C++ Foundation with Data Structures

Lecture 15 : Priority Queues

Implementing Min Priority Queue :

A priority queue is just like a normal queue data structure except that each element inserted is associated with a “priority”.
It supports the usual push(), pop(), top() etc operations, but is specifically designed so that its first element is always the greatest of the elements it contains, i.e. max heap.

In STL, priority queues take three template parameters:

```
template <class T,
        class Container = vector<T>,
        class Compare = less<typename Container::value_type>
        >
```

class priority_queue;

- The first element of the template defines the class of each element. It can be user-defined classes or primitive data-types. Like in you case it can be int, float or double.
- The second element defines the container to be used to store the elements. The standard container classes std::vector and std::dequeue fulfil these requirements. It is usually the vector of the class defined in the first argument. Like in your case it can be vector<int>, vector<float>, vector<double>.
- The third element is the comparative class. By default it is less<T> but can be changed to suit your need. For min heap it can be changed to greater<T>.

Example

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

int main() {
   priority_queue<int, vector<int>, greater<int> > pq;
   pq.push(40);
```

```
    pq.push(320);
    pq.push(42);
    pq.push(42);
    pq.push(65);
    pq.push(12);
    pq.push(245);
    cout << pq.top() << endl;
    return 0;
}
```

The above code used the greater<T> functional. Below is the code using a comparative class which performs operator overloading. The code below will make it clear.

```
#include<iostream>
#include <queue>
using namespace std;
class Comp{
    public:
    bool operator () (int a, int b) {
        return a > b;
    }
};

int main() {
    priority_queue<int, vector<int>, Comp> pq;
    pq.push(40);
    pq.push(320);
    pq.push(42);
    pq.push(42);
    pq.push(65);
    pq.push(12);
    pq.push(245);
    cout<<pq.top()<<endl;
    return 0;
}
```

The output for both the code will be 12.

The priority_queue uses the function inside Comp class to maintain the elements sorted in a way that preserves *heap properties*(i.e., that the element popped is the last according to this *strict weak ordering*).

In above example we have used custom function which will make the heap as min-heap.