Data Structures with C++ Using STL

Part 1: Sequence Containers

STL Containers

Sequence Containers	Adapter Containers	Associative Containers
Vector	Stack	Set, Multiset
Deque	Queue	Map, MutItimap
List	Priority Queue	

Sequence Containers

- Sequence containers store data by linear position

Associative containers

- Associative containers store elements by key;
- A program access an element by key, which may bear no relationship to the location of the element in the container.

Sequence Adapters

- An adapter contains a sequence container as its underlying storage structure;
- The programming interface of an adapter provides only a restricted set of operations supported by the underlying sequence container.

Member Types

```
value_type
allocator_type
size_type
difference_type
iterator
const_iterator
reverse_iterator
const_reverse_iterator
reference
const_reference
key_type
mapped_type
key_compare
```

Iterator

```
begin()
end()
rbegin()
Rend()
```

Element Access

```
front()
back()
[]
at()
```

Stack and Queue Operations

```
push_back()
pop_back()
push_front()
pop_front()
```

List Operations

```
insert(p, x)
insert(p, n, x)
insert(p, first, last)
erase(p)
erase(first, last)
clear()
```

Other Operations

```
size()
empty()
max_size()
capacity()
reserve()
resize()
swap()
get_allocator()
==
!=
<
```

Constructors

```
container()
container(n)
container(n, x)
container(first, last)
container(x)
~container()
```

Assignments

```
operator=(x)
assign(n, x)
assign(first, last)
```

Associative Operations

```
operator[](k)
find(k)
lower_bound(k)
upper_bound(k)
equal_range(k)
key_comp()
value_comp()
```

An array as a container

- An array is a container that stores n elements in a contiguous block of memory
- An array does not know its own size
- The size of the array is fixed at the time of declaration and can not be changed during the run time
- The C++ array does not allow the assignment of one array to the other
- The array supplies subscripting and random-access iterators in the form of ordinary pointers

STL Vectors

- Vector contains contiguous elements stored as in an array
- Accessing or appending elements take constant time
- Locating a value or inserting an element into the vector takes linear time
- Size and capacity are different
- Dangerous to keep pointers to elements in a vector that might be resized
- http://www.cppreference.com/cppvector/

Using reserve() to ensure correctness

```
1.  struct Link {
2.    Link* next;
3.    Link(Link* n=0):next(n) {}
4.    //......
5.  };
6.
7.  vector<Link> v;

8.  void chain(size_t n) {
9.    v.reserve(n);
10.   v.push_back(Link(0));
11.   for (int i=1; i<n; i++)
12.     v.push_back(Link(&v[i-1]));
13.   // ......
14. }
```

STL Lists

- Sequences of elements stored in a link list
- STL list is optimized for insertion and deletion of elements
- http://www.cppreference.com/cpplist/

List Operations

```
template<class T, class A=allocator<T>> class list {
public:
  void splice(iterator pos, list &x);
  void splice(iterator pos, list &x, iterator p);
  void splice(iterator pos, list &x, iterator first,
   iterator last);
  void merge(list &);
  template<class Cmp>
    void merge(list &, Cmp);
  void sort();
  template<class Cmp>
    void sort(Cmp);
  // ……
};
```

STL Iterator

- Sequence Containers:
  - Vector random access
  - Deque random access
  - List bidirectional
- Associated Containers:
  - Set bidirectional
  - Multiset bidirectional
  - Map bidirectional
  - Multimap bidirectional
- Container Adapters:
  - No iterator supported
- Iterator types:
  - Random access, bidirectional, forward, input, output

STL Algorithm

- STL separates the algorithms from the containers
- The elements of containers are accessed through iterators
- Template programming avoids the overhead of virtual function calls
- http://www.cppreference.com/cppalgorithm/

Non-modifying Sequence Operations

```
for_each()
find()
find_if()
find_first_of()
adjacent_find()
count()
count_if()
mismatch()
equal()
search()
find_end()
search_n()
```

Modifying Sequence Operations

```
transform()
copy()
copy_backward()
swap()
iter_swap()
swap_ranges()
replace()
replace_if()
replace_copy()
replace_copy_if()
fill()
fill_n()
generate()
generate_n()
remove()
remove_if()
remove_copy()
remove_copy_if()
unique()
unique_copy()
reserve()
reserve_copy()
rotate()
rotate_copy()
random_shuffle()
```

Sorted Sequences

```
sort()
stable_sort()
partial_sort()
partial_sort_copy()
nth_element()
lower_bound()
upper_bound()
equal_range()
binary_search()
merge()
inplace_merge()
partition()
stable_partition()
```

Function Objects

- An object of a class with an application operator is called a function-like object, a functor or simply a function object
- Predicates
- Arithmetic function objects
- Binder, Adapters, Negaters

Iterator Traits

- Iterator traits provide the type of elements to which the iterator refers
- The value type of iterator type T:
  - `typename std::iterator_traits<T>::value_type`

Using Iterator Traits

```
1.  template <class T>
2.  class doublyList {
3.  public:
4.    class iterator;
5.    friend class iterator;
6.    // ……
7.    class iterator {
8.    public:
9.      typedef bidirectional_iterator_tag
    iterator_category;
10.     typedef ptrdiff_t difference_type;
11.     typedef T value_type;
12.     typedef T* pointer;
13.     typedef T& reference;
14.     // ……
15. };
16.   iterator begin();
17.   iterator end();
18.   // ……
19. };
```

```
1.  template<class iIter, class oIter>
2.  oIter prefix_sum(iIter start, iIter
    end, oIter result) {
3.    typedef typename
    iterator_traits<oIter>::value_type
    value_type;
4.    // ……
5.    value_type __value = *start;
6.    while (++start != end) {
7.      __value = __value + *start;
8.      *++result = __value;
9.    }
10.   // ……
11. };
```

Input Iterator Requirements

1. `TYPE(iter)` — Copy constructor;
2. `iter1 == iter2`
3. `iter1 != iter2`
4. `*iter` — If iter1 == iter2, then *iter1 == *iter2.
5. `iter ->member` — (*iter).member
6. `++iter`
7. `iter++`

- Algorithms on input iterators should be single-pass algorithms, a == b does not imply ++a == ++b;
- Type T is not required to be a reference type, so algorithms on input iterators should not attempt to assign through them.

Allocators

- STL allocators handle the allocation and deallocation of memory
- It's the base for technical solutions of certain memory models, such as shared memory, garbage collection, and object-oriented database
- Default value as a parameter for STL containers:

```
template <class T, class Allocator = allocator<T>>
```

Summery

- STL has three key components: container, iterator and algorithm
- STL organizes its containers into three categories: sequence containers, adapters and associative containers
- STL algorithms are functions that perform such common data manipulations as search, sorting and comparing elements or entire containers

Joke of the Day

Why do programmers think Halloween and Christmas are the same day?