Data Structures & STL

ACM-ICPC 2012 TUTORIAL #2

International College, KMITL: Gateway to Professional Success in the International Arena

Content

- STL – Standard Template Library
- Stack
- Queue
- Priority queue
- Vector
  - Iterator
- Set
- Dictionary

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

STL – Standard Template Library

STL- Standard Template Library

- A standard library in C++
  - Every C++ compiler supports STL
- Use template technique
  - Generic programming (support several data types)
- Contain important data structures
  - Stack, queue, priority queue
  - Vector, deque, list
  - Set, map (dictionary)

  - See the completed list at: http://www.cplusplus.com/reference/stl/

STL- Standard Template Library

- Contain useful algorithms
  - find, find_if, search, search_n
  - count, count_if
  - sort
  - replace, replace_if
  - remove, remove_if
  - min, max, min_element, max_element

  - See the completed list at: http://www.cplusplus.com/reference/algorithm/

Stack

Stack

- LIFO: Last-In, First-Out

Methods of stack

Methods (or operators)	Description
size	Return size
empty	Test whether stack is empty
top	Access next element
push	Add element
pop	Remove element

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main ()
{
  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 << endl;
  return 0;
}
```

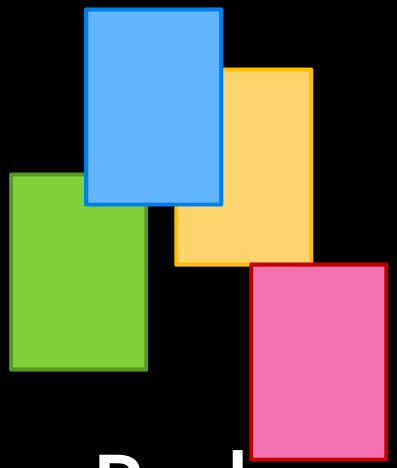
Result

```
Popping out elements... 4 3 2 1 0
Press any key to continue
```

Queue

Queue

- FIFO: First-In, First-Out



Pop

Back

Push

Front

Methods of queue

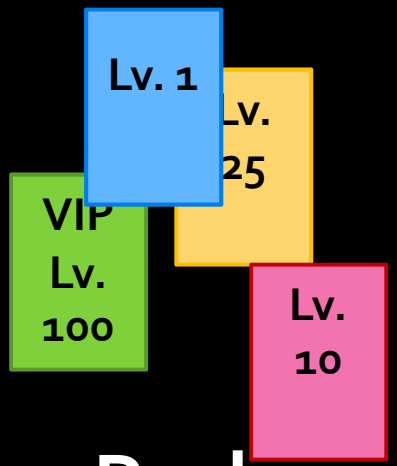
Methods (or operators)	Description
size	Return size
empty	Test whether queue is empty
front	Access next element
back	Access last element
push	Insert element
pop	Remove next element

```
#include <iostream>
#include <queue>
using namespace std;
int main ()
{
  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();
  }
  return 0;
}
```

Priority queue

Priority queue

- Its first element is always the greatest of the elements it contains



Pop

Push

Back

Front

Methods of Priority queue

Methods (or operators)	Description
size	Return size
empty	Test whether priority queue is empty
top	Access top element
push	Insert element
pop	Remove top element

```
#include <iostream>
#include <queue>
using namespace std;
int main ()
{
  priority_queue<int> mypq;
  mypq.push(30);
  mypq.push(100);
  mypq.push(25);
  mypq.push(40);
  cout << "Popping out elements...";
  while (!mypq.empty())
  {
     cout << " " << mypq.top();
     mypq.pop();
  }
  cout << endl;
  return 0;
}
```

Result

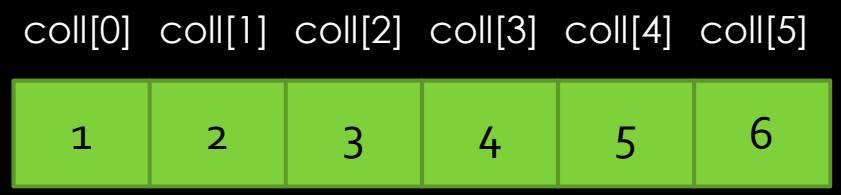
```
Popping out elements... 100 40 30 25
Press any key to continue
```

Vector

Vector

- Similar to an array
  - holds a sequence of values in contiguous memory locations
  - Can access vector elements using []
- But vector has several advantages over an array:
  - Need not to declare the number of elements
  - Automatically increases the size
  - Retrieve the number of elements using simpler syntax

Vector



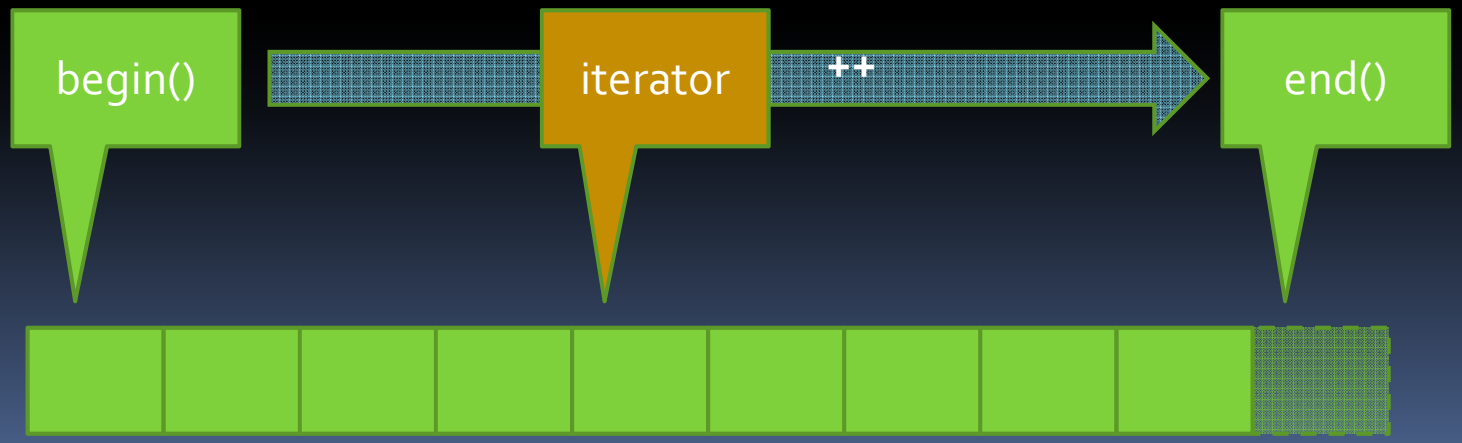

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   vector<int> coll;
   for( int i=1; i<=6;  ++i )
      coll.push_back( i );
   for( i=0; i<coll.size();  ++i )
      cout << coll[i] << ' ';
   cout << endl;
   return 0;
}
```

Result

```
1 2 3 4 5 6
Press any key to continue
```

Iterator

- STL containers are generally accessed using *iterators*.
- Iterators are objects that can 'iterate' containers.
- Every iterator object represents a position in a container.



Iterator

- Operators of iterator

Operators	Description
operator*	Return the element at the position of the iterator
operator++	Advance the iterator one step
operator==	Returns whether two iterators represent the same object (same container, same position)
operator=	Assigns the position of one iterator to another

- Methods of container

Methods	Description
begin()	Returns an iterator representing the first element
end()	Returns an iterator representing the position next to the last element

```cpp
// vector::begin
#include <iostream>
#include <vector>
using namespace std;
int main ()
{
  vector<int> myvector;
  for (int i=1; i<=5; i++)
      myvector.push_back(i);
  vector<int>::iterator it;
  cout << "myvector contains:";
  for ( it=myvector.begin() ; it < myvector.end(); it++ )
    cout << " " << *it;
  cout << endl;
  return 0;
}
```

Result

```
myvector contains: 1 2 3 4 5
Press any key to continue
```

Important methods of vector

Methods (or operators)	Description
operator=	Copy vector content
size	Return size
empty	Test whether vector is empty
operator[]	Access element
at	Access element
front	Access first element
back	Access last element
push_back	Add element to the end
pop_back	Delete last element
insert	Insert elements
erase	Erase elements
clear	Clear content

```
#include <iostream>
#include <vector>
using namespace std;
int main ()
{
  vector<int> myvector (3,100);
  vector<int>::iterator it;
  it = myvector.begin();
  it = myvector.insert ( it , 200 );
  myvector.insert (it,2,300);
  // "it" no longer valid, get a new one:
  it = myvector.begin();
  vector<int> anothervector (2,400);
  myvector.insert (it+2,anothervector.begin(),
                                    anothervector.end());
  int myarray [] = { 501,502,503 };
  myvector.insert (myvector.begin(), myarray, myarray+3);
  cout << "myvector contains:";
  for (it=myvector.begin(); it<myvector.end(); it++)
    cout << " " << *it;
  cout << endl;
  return 0;
}
```

Result

```
myvector contains: 501 502 503 300 300 400 400 200 100 100 100
Press any key to continue
```

Set

Set

- A set is a collection in which each element can only occur once.
- The elements are sorted automatically according to their value.

- If you prefer having the same element occurring several times, use multiset instead of set.

Important methods of set

Methods (or operators)	Description
operator=	Copy container content
size	Return size
empty	Test whether container is empty
insert	Insert elements
erase	Erase elements
clear	Clear content
find	Get iterator to element
count	Count elements with a specific key

```
// set::find
#include <iostream>
#include <set>
using namespace std;
int main ()
{
  set<int> myset;
  set<int>::iterator it;
  // set some initial values:
  for (int i=1; i<=5; i++)
      myset.insert(i*10);    // set: 10 20 30 40 50
  it=myset.find(20);
  myset.erase (it);
  myset.erase (myset.find(40));
  cout << "myset contains:";
  for (it=myset.begin(); it!=myset.end(); it++)
    cout << " " << *it;
  cout << endl;
  return 0;
}
```

Result

```
myset contains: 10 30 50
Press any key to continue
```

```
// set::count
#include <iostream>
#include <set>
using namespace std;
int main ()
{
  set<int> myset;
  int i;
  // set some initial values:
  for (i=1; i<5; i++)
      myset.insert(i*3);   // set:
  for (i=0;i<10; i++)
  {
    cout << i;
    if (myset.count(i)>0)
      cout << " is an element of myset.\n";
    else
      cout << " is not an element of myset.\n";
  }
  return 0;
}
```

Result

```
0 is not an element of myset.
1 is not an element of myset.
2 is not an element of myset.
3 is an element of myset.
4 is not an element of myset.
5 is not an element of myset.
6 is an element of myset.
7 is not an element of myset.
8 is not an element of myset.
9 is an element of myset.
Press any key to continue
```

Dictionary

Dictionary

- Also called associative array or map
- A collection of (key, value) pairs
- Each possible key appears at most once
- Automatically sorted from lower to higher key
- Lookup operation: find the value for a given key

- In STL, use map container class to create a dictionary

Important methods of map

Methods (or operators)	Description
operator=	Copy container content
size	Return size
empty	Test whether container is empty
operator[]	Access element
insert	Insert elements
erase	Erase elements
clear	Clear content
find	Get iterator to element
count	Count elements with a specific key

Iterator of a map

- Use pair data type (class)
  - first
  - second
- Key data members of iterator of a map

```
map<Key,T>::iterator it;
(*it).first;  // the key value (of type Key)
(*it).second; // the mapped value (of type T)
(*it);    // the "element value" (of type pair<const Key,T>)
```

```
it->first;    // same as (*it).first   (the key value)
it->second;   // same as (*it).second  (the mapped value)
```

```
// map::find
#include <iostream>
#include <map>
using namespace std;
int main ()
{
  map<char,int> mymap;
  map<char,int>::iterator it;
  mymap['a']=50;
  mymap['b']=100;
  mymap['c']=150;
  mymap['d']=200;
  it=mymap.find('b');
  mymap.erase (it);
  mymap.erase (mymap.find('d'));
  // print content:
  cout << "elements in mymap:" << endl;
  cout << "a => " << mymap.find('a')->second << endl;
  cout << "c => " << mymap.find('c')->second << endl;
  return 0;
}
```

Result

```
elements in mymap: a => 50 c => 150
Press any key to continue
```

Other useful data structures

- deque (double-ended queue)
  - Like a vector but can push_front, push_back, pop_front, and pop_back
- list
  - Proper for insertion and deletion
- multiset
- multimap

- For more detail, see: http://www.cplusplus.com/reference/stl/