Lecture 10

Object Oriented Programming and C++

Standard Template Library

International School
6th Floor, 182 Nguyen Van Linh st.,
Tel: 365-0403 (ext 601)
http://kdtqt.duytan.edu.vn

Trương Đình Huy
huy.truongdinh@gmail.com
0982.132.352

Learning Objectives

- Iterators
  - Constant and mutable iterators
  - Reverse iterators
- Containers
  - Sequential containers
  - Container adapters stack and queue
- Generic Algorithms
  - Big-O notation
  - Sequence, set, and sorting algorithms

Introduction

- Recall stack and queue data structures
  - We created our own
  - Large collection of standard data structures exists
  - Make sense to have standard portable implementations of them!
- Standard Template Library (STL)
  - Includes libraries for all such data structures
    - Like container classes: stacks and queues

Iterators

- Recall: generalization of a pointer
  - Typically even implemented with pointer!
- "Abstraction" of iterators
  - Designed to hide details of implementation
  - Provide uniform interface across different container classes
- Each container class has "own" iterator type
  - Similar to how each data type has own pointer type

Manipulating Iterators

- Recall using overloaded operators:
  - ++, --, ==, !=
  - *
    - So if p is iterator variable, *p gives access to data pointed to by p
- Vector template class
  - Has all above overloads
  - Also has members begin() and end()

```
c.begin();      //Returns iterator for 1st item in c
c.end();        //Returns "test" value for end
```

Cycling with Iterators

- Recall cycling ability:
  for (p=c.begin();p!=c.end();p++)
  process *p //*p is current data item
- Big picture so far…
- Keep in mind:
  - Each container type in STL has own iterator types
    - Even though they're all used similarly

Key concepts

- Containers
- Iterators
- Algorithms

STL Containers

- Sequence Container
  - Vector
  - Deque
  - List
- Adapter Containers
  - Stack
  - Queue
  - Priority queue
- Associative Container
  - Set, multiset
  - Map, multimap

Links

- http://www.cplusplus.com/
- http://www.tutorialspoint.com/cplusplus/

List of containers

STL container:
vector
list
slist
queue
stack
deque
set

API for the containers

Function:	Purpose:
push_front	Inserts elements before the first (not available for vector)
pop_front	Removes the first element (not available for vector)
push_back	Appends element at the end
pop_back	Removes element from the end
empty	Boolean indicating if the container is empty
size	Returns number of elements
insert	Insert an element in a position
erase	Removes an element at a position
clear	Removes all elements
resize	Resizes the container
front	Returns a reference to the first element
Back	Returns a reference to the last element
[]	Subscripting access without bounds checking
at	Subscripting access with bounds checking

Vector Container

- Generalized array that stores a collection of elements of the same data type
- Vector – similar to an array
  - Vectors allow access to its elements by using an index in the range from 0 to n-1 where n is the size of the vector
- Vector vs array
  - Vector has operations that allow the collection to grow and contract dynamically at the rear of the sequence

Vector Container

Example:

```
#include <vector>
.
.
.
vector<int> scores;
vector<Point> pointList;
```

Demo

```
#include <vector>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    // empty vector of ints
    vector <int> v;
    v.push_back(10);
    v.push_back(20);
    vector <float> f;
    f.push_back(101.3F);
    f.push_back(101.222F);
    vector<int> second (4,100); // four ints with value 100
}
```

Demo

```
#include <vector>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
//using index
    for(unsigned i=0;i< v.size();i++)
    cout<<v[i]<<endl;
//Using iterator
     vector <int>::iterator it;
     for(it=v.begin(); it<v.end();it++)
     cout<<*it<<endl;
```

Demo

```
#include <vector>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
//Delete element
    f.erase(f.begin());
}
```

Demo

```
class Point {
public:
	int x;
	int y;
Point(int a, int b)
{
	x = a;
	y = b;
}
void Display()
{
	cout<<x<<endl; cout<<y<<endl;
} };
```

Demo

```
#include <vector>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    Point p(10,20);
    // Adds a new element is object
    vector <Point> v;
    v.push_back(p);
}
```

List Container

- Stores elements by position
- Each item in the list has both a value and a memory address (pointer) that identifies the next item in the sequence
- To access a specific data value in the list, one must start at the first position (front) and follow the pointers from element to element until data item is located.
- List is not a direct access structure
- Advantage: ability to add and remove items efficiently at any position in the sequence

List Container

emplace_front	Construct and insert element at beginning (public member function)
push_front	Insert element at beginning (public member function)
pop_front	Delete first element (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
Insert	Insert elements (public member function)
Erase	Erase elements (public member function)
Swap	Swap content (public member function)
Resize	Change size (public member function)
Clear	Clear content (public member function)

Demo

```
#include <list>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
      list<int> mylist;
      int myint;
for(int i=0; i<5;i++)
{
    mylist.push_back (i);
}
    cout << "mylist stores " << mylist.size() << " numbers.\n";
  return 0;
}
```

Demo

```
#include <list>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
   int myints[]= {17,89,89,14,22,77};
  list<int> mylist (myints,myints+6);
  mylist.remove(89);
  cout << "mylist contains:";
  for (list<int>::iterator it=mylist.begin(); it!=mylist.end(); ++it)
    cout << ' ' << *it;cout << '\n';
  return 0;

}
```

Demo

```
#include <list>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
   int myints[]= {17,89,89,14,22,77};
  list<int> mylist (myints,myints+6);
  mylist.remove(89);
  cout << "mylist contains:";
  for (list<int>::iterator it=mylist.begin(); it!=mylist.end(); ++it)
    cout << ' ' << *it;cout << '\n';
  return 0;

}
```

Demo

```
#include <list>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
     list<int> mylist;
     list<int>::iterator it;
for (int i=0; i<10; ++i)  mylist.push_back(i*10);
  it = mylist.begin();
  ++it; mylist.erase(it);
  for (it=mylist.begin(); it!=mylist.end(); ++it)
cout << ' ' << *it;
cout << '\n';
  return 0;
}
```

Stack Container

- Adapter Container
- These containers restrict how elements enter and leave a sequence
- Stack
  - allows access at only one end of the sequence (top)
  - Adds objects to container by *pushing* the object onto the stack
  - Removes objects from container by *popping* the stack
  - LIFO ordering (last end, first out)

Functions

- **empty** Test whether container is empty (public member function)
- **size** Return size (public member function)
- **top** Access next element (public member function)
- **push** Insert element (public member function)
- **emplace** Construct and insert element (public member function)
- **pop** Remove top element (public member function)
- **swap** Swap contents (public member function)

Demo

```
#include <stack>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    stack<int> mystack;
    for (int i=0; i<5; ++i) mystack.push(i);
    cout << "Popping out elements...";
while (!mystack.empty())
{
      cout << ' ' << mystack.top();
      mystack.pop();
}
cout << '\n'; return 0;
}
```

Exercise

Transferred from decimal to binary using stack

Queue Container

- Queue
  - Allows access only at the front and rear of the sequence
  - Items enter at the rear and exit from the front
  - Example: waiting line at a grocery store
  - FIFO ordering (first-in first-out)
  - *push(add* object to a queue)
  - *pop* (remove object from queue)

Member functions

- (constructor) Construct queue (public member function)
- empty Test whether container is empty (public member function)
- size Return size (public member function)
- front Access next element (public member function)
- back Access last element (public member function)
- push Insert element (public member function)
- emplace Construct and insert element (public member function)
- pop Remove next element (public member function)
- swap Swap contents (public member function)

Demo

```
#include <queue>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{  queue<int> myqueue;
int myint;
cout << "Please enter some integers (enter 0 to end):\n";
do {
cin >> myint; myqueue.push (myint);
} while (myint);
cout << "myqueue contains: ";
while (!myqueue.empty()) {
cout << ' ' << myqueue.front(); myqueue.pop();}
cout << '\n';
return 0;
}
```

Demo

```
#include <queue>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
queue<int> myqueue;
myqueue.push(12);
myqueue.push(75);   // this is now the back
myqueue.back() -= myqueue.front();
cout << "myqueue.back() is now " << myqueue.back() << '\n';
return 0;
}
```

Algorithm

- **Sort** Sort elements in range (function template)
- **binary_search** Test if value exists in sorted sequence (function template)
- **Rotate** Rotate left the elements in range (function template)
- **Reverse** Reverse range (function template)

Demo

```
#include <vector>
#include <algorithm>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int myints[] ={10,40,30,22,34,3,21,15};
vector<int> myvector(myints,myints+8);
sort(myvector.begin(),myvector.end());
cout<<"Sap xep tang dan"<<endl;
for(int i=0; i<myvector.size();i++)
cout<<' '<<myvector[i];
cout<<endl; return 0;
}
```

Demo

```
#include <vector>
#include <algorithm>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int myints[] ={10,40,30,22,34,3,21,15};
vector<int> myvector(myints,myints+8);
cout<<"Sap xep tang dan 4 phan tu dau"<<endl;
sort (myvector.begin(), myvector.begin()+4);
for(int i=0; i<myvector.size();i++)
cout<<' '<<myvector[i];
cout<<endl;
```

Demo

```
#include <vector>
#include <algorithm>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int myints[] ={10,40,30,22,34,3,21,15};
vector<int> myvector(myints,myints+8);
cout<<"Sap xep tang dan 4 phan tu dau"<<endl;
sort (myvector.begin(), myvector.begin()+4);
for(int i=0; i<myvector.size();i++)
cout<<' '<<myvector[i];
cout<<endl;
*/
cout<<"Sap xep tang dan 4 phan tu sau"<<endl;
sort (myvector.begin()+4, myvector.end());
for(int i=0; i<myvector.size();i++)
cout<<' '<<myvector[i];
cout<<endl;
return 0;}
```

Demo

```
#include <vector>
#include <algorithm>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int myints[] ={10,40,30,22,34,3,21,15};
vector<int> myvector(myints,myints+8);
cout<<"Sap xep tang dan 4 phan tu sau"<<endl;
sort (myvector.begin()+4, myvector.end());
for(int i=0; i<myvector.size();i++)
cout<<' '<<myvector[i];
cout<<endl;
return 0;

}
```

Demo

```
#include <vector>
#include <algorithm>
using namespace std;
bool myfunction (int i,int j) { return (i>j); }
int _tmain(int argc, _TCHAR* argv[])
{
int myints[] ={10,40,30,22,34,3,21,15};
vector<int> myvector(myints,myints+8);
cout<<"Sap xep tang giam dan"<<endl;
sort (myvector.begin(), myvector.end(),myfunction);
for(int i=0; i<myvector.size();i++)
cout<<' '<<myvector[i];
cout<<endl;
return 0;

}
```

Demo

```
#include <vector>
#include <algorithm>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int myints[] ={10,40,30,22,34,3,21,15};
vector<int> myvector(myints,myints+8);
reverse(myvector.begin(),myvector.end());
cout<<"Dao nguoc vector"<<endl;
for(int i=0; i<myvector.size();i++)
cout<<' '<<myvector[i];
cout<<endl;
return 0;
}
```

Summary 1

- Iterator is "generalization" of a pointer
  - Used to move through elements of container
- Container classes with iterators have:
  - Member functions end() and begin() to assist cycling
- Main kinds of iterators:
  - Forward, bi-directional, random-access
- Given constant iterator p, *p is read-only version of element

Summary 2

- Given mutable iterator p → *p can be assigned value
- Bidirectional container has reverse iterators allowing reverse cycling
- Main STL containers: list, vector, deque
  - stack, queue: container adapter classes
- set, map, multiset, multimap containers store in sorted order
- STL implements generic algorithms
  - Provide maximum running time guarantees