PRIORITY QUEUE ← (STL)
COMPARISON CLASSES

Problem Solving with Computers-II

Heaps → min-Heap

→ max-Heap

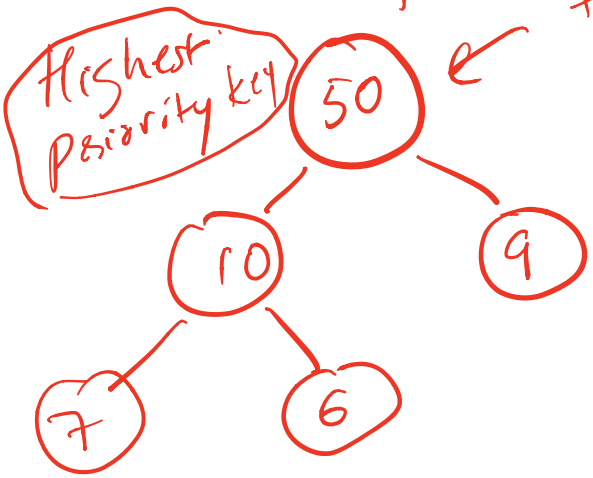
C++

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

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

GitHub

max-Heap



Highest (max) top

Default priority queue is a max-Heap

but it can be configured as a min-Heap

From last class….

type of the key

Default configuration

```
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;
}
```

What is the output of this code?

A. 10 2 80

B. 2 10 80

C. 80 10 2

D. 80 2 10

E. None of the above

sorted order

min-heap:

priority-queue<int, vector<int>, greater<int>> pq;

comparison class

std::priority_queue template arguments

The template for priority_queue takes 3 arguments:

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

type of keys

data structure used to represent the heap

- The first is the type of the elements contained in the queue.
- If it is the only template argument used, the remaining 2 get their default values:
  - a **vector<T>** is used as the internal store for the queue,
  - **less is a comparison** class that provides priority comparisons

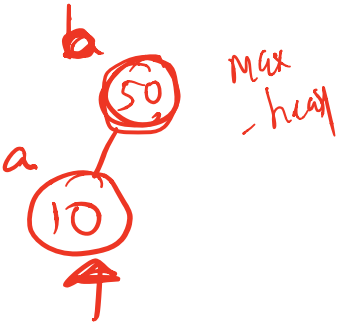
priority-queue < int, vector<int>, cmp > pq;

custom class that tells pq how to compare keys.

① how does pq use the cmp?

if (cmp(a, b)) {
// a has less priority over b

} else { // a has higher priority



② how can we define our own compare class.

Comparison class

- A class used to perform comparisons.
- Implements a function operator that compares two keys

greater

function operator

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

functor ⇒ function operator

```
//Use cmp to compare any two keys
cmp foo;
cout<<foo(x, y);
```

// use object like a function

foo.operator()(x,y) a

Configure PQ with a comparison class

```
class cmp{
        bool operator()(int& a, int& b) const {
                return a > b;
        }
};
int main(){
        int arr[]={10, 2, 80};
        priority_queue<int, vector<int>, cmp> pq;
        for(int i=0; i < 3; i++)
                pq.push(arr[i]);

        while(!pq.empty()){
                cout<<pq.top()<<endl;
                pq.pop();
        }
        return 0;
}
```

cmp (80, 2)

cmp (2, 10)

false

2 has higher priority than 10

What is the output of this code?

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

Practice functors and PQs:

0x8000 0x8004 0x8008

```
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;
}
```

What is the output of this code?

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

2, 10, 80

```
template <class T>
class cmp {
    public:
        bool operator()(T* a, T* b){
            return *a > *b;
        }
}
```

cannot overload < > == for pointers because pointer is a basic type

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?

80, 2, 10

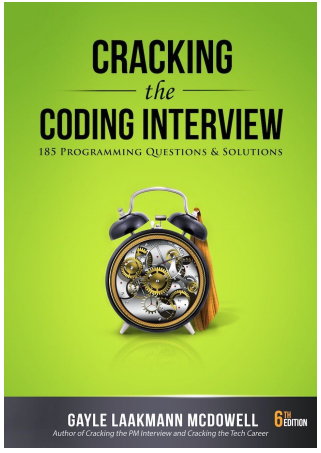
2, 10, 80

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;
}
```

Small group exercise



Write a ADT called in minStack that provides the following methods

- push() // inserts an element to the “top” of the minStack
- pop() // removes the last element that was pushed on the stack
- top () // returns the last element that was pushed on the stack
- min() // returns the minimum value of the elements stored so far
- empty()

5, 3, 9, 7, 3

Practice the following in breakout rooms

- Ask clarifying questions:

  e.g.
  - Does the stack handle only integer keys? → yes
  - What is the expected running time of min()? O(1)
  - Can we use the STL in our implementations? Yes

- Come up with an overall strategy & demonstrate using small examples.
  - Show how you can solve the problem using your strategy
  - Better still, practice thinking aloud to show how you came up with a solution.

- Code your solution → use a real language (in our case C++)