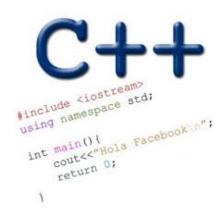
HEAPS: IMPLEMENTATION PRIORITY QUEUES COMPARISON CLASSES

Problem Solving with Computers-II





std::priority_queue (STL's version of heap)

A C++ priority_queue is a generic container, and can store any data type on which an ordering can be defined: for example ints, structs (Card), pointers etc.

```
#include <queue>
priority_queue<int> pq;
```

Methods:

```
*push() //insert
*pop() //delete max priority item
*top() //get max priority item
*empty() //returns true if the priority queue is empty
```

- You can extract object of highest priority in O(log N)
- To determine priority: objects in a priority queue must be comparable to each other

STL Heap implementation: Priority Queues in C++

```
What is the output of this code?
```

```
priority queue<int> pq;
pq.push(10);
pq.push(2);
pq.push(80);
cout<<pre>cout<<pre>cout<</pre>
pq.pop();
cout<<pq.top();
pq.pop();
cout<<pre>cout<<<pre>pq.top();
pq.pop();
```

```
A.10 2 80
B.2 10 80
C.80 10 2
D.80 2 10
E. None of the above
```

std::priority_queue template arguments

```
template <
    class T,
    class Container= vector<T>,
    class Compare = less <T>
        class priority_queue;
```

The template for priority_queue takes 3 arguments:

- 1. Type elements contained in the queue.
- 2. Container class used as the internal store for the priority_queue, the default is vector<T>
- 3. Class that provides priority comparisons, the default is less

Comparison class

 Comparison class: A class that implements a function operator for comparing objects

```
class compareClass{
    bool operator()(int& a, int & b) const {
        return a>b;
    }
};
```

Comparison class

```
class compareClass{
       bool operator()(int& a, int & b) const {
             return a>b;
};
int main(){
                              What is the output of this code?
    compareClass c;
                              A. 1
    cout << c(10, 20) << endl; B.0
                               C. Error
```

STL Heap implementation: Priority Queues in C++

```
class cmp{
       bool operator()(int& a, int & b) const {
               return a>b;
priority_queue<int, vector<int>, cmp> pq;
pq.push(10);
pq.push(2);
pq.push(80);
cout<<pre><<pre>pq.top();
                       Output:
pq.pop();
cout<<pq.top();
                       pq is a
                                       heap
pq.pop();
cout<<pre><<pre>pq.top();
pq.pop();
```

std::priority_queue template arguments

```
//Template parameters for a max-heap
priority_queue<int, vector<int>, std::less<int>> pq;

//Template parameters for a min-heap
priority_queue<int, vector<int>, std::greater<int>> pq;
```

Sort array elements using a pq storing pointers

```
int main(){
     int arr[]=\{10, 2, 80\};
     priority queue<int*> pq;
     for(int i=0; i < 3; i++)
          pq.push(arr+i);
     while(!pq.empty()){
          cout << *pq.top() << endl;
         pq.pop();
     return 0;
```

How can we change the way pq prioritizes pointers?

Write a comparison class to print the integers in the array in sorted order

```
int main(){
     int arr[]=\{10, 2, 80\};
     priority queue<int*, vector<int*>, cmpPtr> pq;
     for(int i=0; i < 3; i++)
           pq.push(arr+i);
     while(!pq.empty()){
           cout << *pq.top() << endl;
         pq.pop();
     return 0;
```