C++ STL PRIORITY QUEUE

STL Priority Queue

- Implemented using a heap
- Operations:
  - push(new_item)
  - pop(): removes but does not return top item
  - top() return top item (item at back/end of the container)
  - size()
  - empty()
- http://www.cplusplus.com/reference/stl/priority_queue/push/
- By default, implemented using a **max heap** but can use comparator functors to use **min heap**
- **Runtime: O(log(n)) push and pop while all other functions are constant (i.e. O(1))**

```
// priority_queue::push/pop
#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;
 }
```

Code here will print
100 40 30 25

STL Priority Queue Template

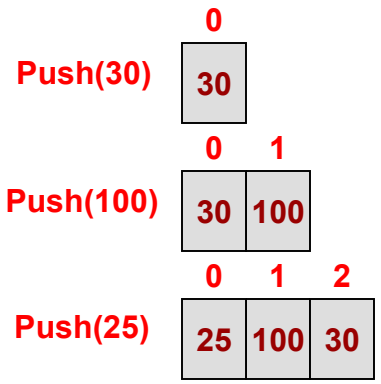
- Template that allows type of element, container class, and comparison operation for ordering to be provided
- First template parameter should be type of element stored
- Second template parameter should be the container class you want to use to store the items (usually `vector<type_of_elem>`)
- Third template parameters should be comparison functor that will define the order from first to last in the container

```
// priority_queue::push/pop
#include <iostream>
#include <queue>
using namespace std;

int main ()
{ priority_queue<int, vector<int>, greater<int>> mypq;
  mypq.push(30); mypq.push(100); mypq.push(25);
  cout<< "Popping out elements...";
  while (!mypq.empty()) {
    cout<< " " << mypq.top();
    mypq.pop();
  }
}
```

Code here will print
25, 30, 100

greater<int> will yield a min-heap
less<int> will yield a max-heap



Push(n): adds n to PQ
Top(): Return min element in PQ
Pop(): Removes min element in PQ

C++ less and greater

- For classes that have operators < or > and no need for newly written functor: use the C++ built-in functors `less` and `greater`
- Less
  - Compares two objects of type T using the operator< defined for T
- Greater
  - Compares two objects of type T using the operator> defined for T

```
template <typename T>
struct less
{
  bool operator()(const T& v1, const T& v2){
    return v1 < v2;
  }
};

template <typename T>
struct greater
{
  bool operator()(const T& v1, const T& v2){
    return v1 > v2;
  }
};
```

STL Priority Queue Template

- User defined classes must implement operator<() for max-heap or operator>() for min-heap **OR** a custom functor
- Main() will print names in the following order:
  - Jane
  - Charlie
  - Bill

```
// priority_queue::push/pop
#include <iostream>
#include <queue>
#include <string>
using namespace std;

class Item {
 public:
  int score;
  string name;

  Item(int s, string n) { score = s; name = n;}
  bool operator>(const Item &rhs) const
  { if(this->score > rhs.score) return true;
    else return false;
  }
};

int main ()
{
  priority_queue<Item, vector<Item>, greater<Item> > mypq;
  Item i1(25,”Bill”);    mypq.push(i1);
  Item i2(5,”Jane”);     mypq.push(i2);
  Item i3(10,”Charlie”); mypq.push(i3);
  cout<< "Popping out elements...";
  while (!mypq.empty()) {
    cout<< " " << mypq.top().name;
    mypq.pop();
} }
```