.2.1. FileStream კლასი გამოიყენება:
ა. ფაილებში ბაიტების ჩაწერა-წაკითხვის ოპერაციების შესასრულებლად
ბ. ფაილებში სიმბოლოების ჩაწერა-წაკითხვის ოპერაციების შესასრულებლად
გ. ფაილში მხოლოდ სიმბოლოების ჩასაწერად

9.2.2. ფაილთან დაკავშირებული ბაიტების ნაკადის შესაქმნელად საჭიროა:
ა. MemoryStream ობიექტის ფორმირება
ბ. Object ობიექტის ფორმირება
გ. FileStream ობიექტის ფორმირება

9.2.3. FileMode რეჟიმი განსაზღვრავს:
ა. ფაილში ჩაწერის რეჟიმს
ბ. ფაილის გახსნის რეჟიმს
გ. ფაილიდან წაკითხვის რეჟიმს

9.2.4. FileMode რეჟიმის Append მნიშვნელობა მიუთითებს, რომ ხდება:
ა. მონაცემების წაკითხვა
ბ. გამოსასვლელი მონაცემების დამატება ფაილის ბოლოში
გ. მონაცემების ჩაწერა ფაილის დასაწყისში

9.2.5. FileMode რეჟიმის Create მნიშვნელობა მიუთითებს, რომ:
ა. იქმნება ახალი გამოსასვლელი ფაილი და ამავე სახელის მქონე არსებული ფაილი იშლება

ბ. იქმნება ახალი გამოსასვლელი ფაილი და ამავე სახელის მქონე არსებული ფაილი არ იშლება
გ. იხსნება არსებული ფაილი

9.2.6. FileMode რეჟიმის CreateNew მნიშვნელობა მიუთითებს, რომ:
ა. იქმნება ახალი გამოსასვლელი ფაილი
ბ. იშლება არსებული ფაილი
გ. იხსნება არსებული ფაილი

9.2.7. FileMode რეჟიმის Open მნიშვნელობა მიუთითებს, რომ:
ა. იხურება არსებული ფაილი
ბ. იშლება არსებული ფაილი
გ. იხსნება არსებული ფაილი

9.2.8. FileMode რეჟიმის OpenCreate მნიშვნელობა მიუთითებს, რომ:
ა. თუ ფაილი არსებობს, მაშინ ის გაიხსნება, თუ არ არსებობს, მაშინ ის შეიქმნება
ბ. თუ ფაილი არსებობს, მაშინ ის წაიშლება
გ. თუ ფაილი არსებობს, მაშინ ის შეიქმნება

9.2.9. FileMode რეჟიმის Truncate მნიშვნელობა მიუთითებს, რომ:
ა. იხსნება არსებული ფაილი, მაგრამ მისი სიგრძე არ ჩამოიჭრება ნულამდე
ბ. იქმნება ახალი ფაილი
გ. იხსნება არსებული ფაილი, მაგრამ მისი სიგრძე ჩამოიჭრება ნულამდე

9.2.10. FileStraem file_in = new FileStream(“filetest.dat”, FileMode.Open);
ოპერატორის შესრულების შედეგად:
ა. წაიშლება filetest.dat ფაილი
ბ. გაიხსნება filetest.dat ფაილი
გ. შეიცვლება filetest.dat ფაილი

9.2.11. FileAccess მიმართვის სახე იღებს მნიშვნელობას:
ა. Read, Write და ReadWrite
ბ. Read, Open
გ. Close, Open და Write

9.2.12. FileStraem file_in = new FileStream(“filetest.dat”, FileMode.Open, FileAccess.Read); ოპერატორის შესრულების შედეგად:
ა. filetest.dat ფაილი გაიხსნება ჩაწერის რეჟიმში
ბ. filetest.dat ფაილი გაიხსნება დამატების რეჟიმში
გ. filetest.dat ფაილი გაიხსნება წაკითხვის რეჟიმში

9.2.13. FileStraem კლასის ReadByte() მეთოდი:
ა. ფაილში წერს ერთ ბაიტს და გასცემს მის მთელრიცხვა მნიშვნელობას
ბ. ფაილიდან კითხულობს ერთ ბაიტს და გასცემს მის მთელრიცხვა მნიშვნელობას
გ. ფაილიდან კითხულობს ერთ ბაიტს და არ გასცემს მის მთელრიცხვა

მნიშვნელობას

9.2.14. FileStraem კლასის Read(byte[ ] ბუფერი, int პოზიცია,
int ბაიტების_რაოდენობა) მეთოდი:
ა. ფაილიდან კითხულობს მითითებული რაოდენობის ბაიტს და ათავსებს მათ ბუფერში
ბ. ფაილში წერს მითითებული რაოდენობის ბაიტს და ათავსებს მათ ბუფერში
გ. ფაილიდან კითხულობს მითითებული რაოდენობის ბაიტს და არ ათავსებს მათ ბუფერში

9.2.15. FileStraem კლასის ReadByte() მეთოდი ფაილის დასასრულის სიმბოლოს წაკითხვის შემთხვევაში გასცემს:
ა. -1
ბ. 0
გ. 1

9.2.16. FileStraem კლასის Read(byte[ ] ბუფერი, int პოზიცია,
int ბაიტების_რაოდენობა) მეთოდის შესრულების შედეგად გაიცემა:
ა. წასაკითხი ბაიტების მითითებული რაოდენობა
ბ. რეალურად ჩაწერილი ბაიტების რაოდენობა
გ. რეალურად წაკითხული ბაიტების რაოდენობა

9.2.17. FileStraem კლასის WriteByte() მეთოდი:
ა. ფაილში წერს სიმბოლოს
ბ. ფაილში წერს 1 ბაიტს
გ. ფაილში წერს 2 ბაიტს

9.2.18. FileStraem კლასის Write(byte[ ] ბუფერი, int პოზიცია,
int ბაიტების_რაოდენობა) მეთოდი:
ა. ბუფერიდან ფაილში წერს მხოლოდ ერთ ბაიტს
ბ. ბუფერიდან ფაილში კითხულობს მითითებული რაოდენობის ბაიტს
გ. ბუფერიდან ფაილში წერს მითითებული რაოდენობის ბაიტს

9.2.19. FileStraem კლასის Write(byte[ ] ბუფერი, int პოზიცია,
int ბაიტების_რაოდენობა) მეთოდის შესრულების შედეგად გაიცემა:
ა. რეალურად ჩაწერილი ბაიტების რაოდენობა
ბ. რეალურად წაკითხული ბაიტების რაოდენობა
გ. ჩასაწერი ბაიტების მითითებული რაოდენობა

9.2.20. ფაილში მონაცემები იწერება მაშინ, როცა ბუფერი:
ა. სავსეა
ბ. ცარიელია
გ. ნახევრად სავსეა

9.2.21. FileStraem კლასის Flush() მეთოდი ახდენს ფაილში ჩაწერას, როცა:

ა. ბუფერი ცარიელია
ბ. ბუფერი სავსეა
გ. ბუფერში მხოლოდ სიმბოლოებია

9.2.22. FileStraem კლასის Close() მეთოდი:
ა. ბუფერიდან მონაცემებს არ წერს ფაილში და ხურავს ფაილს
ბ. ბუფერიდან მონაცემებს წერს ფაილში და ხსნის ფაილს
გ. ბუფერიდან მონაცემებს წერს ფაილში და ხურავს ფაილს

ფაილში სიმბოლოების შეტანა-გამოტანა

9.3.1. სიმბოლურ ფაილებთან მუშაობისას FileStream ობიექტი უნდა ჩავრთოთ:
ა. object კლასის შემადგენლობაში
ბ. StreamReader ან StreamWriter კლასის შემადგენლობაში
გ. MemoryStream კლასის შემადგენლობაში

9.3.2. FileStream file_out = new FileStream("F1.txt", FileMode.Create);
StreamWriter file_strem_out = new StreamWriter(file_out);
ოპერატორების შესრულების შედეგად:
ა. იქმნება F1.txt ფაილი სიმბოლოების ჩასაწერად
ბ. იხსნება F1.txt ფაილი სიმბოლოების ჩასაწერად
გ. იშლება F1.txt ფაილი

9.3.3. FileStream file_in = new FileStream("F1.txt", FileMode.Open);
StreamReader file_strem_in = new StreamReader(file_in);
ოპერატორების შესრულების შედეგად:
ა. იქმნება F1.txt ფაილი სიმბოლოების წასაკითხად
ბ. იხსნება F1.txt ფაილი სიმბოლოების ჩასაწერად
გ. იხსნება F1.txt ფაილი სიმბოლოების წასაკითხად

ორობითი მონაცემების შეტანა-გამოტანა

9.4.1. ფაილში ორობითი მონაცემების ჩასაწერად გამოიყენება:
ა. TextWriter კლასი
ბ. BinaryReader კლასი
გ. BinaryWriter კლასი

9.4.2. ფაილში ორობითი მონაცემების წასაკითხად გამოიყენება:
ა. BinaryReader კლასი
ბ. BinaryWriter კლასი
გ. TextReader კლასი

9.4.3. BinaryWriter კლასის Write(sbyte ცვლადი) მეთოდი ახდენს ფაილში:
ა. ნიშნიანი ბაიტის წაკითხვას
ბ. ნიშნიანი ბაიტის ჩაწერას

გ. უნიშნო ბაიტის ჩაწერას

9.4.4. BinaryWriter კლასის Write(byte ცვლადი) მეთოდი ახდენს ფაილში:
ა. ნიშნიანი ბაიტის ჩაწერას
ბ. უნიშნო ბაიტის ჩაწერას
გ. უნიშნო ბაიტის წაკითხვას

9.4.5. BinaryWriter კლასის Write(byte[ ] ბუფერი) მეთოდი ახდენს ფაილში:
ა. მხოლოდ 1 ბაიტის ჩაწერას
ბ. ბაიტების მასივის წაკითხვას
გ. ბაიტების მასივის ჩაწერას

9.4.6. BinaryWriter კლასის Write(short ცვლადი) მეთოდი ახდენს ფაილში:
ა. მოკლე მთელი რიცხვის ჩაწერას
ბ. გრძელი მთელი რიცხვის ჩაწერას
გ. მოკლე მთელი რიცხვის წაკითხვას

9.4.7. BinaryWriter კლასის Write(ushort ცვლადი) მეთოდი ახდენს ფაილში:
ა. ნიშნიანი მთელი რიცხვის ჩაწერას
ბ. უნიშნო მთელი რიცხვის ჩაწერას
გ. უნიშნო მთელი რიცხვის წაკითხვას

9.4.8. BinaryWriter კლასის Write(int ცვლადი) მეთოდი ახდენს ფაილში:
ა. უნიშნო მთელი რიცხვის ჩაწერას
ბ. მთელი რიცხვის წაკითხვას
გ. მთელი რიცხვის ჩაწერას

9.4.9. BinaryWriter კლასის Write(uint ცვლადი) მეთოდი ახდენს ფაილში:
ა. უნიშნო მთელი რიცხვის წაკითხვას
ბ. უნიშნო მთელი რიცხვის ჩაწერას
გ. ნიშნიანი მთელი რიცხვის ჩაწერას

9.4.10. BinaryWriter კლასის Write(long ცვლადი) მეთოდი ახდენს ფაილში:
ა. გრძელი მთელი რიცხვის ჩაწერას
ბ. გრძელი მთელი რიცხვის წაკითხვას
გ. მოკლე მთელი რიცხვის ჩაწერას

9.4.11. BinaryWriter კლასის Write(ulong ცვლადი) მეთოდი ახდენს ფაილში:
ა. უნიშნო გრძელი მთელი რიცხვის ჩაწერას
ბ. უნიშნო გრძელი მთელი რიცხვის წაკითხვას
გ. ნიშნიანი გრძელი მთელი რიცხვის ჩაწერას

9.4.12. BinaryWriter კლასის Write(float ცვლადი) მეთოდი ახდენს ფაილში:
ა. ორმაგი სიზუსტის წილადის ჩაწერას
ბ. ერთმაგი სიზუსტის წილადის წაკითხვას

გ. ერთმაგი სიზუსტის წილადის ჩაწერას

9.4.13. BinaryWriter კლასის Write(double ცვლადი) მეთოდი ახდენს ფაილში:
ა. ერთმაგი სიზუსტის წილადის ჩაწერას
ბ. ორმაგი სიზუსტის წილადის ჩაწერას
გ. ორმაგი სიზუსტის წილადის წაკითხვას

9.4.14. BinaryWriter კლასის Write(char ცვლადი) მეთოდი ახდენს ფაილში:
ა. სიმბოლოს ჩაწერას
ბ. სიმბოლოს წაკითხვას
გ. სტრიქონის ჩაწერას

9.4.15. BinaryWriter კლასის Write(char[ ] მასივი) მეთოდი ახდენს ფაილში:
ა. სტრიქონების მასივის ჩაწერას
ბ. სიმბოლოების მასივის წაკითხვას
გ. სიმბოლოების მასივის ჩაწერას

9.4.16. BinaryWriter კლასის Write(string ცვლადი) მეთოდი ახდენს ფაილში:
ა. სტრიქონის წაკითხვას
ბ. სტრიქონის ჩაწერას
გ. სიმბოლოების მასივის წაკითხვას

9.4.17. BinaryReader კლასის Read() მეთოდი ფაილიდან:
ა. კითხულობს შესასვლელი ნაკადის სიმბოლოს და გასცემს მის მთელრიცხვა მნიშვნელობას
ბ. კითხულობს შესასვლელი ნაკადის სიმბოლოს და არ გასცემს მის მთელრიცხვა მნიშვნელობას
გ. შესასვლელ ნაკადში წერს სიმბოლოს და გასცემს მის მთელრიცხვა მნიშვნელობას

9.4.18. BinaryReader კლასის Read(byte[ ] ბუფერი, int პოზიცია,
int ბაიტების_რაოდენობა) მეთოდი:
ა. ფაილიდან კითხულობს მხოლოდ 1 ბაიტს
ბ. ფაილში წერს მითითებული რაოდენობის ბაიტს ბუფერიდან
გ. ფაილიდან კითხულობს მითითებული რაოდენობის ბაიტს და ათავსებს მათ ბუფერში

9.4.19. BinaryReader კლასის Read(char[ ] ბუფერი, int პოზიცია,
int ბაიტების_რაოდენობა) მეთოდი:
ა. ფაილში წერს მითითებული რაოდენობის სიმბოლოს ბუფერიდან
ბ. ფაილიდან კითხულობს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ ბუფერში
გ. ფაილიდან კითხულობს მხოლოდ 1 სიმბოლოს

9.4.20. BinaryReader კლასის ReadBoolean() მეთოდი ახდენს:
ა. ლოგიკური მონაცემის წაკითხვას

ბ. ლოგიკური მონაცემის ჩაწერას
გ. მთელი რიცხვის წაკითხვას

9.4.21. BinaryReader კლასის ReadByte() მეთოდი ახდენს:
ა. ნიშნიანი ბაიტის წაკითხვას
ბ. უნიშნო ბაიტის წაკითხვას
გ. უნიშნო ბაიტის ჩაწერას

9.4.22. BinaryReader კლასის ReadSByte() მეთოდი ახდენს:
ა. ნიშნიანი ბაიტის წაკითხვას
ბ. უნიშნო ბაიტის წაკითხვას
გ. ნიშნიანი ბაიტის ჩაწერას

9.4.23. BinaryReader კლასის ReadBytes(int ბაიტების_რაოდენობა) მეთოდი ახდენს:
ა. მხოლოდ 1 ბაიტების წაკითხვას
ბ. მითითებული რაოდენობის ბაიტების ჩაწერას
გ. მითითებული რაოდენობის ბაიტების წაკითხვას

9.4.24. BinaryReader კლასის ReadChar() მეთოდი ახდენს:
ა. სიმბოლოს წაკითხვას
ბ. სიმბოლოს ჩაწერას
გ. ბაიტის წაკითხვას

9.4.25. BinaryReader კლასის ReadChars(int სიმბოლოების_რაოდენობა) მეთოდი ახდენს:
ა. მითითებული რაოდენობის სიმბოლოების ჩაწერას
ბ. მითითებული რაოდენობის სიმბოლოების წაკითხვას
გ. მხოლოდ 1 სიმბოლოების წაკითხვას

9.4.26. BinaryReader კლასის ReadDouble() მეთოდი ახდენს:
ა. ერთმაგი სიზუსტის წილადის წაკითხვას
ბ. ორმაგი სიზუსტის წილადის ჩაწერას
გ. ორმაგი სიზუსტის წილადის წაკითხვას

9.4.27. BinaryReader კლასის ReadSingle() მეთოდი ახდენს:
ა. ერთმაგი სიზუსტის წილადის წაკითხვას
ბ. ორმაგი სიზუსტის წილადის წაკითხვას
გ. ერთმაგი სიზუსტის წილადის ჩაწერას

9.4.28. BinaryReader კლასის ReadInt16() მეთოდი ახდენს:
ა. მოკლე მთელი რიცხვის წაკითხვას
ბ. მოკლე მთელი რიცხვის ჩაწერას
გ. გრძელი მთელი რიცხვის წაკითხვას

9.4.29. BinaryReader კლასის ReadInt32() მეთოდი ახდენს:
ა. მთელი რიცხვის წაკითხვას

ბ. მთელი რიცხვის ჩაწერას
გ. გრძელი მთელი რიცხვის წაკითხვას

9.4.30. BinaryReader კლასის ReadInt64() მეთოდი ახდენს:
ა. გრძელი მთელი რიცხვის წაკითხვას
ბ. გრძელი მთელი რიცხვის ჩაწერას
გ. მთელი რიცხვის წაკითხვას

9.4.31. BinaryReader კლასის ReadUInt16() მეთოდი ახდენს:
ა. უნიშნო მოკლე მთელი რიცხვის წაკითხვას
ბ. უნიშნო მოკლე მთელი რიცხვის ჩაწერას
გ. ნიშნიანი მოკლე მთელი რიცხვის წაკითხვას

9.4.32. BinaryReader კლასის ReadUInt32() მეთოდი ახდენს:
ა. უნიშნო მთელი რიცხვის ჩაწერას
ბ. უნიშნო მთელი რიცხვის წაკითხვას
გ. უნიშნო მოკლე მთელი რიცხვის წაკითხვას

9.4.33. BinaryReader კლასის ReadUInt64() მეთოდი ახდენს:
ა. უნიშნო გრძელი მთელი რიცხვის ჩაწერას
ბ. გრძელი მთელი რიცხვის წაკითხვას
გ. უნიშნო გრძელი მთელი რიცხვის წაკითხვას

9.4.34. BinaryReader კლასის ReadString() მეთოდი ახდენს:
ა. სტრიქონის ჩაწერას
ბ. სტრიქონის წაკითხვას
გ. სიმბოლოს ჩაწერას

9.4.35. ricxvi = file_in.ReadInt32(); ოპერატორის შესრულების შედეგად:
ა. ricxvi ცვლადს მიენიჭება მოკლე მთელი რიცხვი
ბ. ricxvi ცვლადს მიენიჭება წილადი
გ. ricxvi ცვლადს მიენიჭება მთელი რიცხვი

9.4.36. ricxvi = file_in.ReadInt16(); ოპერატორის შესარულების შედეგად:
ა. ricxvi ცვლადს მიენიჭება წილადი
ბ. ricxvi ცვლადს მიენიჭება მთელი რიცხვი
გ. ricxvi ცვლადს მიენიჭება მოკლე მთელი რიცხვი

9.4.37. ricxvi = file_in.ReadInt64(); ოპერატორის შესარულების შედეგად:
ა. ricxvi ცვლადს მიენიჭება გრძელი მთელი რიცხვი
ბ. ricxvi ცვლადს მიენიჭება მთელი რიცხვი
გ. ricxvi ცვლადს მიენიჭება მოკლე მთელი რიცხვი

9.4.38. ricxvi = file_in.ReadUInt64(); ოპერატორის შესარულების შედეგად:
ა. ricxvi ცვლადს მიენიჭება უნიშნო გრძელი მთელი რიცხვი

ბ. ricxvi ცვლადს მიენიჭება გრძელი მთელი რიცხვი
გ. ricxvi ცვლადს მიენიჭება მოკლე მთელი რიცხვი

9.4.39. ricxvi = file_in.ReadUInt32(); ოპერატორის შესრულების შედეგად:
ა. ricxvi ცვლადს მიენიჭება უნიშნო გრძელი მთელი რიცხვი
ბ. ricxvi ცვლადს მიენიჭება უნიშნო მთელი რიცხვი
გ. ricxvi ცვლადს მიენიჭება მოკლე მთელი რიცხვი

9.4.40. ricxvi = file_in.ReadUInt16(); ოპერატორის შესრულების შედეგად:
ა. ricxvi ცვლადს მიენიჭება უნიშნო გრძელი მთელი რიცხვი
ბ. ricxvi ცვლადს მიენიჭება გრძელი მთელი რიცხვი
გ. ricxvi ცვლადს მიენიჭება უნიშნო მოკლე მთელი რიცხვი

ფაილებთან პირდაპირი მიმართვა

9.5.1. რომელი მსჯელობაა არასწორი:
ა. ფაილებთან შესაძლებელია ლოგიკური მიმართვა
ბ. ფაილებთან შესაძლებელია მიმდევრობითი მიმართვა
გ. ფაილებთან შესაძლებელია პირდაპირი მიმართვა

9.5.2. ფაილთან მიმდევრობითი მიმართვის დროს:
ა. პირდაპირ სრულდება ფაილში მონაცემების ჩაწერა და წაკითხვა
ბ. მიმდევრობით სრულდება ფაილში მონაცემების ჩაწერა და წაკითხვა
გ. მიმდევრობით სრულდება მონაცემების ჩაწერა და პირდაპირ სრულდება წაკითხვა

9.5.3. ფაილთან პირდაპირი მიმართვის დროს:
ა. მიმდევრობით სრულდება მონაცემების ჩაწერა და პირდაპირ სრულდება წაკითხვა
ბ. მიმდევრობით სრულდება ფაილში მონაცემების ჩაწერა და წაკითხვა
გ. პირდაპირ სრულდება ფაილში მონაცემების ჩაწერა და წაკითხვა

9.5.4. ფაილთან პირდაპირი მიმართვის დროს გამოიყენება:
ა. ლოგიკური მიმთითებელი
ბ. საფაილო მიმთითებელი
გ. სიმბოლური მიმთითებელი

9.5.5. საფაილო მიმთითებელი არის:
ა. ფაილის მიმდინარე ელემენტის პოზიციის ნომერი
ბ. ფაილის სახელი
გ. ფაილის ტიპი

9.5.6. Seek() მეთოდი გამოიყენება:
ა. ფაილის ტიპის შესაცვლელად
ბ. ფაილის სახელის შესაცვლელად

გ. საფაილო ცვლადის მნიშვნელობის შესაცვლელად

9.5.7. საფაილო ცვლადის მნიშვნელობის შესაცვლელად გამოიყენება:
ა. object ობიექტი
ბ. FileStream კლასი
გ. Seek() მეთოდი

9.5.8. SeekOrigin.Begin მნიშვნელობა მიუთითებს იმაზე, რომ:
ა. გადათვლა უნდა შესრულდეს ფაილის დასაწყისიდან
ბ. გადათვლა უნდა შესრულდეს ფაილის ბოლოდან
გ. გადათვლა უნდა შესრულდეს ფაილის შუიდან

9.5.9. SeekOrigin.End მნიშვნელობა მიუთითებს იმაზე, რომ:
ა. გადათვლა უნდა შესრულდეს ფაილის დასაწყისიდან
ბ. გადათვლა უნდა შესრულდეს ფაილის მიმდინარე პოზიციიდან
გ. გადათვლა უნდა შესრულდეს ფაილის ბოლოდან

9.5.10. SeekOrigin.Current მნიშვნელობა მიუთითებს იმაზე, რომ:
ა. გადათვლა უნდა შესრულდეს ფაილის ბოლოდან
ბ. გადათვლა უნდა შესრულდეს ფაილის მიმდინარე პოზიციიდან
გ. გადათვლა უნდა შესრულდეს ფაილის დასაწყისიდან

9.5.11. file1.Seek(10, SeekOrigin.Begin); ოპერატორის შესრულების შედეგად:
ა. შესრულდება 10 ბაიტის გადათვლა ფაილის ბოლოდან
ბ. შესრულდება 10 ბაიტის გადათვლა ფაილის დასაწყისიდან
გ. შესრულდება 10 ბაიტის გადათვლა ფაილის მიმდინარე პოზიციიდან

9.5.12. file1.Seek(10, SeekOrigin.Current); ოპერატორის შესრულების შედეგად:
ა. შესრულდება 10 ბაიტის გადათვლა ფაილის დასაწყისიდან
ბ. შესრულდება 10 ბაიტის გადათვლა ფაილის ბოლოდან
გ. შესრულდება 10 ბაიტის გადათვლა ფაილის მიმდინარე პოზიციიდან

9.5.13. file1.Seek(10, SeekOrigin.End); ოპერატორის შესრულების შედეგად:
ა. შესრულდება 10 ბაიტის გადათვლა ფაილის ბოლოდან
ბ. შესრულდება 10 ბაიტის გადათვლა ფაილის მიმდინარე პოზიციიდან
გ. შესრულდება 10 ბაიტის გადათვლა ფაილის დასაწყისიდან

თავი 10. განსაკუთრებული სიტუაციები

განსაკუთრებული სიტუაციების დამუშავების საფუძვლები

10.1.1. განსაკუთრებული სიტუაცია არის სიტუაცია, რომელიც:
ა. არ იწვევს პროგრამის ავარიულად დამთავრებას
ბ. არ არის გამოწვეული პროგრამის შესრულების დროს აღძრული შეცდომის მიერ
გ. გამოწვეულია პროგრამის შესრულების დროს აღძრული შეცდომის მიერ

10.1.2. განსაკუთრებული სიტუაციებია:
ა. მასივის გამოცხადება და გადავსება
ბ. ნულზე გაყოფა და არარსებული ფაილის გახსნა
გ. კლასის გამოცხადება და მასივის ინდექსის გასვლა დიაპაზონის გარეთ

10.1.3. რაშია განსაკუთრებული სიტუაციების დამუშავების უპირატესობა:
ა. იმაში, რომ პროგრამას შეუძლია შეცდომის დამუშავება და შესრულების გაგრძელება
ბ. იმაში, რომ პროგრამას არ ეუძლია შეცდომის დამუშავება და შესრულების გაგრძელება
გ. იმაში, რომ პროგრამას შეუძლია შეცდომის დამუშავება, მაგრამ ავარიულად ამთავრებს მუშაობას

10.1.4. განსაკუთრებული სიტუაცია იწვევს:
ა. პროგრამის შესრულების ავარიულად დამთავრებას
ბ. პროგრამის შესრულების ნორმალურად დამთავრებას
გ. პროგრამის შესრულების გაგრძელებას

10.1.5. განსაკუთრებული სიტუაციების დამამუშავებლები იძლევიან:
ა. კლასების გამოცხადების საშუალებას საშუალებას
ბ. პროგრამის შესრულების ავარიულად დამთავრების საშუალებას
გ. პროგრამის შესრულების გაგრძელების საშუალებას

10.1.6. რომელია განსაკუთრებული სიტუაციების საბაზო კლასი:
ა. object კლასი
ბ. Exception კლასი
გ. FileStream კლასი

10.1.7. განსაკუთრებული სიტუაციების ყველა კლასი მემკვიდრეობით მიიღება:
ა. object ობიექტისაგან
ბ. Exception კლასისაგან
გ. FileStream კლასისაგან

10.1.8. Exception კლასი არის:
ა. System სახელების სივრცის ნაწილი
ბ. System.Data სახელების სივრცის ნაწილი
გ. System.Drawing სახელების სივრცის ნაწილი

10.1.9. განსაკუთრებული სიტუაციების რომელი კატეგორიები არსებობს:
ა. არსებობს ერთი კატეგორია: გენერირებული C# ენის შესრულების გარემოს მიერ
ბ. არსებობს ერთი კატეგორია: გენერირებული პროგრამა-დანართების მიერ
გ. არსებობს ორი კატეგორია: გენერირებული C# ენის შესრულების გარემოს მიერ და გენერირებული პროგრამა-დანართების მიერ

10.1.10. C# ენის შესრულების გარემოს (CLR) მიერ გენერირებული განსაკუთრებული სიტუაციები აღწერილია:
ა. object ობიექტში
ბ. SystemException კლასში
გ. ApplicationException კლასში

10.1.11. პროგრამა-დანართების მიერ გენერირებული განსაკუთრებული სიტუაციები აღწერილია:
ა. object ობიექტში
ბ. SystemException კლასში
გ. ApplicationException კლასში

10.1.12. რომელი განსაკუთრებული სიტუაცია აღიძვრება ნულზე გაყოფის დროს:
ა. ArrayTypeMismatchException
ბ. DivideByZeroException
გ. IndexOutOfRangeException

10.1.13. ArrayTypeMismatchException განსაკუთრებული სიტუაცია აღიძვრება მაშინ, როცა:
ა. მასივის ინდექსი გადის დიაპაზონის გარეთ
ბ. ადგილი აქვს ნულზე გაყოფას
გ. მნიშვნელობის ტიპი შეუთავსებია მასივის ტიპთან

10.1.14. DivideByZeroException განსაკუთრებული სიტუაცია აღიძვრება მაშინ, როცა:
ა. ადგილი აქვს ნულზე გაყოფას
ბ. მნიშვნელობის ტიპი შეუთავსებია მასივის ტიპთან
გ. მასივის ინდექსი გადის დიაპაზონის გარეთ

10.1.15. IndexOutOfRangeException განსაკუთრებული სიტუაცია აღიძვრება მაშინ, როცა:
ა. ადგილი აქვს არაკორექტულ გარდაქმნას პროგრამის შესრულების დროს
ბ. მასივის ინდექსი გადის დიაპაზონის გარეთ
გ. new ოპერატორის გამოძახება იყო არაკორექტული მეხსიერების უკმარისობის გამო

10.1.16. InvalidCastException განსაკუთრებული სიტუაცია აღიძვრება მაშინ, როცა:
ა. new ოპერატორის გამოძახება იყო არაკორექტული მეხსიერების უკმარისობის გამო
ბ. ადგილი აქვს არაკორექტულ გარდაქმნას პროგრამის შესრულების დროს
გ. ადგილი აქვს სტეკის გადავსებას

10.1.17. OutOfMemoryException განსაკუთრებული სიტუაცია აღიძვრება მაშინ, როცა:
ა. new ოპერატორის გამოძახება იყო არაკორექტული მეხსიერების უკმარისობის

გამო
ბ. მასივის ინდექსი გადის დიაპაზონის გარეთ
გ. ადგილი აქვს არაკორექტულ გარდაქმნას პროგრამის შესრულების დროს

10.1.18. OverflowException განსაკუთრებული სიტუაცია აღიძვრება მაშინ, როცა:
ა. ადგილი აქვს ნულზე გაყოფას
ბ. ადგილი აქვს სტეკის გადავსებას
გ. ადგილი აქვს გადავსებას არითმეტიკული ოპერაციის შესრულების დროს

10.1.19. StackOverflowException განსაკუთრებული სიტუაცია აღიძვრება მაშინ, როცა:
ა. ადგილი აქვს სტეკის გადავსებას
ბ. ადგილი აქვს გადავსებას არითმეტიკული ოპერაციის შესრულების დროს
გ. ადგილი აქვს ნულზე გაყოფას

10.1.20. განსაკუთრებული სიტუაციების დამუშავება სრულდება შემდეგი ოპერატორების გამოყენებით:
ა. try, catch, params
ბ. try, catch, throw, finally
გ. throw, finally, object

10.1.21. try ბლოკში მოთავსებულია ის ოპერატორები, რომელთა:
ა. შესრულებასაც თვალყური არ უნდა ვადევნოთ
ბ. შესრულებასაც თვალყური უნდა ვადევნოთ
გ. შეცვლაც გვინდა

10.1.22. განსაკუთრებულ სიტუაციას იჭერს და ამუშავებს:
ა. try ბლოკი
ბ. finally ბლოკი
გ. catch ბლოკი

10.1.23. განსაკუთრებული სიტუაციის ხელოვნურად გენერირებას იწვევს:
ა. catch ოპერატორი
ბ. try ოპერატორი
გ. throw ოპერატორი

10.1.24. finally ბლოკში მოთავსებულია კოდი, რომელიც:
ა. ყოველთვის შესრულდება
ბ. ყოველთვის არ შესრულდება
გ. ხანდახან შესრულდება

try და catch ბლოკები

10.2.1. try ბლოკთან შეიძლება დაკავშირებული იყოს:
ა. რამდენიმე catch ოპერატორი

ბ. მხოლოდ 1 catch ოპერატორი
გ. მხოლოდ 2 catch ოპერატორი

10.2.2. catch ოპერატორი იჭერს იმ განსაკუთრებულ სიტუაციას, რომელიც:
ა. გამოცხადებულია FileStream კლასში
ბ. არ შეესაბამება catch ოპერატორში მითითებულ ტიპს
გ. შეესაბამება catch ოპერატორში მითითებულ ტიპს

10.2.3. თუ განსაკუთრებული სიტუაციის გენერირება არ ხდება, მაშინ:
ა. catch ბლოკი შესრულდება
ბ. catch ბლოკი არ შესრულდება
გ. შესრულდება უპარამეტრებო catch ოპერატორი

10.2.4. catch ოპერატორში პარამეტრის მითითება:
ა. არ არის აუცილებელი
ბ. აუცილებელია
გ. ნაწილობრივაა აუცილებელი

10.2.5. catch ოპერატორში პარამეტრის მითითება საჭიროა მაშინ, როცა:
ა. არ არის აუცილებელი განსაკუთრებული სიტუაციის ობიექტთან მიმართვა
ბ. აუცილებელია განსაკუთრებული სიტუაციის ობიექტთან მიმართვა
გ. გამოტოვებულია try ბლოკი

10.2.6. განსაკუთრებული სიტუაციის დაჭერა და დამუშავება თავიდან აგვაცილებს:
ა. try ბლოკის შესრულებას
ბ. პროგრამის ნორმალურად დამთავრებას
გ. პროგრამის ავარიულად დამთავრებას

10.2.7. თუ პროგრამა არ ახდენს განსაკუთრებული სიტუაციის დაჭერას, მაშინ განსაკუთრებულ სიტუაციას დაიჭერს:
ა. System სახელების სივრცე
ბ. Windows ოპერაციული სისტემა
გ. C#-ის შესრულების სისტემა

10.2.8. რომელი მსჯელობაა სწორი:
ა. თუ განსაკუთრებული სიტუაციის ტიპი შეესაბამება catch ოპერატორში მითითებულ ტიპს, მაშინ ეს catch ოპერატორი ვერ დაიჭერს ამ განსაკუთრებულ სიტუაციას
ბ. თუ განსაკუთრებული სიტუაციის ტიპი შეესაბამება catch ოპერატორში მითითებულ ტიპს, მაშინ ეს catch ოპერატორი დაიჭერს ამ განსაკუთრებულ სიტუაციას
გ. თუ განსაკუთრებული სიტუაციის ტიპი არ შეესაბამება catch ოპერატორში მითითებულ ტიპს, მაშინ ეს catch ოპერატორი დაიჭერს ამ განსაკუთრებულ სიტუაციას

10.2.9. ყველა განსაკუთრებული სიტუაციის დასაჭერად უნდა გამოვიყენოთ catch ოპერატორი:

ა. პარამეტრების გარეშე
ბ. ერთი პარამეტრით
გ. ორი პარამეტრით

10.2.10. catch ოპერატორები მოწმდება:
ა. იმ მიმდევრობით, როგორც არიან პროგრამაში მითითებული
ბ. უკუმიმდევრობით
გ. დაწყებული შუა ოპერატორიდან

10.2.11. რისთვის გამოიყენება უპარამეტრებო catch ოპერატორი:
ა. ზოგიერთი განსაკუთრებული სიტუაციის დასაჭერად
ბ. ყველა განსაკუთრებული სიტუაციის დასაჭერად
გ. არ აჭერს არც ერთ განსაკუთრებულ სიტუაციას

ჩადგმული try ბლოკები

10.3.1. რომელი მსჯელობაა სწორი:
ა. try ბლოკში უნდა მოთავსთდეს მხოლოდ finally ბლოკი
ბ. არაა შესაძლებელი try ბლოკების ერთმანეთში მოთავსება
გ. შესაძლებელია try ბლოკების ერთმანეთში მოთავსება

10.3.2. განსაკუთრებული სიტუაცია, რომელიც ვერ დაიჭირა შიდა try ბლოკის შესაბამისმა catch ოპერატორმა:
ა. ვრცელდება გარე try ბლოკზე
ბ. არ ვრცელდება გარე try ბლოკზე
გ. ვრცელდება მხოლოდ შიდა try ბლოკზე

throw ოპერატორი

10.4.1. რომელი მსჯელობაა სწორი:
ა. რადგან throw ოპერატორი ოპერირებს ობიექტებზე, ამიტომ მის გამოყენებამდე საჭიროა ობიექტის შექმნა
ბ. რადგან throw ოპერატორი არ ოპერირებს ობიექტებზე, ამიტომ მის გამოყენებამდე საჭიროა ობიექტის შექმნა
გ. რადგან throw ოპერატორი ოპერირებს ობიექტებზე, ამიტომ მის გამოყენებამდე არ არის საჭირო ობიექტის შექმნა

finally ბლოკი

10.5.1. რომელი მსჯელობაა სწორი:
ა. finally ბლოკი არასოდეს არ გამოიძახება
ბ. finally ბლოკი გამოიძახება იმისგან დამოუკიდებლად გენერირდება თუ არა განსაკუთრებული სიტუაცია
გ. finally ბლოკი გამოიძახება მხოლოდ catch ბლოკში

10.5.2. finally ბლოკის კოდი:
ა. შესრულდება მხოლოდ catch ოპერატორში
ბ. არასოდეს არ შესრულდება
გ. ყოველთვის შესრულდება

checked და unchecked ოპერატორები

10.6.1. checked ოპერატორი გამოიყენება იმ:
ა. გამოსახულების მისათითებლად, რომლის შემოწმებაც უნდა მოხდეს გადავსების არსებობაზე
ბ. გამოსახულების მისათითებლად, რომლის შემოწმებაც არ უნდა მოხდეს გადავსების არსებობაზე
გ. ოპერატორის ბლოკის მისათითებლად, რომლის შემოწმებაც არ უნდა მოხდეს გადავსების არსებობაზე

10.6.2. checked ოპერატორი გამოიყენება იმ:
ა. ოპერატორების ბლოკის მისათითებლად, რომლის შემოწმებაც არ უნდა მოხდეს გადავსების არსებობაზე
ბ. ოპერატორების ბლოკის მისათითებლად, რომლის შემოწმებაც უნდა მოხდეს გადავსების არსებობაზე
გ. გამოსახულების მისათითებლად, რომლის შემოწმებაც არ უნდა მოხდეს გადავსების არსებობაზე

10.6.3. როგორია checked ოპერატორის სინტაქსი, რომელიც გამოსახულებას ამოწმებს:
ა. checked
ბ. checked { შესამოწმებელი ოპერატორები }
გ. checked (გამოსახულება)

10.6.4. როგორია checked ოპერატორის სინტაქსი, რომელიც ოპერატორების ბლოკს ამოწმებს:
ა. checked
ბ. checked { შესამოწმებელი ოპერატორები }
გ. checked (გამოსახულება)

10.6.5. checked ოპერატორი იწვევს შემდეგი განსაკუთრებული სიტუაციის გენერირებას:
ა. OverflowException
ბ. IndexOutOfRangeException
გ. DivideByZeroException

10.6.6. unchecked ოპერატორი გამოიყენება იმ:
ა. ოპერატორების ბლოკის მიმართ, რომლისთვისაც არ უნდა შესრულდეს გადავსების იგნორირება
ბ. გამოსახულების მიმართ, რომლისთვისაც არ უნდა შესრულდეს გადავსების იგნორირება
გ. გამოსახულების მიმართ, რომლისთვისაც უნდა შესრულდეს გადავსების

იგნორირება

10.6.7. unchecked ოპერატორი გამოიყენება იმ:
ა. ოპერატორების ბლოკის მიმართ, რომლისთვისაც არ უნდა შესრულდეს გადავსების იგნორირება
ბ. ოპერატორების ბლოკის მიმართ, რომლისთვისაც უნდა შესრულდეს გადავსების იგნორირება
გ. გამოსახულების მიმართ, რომლისთვისაც არ უნდა შესრულდეს გადავსების იგნორირება

10.6.8. თუ unchecked ოპერატორის გამოყენების შემთხვევაში ადგილი ექნა გადავსებას, მაშინ:
ა. შესრულდება შედეგის ჩამოჭრა
ბ. არ შესრულდება შედეგის ჩამოჭრა
გ. გენერირდება OverflowException განსაკუთრებული სიტუაცია

10.6.9. unchecked ოპერატორის გამოყენების შემთხვევაში შედეგის ჩამოჭრა ხდება იმ მიზნით, რომ:
ა. გამოსახულების ტიპი არ დაემთხვეს შედეგის ტიპს
ბ. გამოსახულების ტიპი დაემთხვეს შედეგის ტიპს
გ. შედეგის ტიპი გახდეს int

10.6.10. როგორია unchecked ოპერატორის სინტაქსი, რომელიც ახდენს გადავსების იგნორირებას გამოსახულებისათვის:
ა. unchecked
ბ. unchecked { შესამოწმებელი ოპერატორები }
გ. unchecked (გამოსახულება)

10.6.11. როგორია unchecked ოპერატორის სინტაქსი, რომელიც ახდენს გადავსების იგნორირებას ოპერატორების ბლოკისათვის:
ა. unchecked
ბ. unchecked { შესამოწმებელი ოპერატორები }
გ. unchecked (გამოსახულება)

თავი 11. დელეგატები, მოვლენები, ინტერფეისები და სახელების სივრცე

დელეგატები

11.1.1. დელეგატი არის ობიექტი, რომელიც:
ა. მეთოდს მიმართავს
ბ. მეთოდს არ მიმართავს
გ. ობიექტს მიმართავს

11.1.2. რომელი მსჯელობაა სწორი:
ა. დელეგატი არის ობიექტი

ბ. დელეგატი არ არის ობიექტი
გ. დელეგატი არის მეთოდი

11.1.3. დელეგატების გამოყენებით შესასრულებელი მეთოდის არჩევა ხდება:
ა. სტატიკურად
ბ. დინამიურად
გ. არადინამიურად

11.1.4. დელეგატები უზრუნველყოფენ:
ა. კლასებს
ბ. მასივებს
გ. მოვლენებს

11.1.5. რომელი მსჯელობაა სწორი:
ა. დელეგატი შეიცავს მეთოდის სიგნატურას
ბ. დელეგატი არ შეიცავს მეთოდის სიგნატურას
გ. დელეგატი შეიცავს მასივს

11.1.6. მეთოდის სიგნატურა არის:
ა. მეთოდის სახელი და პარამეტრების სია
ბ. მეთოდის მიერ დაბრუნებული მნიშვნელობის ტიპი
გ. მეთოდის სახელი, მის მიერ დაბრუნებული მნიშვნელობის ტიპი და პარამეტრების სია

11.1.7. რომელი მსჯელობაა სწორი:
ა. დელეგატი არ ინახავს მეთოდის მისამართს
ბ. დელეგატი ინახავს მეთოდის მისამართს
გ. დელეგატი ინახავს კლასის მისამართს

11.1.8. ერთი დელეგატი:
ა. შეგვიძლია გამოვიყენოთ რამდენიმე მეთოდის გამოძახებისათვის
ბ. არ შეგვიძლია გამოვიყენოთ რამდენიმე მეთოდის გამოძახებისათვის
გ. შეგვიძლია გამოვიყენოთ მხოლოდ ორი მეთოდის გამოძახებისათვის

11.1.9. ერთი დელეგატი:
ა. შეიძლება ინახავდეს რამდენიმე მეთოდის მისამართს
ბ. არ შეიძლება ინახავდეს რამდენიმე მეთოდის მისამართს
გ. შეიძლება ინახავდეს მხოლოდ ერთი მეთოდის მისამართს

11.1.10. დელეგატის გამოცხადებისათვის გამოიყენება:
ა. public სიტყვა
ბ. delegate სიტყვა
გ. abstract სიტყვა

11.1.11. დელეგატს შეუძლია გამოიძახოს მხოლოდ ის მეთოდი, რომლის:

ა. ტიპია int
ბ. სიგნატურა არ შეესაბამება დელეგატის სიგნატურას
გ. სიგნატურა შეესაბამება დელეგატის სიგნატურას

11.1.12. დელეგატის საშუალებით შეიძლება გამოვიძახოთ:
ა. მხოლოდ ობიექტის მეთოდი
ბ. ობიექტის მეთოდი ან კლასის სტატიკური მეთოდი
გ. მხოლოდ სტატიკური მეთოდი

11.1.13. მისამართების მიმდევრობით ინიციალიზების გზით დელეგატს:
ა. შეუძლია ერთმანეთის მიყოლებით გამოიძახოს შესაბამისი მეთოდები
ბ. არ შეუძლია ერთმანეთის მიყოლებით გამოიძახოს შესაბამისი მეთოდები
გ. შეუძლია ერთდროულად გამოიძახოს ყველა მეთოდი

11.1.14. დელეგატის მრავალმისამართიანობა არის მისი თვისება:
ა. არ შეინახოს რამდენიმე მისამართი
ბ. შეინახოს რამდენიმე მისამართი
გ. შეინახოს მხოლოდ 1 მისამართი

11.1.15. მეთოდებზე მიმართვების მწკრივი არის:
ა. გამოსაძახებელი მეთოდების მისამართების მიმდევრობა
ბ. მხოლოდ სტატიკური მეთოდების მისამართების მიმდევრობა
გ. არაგამოსაძახებელი მეთოდების მისამართების მიმდევრობა

11.1.16. მეთოდებზე მიმართვების მწკრივისათვის მისამართის დასამატებლად გამოიყენება:
ა. -= ოპერატორი
ბ. += ოპერატორი
გ. + ოპერატორი

11.1.17. მეთოდებზე მიმართვების მწკრივიდან მისამართის წასაშლელად გამოიყენება:
ა. += ოპერატორი
ბ. -= ოპერატორი
გ. - ოპერატორი

11.1.18. დელეგატს, რომელიც რამდენიმე მიმართვას ინახავს, უნდა ჰქონდეს დასაბრუნებელი მნიშვნელობის ტიპი:
ა. double
ბ. int
გ. void

მოვლენები

11.2.1. რომელი მსჯელობაა სწორი:

ა. მოვლენა არის სტატიკური მეთოდის რეალიზება
ბ. მოვლენა არ არის ავტომატური უწყება რაიმე მომხდარ მოქმედებაზე
გ. მოვლენა არის ავტომატური უწყება რაიმე მომხდარ მოქმედებაზე

11.2.2. მოვლენის მაგალითებია:
ა. კლავიატურის კლავიშზე დაჭერა
ბ. მასივის განულება
გ. ცვლადის ინიციალიზაცია

11.2.3. მოვლენის დამამუშავებელი იქმნება:
ა. მოვლენის საფუძველზე
ბ. დელეგატის საფუძველზე
გ. მასივის საფუძველზე

11.2.4. მოვლენის გამოცხადებისათვის გამოიყენება:
ა. delegate სიტყვა
ბ. event სიტყვა
გ. string სიტყვა

11.2.5. მოვლენის დამამუშავებელი აქტიურდება:
ა. დელეგატის საშუალებით
ბ. მოვლენის საშუალებით
გ. კლასის საშუალებით

11.2.6. რომელი მსჯელობაა სწორი:
ა. მოვლენა არ შეიძლება იყოს მრავალმისამართიანი
ბ. მოვლენა შეიძლება იყოს მრავალმისამართიანი
გ. მოვლენა შეიძლება იყოს მხოლოდ ერთმისამართიანი

11.2.7. მოვლენის დამამუშავებლისათვის უნდა მიეთითოს დასაბრუნებელი მნიშვნელობის ტიპი:
ა. bool
ბ. int
გ. void

11.2.8. ერთ მოვლენას შეიძლება ჰქონდეს:
ა. მხოლოდ 1 დამამუშავებელი
ბ. რამდენიმე დამამუშავებელი
გ. მხოლოდ 2 დამამუშავებელი

11.2.9. მოვლენას, რომლის დამამუშავებლების ნაწილი შეიძლება განსაზღვრული იყოს სხვადასხვა ობიექტებში:
ა. ფართოსამაუწყებლო ეწოდება
ბ. სტატიკური ეწოდება
გ. ვირტუალური ეწოდება

11.2.10. ფართოსამაუწყებლო მოვლენა ბევრ ობიექტს საშუალებას:
ა. აძლევს რეაგირება არ მოახდინოს მოვლენის შესახებ უწყებაზე
ბ. არ აძლევს რეაგირება მოახდინოს მოვლენის შესახებ უწყებაზე
გ. აძლევს რეაგირება მოახდინოს მოვლენის შესახებ უწყებაზე

11.2.11. მოვლენის შესახებ უწყების მისაღებად წინასწარ:
ა. არ უნდა იყოს რეგისტრირებული ამ მოვლენის დამამუშავებელი თითოეული ობიექტისათვის
ბ. უნდა იყოს რეგისტრირებული ამ მოვლენის დამამუშავებელი ობიექტის გარეშე
გ. უნდა იყოს რეგისტრირებული ამ მოვლენის დამამუშავებელი თითოეული ობიექტისათვის

11.2.12. რადგან მოვლენის დამამუშავებელი ცალ-ცალკე რეგისტრირდება თითოეული ობიექტისათვის, ამიტომ:
ა. ყველა ობიექტი ერთდროულად იღებს უწყებას მომხდარი მოვლენის შესახებ
ბ. თითოეული ობიექტი ცალკე იღებს უწყებას მომხდარი მოვლენის შესახებ
გ. არც ერთი ობიექტი არ იღებს უწყებას მომხდარი მოვლენის შესახებ

ინტერფეისები

11.3.1. ინტერფეისში შეიძლება განისაზღვროს:
ა. ინდექსატორები და მასივები
ბ. მეთოდები და სტრიქონები
გ. მეთოდები, თვისებები, ინდექსატორები და მოვლენები

11.3.2. კლასის ინტერფეისი ამ კლასის რეალიზაციისაგან შეიძლება:
ა. მთლიანად განვაცალკევოთ
ბ. ნაწილობრივ განვაცალკევოთ
გ. ვერ განვაცალკევოთ

11.3.3. ინტერფეისის რეალიზება:
ა. შეუძლებელია რამდენიმე კლასის საშუალებით
ბ. შესაძლებელია რამდენიმე კლასის საშუალებით
გ. შესაძლებელია მხოლოდ 1 კლასის საშუალებით

11.3.4. კლასი, რომელიც ახდენს ინტერფეისის რეალიზებას:
ა. არ შეიძლება შეიცავდეს დამატებით ელემენტებს
ბ. შეიძლება შეიცავდეს დამატებით ელემენტებს
გ. ნაწილობრივ შეიძლება შეიცავდეს დამატებით ელემენტებს

11.3.5. ინტერფეისში არ შეიძლება გამოვაცხადოთ:
ა. ველები (ცვლადები)
ბ. მეთოდები
გ. ინტერფეისები

11.3.6. ინტერფეისში მეთოდების გამოცხადება სრულდება:
ა. კლასის საშუალებით
ბ. მასივის საშუალებით
გ. მათი სიგნატურისა და დასაბრუნებელი მნიშვნელობის ტიპის მითითების გზით

11.3.7. ინტერფეისში გამოცხადებული მეთოდებისათვის იგულისხმება:
ა. protected მოდიფიკატორი
ბ. private მოდიფიკატორი
გ. public მოდიფიკატორი

11.3.8. ინტერფეისში არ შეიძლება გამოვაცხადოთ:
ა. ინდექსატორები
ბ. სტატიკური წევრები
გ. მოვლენები

11.3.9. ინტერფეისში არ შეიძლება გამოვაცხადოთ:
ა. თვისებები
ბ. მოვლენები
გ. კონსტრუქტორები

11.3.10. ინტერფეისში არ შეიძლება გამოვაცხადოთ:
ა. ინდექსატორები
ბ. დესტრუქტორები
გ. მოვლენები

11.3.11. კლასმა ინტერფეისის რეალიზება უნდა მოახდინოს:
ა. მთლიანად
ბ. ნაწილობრივ
გ. მიმდევრობით

11.3.12. ერთ კლასს შეუძლია:
ა. მხოლოდ ერთი ინტერფეისის რეალიზება
ბ. ერთზე მეტი ინტერფეისის რეალიზება
გ. მხოლოდ ორი ინტერფეისის რეალიზება

11.3.13. ინტერფეისული ცვლადი შეიძლება მიმართავდეს ნებისმიერ ობიექტს, რომელიც:
ა. ახდენს სხვა ინტერფეისის რეალიზებას
ბ. არ ახდენს მისი ინტერფეისის რეალიზებას
გ. ახდენს მისი ინტერფეისის რეალიზებას

11.3.14. ინტერფეისში თვისების განსაზღვრისას:
ა. ეთითება თვისების ტანი
ბ. არ ეთითება თვისების ტანი

გ. ეთითება ინდექსატორი

11.3.15. ცხადი რეალიზაციის დროს:
ა. ინტერფეისის ელემენტის წინ ეთითება ინტერფეისის სახელი
ბ. ინტერფეისის ელემენტის წინ არ ეთითება ინტერფეისის სახელი
გ. ინტერფეისის ელემენტის წინ ეთითება კლასის სახელი

11.3.16. ცხადი რეალიზაციის გამოყენება საჭიროა მაშინ, როცა:
ა. პროგრამაში გამოცხადებულია ორი ინტერფეისი, რომლებიც არ შეიცავენ ერთნაირი სახელის მქონე მეთოდებს
ბ. პროგრამაში გამოცხადებულია ორი ინტერფეისი, რომლებიც შეიცავენ ერთნაირი სახელის მქონე მეთოდებს
გ. პროგრამაში გამოცხადებულია სამი ინტერფეისი, რომლებიც არ შეიცავენ ერთნაირი სახელის მქონე მეთოდებს

სახელების სივრცე

11.4.1. სახელების სივრცე კლასების სახელებს შორის კონფლიქტს:
ა. თავიდან გვაცილებს
ბ. თავიდან არ გვაცილებს
გ. იწვევს

11.4.2. სახელების სივრცეში სახელების ერთი ნაკრები:
ა. არ შეგვიძლია შევინახოთ სახელების სხვა ნაკრებებისაგან ცალკე
ბ. შეგვიძლია შევინახოთ სახელების სხვა ნაკრებებისაგან ცალკე
გ. შეგვიძლია შევინახოთ სახელების სხვა ნაკრებებთან ერთად

11.4.3. სახელების ერთ სივრცეში გამოცხადებული სახელები:
ა. არ არიან კონფლიქტში სახელების სხვა სივრცეში გამოცხადებულ ასეთივე სახელებთან
ბ. არიან კონფლიქტში სახელების სხვა სივრცეში გამოცხადებულ ასეთივე სახელებთან
გ. არ არიან კონფლიქტში სახელების ამავე სივრცეში გამოცხადებულ ასეთივე სახელებთან

11.4.4. სახელების სივრცეში განსაზღვრული ყველა წევრი:
ა. ნაწილობრივ იმყოფება ამ სახელების სივრცის მხედველობის უბნის საზღვრებში
ბ. არ იმყოფება ამ სახელების სივრცის მხედველობის უბნის საზღვრებში
გ. იმყოფება ამ სახელების სივრცის მხედველობის უბნის საზღვრებში

11.4.5. სახელების სივრცეში შეიძლება განსაზღვრული იყოს:
ა. ჩამოთვლები და მასივები
ბ. კლასები და დელეგატები
გ. ინტერფეისები და ცვლადები

11.4.6. თუ ორი სახელების სივრცე შეიცავს ერთნაირი სახელის მქონე კლასს, მაშინ იმისათვის, რომ ეს ორი კლასი ერთმანეთისაგან განვასხვავოთ:
ა. საჭირო არ არის კლასის სახელის წინ სახელების სივრცის მითითება
ბ. საჭიროა კლასის სახელის წინ სახელების სივრცის მითითება
გ. საჭიროა მხოლოდ კლასის სახელის მითითება

11.4.7. სახელების სივრცე პროგრამაში ხილული ხდება:
ა. using დირექტივაში მისი მითითების შემდეგ
ბ. მასივში მისი გამოცხადების შემდეგ
გ. public მოდიფიკატორის მითითების შემდეგ

11.4.8. using დირექტივა მითითებული უნდა იყოს:
ა. პროგრამის შუაში
ბ. პროგრამის ბოლოში
გ. პროგრამის დასაწყისში

11.4.9. თუ პროგრამაში გამოცხადებულია ორი სახელების სივრცე, მაშინ:
ა. სახელების პირველი სივრცე არ მალავს სახელების მეორე სივრცეს
ბ. სახელების პირველი სივრცე მალავს სახელების მეორე სივრცეს
გ. სახელების მეორე სივრცე მალავს სახელების პირველი სივრცეს

11.4.10. სახელების სივრცის გამოცხადებისას იგი:
ა. თავის სახელებს არ უმატებს უკვე არსებულ სახელებს
ბ. თავის სახელებს უმატებს უკვე არსებულ სახელებს
გ. ფარავს უკვე არსებულ სახელებს

11.4.11. სახელების ერთი სივრცე:
ა. შეიძლება ნაწილობრივ მოვათავსოთ სახელების მეორე სივრცის შიგნით
ბ. არ შეიძლება მოვათავსოთ სახელების მეორე სივრცის შიგნით
გ. შეიძლება მოვათავსოთ სახელების მეორე სივრცის შიგნით

11.4.12. სახელების სივრცის იერარქია:
ა. შეიძლება გადანაწილებული იყოს სხვადასხვა პროგრამაში
ბ. არ შეიძლება გადანაწილებული იყოს სხვადასხვა პროგრამაში
გ. ნაწილობრივ შეიძლება გადანაწილებული იყოს სხვადასხვა პროგრამაში

11.4.13. თუ პროგრამაში არ არის გამოცხადებული სახელების სივრცე, მაშინ:
ა. არ გამოიყენება ავტომატურად განსაზღვრული სახელების სივრცე
ბ. გამოიყენება ავტომატურად განსაზღვრული სახელების სივრცე
გ. არც ერთი სახელების სივრცე არ გამოიყენება

11.4.14. ავტომატურად განსაზღვრული სახელების სივრცე:
ა. არასოდეს არ გამოიყენება
ბ. დიდი და რთული პროგრამებისათვის
გ. გამოიყენება მოკლე და მარტივი პროგრამებისათვის

თავი 12. სტრიქონები

სტრიქონები

12.1.1. სტრიქონები წარმოადგენენ:
ა. მიმართვითი ტიპის მქონე ობიექტებს
ბ. ჩვეულებრივი ტიპის მქონე ობიექტებს
გ. მხოლოდ ლოგიკურ მონაცემებს

12.1.2. სტრიქონი არის:
ა. სიმბოლოების მასივი
ბ. მთელრიცხვა მასივი
გ. წილადების მასივი

12.1.3. სტრიქონებში პირველი ელემენტის ინდექსია:
ა. 1
ბ. 0
გ. -1

12.1.4. სტრიქონებს Length თვისება:
ა. არ აქვთ
ბ. აქვთ
გ. ხანდახან აქვთ

12.1.5. სტრიქონის ცალკეული სიმბოლოს მისაღებად უნდა მივუთითოთ:
ა. სტრიქონის სახელი და ინდექსი
ბ. მხოლოდ ინდექსი
გ. მხოლოდ სახელი

12.1.6. Insert მეთოდი:
ა. გამომძახებელ სტრიქონში წაშლის მითითებულ სტრიქონს
ბ. გამომძახებელ სტრიქონში შეცვლის მითითებულ სტრიქონს
გ. გამომძახებელ სტრიქონში ჩასვამს მითითებულ სტრიქონს

12.1.7. Remove მეთოდი:
ა. გამომძახებელ სტრიქონში ჩასვამს მითითებული რაოდენობის სიმბოლოს
ბ. გამომძახებელ სტრიქონში შეცვლის მითითებული რაოდენობის სიმბოლოს
გ. გამომძახებელ სტრიქონში წაშლის მითითებული რაოდენობის სიმბოლოს

12.1.8. Replace მეთოდი:
ა. გამომძახებელ სტრიქონში ჩასვამს მითითებულ სტრიქონს
ბ. გამომძახებელ სტრიქონში წაშლის მითითებულ სტრიქონს
გ. გამომძახებელ სტრიქონში მითითებულ სტრიქონს ახალი სტრიქონით შეცვლის

12.1.9. + ოპერატორი გამოიყენება:
ა. ერთ სტრიქონში მეორე სტრიქონის ჩასასმელად
ბ. სტრიქონების კონკატენაციის ოპერაციის შესასრულებლად
გ. ერთი სტრიქონიდან ქვესტრიქონის წასაშლელად

12.1.10. LastIndexOf მეთოდი:
ა. გასცემს გამომძახებელი სტრიქონის ინდექსს, რომელშიც უკანასკნელად იყო ნაპოვნი მისი არგუმენტი
ბ. გასცემს გამომძახებელი სტრიქონის ინდექსს, რომელშიც პირველად იყო ნაპოვნი მისი არგუმენტი
გ. ადარებს გამომძახებელ სტრიქონსა და თავის არგუმენტს

12.1.11. IndexOf მეთოდი:
ა. გამომძახებელი სტრიქონის მითითებული პოზიციიდან გასცემს მითითებული რაოდენობის სიმბოლოსაგან შემდგარ ქვესტრიქონს
ბ. გასცემს გამომძახებელი სტრიქონის ინდექსს, რომელშიც პირველად იყო ნაპოვნი მისი არგუმენტი
გ. გასცემს გამომძახებელი სტრიქონის ინდექსს, რომელშიც უკანასკნელად იყო ნაპოვნი მისი არგუმენტი

12.1.12. CompareTo მეთოდი:
ა. გასცემს გამომძახებელი სტრიქონის ინდექსს, რომელშიც უკანასკნელად იყო ნაპოვნი მისი არგუმენტი
ბ. გასცემს გამომძახებელი სტრიქონის ინდექსს, რომელშიც პირველად იყო ნაპოვნი მისი არგუმენტი
გ. ადარებს გამომძახებელ სტრიქონსა და თავის არგუმენტს

12.1.13. Copy მეთოდი:
ა. გასცემს გამომძახებელი სტრიქონის ინდექსს, რომელშიც პირველად იყო ნაპოვნი მისი არგუმენტი
ბ. გამომძახებელი სტრიქონის მითითებული პოზიციიდან გასცემს მითითებული რაოდენობის სიმბოლოსაგან შემდგარ ქვესტრიქონს
გ. გამომძახებელ სტრიქონს ანიჭებს თავისი არგუმენტის ასლს

12.1.14. Substring მეთოდი:
ა. გამომძახებელი სტრიქონის მითითებული პოზიციიდან გასცემს მითითებული რაოდენობის სიმბოლოსაგან შემდგარ ქვესტრიქონს
ბ. ადარებს გამომძახებელ სტრიქონსა და თავის არგუმენტს
გ. გამომძახებელ სტრიქონს ანიჭებს თავისი არგუმენტის ასლს

12.1.15. სტრიქონების მასივის გამოცხადებისათვის string სიტყვის შემდეგ ეთითება:
ა. მრგვალი ფრჩხილები
ბ. კვადრატული ფრჩხილები
გ. ფიგურული ფრჩხილები

12.1.16. გვაქვს სტრიქონი string striqoni = “C# დაპროგრამების ენაა”; დასაშვებია თუ არა ასეთი სახის მინიჭება striqoni[1] = ‘A’; :

ა. კი

ბ. არა

გ. ხანდახან

თავი 17. შესავალი მონაცემთა ბაზებში

მონაცემთა ბაზების აგების პრინციპები

17.1.1. მონაცემთა ბაზა არის:

ა. ცხრილების ერთობლიობა

ბ. ფაილების ერთობლიობა

გ. კატალოგების ერთობლიობა

17.1.2. მონაცემთა ბაზა არის:

ა. მონაცემების არაორგანიზებული ერთობლიობა

ბ. მონაცემების ორგანიზებული ერთობლიობა

გ. მონაცემების ნახევრად ორგანიზებული ერთობლიობა

17.1.3. ცხრილი არის:

ა. კატალოგების ერთობლიობა

ბ. ფაილების ერთობლიობა

გ. ჩანაწერების ერთობლიობა

17.1.4. ჩანაწერი არის:

ა. სიმბოლოების ერთობლიობა

ბ. ველების ერთობლიობა

გ. რიცხვების ერთობლიობა

17.1.5. ველი შეიცავს:

ა. მონაცემებს

ბ. ფაილებს

გ. მხოლოდ რიცხვებს

17.1.6. ცხრილის ერთი ჩანაწერი შეესაბამება:

ა. რამდენიმე ობიექტს

ბ. ორ ობიექტს

გ. ერთ ობიექტს

17.1.7. ველის სახელი უნდა იწყებოდეს:

ა. ხაზგასმის სიმბოლოთი

ბ. ციფრით

გ. ასოთი

17.1.8. პირველადი გასაღების მნიშვნელობა:
ა. უნდა იყოს უნიკალური
ბ. არ უნდა იყოს უნიოკალური
გ. ხანდახან უნდა იყოს უნიკალური

17.1.9. პირველადი გასაღები თითოეულ ჩანაწერს:
ა. ცალსახად არ განსაზღვრავს
ბ. ცალსახად განსაზღვრავს
გ. ცალსახად ნაწილობრივ განსაზღვრავს

17.1.10. პირველადი გასაღები მონაცემთა ბაზაში:
ა. უზრუნველყოფს ინფორმაციის არაწინააღმდეგობრიობას
ბ. არ უზრუნველყოფს ინფორმაციის არაწინააღმდეგობრიობას
გ. უზრუნველყოფს ინფორმაციის წინააღმდეგობრიობას

17.1.11. ბმულ ცხრილებში:
ა. ორივე მთავარი ცხრილია
ბ. ერთი ცხრილი მთავარია, მეორე კი დეტალების
გ. ორივე დეტალების ცხრილია

17.1.12. მონაცემთა რელაციური ბაზა არის:
ა. ბმული ცხრილების ერთობლიობა
ბ. ბმული ველების ერთობლიობა
გ. ბმული ჩანაწერების ერთობლიობა

17.1.13. ჩანაწერი შედგება:
ა. ცხრილებისაგან
ბ. ჩანაწერებისაგან
გ. ველებისაგან

17.1.14. მონაცემთა ბაზის მართვის სისტემა არის:
ა. პროგრამა, რომელიც მართავს მონაცემთა ბაზაში ინფორმაციის ჩამატებას
ბ. პროგრამული უზრუნველყოფა, რომელიც მართავს მონაცემთა ბაზაში ინფორმაციის მხოლოდ შეცვლას
გ. პროგრამული უზრუნველყოფა, რომელიც მართავს მონაცემთა ბაზაში ინფორმაციის ჩამატებას, შეცვლას, წაშლას და ა.შ.

17.1.15. პირველადი გასაღები არის:
ა. ველი, რომელიც ცალსახად არ განსაზღვრავს თითოეულ ჩანაწერს ცხრილში
ბ. ველი, რომელიც ცალსახად განსაზღვრავს თითოეულ ჩანაწერს ცხრილში
გ. ველი, რომელიც ცალსახად განსაზღვრავს რამდენიმე ჩანაწერს ცხრილში

17.1.16. შეიძლება თუ არა გამეორდეს პირველადი გასაღების მნიშვნელობა:
ა. არა

ბ. კი
გ. ხანდახან

17.1.17. კავშირი ერთი-ბევრთან ნიშნავს იმას, რომ:
ა. პირველი ცხრილის ერთი ჩანაწერი მიმართავს მეორე ცხრილის რამდენიმე ჩანაწერს
ბ. პირველი ცხრილის რამდენიმე ჩანაწერი მიმართავს მეორე ცხრილის რამდენიმე ჩანაწერს
გ. პირველი ცხრილის ერთი ჩანაწერი მიმართავს მეორე ცხრილის ერთ ჩანაწერს

17.1.18. რის საშუალებით ხდება ცხრილებს შორის ერთი-ბევრთან კავშირის დამყარება:
ა. შიდა გასაღების საშუალებით
ბ. გარე გასაღების საშუალებით
გ. რომელიმე ჩანაწერის საშუალებით

17.1.19. დეტალების ცხრილი ეწოდება ცხრილს, რომელიც:
ა. მხოლოდ პირველად გასაღებს შეიცავს
ბ. გარე გასაღებს არ შეიცავს
გ. გარე გასაღებს შეიცავს

17.1.20. მთავარი ცხრილი ეწოდება ცხრილს, რომელსაც:
ა. გარე გასაღები მიმართავს
ბ. პირველადი გასაღები მიმართავს
გ. გარე გასაღები არ მიმართავს

17.1.21. კავშირს განხორციელებულს გარე გასაღების საშუალებით ეწოდება:
ა. მთავარი-შვილი
ბ. მთავარი-დეტალი
გ. მშობელი-დეტალი

17.1.22. ბმული ცხრილების ერთობლიობას ეწოდება:
ა. მონაცემთა რელაციური ბაზა
ბ. მონაცემთა ლოგიკური ბაზა
გ. მონაცემთა ფიზიკური ბაზა

17.1.23. ორი ცხრილი ერთმანეთს უკავშირდება:
ა. როგორც გარე, ისე პირველადი გასაღების საშუალებით
ბ. პირველადი გასაღების საშუალებით
გ. გარე გასაღების საშუალებით

17.1.24. რომელი განმარტებაა სწორი:
ა. დეტალების ცხრილის გარე გასაღები უკავშირდება მთავარი ცხრილის პირველად გასაღებს
ბ. დეტალების ცხრილის პირველადი გასაღები უკავშირდება მთავარი ცხრილის

გარე გასაღებს
გ. დეტალების ცხრილის პირველადი გასაღები უკავშირდება მთავარი ცხრილის პირველად გასაღებს

17.1.25. "მთავარი-დეტალი" კავშირი ეწოდება კავშირს განხორციელებულს:
ა. როგორც გარე, ისე პირველადი გასაღების საშუალებით
ბ. გარე გასაღების საშუალებით
გ. პირველადი გასაღების საშუალებით

17.1.26. მონაცემთა ბაზასთან მუშაობა ხორციელდება:
ა. საოფისე პროგრამების საშუალებით
ბ. ოპერაციული სისტემის საშუალებით
გ. მონაცემთა ბაზის მართვის სისტემის საშუალებით

17.1.27. მონაცემთა ბაზების მართვის სისტემებია:
ა. Microsoft SQL, Oracle, Paradox, Access
ბ. Word, Excel, PowerPoint
გ. MsPaint, Winrar

17.1.28. Paradox და dBase სისტემები მონაცემთა ბაზას ინახავენ:
ა. ფაილში
ბ. კატალოგში
გ. სერვერზე

17.1.29. Access და InterBaseსისტემები მონაცემთა ბაზას ინახავენ:
ა. სერვერზე
ბ. კატალოგში
გ. ფაილში

17.1.30. Microsoft SQL, Oracle და SyBase სისტემები მონაცემთა ბაზას ინახავენ:
ა. სერვერზე
ბ. ფაილში
გ. კატალოგზე

17.1.31. მონაცემთა ბაზების მოდელებია:
ა. სტრუქტურული, ლოგიკური, ფიზიკური
ბ. ავტონომური, ფაილ/სერვერი, კლიენტ/სერვერი, მრავალდონიანი
გ. ავტონომური, სტრუქტურული, ფიზიკური

17.1.32. ავტონომური მონაცემთა ბაზა ინახება:
ა. რამდენიმე კომპიუტერზე
ბ. სერვერზე
გ. ლოკალურ კომპიუტერზე

17.1.33. ავტონომურ მონაცემთა ბაზასთან მუშაობისას:

ა. არ გამოიყენება ქსელი
ბ. გამოიყენება ქსელი
გ. გამოიყენება სერვერი

17.1.34. ფაილ/სერვერული მონაცემთა ბაზა ინახება:
ა. ლოკალურ კომპიუტერზე
ბ. სერვერზე
გ. როგორც ლოკალურ კომპიუტერზე, ისე სერვერზე

17.1.35. ფაილ/სერვერული მონაცემთა ბაზების ნაკლია:
1. ქსელის არამწარმოებლური დატვირთვა
2. მონაცემების მთლიანობას უზრუნველყოფს კლიენტის პროგრამა
3. მონაცემების მთლიანობას უზრუნველყოფს სერვერი
ა. 2, 3
ბ. 1, 3
გ. 1,2

17.1.36. ფაილ/სერვერული მონაცემთა ბაზებში ერთი ჩანაწერის შეცვლისას სრულდება:
ა. ყველა ჩანაწერის გაახლება
ბ. ერთი ჩანაწერის გაახლება
გ. რამდენიმე ჩანაწერის გაახლება

17.1.37. კლიენტ/სერვერულ მოდელში მონაცემთა ბაზასთან მიმართვა ხდება:
ა. როგორც კლიენტის, ისე სერვერის საშუალებით
ბ. კლიენტის საშუალებით
გ. სერვერის საშუალებით

17.1.38. რომელი მსჯელობაა სწორი:
1. კლიენტ/სერვერული მოდელი განტვირთავს ქსელს
2. კლიენტ/სერვერული მოდელი უზრუნველყოფს მონაცემების მთლიანობას
3. კლიენტ/სერვერული მოდელი არ განტვირთავს ქსელს :
ა. 1,3
ბ. 1,2
გ. 2,3

17.1.39. მრავალდონიანი მონაცემთა ბაზების რამდენდონიანი ვარიანტია გავრცელებული:
ა. ერთდონიანი
ბ. სამდონიანი
გ. ხუთდონიანი

17.1.40. მრავალდონიანი მონაცემთა ბაზების პირველ დონეზე მოთავსებულია:
ა. კლიენტების პროგრამები
ბ. პროგრამების სერვერი

გ. მონაცემთა ბაზების დაშორებული სერვერი

17.1.41. მრავალდონიანი მონაცემთა ბაზების მეორე დონეზე მოთავსებულია:
ა. მონაცემთა ბაზების დაშორებული სერვერი
ბ. კლიენტების პროგრამები
გ. პროგრამების სერვერი

17.1.42. მრავალდონიანი მონაცემთა ბაზების მესამე დონეზე მოთავსებულია:
ა. მონაცემთა ბაზების დაშორებული სერვერი
ბ. პროგრამების სერვერი
გ. კლიენტების პროგრამები

17.1.43. შენახული პროცედურა:
ა. ინახება და სრულდება კლიენტის კომპიუტერზე მოთავსებულ მონაცემთა ბაზაში
ბ. ინახება და სრულდება ლოკალურ კომპიუტერზე მოთავსებულ მონაცემთა ბაზაში
გ. ინახება და სრულდება სერვერზე მოთავსებულ მონაცემთა ბაზაში

17.1.44. SQL-სერვერზე შენახული პროცედურა იწერება:
ა. C# ენაზე
ბ. Transact_SQL ენაზე
გ. J# ენაზე

თავი18. SQL-სერვერი

SQL-მოთხოვნის ტიპები

18.1.1. DML არის
ა. მონაცემების დამუშავების ენა
ბ. მონაცემების განსაზღვრის ენა
გ. დაპროგრამების მაღალი დონის ენა

18.1.2. DML მოთხოვნის რამდენი ტიპი არსებობს:
ა. 5
ბ. 3
გ. 4

18.1.3. SQL არის
ა. სტანდარტული ენა მონაცემთა რელაციურ ბაზებთან სამუშაოდ
ბ. დაპროგრამების მაღალი დონის ენა
გ. დაპროგრამების დაბალი დონის ენა

18.1.4. DDL არის:
ა. მონაცემების დამუშავების ენა

ბ. დაპროგრამების მაღალი დონის ენა
გ. მონაცემების განსაზღვრის ენა

18.1.5. რა ფუნქციებს ასრულებენ DML მოთხოვნები:
ა. მონაცემთა ბაზაში მონაცემების მხოლოდ დამატება
ბ. მონაცემთა ბაზაში მონაცემების დამატება, წაშლა და შეცვლა
გ. უზრუნველყოფენ მონაცემთა ბაზაში ცხრილების შექმნის საშუალებებს

18.1.6. რა ფუნქციებს ასრულებენ DDL მოთხოვნები:
ა. უზრუნველყოფენ მონაცემთა ბაზაში ცხრილების შექმნის საშუალებებს
ბ. მონაცემთა ბაზაში მონაცემების წაშლა
გ. მონაცემთა ბაზაში მონაცემების დამატება, წაშლა და შეცვლა

18.1.7. ქვემოთ ჩამოთვლილიდან რომელი მიეკუთვნება DML-მოთხოვნებს:
ა. SELECT, BETWEEN, IS NULL, DELETE
ბ. INSERT, UPDATE, WHERE, ORDER BY
გ. SELECT, INSERT, UPDATE, DELETE

მონაცემთა ტიპები

18.1.8. bigint ტიპი არის:
ა. მთელი რიცხვი დიაპაზონში $-2^{63} \div 2^{63}-1$
ბ. მთელი რიცხვი დიაპაზონში $-2^{31} \div 2^{31}-1$
გ. მცურავწერტილიანი მნიშვნელობა დიაპაზონში -3,40E+38 ÷ 3,40E+38

18.1.9. int ტიპი არის:
ა. მთელი რიცხვი დიაპაზონში $-2^{63} \div 2^{63}-1$
ბ. მთელი რიცხვი დიაპაზონში $-2^{31} \div 2^{31}-1$
გ. მცურავწერტილიანი მნიშვნელობა დიაპაზონში -1,79E+308 ÷ 1,79E+308

18.1.10. datetime ტიპი არის:
ა. ცვლადი სიგრძის ორობითი მონაცემები $2^{31}-1$ ბაიტამდე
ბ. მთელი რიცხვი მნიშვნელობით 0 ან 1
გ. თარიღი და დრო 1753 წლის 1 იანვრიდან 9999 წლის 31 დეკემბრამდე

18.1.11. float ტიპი არის:
ა. მთელი რიცხვი დიაპაზონში $-2^{31} \div 2^{31}-1$
ბ. მცურავწერტილიანი მნიშვნელობა დიაპაზონში -1,79E+308 ÷ 1,79E+308
გ. მცურავწერტილიანი მნიშვნელობა დიაპაზონში -3,40E+38 ÷ 3,40E+38

18.1.12. char ტიპი არის:
ა. ფიქსირებული სიგრძის სტრიქონი 8000 სიმბოლომდე
ბ. მთელი რიცხვი დიაპაზონში $-2^{63} \div 2^{63}-1$
გ. მცურავწერტილიანი მნიშვნელობა დიაპაზონში -1,79E+308 ÷ 1,79E+308

18.1.13. nvarchar ტიპი არის:

ა. ცვლადი სიგრძის Unicode სიმბოლოების სტრიქონი 4000 სიმბოლომდე

ბ. ცვლადი სიგრძის ორობითი მონაცემები 8000 ბაიტამდე

გ. ცვლადი სიგრძის ორობითი მონაცემები 2^{31} -1 ბაიტამდე

18.1.14. image ტიპი არის:

ა. ცვლადი სიგრძის ორობითი მონაცემები 8000 ბაიტამდე

ბ. ცვლადი სიგრძის ორობითი მონაცემები 2^{31} -1 ბაიტამდე

გ. ფიქსირებული სიგრძის ორობითი მონაცემები 8000 ბაიტამდე

18.1.15. smallint ტიპი არის:

ა. ცვლადი სიგრძის ორობითი მონაცემები 2^{31}-1 ბაიტამდე

ბ. მთელი რიცხვი მნიშვნელობით 0 ან 1

გ. მთელი რიცხვი დიაპაზონში $-2^{15} \div 2^{15}-1$

18.1.16. tinyint ტიპი არის:

ა. მთელი რიცხვი მნიშვნელობით 0 ან 1

ბ. მთელი რიცხვი დიაპაზონში 0÷255

გ. მთელი რიცხვი დიაპაზონში $-2^{63} \div 2^{63}-1$

18.1.17. bit ტიპი არის:

ა. მთელი რიცხვი მნიშვნელობით 0 ან 1

ბ. ფულის ერთეული მნიშვნელობით $-2^{63} \div 2^{63}-1$

გ. მთელი რიცხვი დიაპაზონში 0÷255

18.1.18. money ტიპი არის:

ა. მთელი რიცხვი დიაპაზონში $-2^{15} \div 2^{15}-1$

ბ. მცურავწერტილიანი მნიშვნელობა დიაპაზონში -3,40E+38÷3,40E+38

გ. ფულის ერთეული მნიშვნელობით $-2^{63} \div 2^{63}-1$

18.1.19. small money ტიპი არის:

ა. ცვლადი სიგრძის ორობითი მონაცემები 2^{31}-1 ბაიტამდე

ბ. ფულის ერთეული მნიშვნელობით $-2^{31} \div 2^{31}-1$

გ. მთელი რიცხვი დიაპაზონში $-2^{15} \div 2^{15}-1$

18.1.20. real ტიპი არის:

ა. ფულის ერთეული მნიშვნელობით $-2^{63} \div 2^{63}-1$

ბ. ცვლადი სიგრძის ორობითი მონაცემები 2^{31}-1 ბაიტამდე

გ. მცურავწერტილიანი მნიშვნელობა დიაპაზონში -3,40E+38÷3,40E+38

18.1.21. smalldatetime ტიპი არის:

ა. თარიღი და დრო 1900 წლის 1 იანვრიდან 2079 წლის 6 ივნისამდე

ბ. თარიღი და დრო 1753 წლის 1 იანვრიდან 9999 წლის 31 დეკემბრამდე

გ. ფიქსირებული სიგრძის ორობითი მონაცემები 8000 ბაიტამდე

18.1.22. varchar ტიპი არის:
ა. ცვლადი სიგრძის სტრიქონი 8000 სიმბოლომდე
ბ. ფიქსირებული სიგრძის სტრიქონი 8000 სიმბოლომდე
გ. მთელი რიცხვი დიაპაზონში $-2^{63} \div 2^{63}-1$

18.1.23. text ტიპი არის:
ა. ცვლადი სიგრძის Unicode სიმბოლოების სტრიქონი $2^{31}-1$ სიმბოლომდე
ბ. ცვლადი სიგრძის სტრიქონი $2^{31}-1$ სიმბოლომდე
გ. ფიქსირებული სიგრძის სტრიქონი 8000 სიმბოლომდე

18.1.24. ntext ტიპი არის:
ა. ცვლადი სიგრძის სტრიქონი 8000 სიმბოლომდე
ბ. ფიქსირებული სიგრძის ორობითი მონაცემები 8000 ბაიტამდე
გ. ცვლადი სიგრძის Unicode სიმბოლოების სტრიქონი $2^{31}-1$ სიმბოლომდე

18.1.25. nchar ტიპი არის:
ა. ცვლადი სიგრძის სტრიქონი $2^{31}-1$ სიმბოლომდე
ბ. ფიქსირებული სიგრძის Unicode სიმბოლოების სტრიქონი 4000 სიმბოლომდე
გ. ცვლადი სიგრძის სტრიქონი 8000 სიმბოლომდე

18.1.26. binary ტიპი არის:
ა. ფიქსირებული სიგრძის ორობითი მონაცემები 8000 ბაიტამდე
ბ. ცვლადი სიგრძის ორობითი მონაცემები $2^{31}-1$ ბაიტამდე
გ. ცვლადი სიგრძის ორობითი მონაცემები 8000 ბაიტამდე

18.1.27. varbinary ტიპი არის:
ა. ცვლადი სიგრძის ორობითი მონაცემები 8000 ბაიტამდე
ბ. ფიქსირებული სიგრძის ორობითი მონაცემები 8000 ბაიტამდე
გ. ცვლადი სიგრძის ორობითი მონაცემები $2^{31}-1$ ბაიტამდე

მონაცემთა ბაზისა და ცხრილის შექმნა და წაშლა

18.1.28. მონაცემთა ბაზის შესაქმნელად გამოიყენება ბრძანება:
ა. DROP DATABASE
ბ. CREATE DATABASE
გ. CREATE TABLE

18.1.29. ცხრილის წასაშლელად გამოიყენება ბრძანება:
ა. DELETE TABLE
ბ. CREATE TABLE
გ. DROP TABLE

18.1.30. NOT NULL საკვანძო სიტყვა მიუთითებს, რომ
ა. ველი არის პირველადი გასაღები
ბ. ველის მნიშვნელობა აუცილებლად უნდა იყოს მითითებული

გ. ველის მნიშვნელობა უნდა შემოწმდეს

18.1.31. ცხრილის შესაქმნელად გამოიყენება ბრძანება:
ა. INSERT
ბ. DROP TABLE
გ. CREATE TABLE

18.1.32. მონაცემთა ბაზის წასაშლელად გამოიყენება ბრძანება:
ა. DROP DATABASE
ბ. CREATE DATABASE
გ. DROP TABLE

18.1.33. PTIMARY KEY საკვანძო სიტყვა მიუთითებს, რომ:
ა. ველის მნიშვნელობა აუცილებლად უნდა მიეთითოს
ბ. ველი არის პირველადი გასაღები
გ. ველის მნიშვნელობა უნდა შემოწმდეს

18.1.34. DEFAULT საკვანძო სიტყვა მიუთითებს:
ა. რომ ველის მნიშვნელობა უნდა შემოწმდეს
ბ. რომ ველი არის პირველადი გასაღები
გ. ველის ავტომატურ მნიშვნელობას

18.1.35. UNIQUE საკვანძო სიტყვა მიუთითებს:
ა. პირველად გასაღებზე
ბ. ველის ავტომატურ მნიშვნელობას
გ. ველის უნიკალურ მნიშვნელობას

18.1.36. CHECK საკვანძო სიტყვა განსაზღვრავს:
ა. პირველად გასაღებს
ბ. ველის მნიშვნელობაზე დადებულ შეზღუდვებს
გ. ველის ავტომატურ მნიშვნელობას

18.1.37. რომელ ბრძანებაში არ არის შეცდომა:
ა. CREATE TABLE Table1
ბ. TABLE CREATE Table1
გ. TABLE DROP Table1

SELECT ბრძანება

18.1.38. SELECT ბრძანება გამოიყენება:
ა. ცხრილიდან მონაცემების ამოსაღებად
ბ. ცხრილში მონაცემების შესაცვლელად
გ. ცხრილში მონაცემების ჩასამატებლად

18.1.39. SELECT * FROM Salespeople ბრძანება Salespeople ცხრილიდან ამოირჩევს:

ა. პირველი ველის მნიშვნელობას
ბ. ყველა ველის მნიშვნელობას
გ. უკანასკნელი ველის მნიშვნელობას

18.1.40. SELECT სიტყვის შემდეგ ეთითება:
ა. მონაცემთა ბაზის სახელი
ბ. ცხრილის სახელი
გ. ველის სახელები

18.1.41. FROM სიტყვის შემდეგ ეთითება:
ა. ველების სახელები
ბ. ცხრილის სახელი
გ. მონაცემთა ბაზის სახელი

18.1.42. ცხრილის ყველა ველის მნიშვნელობის გამოსატანად უნდა მივუთითოთ:
ა. ვარსკვლავი (*)
ბ. კითხვის ნიშანი (?)
გ. პირველი ველის სახელი

18.1.43. DISTINCT საკვანძო სიტყვა:
ა. არ კრძალავს გამეორებადი მნიშვნელობების გამოტანას
ბ. კრძალავს გამეორებადი მნიშვნელობების გამოტანას
გ. კრძალავს არაგამეორებადი მნიშვნელობების გამოტანას

18.1.44. WHERE საკვანძო სიტყვის შემდეგ ეთითება:
ა. პირობა
ბ. ველის სახელი
გ. ცხრილის სახელი

შედარებისა და ლოგიკის ოპერატორები

18.1.45. ლოგიკის ოპერატორია:
ა. OR
ბ. >=
გ. SELECT

18.1.46. შედარების ოპერატორია:
ა. AND
ბ. <
გ. WHERE

18.1.47. რომელი ოპერატორი გასცემს true მნიშვნელობას, თუ მისი ორივე არგუმენტის მნიშვნელობაა true:
ა. AND
ბ. NOT

გ. DEFAULT

18.1.48. SELECT * FROM Customers WHERE qalaqi = ‘თბილისი’ OR qalaqi = ‘ბათუმი’ მოთხოვნა გასცემს ჩანაწერებს, სადაც:
ა. qalaqi ველი შეიცავს მნიშვნელობას ‘თბილისი’
ბ. qalaqi ველი შეიცავს მნიშვნელობას ‘ბათუმი’
გ. qalaqi ველი შეიცავს მნიშვნელობებს ‘თბილისი’ ან ‘ბათუმი’

18.1.49. რომელი ოპერატორი ახორციელებს შედარებას მეტობაზე:
ა. <=
ბ. >
გ. NOT

18.1.50. რომელი ოპერატორი ახორციელებს შედარებას ტოლობაზე:
ა. AND
ბ. =
გ. >=

18.1.51. რომელი ოპერატორი ახორციელებს შედარებას ნაკლებობაზე:
ა. <
ბ. >=
გ. OR

18.1.52. რომელი ოპერატორი ახორციელებს შედარებას უტოლობაზე:
ა. NOT
ბ. <=
გ. <>

18.1.53. რომელი ოპერატორია ლოგიკური “და”:
ა. OR
ბ. =
გ. AND

18.1.54. რომელი ოპერატორია ლოგიკური “ან”:
ა. OR
ბ. <>
გ. AND

18.1.55. რომელი ოპერატორია ლოგიკური “არა”:
ა. NOT
ბ. OR
გ. >=

18.1.56. რომელი ოპერატორი გასცემს true მნიშვნელობას, თუ მისი ერთი არგუმენტის მნიშვნელობაა true, მეორე არგუმენტის კი - false:

ა. NOT
ბ. AND
გ. OR

18.1.57. როგორ შედეგს გასცემს AND ოპერატორი, თუ მისი ორივე არგუმენტის მნიშვნელობაა true:
ა. false
ბ. true
გ. NOT

18.1.58. როგორ შედეგს გასცემს AND ოპერატორი, თუ მისი ორივე არგუმენტის მნიშვნელობაა false:
ა. true
ბ. false
გ. AND

18.1.59. როგორ შედეგს გასცემს NOT ოპერატორი, თუ მისი არგუმენტის მნიშვნელობაა true:
ა. AND
ბ. true
გ. false

18.1.60. როგორ შედეგს გასცემს NOT ოპერატორი, თუ მისი არგუმენტის მნიშვნელობაა false:
ა. OR
ბ. false
გ. true

18.1.61. როგორ შედეგს გასცემს AND ოპერატორი, თუ მისი ერთი არგუმენტის მნიშვნელობაა false, მეორესი კი - true:
ა. false
ბ. true
გ. OR

18.1.62. როგორ შედეგს გასცემს OR ოპერატორი, თუ მისი ორივე არგუმენტის მნიშვნელობაა false:
ა. false
ბ. true
გ. NOT

18.1.63. როგორ შედეგს გასცემს OR ოპერატორი, თუ მისი ორივე არგუმენტის მნიშვნელობაა true:
ა. false
ბ. true
გ. NOT

18.1.64. როგორ შედეგს გასცემს OR ოპერატორი, თუ მისი ერთი არგუმენტის მნიშვნელობაა false, მეორესი კი - true:

ა. false

ბ. OR

გ. true

18.1.65. SELECT * FROM Customers
WHERE qalaqi = 'qutaisi' AND rating = 200
მოთხოვნა გასცემს ჩანაწერებს, სადაც:

ა. qalaqi ველი შეიცავს მნიშვნელობას 'ქუთაისი' ან rating ველი შეიცავს მნიშვნელობას 200

ბ. qalaqi ველი შეიცავს მნიშვნელობას 'ქუთაისი' და rating ველი შეიცავს მნიშვნელობას 200

გ. qalaqi ველი შეიცავს მნიშვნელობას 'ქუთაისი'

IN, BETWEEN, LIKE, IS NULL ოპერატორები

18.1.66. SELECT * FROM Customers WHERE cname IS NULL
მოთხოვნა გასცემს ჩანაწერებს, სადაც:

ა. არის შეტანილი cname ველის მნიშვნელობა

ბ. არ არის შეტანილი cname ველის მნიშვნელობა

გ. cname ველის მნიშვნელობაა 5

18.1.67. SELECT * FROM Salespeople
WHERE qalaqi IN ('თბილისი', 'ქუთაისი')
მოთხოვნა გასცემს ჩანაწერებს, სადაც:

ა. qalaqi ველის მნიშვნელობა შედის მითითებულ სიაში

ბ. qalaqi ველის მნიშვნელობაა 'თბილისი'

გ. qalaqi ველის მნიშვნელობაა 'ქუთაისი'

18.1.68. SELECT * FROM Salespeople
WHERE comm BETWEEN 9 AND 15
მოთხოვნა გასცემს ჩანაწერებს, სადაც:

ა. comm ველის მნიშვნელობაა 9

ბ. comm ველის მნიშვნელობაა 15

გ. comm ველის მნიშვნელობა მოთავსებულია 9-სა და 15-ს შორის

18.1.69. SELECT * FROM Customers WHERE cname LIKE 'a%'
მოთხოვნა გასცემს ჩანაწერებს, სადაც:

ა. cname ველის მნიშვნელობები 'a' ასოთი მთავრდება

ბ. cname ველის მნიშვნელობები 'a' ასოთი არ იწყება

გ. cname ველის მნიშვნელობები 'a' ასოთი იწყება

18.1.70. SELECT * FROM Customers WHERE cname LIKE 'a_m'

მოთხოვნა გასცემს ჩანაწერებს, სადაც:

ა. პირველი სიმბოლოა ‘a’, მესამე კი - ‘m’

ბ. პირველი სიმბოლოა ‘m’, მესამე კი - ‘a’

გ. მეორე სიმბოლოა ‘a’, მესამე კი - ‘m’

აგრეგატული ფუნქციები

18.1.71. COUNT ფუნქცია ასრულებს:

ა. ველის მნიშვნელობების საშუალო არითმეტიკულის გამოთვლას

ბ. ჩანაწერების დათვლას

გ. ველის მნიშვნელობების შეკრებას

18.1.72. რომელი ბრძანება გასცემს Orders ცხრილის tanxa ველის მნიშვნელობების ჯამს:

ა. SELECT MIN(tanxa) FROM Orders

ბ. SELECT AVG(tanxa) FROM Orders

გ. SELECT SUM(tanxa) FROM Orders

18.1.73. SUM ფუნქცია ასრულებს:

ა. ველის მნიშვნელობების შეკრებას

ბ. ჩანაწერების დათვლას

გ. ველის მნიშვნელობებს შორის მაქსიმალურის პოვნას

18.1.74. AVG ფუნქცია ასრულებს:

ა. ველის მნიშვნელობებს შორის მინიმალურის პოვნას

ბ. ველის მნიშვნელობების საშუალო არითმეტიკულის გამოთვლას

გ. ველის მნიშვნელობების შეკრებას

18.1.75. MIN ფუნქცია ასრულებს:

ა. ველის მნიშვნელობებს შორის მაქსიმალურის პოვნას

ბ. ველის მნიშვნელობების შეკრებას

გ. ველის მნიშვნელობებს შორის მინიმალურის პოვნას

18.1.76. MAX ფუნქცია ასრულებს:

ა. ველის მნიშვნელობებს შორის მაქსიმალურის პოვნას

ბ. ველის მნიშვნელობებს შორის მინიმალურის პოვნას

გ. ჩანაწერების დათვლას

18.1.77. SUM ფუნქცია გამოიყენება:

ა. სიმბოლური ველების მიმართ

ბ. რიცხვითი ველების მიმართ

გ. როგორც რიცხვითი, ისე სიმბოლური ველების მიმართ

18.1.78. AVG ფუნქცია გამოიყენება:

ა. სიმბოლური ველების მიმართ

ბ. რიცხვითი ველების მიმართ
გ. როგორც რიცხვითი, ისე სიმბოლური ველების მიმართ

18.1.79. COUNT ფუნქცია გამოიყენება:
ა. მხოლოდ სიმბოლური ველების მიმართ
ბ. მხოლოდ რიცხვითი ველების მიმართ
გ. როგორც რიცხვითი, ისე სიმბოლური ველების მიმართ

18.1.80. MAX ფუნქცია გამოიყენება:
ა. მხოლოდ რიცხვითი ველების მიმართ
ბ. მხოლოდ სიმბოლური ველების მიმართ
გ. როგორც რიცხვითი, ისე სიმბოლური ველების მიმართ

18.1.81. MIN ფუნქცია გამოიყენება:
ა. როგორც რიცხვითი, ისე სიმბოლური ველების მიმართ
ბ. მხოლოდ რიცხვითი ველების მიმართ
გ. მხოლოდ სიმბოლური ველების მიმართ

18.1.82. SELECT SUM(tanxa) FROM Orders ბრძანება გასცემს:
ა. tanxa ველის მნიშვნელობების ჯამს
ბ. tanxa ველის მნიშვნელობებს შორის მაქსიმალურს
გ. tanxa ველის მნიშვნელობების საშუალო არითმეტიკულს

18.1.83. SELECT AVG(tanxa) FROM Orders ბრძანება გასცემს:
ა. tanxa ველის მნიშვნელობებს შორის მინიმალურს
ბ. tanxa ველის მნიშვნელობების საშუალო არითმეტიკულს
გ. tanxa ველის მნიშვნელობების ჯამს

18.1.84. GROUP BY საკვანძო სიტყვა ჩანაწერებს:
ა. არ აჯგუფებს მითითებული ველის მნიშვნელობების მიხედვით
ბ. ხანდახან აჯგუფებს მითითებული ველის მნიშვნელობების მიხედვით
გ. აჯგუფებს მითითებული ველის მნიშვნელობების მიხედვით

18.1.85. SELECT onum, cnum, MIN(tanxa) FROM Orders
GROUP BY cnum
ბრძანება ჩანაწერებს აჯგუფებს:
ა. cnum ველის მიხედვით
ბ. onum ველის მიხედვით
გ. tanxa ველის მიხედვით

18.1.86. GROUP BY საკვანძო სიტყვის მიერ ჯგუფის ფორმირების პირობაა:
ა. ველის ნულოვანი მნიშვნელობები
ბ. ველის სხვადასხვა მნიშვნელობები
გ. ველის ერთნაირი მნიშვნელობები

18.1.87. SELECT odate, cnum, AVG(tanxa) FROM Orders
GROUP BY cnum, odate
ბრძანება ჩანაწერებს აჯგუფებს:

ა. odate და cnum ველების მიხედვით
ბ. cnum და odate ველების მიხედვით
გ. მხოლოდ odate ველის მიხედვით

18.1.88. Having საკვანძო სიტყვა გამოიყენება:

ა. მიღებული შედეგიდან ჩანაწერების ჯგუფის გამოსაყოფად
ბ. ველების მნიშვნელობების შესაკრებად
გ. ცარიელი ველების საპოვნელად

რამდენიმე ცხრილიდან ჩანაწერების ამორჩევა

18.1.89. SELECT Customers.cname, Salespeople.sname
FROM Customers, Salespeople
მოთხოვნა ჩანაწერებს ამოიღებს:

ა. მხოლოდ Customers ცხრილიდან
ბ. მხოლოდ Salespeople ცხრილიდან
გ. Customers და Salespeople ცხრილებიდან

18.1.90. SELECT Customers.cname, Salespeople.sname, Orders.tarigi
FROM Customers, Salespeople, Orders
მოთხოვნა ჩანაწერებს ამოიღებს:

ა. Customers და Salespeople ცხრილებიდან
ბ. Orders, Customers და Salespeople ცხრილებიდან
გ. Orders და Salespeople ცხრილებიდან

ტესტების პასუხები

თავი 2. C# ენის საფუძვლები

საწყისი ცნებები

2.1.1. ბ
2.1.2. გ
2.1.3. ა
2.1.4. ა
2.1.5. ა
2.1.6. ბ
2.1.7. გ
2.1.8. ბ
2.1.9. ა
2.1.10. ბ
2.1.11. ა
2.1.12. გ
2.1.13. ბ
2.1.14. ა
2.1.15. ა
2.1.16. გ
2.1.17. ა
2.1.18. გ
2.1.19. ა
2.1.20. ბ
2.1.21. გ
2.1.22. ა
2.1.23. ბ

C# ენის საბაზო ტიპები

2.2.1. ა
2.2.2. ბ
2.2.3. ა
2.2.4. გ
2.2.5. ბ
2.2.6. ა
2.2.7. გ
2.2.8. ა
2.2.9. ა
2.2.10 გ
2.2.11. ბ
2.2.12. ა
2.2.13. გ
2.2.14. გ
2.2.15. გ
2.2.16. ბ
2.2.17. ბ
2.2.18. ა
2.2.19. ა
2.2.20. ბ
2.2.21. გ
2.2.22. ა
2.2.23. ა
2.2.24. გ
2.2.25. ა
2.2.26. გ
2.2.27. ბ
2.2.28. ა
2.2.29. ბ
2.2.30. ბ
2.2.31. გ
2.2.32. ა
2.2.33. გ
2.2.34. ა
2.2.35. ბ
2.2.36. გ
2.2.37. ა
2.2.38. ბ
2.2.39. ბ
2.2.40. გ
2.2.41. გ
2.2.42. ა
2.2.43. ა
2.2.44. ბ
2.2.45. გ
2.2.46. ა
2.2.47. ბ
2.2.48. ა
2.2.49. ა

ცვლადები და მათი ინიციალიზაცია

2.3.1. ა
2.3.2. ბ
2.3.3. ა
2.3.4. ბ
2.3.5. გ
2.3.6 ა

ოპერატორები

2.4.1. ა
2.4.2. გ
2.4.3. გ
2.4.4. გ
2.4.5. გ
2.4.6. გ
2.4.7. ა
2.4.8. გ
2.4.9. გ
2.4.10. გ
2.4.11. ა
2.4.12. გ
2.4.13. გ
2.4.14. გ
2.4.15. ა
2.4.16. გ
2.4.17. გ
2.4.18. ა
2.4.19. გ
2.4.20. გ
2.4.21. ა
2.4.22. გ
2.4.23. გ
2.4.24. გ
2.4.25. გ
2.4.26. გ
2.4.27. გ
2.4.28. გ
2.4.29. გ
2.4.30. გ
2.4.31. გ
2.4.32. ა

შედარებისა და ლოგიკის ოპერატორები

2.5.1. გ
2.5.2. გ
2.5.3. ა
2.5.4. ა
2.5.5. გ
2.5.6. გ
2.5.7. გ
2.5.8. გ
2.5.9. გ
2.5.10. ა
2.5.11. გ
2.5.12. გ
2.5.13. გ
2.5.14. გ
2.5.15. ა
2.5.16. ა
2.5.17. გ
2.5.18. ა
2.5.19. გ
2.5.20. გ
2.5.21. ა
2.5.22. გ
2.5.23. ა
2.5.24. ა
2.5.25. გ
2.5.26. გ
2.5.27. ა
2.5.28. ა
2.5.29. გ
2.5.30. გ
2.5.31. ა

მინიჭების ოპერატორი. შედგენილი ოპერატორები

2.6.1. ა
2.6.2. გ
2.6.3. ა
2.6.4. გ
2.6.5. ა
2.6.6. გ
2.6.7. ა
2.6.8. გ
2.6.9. გ
2.6.10. გ
2.6.11. გ
2.6.12. ა

ოპერატორების პრიორიტეტები

2.7.1. ა
2.7.2. გ
2.7.3. ა
2.7.4. გ
2.7.5. ა

გამოსახულებები. მათემატიკური მეთოდები

2.8.1. გ
2.8.2. ა
2.8.3. ბ
2.8.4. გ
2.8.5. ბ
2.8.6. ა
2.8.7. ა
2.8.8. გ
2.8.9. ბ
2.8.10. ბ
2.8.11. ა
2.8.12. გ
2.8.13. გ
2.8.14. ბ
2.8.15. ა
2.8.16. ბ
2.8.17. ა

ტიპების გარდაქმნა

2.9.1. გ
2.9.2. ბ
2.9.3. ა
2.9.4. ა
2.9.5. გ
2.9.6. ა
2.9.7. ა
2.9.8. გ
2.9.9. გ
2.9.10. ა
2.9.11. ბ
2.9.12. ა
2.9.13. ა
2.9.14. გ
2.9.15. ბ
2.9.16. ა
2.9.17. გ
2.9.18. ბ
2.9.19. ბ
2.9.20. ბ
2.9.21. გ
2.9.22. ბ
2.9.23. ა
2.9.24. ა
2.9.25. გ
2.9.26. გ
2.9.27. გ
2.9.28. ა
2.9.29. ა
2.9.30. ბ
2.9.31. ბ
2.9.32. ა
2.9.33. ბ
2.9.34. გ
2.9.35. გ
2.9.36. ა
2.9.37. ბ
2.9.38. ა
2.9.39. ა
2.9.40. ბ
2.9.41. გ
2.9.42. გ
2.9.43. ა
2.9.44. ბ
2.9.45. ა
2.9.46. გ
2.9.47. ბ
2.9.48. გ
2.9.49. ბ
2.9.50. ა

თავი 3. მმართველი ოპერატორები

ამორჩევის ოპერატორები. if ოპერატორი

3.1.1. გ
3.1.2. ა
3.1.3. ა
3.1.4. გ
3.1.5. ბ
3.1.6. გ
3.1.7. ა
3.1.8. გ
3.1.9. ბ

ხილვადობის უბანი

3.2.1. ა
3.2.2. ბ
3.2.3. ა
3.2.4. ა

3.2.5. გ
3.2.6. გ
3.2.7. ა
3.2.8. ბ
3.2.9. გ
3.2.10. ბ
3.2.11. ა
3.2.12. ა
3.2.13. ბ
3.2.14. გ
3.2.15. ა
3.2.16. ბ
3.2.17. ა
3.2.18. ა
3.2.19. ა
3.2.20. ბ
3.2.21. ა
3.2.22. გ
3.2.23. გ
3.2.24. ა

ამორჩევის ოპერატორები. switch ოპერატორი

3.3.1. ა
3.3.2. ბ
3.3.3. ა
3.3.4. გ
3.3.5. ა
3.3.6. ბ
3.3.7. ა
3.3.8. გ
3.3.9. ა
3.3.10. გ
3.3.11. ბ
3.3.12. გ
3.3.13. ბ

იტერაციის ოპერატორები. for ოპერატორი

3.4.1. გ
3.4.2. ბ
3.4.3. ა
3.4.4. ბ
3.4.5. გ
3.4.6. ა
3.4.7. ბ
3.4.8. ბ
3.4.9. გ
3.4.10. გ
3.4.11. ა
3.4.12. ბ
3.4.13. ა
3.4.14. ბ
3.4.15. გ
3.4.16. გ
3.4.17. გ
3.4.18. გ
3.4.19. ა
3.4.20. ბ
3.4.21. გ
3.4.22. ა
3.4.23. გ

იტერაციის ოპერატორები. while ოპერატორი

3.5.1. ა
3.5.2. ბ
3.5.3. გ
3.5.4. ა
3.5.5. ბ

იტერაციის ოპერატორები. do-while ოპერატორი

3.6.1. ა
3.6.2. ბ
3.6.3. გ
3.6.4. ა
3.6.5. ბ

გადასვლის ოპერატორები. break ოპერატორი

3.7.1. ა
3.7.2. ბ
3.7.3. გ
3.7.4. ბ
3.7.5. გ
3.7.6. ა
3.7.7. გ

გადასვლის ოპერატორები. continue

ოპერატორი

3.8.1. ა
3.8.2. ბ
3.8.3. გ
3.8.4. ბ
3.8.5. ბ
3.8.6. გ

გადასვლის ოპერატორები. goto ოპერატორი

3.9.1. ა
3.9.2. ბ
3.9.3. გ

ტერნარული ოპერატორი

3.10.1. ბ
3.10.2. ა
3.10.3. ბ
3.10.4. გ
3.10.5. ა
3.10.6. ბ

თავი 4. მასივები, სტრიქონები და ბიტობრივი ოპერაციები

ერთგანზომილებიანი მასივები

4.1.1. ა
4.1.2. ბ
4.1.3. გ
4.1.4. ბ
4.1.5. ა
4.1.6. ბ
4.1.7. გ
4.1.8. ა
4.1.9. ა
4.1.10. ბ
4.1.11. ბ
4.1.12. გ
4.1.13. ა
4.1.14. ბ
4.1.15. გ
4.1.16. გ
4.1.17. ა

ორგანზომილებიანი მასივები

4.2.1. ბ
4.2.2. ბ
4.2.3. ა
4.2.4. ბ
4.2.5. გ
4.2.6. ბ
4.2.7. გ
4.2.8. ა
4.2.9. გ

გაუსწორებელი მასივები

4.3.1. ა
4.3.2. გ
4.3.3. ბ

Length თვისება

4.4.1. ბ
4.4.2. ა
4.4.3. ბ
4.4.4. გ
4.4.5. ა
4.4.6. გ
4.4.7. ა

foreach ოპერატორი

4.5.1. ა
4.5.2. გ
4.5.3. ბ
4.5.4. ა

ბიტობრივი ოპერატორები

4.6.1. ბ
4.6.2. ა
4.6.3. ბ
4.6.4. გ
4.6.5. გ
4.6.6. ბ
4.6.7. ბ
4.6.8. გ
4.6.9. გ
4.6.10. ბ
4.6.11. ა
4.6.12. ბ
4.6.13. ა
4.6.14. ბ

4.6.15. გ
4.6.16. ა
4.6.17. ა

ჩამოთვლები

4.7.1. ა
4.7.2. ბ
4.7.3. გ
4.7.4. ა
4.7.5. გ
4.7.6. ბ
4.7.7. ა
4.7.8. გ
4.7.9. ბ
4.7.10. ა

თავი 5. კლასები, ინკაფსულაცია, მეთოდები

კლასების გამოცხადება

5.1.1. გ
5.1.2. ა
5.1.3. ბ
5.1.4. გ
5.1.5. ბ
5.1.6. ბ
5.1.7. ა
5.1.8. ა
5.1.9. გ
5.1.10. ა
5.1.11. ბ
5.1.12. ბ
5.1.13. ა
5.1.14. გ
5.1.15. გ
5.1.16. გ
5.1.17. ბ

ინკაფსულაცია

5.2.1. ბ
5.2.2. ბ
5.2.3. ა
5.2.4. გ
5.2.5. ბ
5.2.6. გ
5.2.7. ა
5.2.8. ა
5.2.9. ბ
5.2.10. გ
5.2.11. ბ
5.2.12. ა
5.2.13. ბ
5.2.14. გ
5.2.15. ბ
5.2.16. ა
5.2.17. ბ
5.2.18. ა
5.2.19. გ

მეთოდები

5.3.1. ა
5.3.2. ბ
5.3.3. გ
5.3.4. ბ
5.3.5. ბ
5.3.6. ა
5.3.7. გ
5.3.8. გ
5.3.9. ა
5.3.10. ბ
5.3.11. ბ
5.3.12. ა
5.3.13. გ
5.3.14. გ
5.3.15. ა
5.3.16. ბ
5.3.17. გ
5.3.18. ბ
5.3.19. ა
5.3.20. ბ
5.3.21. ა

კონსტრუქტორი

5.4.1. ა
5.4.2. ა
5.4.3. გ
5.4.4. ა
5.4.5. ბ
5.4.6. გ
5.4.7. ა

5.4.8. ბ
5.4.9. გ
5.4.10. ა
5.4.11. გ
5.4.12. ბ
5.4.13. ბ
5.4.14. გ

დესტრუქტორი
5.5.1. ა
5.5.2. ბ
5.5.3. გ
5.5.4. ა
5.5.5. ბ
5.5.6. ბ
5.5.7. გ

this სიტყვა
5.6.1. ბ
5.6.2. ბ
5.6.3. გ
5.6.4. გ

შემთხვევითი რიცხვების გენერატორი
5.7.1. ა
5.7.2. ბ
5.7.3. გ
5.7.4. ბ
5.7.5. ბ

თავი 6. მეთოდები უფრო დაწვრილებით. პოლიმორფიზმი

მეთოდისთვის ობიექტის გადაცემა
6.1.1. ბ
6.1.2. ა

მეთოდისთვის არგუმენტების გადაცემის ხერხები
6.2.1. ა
6.2.2. ბ
6.2.3. ბ
6.2.4. გ
6.2.5. ა
6.2.6. ა
6.2.7. ბ

ref და out მოდიფიკატორები
6.3.1. გ
6.3.2. ა
6.3.3. ბ
6.3.4. ა
6.3.5. ბ
6.3.6. გ

params მოდიფიკატორი
6.4.1. გ
6.4.2. გ
6.4.3. ა

მეთოდის გადატვირთვა. პოლიმორფიზმი
6.5.1. ა
6.5.2. ბ
6.5.3. გ
6.5.4. ა
6.5.5. ბ
6.5.6. გ
6.5.7. გ
6.5.8. ა
6.5.9. ბ

კონსტრუქტორების გადატვირთვა
6.6.1. ა
6.6.2. გ
6.6.3. ბ
6.6.4. ა

რეკურსია
6.7.1. ა
6.7.2. ბ
6.7.3. ბ

static მოდიფიკატორი
6.8.1. ა
6.8.2. ბ
6.8.3. გ
6.8.4. ბ
6.8.5. ა
6.8.6. გ

6.8.7. გ
6.8.8. ა
6.8.9. ა
6.8.10. ბ

თავი 7. ოპერატორების გადატვირთვა, თვისებები და ინდექსატორები

ოპერატორების გადატვირთვა

7.1.1. ა
7.1.2. გ
7.1.3. ბ
7.1.4. ბ
7.1.5. ა
7.1.6. გ
7.1.7. გ
7.1.8. ბ
7.1.9. ა
7.1.10. ბ
7.1.11. გ
7.1.12. გ
7.1.13. ა
7.1.14. ა
7.1.15. ა
7.1.16. გ
7.1.17. ბ
7.1.18. გ
7.1.19. გ
7.1.20. ბ
7.1.21. ა

თვისებები

7.2.1. ბ
7.2.2. გ
7.2.3. ა
7.2.4. გ
7.2.5. ა
7.2.6. ბ
7.2.7. ბ
7.2.8. ა
7.2.9. გ
7.2.10. ა
7.2.11. გ
7.2.12. ბ
7.2.13. ა

ინდექსატორი

7.3.1. ა
7.3.2. ა
7.3.3. გ
7.3.4. ბ
7.3.5. ბ
7.3.6. ა

თავი 8. მემკვიდრეობითობა. ვირტუალური მეთოდები

მემკვიდრეობითობის საფუძვლები

8.1.1. გ
8.1.2. გ
8.1.3. ა
8.1.4. ბ
8.1.5. ა
8.1.6. გ

protected მოდიფიკატორი

8.2.1. ბ
8.2.2. გ
8.2.3. ა
8.2.4. ა

კონსტრუქტორები და მემკვიდრეობითობა

8.3.1. ბ
8.3.2. გ
8.3.3. გ
8.3.4. ბ
8.3.5. ა
8.3.6. ბ

ცვლადების დამალვა მემკვიდრეობითობის დროს

8.4.1. გ
8.4.2. გ
8.4.3. ა
8.4.4. ბ
8.4.5. ბ
8.4.6. გ

კლასების იერარქია

8.5.1. ა
8.5.2. ა
8.5.3. ბ

მიმართვები წინაპარი და მემკვიდრე კლასის ობიექტებზე

8.6.1. გ
8.6.2. ა
8.6.3. ა
8.6.4. ბ

ვირტუალური მეთოდები

8.7.1. გ
8.7.2. ა
8.7.3. გ
8.7.4. ბ
8.7.5. ა
8.7.6. ბ
8.7.7. ა
8.7.8. გ
8.7.9. ბ
8.7.10. ა
8.7.11. ბ
8.7.12. გ
8.7.13. ბ

აბსტრაქტული კლასები

8.8.1. ბ
8.8.2. ბ
8.8.3. ა
8.8.4. გ
8.8.5. ბ
8.8.6. ა
8.8.7. გ
8.8.8. ა
8.8.9. ბ
8.8.10. ბ

მემკვიდრეობითობის აკრძალვა

8.9.1. ა

თავი 9. ინფორმაციის შეტანა-გამოტანა

შეტანა-გამოტანის ნაკადების კლასები

9.1.1. ა
9.1.2. გ
9.1.3. გ
9.1.4. ბ
9.1.5. ა
9.1.6. ბ
9.1.7. ა
9.1.8. ა
9.1.9. ბ
9.1.10. გ
9.1.11. გ
9.1.12. ა
9.1.13. ა
9.1.14. ბ
9.1.15. გ
9.1.16. ბ
9.1.17. ა
9.1.18. გ
9.1.19. გ
9.1.20. ბ
9.1.21. ა
9.1.22. ა
9.1.23. ბ
9.1.24. გ
9.1.25. ბ
9.1.26. ბ
9.1.27. გ
9.1.28. ა
9.1.29. ბ
9.1.30. ბ
9.1.31. გ
9.1.32. გ
9.1.33. ა
9.1.34. ა
9.1.35. გ
9.1.36. ბ
9.1.37. ბ
9.1.38. ბ
9.1.39. ა
9.1.40. გ
9.1.41. გ
9.1.42. ა
9.1.43. ბ

FileStream კლასი

9.2.1. ა
9.2.2. გ
9.2.3. ბ
9.2.4. ბ
9.2.5. ა
9.2.6. ა
9.2.7. გ
9.2.8. ა
9.2.9. გ
9.2.10. ბ
9.2.11. ა
9.2.12. გ
9.2.13. ბ
9.2.14. ა
9.2.15. ა
9.2.16. გ
9.2.17. ბ
9.2.18. გ
9.2.19. ა
9.2.20. ა
9.2.21. ბ
9.2.22. გ

ფაილში სიმბოლოების შეტანა-გამოტანა

9.3.1. ბ
9.3.2. ა
9.3.3. გ

ორობითი მონაცემების შეტანა-გამოტანა

9.4.1. გ
9.4.2. ა
9.4.3. ბ
9.4.4. ბ
9.4.5. გ
9.4.6. ა
9.4.7. ბ
9.4.8. გ
9.4.9. ბ
9.4.10. ა
9.4.11. ა
9.4.12. გ
9.4.13. ბ
9.4.14. ა
9.4.15. გ
9.4.16. ბ
9.4.17. ა
9.4.18. გ
9.4.19. ბ
9.4.20. ა
9.4.21. ბ
9.4.22. ა
9.4.23. გ
9.4.24. ა
9.4.25. ბ
9.4.26. გ
9.4.27. ა
9.4.28. ა
9.4.29. ა
9.4.30. ა
9.4.31. ა
9.4.32. ბ
9.4.33. გ
9.4.34. ბ
9.4.35. გ
9.4.36. გ
9.4.37. ა
9.4.38. ა
9.4.39. ბ
9.4.40. გ

ფაილებთან პირდაპირი მიმართვა

9.5.1. ა
9.5.2. ბ
9.5.3. გ
9.5.4. ბ
9.5.5. ა
9.5.6. გ
9.5.7. გ
9.5.8. ა
9.5.9. გ
9.5.10. ბ
9.5.11. ბ
9.5.12. გ
9.5.13. ა

თავი 10. განსაკუთრებული სიტუაციები

სიტუაციების დამუშავების საფუძვლები

10.1.1. გ
10.1.2. ბ
10.1.3. ა
10.1.4. ა
10.1.5. გ
10.1.6. ბ
10.1.7. ბ
10.1.8. ა
10.1.9. გ
10.1.10. ბ
10.1.11. გ
10.1.12. ბ
10.1.13. გ
10.1.14. ა
10.1.15. ბ
10.1.16. ბ
10.1.17. ა
10.1.18. გ
10.1.19. ა
10.1.20. ბ
10.1.21. ბ
10.1.22. გ
10.1.23. გ
10.1.24. ა

try და catch ბლოკები

10.2.1. ა
10.2.2. გ
10.2.3. ბ
10.2.4. ა
10.2.5. ბ
10.2.6. გ
10.2.7. გ
10.2.8. ბ
10.2.9. ა
10.2.10. ა
10.2.11. ბ

ჩადგმული try ბლოკები

10.3.1. გ
10.3.2. ა

throw ოპერატორი

10.4.1. ა

finally ბლოკი

10.5.1. ბ
10.5.2. გ

checked და unchecked ოპერატორები

10.6.1. ა
10.6.2. ბ
10.6.3. გ
10.6.4. ბ
10.6.5. ა
10.6.6. გ
10.6.7. ბ
10.6.8. ა
10.6.9. ბ
10.6.10. გ
10.6.11. ბ

თავი 11. დელეგატები, მოვლენები, ინტერფეისები და სახელების სივრცე

დელეგატები

11.1.1. ა
11.1.2. ა
11.1.3. ბ
11.1.4. გ
11.1.5. ა
11.1.6. გ
11.1.7. ბ
11.1.8. ა
11.1.9. ა
11.1.10. ბ
11.1.11. გ
11.1.12. ბ
11.1.13. ა
11.1.14. ბ
11.1.15. ა
11.1.16. ბ
11.1.17. ბ
11.1.18. გ

მოვლენები

11.2.1. გ
11.2.2. ა
11.2.3. ბ

11.2.4. გ
11.2.5. ა
11.2.6. გ
11.2.7. გ
11.2.8. გ
11.2.9. ა
11.2.10. გ
11.2.11. გ
11.2.12. გ

ინტერფეისები

11.3.1. გ
11.3.2. ა
11.3.3. გ
11.3.4. გ
11.3.5. ა
11.3.6. გ
11.3.7. გ
11.3.8. გ
11.3.9. გ
11.3.10. გ
11.3.11. ა
11.3.12. გ
11.3.13. გ
11.3.14. ა
11.3.15. ა
11.3.16. გ

სახელების სივრცე

11.4.1. ა
11.4.2. გ
11.4.3. ა
11.4.4. გ
11.4.5. გ
11.4.6. გ
11.4.7. ა
11.4.8. გ
11.4.9. ა
11.4.10. გ
11.4.11. გ
11.4.12. ა
11.4.13. გ
11.4.14. გ

თავი 12. სტრიქონები

სტრიქონები

12.1.1. ა
12.1.2. ა
12.1.3. გ
12.1.4. გ
12.1.5. ა
12.1.6. გ
12.1.7. გ
12.1.8. გ
12.1.9. გ
12.1.10. ა
12.1.11. გ
12.1.12. გ
12.1.13. გ
12.1.14. ა
12.1.15. გ
12.1.16. გ

თავი 17. შესავალი მონაცემთა ბაზებში

მონაცემთა ბაზების აგების პრინციპები

17.1.1. ა
17.1.2. გ
17.1.3. გ
17.1.4. გ
17.1.5. ა
17.1.6. გ
17.1.7. გ
17.1.8. ა
17.1.9. გ
17.1.10. ა
17.1.11. გ
17.1.12. ა
17.1.13. გ
17.1.14. გ
17.1.15. გ
17.1.16. ა
17.1.17. ა
17.1.18. გ
17.1.19. გ
17.1.20. ა
17.1.21. გ
17.1.22. ა
17.1.23 გ

17.1.24. ა
17.1.25. ბ
17.1.26. გ
17.1.27. ა
17.1.28. ბ
17.1.29. გ
17.1.30. ა
17.1.31. ბ
17.1.32. გ
17.1.33. ა
17.1.34. ბ
17.1.35. გ
17.1.36. ა
17.1.37. გ
17.1.38. ბ
17.1.39. ბ
17.1.40. ა
17.1.41. გ
17.1.42. ა
17.1.43. გ
17.1.44. ბ

თავი18. SQL-სერვერი

SQL-მოთხოვნის ტიპები

18.1.1. ა
18.1.2. გ
18.1.3. ა
18.1.4. გ
18.1.5. ბ
18.1.6. ა
18.1.7. გ

მონაცემთა ტიპები

18.1.8. ა
18.1.9. ბ
18.1.10. გ
18.1.11. ბ
18.1.12. ა
18.1.13. ა
18.1.14. ბ
18.1.15. გ
18.1.16. ბ
18.1.17. ა
18.1.18. გ
18.1.19. ბ
18.1.20. გ
18.1.21. ა
18.1.22. ა
18.1.23. ბ
18.1.24. გ
18.1.25. ბ
18.1.26. ა
18.1.27. ა

მონაცემთა ბაზისა და ცხრილის შექმნა და წაშლა

18.1.28. ბ
18.1.29. გ
18.1.30. ბ
18.1.31. გ
18.1.32. ა
18.1.33. ბ
18.1.34. გ
18.1.35. გ
18.1.36. ბ
18.1.37. ა

SELECT ბრძანება

18.1.38. ა
18.1.39. ბ
18.1.40. გ
18.1.41. ბ
18.1.42. ა
18.1.43. ბ
18.1.44. ა

შედარებისა და ლოგიკის ოპერატორები

18.1.45. ა
18.1.46. ბ
18.1.47. ა
18.1.48. გ
18.1.49. ბ
18.1.50. ბ
18.1.51. ა
18.1.52. გ
18.1.53. გ
18.1.54. ა
18.1.55. ა
18.1.56. გ
18.1.57. ბ

18.1.58. ბ
18.1.59. გ
18.1.60. გ
18.1.61. ა
18.1.62. ა
18.1.63. ბ
18.1.64. გ
18.1.65. ბ

IN, BETWEEN, LIKE, IS NULL ოპერატორები

18.1.66. ბ
18.1.67. ა
18.1.68. გ
18.1.69. გ
18.1.70. ა

აგრეგატული ფუნქციები

18.1.71. ბ
18.1.72. გ
18.1.73. ა
18.1.74. ბ
18.1.75. გ
18.1.76. ა
18.1.77. ბ
18.1.78. ბ
18.1.79. გ
18.1.80. გ
18.1.81. ა
18.1.82. ა
18.1.83. ბ
18.1.84. გ
18.1.85. ა
18.1.86. გ
18.1.87. ბ
18.1.88. ა

რამდენიმე ცხრილიდან ჩანაწერების ამორჩევა

18.1.89. გ
18.1.90. ბ

ლიტერატურა

1. Шилдт. Г. C# .
2. Прайс дж., Гандерлой М. Visual C#.NET .
3. Смайли Дж. Учимся программироваить на C# .
4. Davis. H. Visual C#.NET Programming.
5. Butow E., Ryan T. C# .
6. Hejlsberg A., Wiltamuth S. C# Language Reference.
7. O'Reilly. Programming C# .
8. Г. Дейтел. Введение в операционные системы. В 2-х томах. Пер. с англ. - М.: Мир, 1987.
9. Э. Таненбаум. Современные операционные системы. - СПб.: Питер, 2002. - 1040 с.: ил.
10. Saurabh Nandu. C#.NET. Web Developer's Guide.
11. O'Reilly. Programming C#.
12. Разработка Windows-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# -NET. Учебный курс MCAD/MCSD/Пер. с англ. - М.: Издательско-торговый дом "Русская Редакция", 2003. - 512 стр.: ил.
13. E.Butow, T. Ryan. Your visual blueprint for building .NET applications.
14. Harold Davis. Visual C# .NET Programming.
15. A. Hejlsberg, S.Wiltamuth. C# Language Reference.
16. Г. Шилдт. Полный справочник по C#.
17. C# Frequently Asked Questions <http://blogs.msdn.com/csharpfaq/default.aspx>.

კომპიუტერული უზრუნველყოფა ლ. გაჩეჩილაძის

იბეჭდება ავტორის მიერ წარმოდგენილი სახით

გადაეცა წარმოებას 26.05.2008. ხელმოწერილია დასაბეჭდად 06.06.2008. ბეჭდვა ოფსეტური. პირობითი ნაბეჭდი თაბახი 20,75. ტირაჟი 100 ეგზ. შეკვეთა #

გამომცემლობა “ტექნიკური უნივერსიტეტი”, თბილისი, კოსტავას 77