CS24 - Problem Solving with Computers II

Heaps (Priority Queues)

Heaps

Not to be confused with Heap memory, Heap is also a data structure

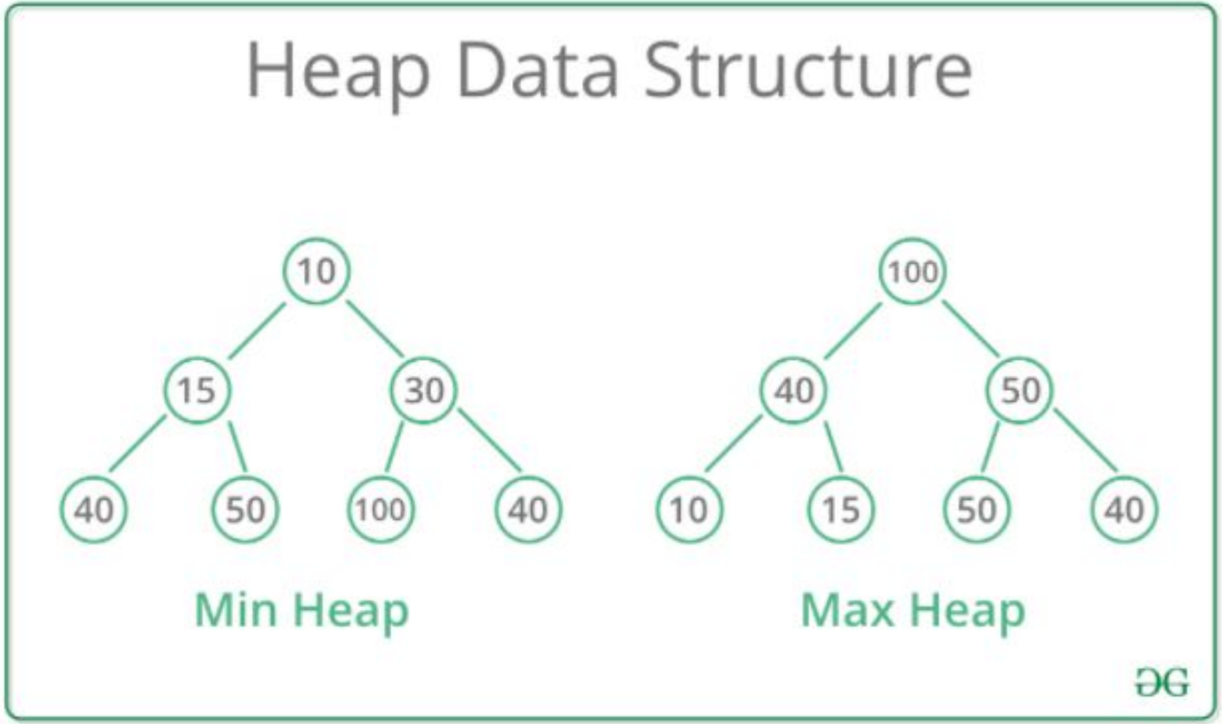
Useful for sorting data

Sorted data can be used to very quickly access max or min (O(1))

Many different types of heap, “binary heap” is typically used for a min/max heap

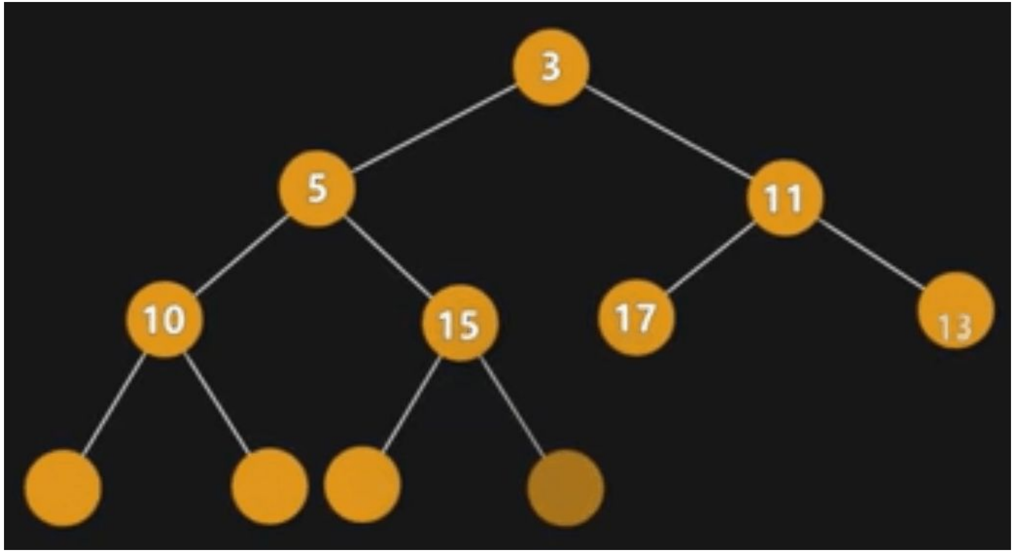
- Binary heaps are complete binary trees (all nodes have two children except for last row which is filled left to right)
- Root of binary tree is the min/max

Heaps


Heap Data Structure
10
15
30
40
50
100
40
Min Heap
100
40
50
10
15
50
40
Max Heap
ƏG

Heaps

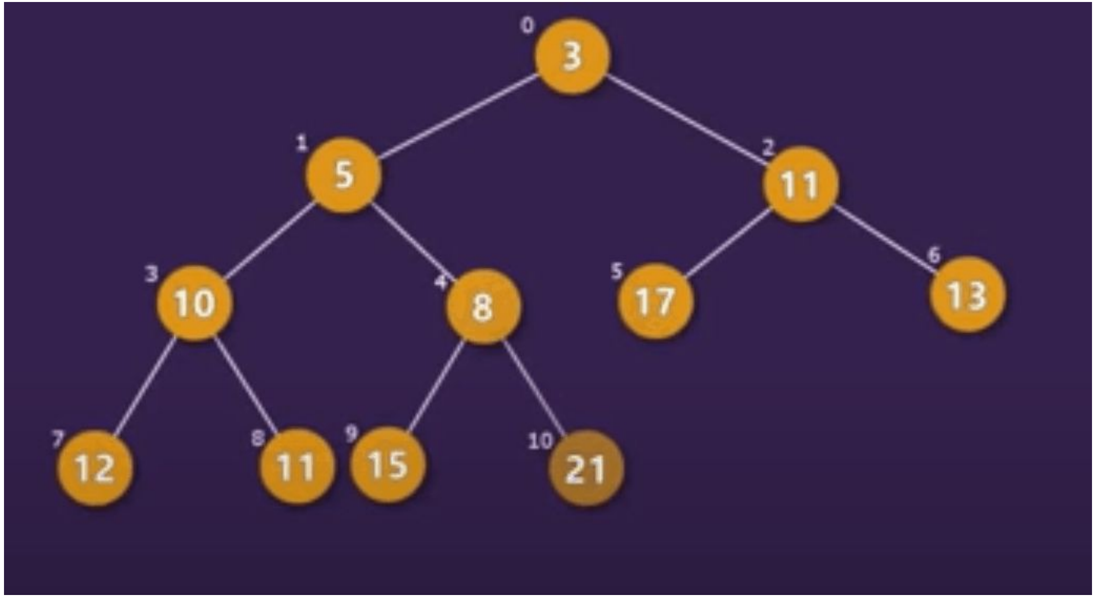
Creating a heap: Insert nodes into the binary tree from left to right, compare them to previous nodes along the branch and rearrange if necessary



Heaps

Deleting nodes is more complicated…

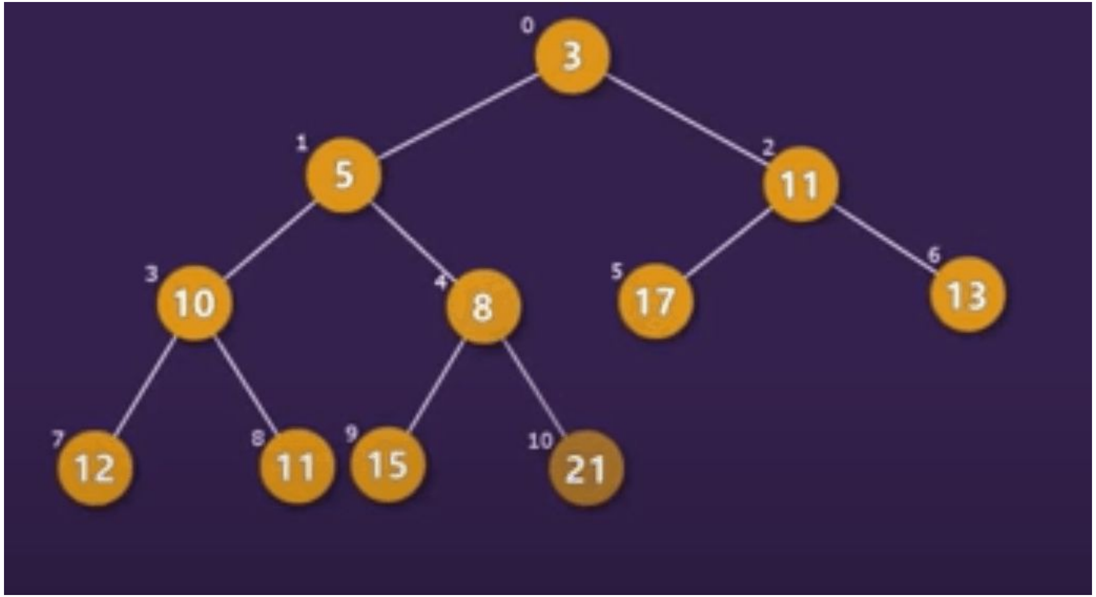
1) Delete the target node, move all nodes in the branch to fill the empty space
2) Move the last node added to the root position
3) Compare on the branch



Heaps

Deleting nodes is more complicated…

1) Delete the target node, move all nodes in the branch to fill the empty space
2) Move the last node added to the root position
3) Compare on the branch



Notice the numbering!

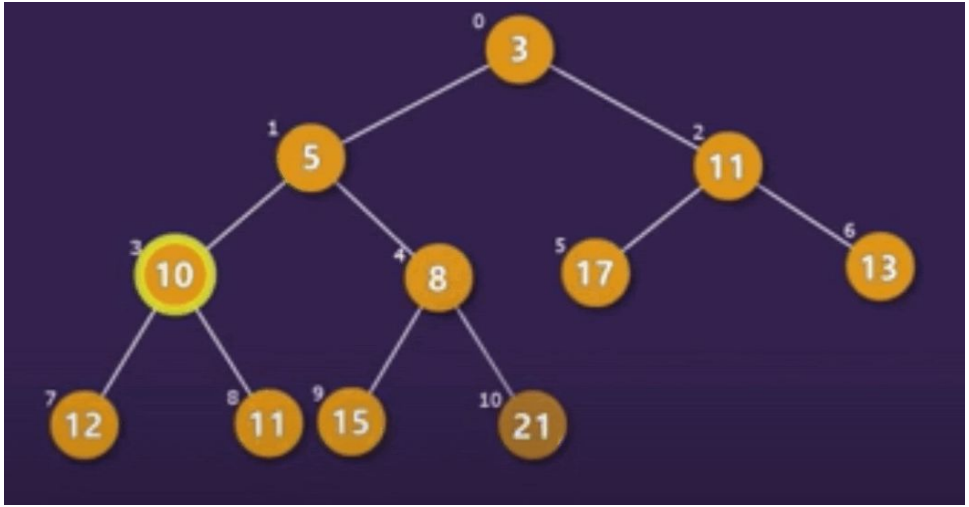
Heaps

In the backend, heaps can be handled as vectors -> {3,5,11,10,8,17,13,12,11,15,21}

Index of parent is floor((i-1)/2)

More space efficient (no need to store parent nodes)

Time complexity benefits?



Heaps

In the backend, heaps can be handled as vectors -> {3,5,11,10,8,17,13,12,11,15,21}

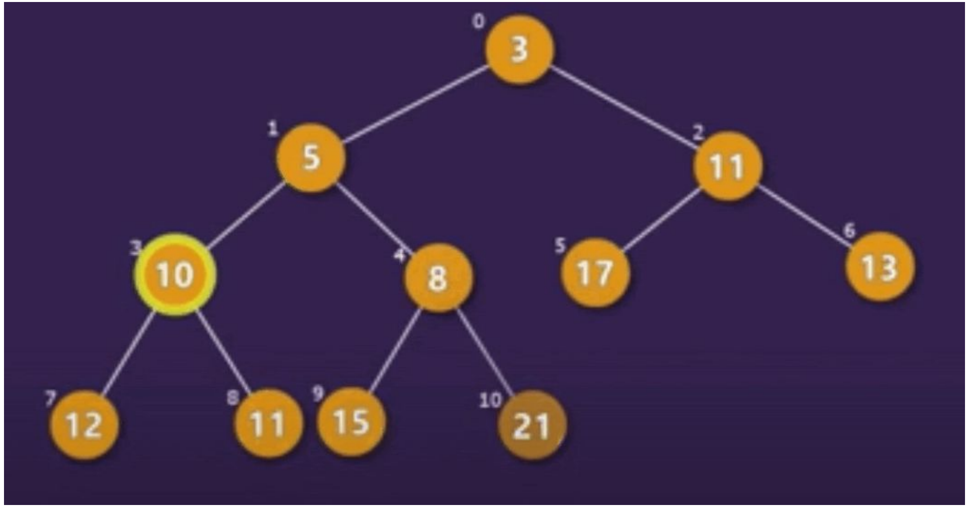
Index of parent is floor((i-1)/2)

More space efficient (no need to store parent nodes)

Time complexities of min/max heaps:

min/max: O(1)

push/pop: O(logN)



Heaps

“Heap” exists in STL through conversion of a vector:

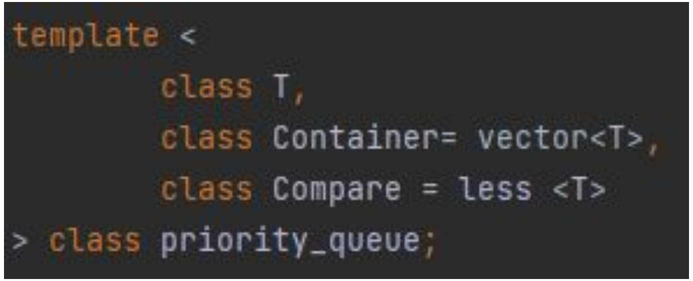
```
#include<bits/stdc++.h>
makeHeap(vector.begin(), vector.end());
vector.end()),
vector.push_back(#),
vector.pop_back()
vector.front()
```

Priority Queues

Another STL implementation of a heap is the priority queue

With priority queues, we get to specify what criteria determines the heap’s order

Type, Container, Comparison ->priority_queue < int, vector<int>, std::less<int>> nameOfPQ;



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

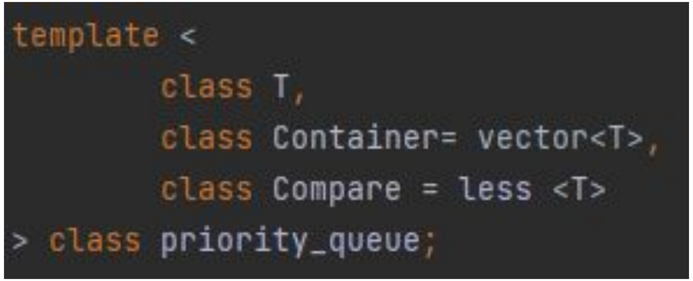
Priority Queues

Another STL implementation of a heap is the priority queue

With priority queues, we get to specify what criteria determines the heap’s order

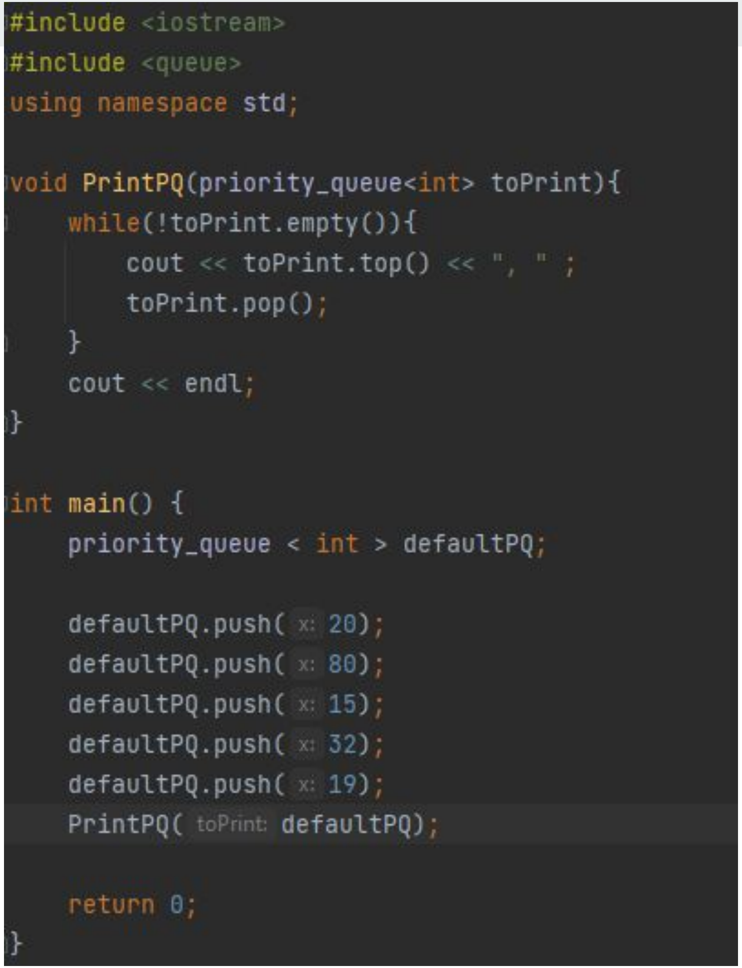
Type, Container, Comparison ->priority_queue < int, vector<int>, std::less<int>> nameOfPQ;

push(#)
pop()
top()
empty()
size()



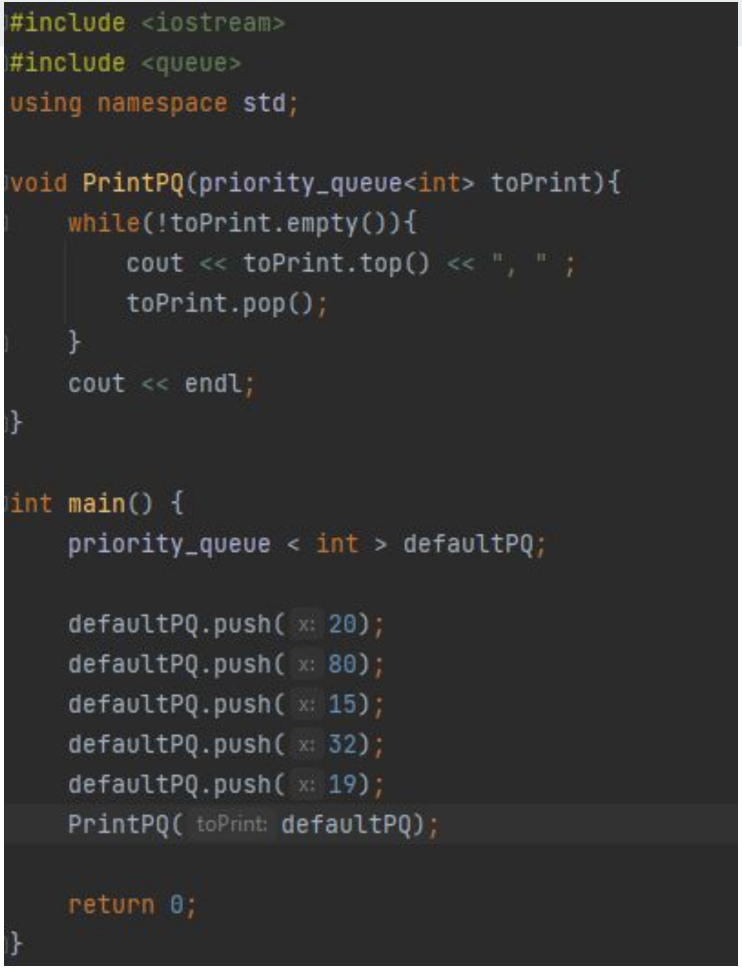
```
template <
        class T,
        class Container= vector<T>,
        class Compare = less <T>
> class priority_queue;
```

Priority Queues

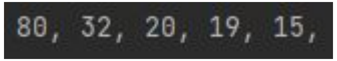


```cpp
#include <iostream>
#include <queue>
using namespace std;

void PrintPQ(priority_queue<int> toPrint){
    while(!toPrint.empty()){
        cout << toPrint.top() << ", " ;
        toPrint.pop();
    }
    cout << endl;
}

int main() {
    priority_queue < int > defaultPQ;

    defaultPQ.push( x: 20);
    defaultPQ.push( x: 80);
    defaultPQ.push( x: 15);
    defaultPQ.push( x: 32);
    defaultPQ.push( x: 19);
    PrintPQ( toPrint: defaultPQ);

    return 0;
}
```

Priority Queues

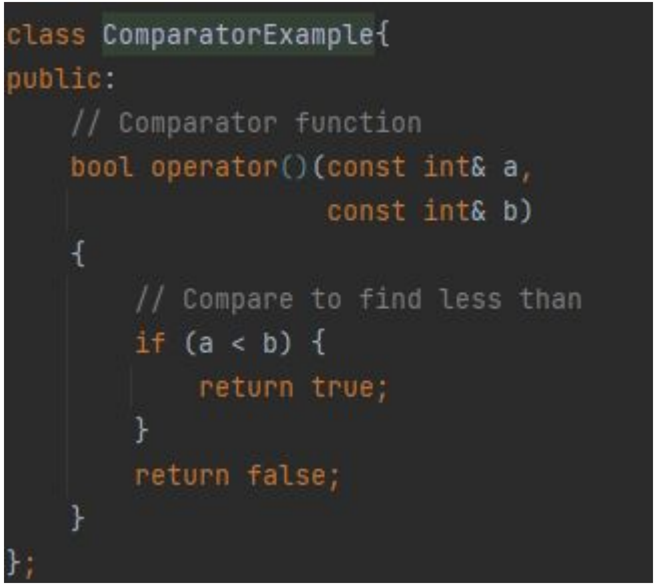


```
#include <iostream>
#include <queue>
using namespace std;

void PrintPQ(priority_queue<int> toPrint){
    while(!toPrint.empty()){
        cout << toPrint.top() << ", " ;
        toPrint.pop();
    }
    cout << endl;
}

int main() {
    priority_queue < int > defaultPQ;

    defaultPQ.push( x: 20);
    defaultPQ.push( x: 80);
    defaultPQ.push( x: 15);
    defaultPQ.push( x: 32);
    defaultPQ.push( x: 19);
    PrintPQ( toPrint: defaultPQ);

    return 0;
}
```

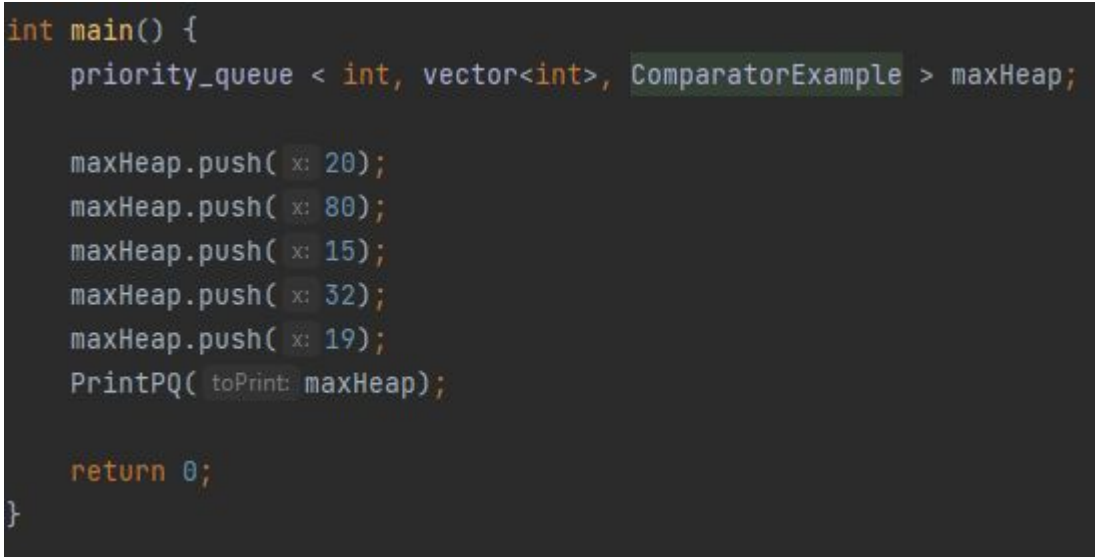


```
80, 32, 20, 19, 15,
```

Priority Queues

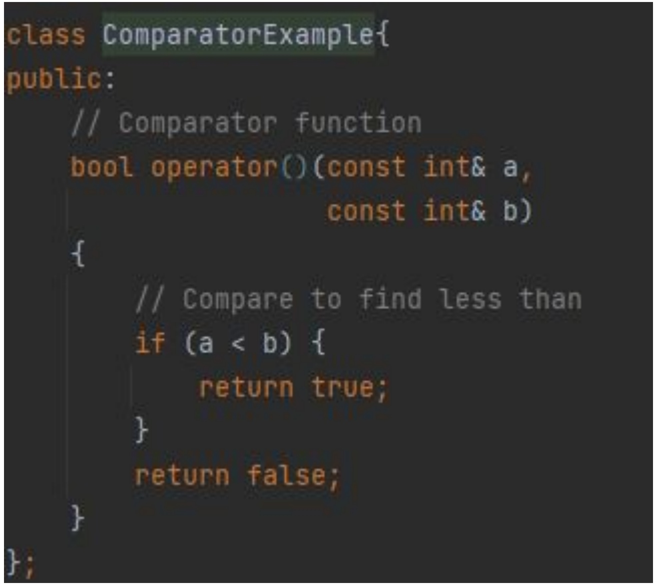


```
class ComparatorExample{
public:
    // Comparator function
    bool operator()(const int& a,
                    const int& b)
    {
        // Compare to find less than
        if (a < b) {
            return true;
        }
        return false;
    }
};
```

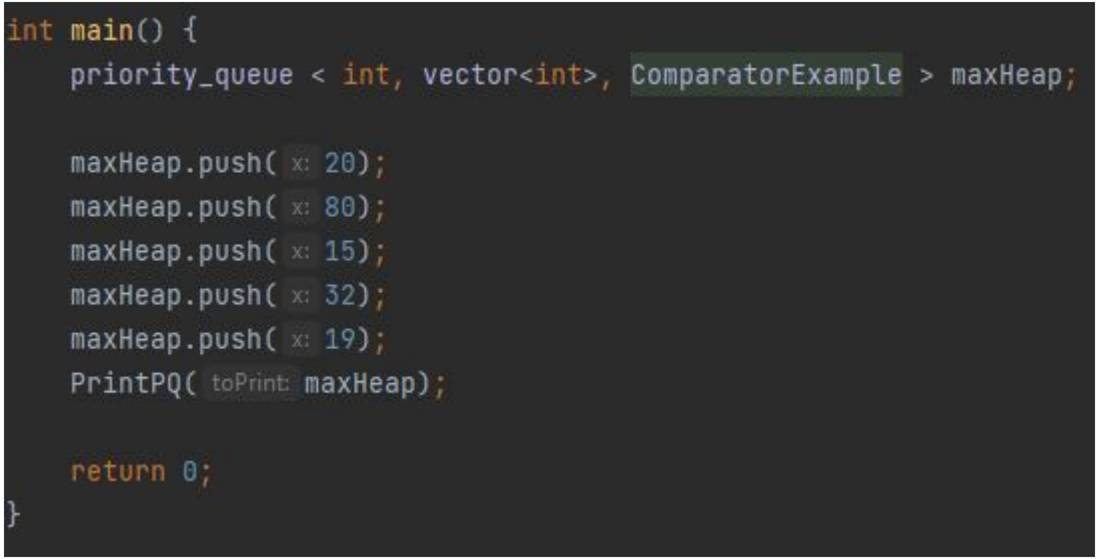


```
int main() {
    priority_queue < int, vector<int>, ComparatorExample > maxHeap;

    maxHeap.push( x: 20);
    maxHeap.push( x: 80);
    maxHeap.push( x: 15);
    maxHeap.push( x: 32);
    maxHeap.push( x: 19);
    PrintPQ( toPrint: maxHeap);

    return 0;
}
```

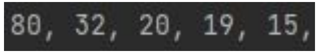
Priority Queues



```
class ComparatorExample{
public:
    // Comparator function
    bool operator()(const int& a,
                    const int& b)
    {
        // Compare to find less than
        if (a < b) {
            return true;
        }
        return false;
    }
};
```

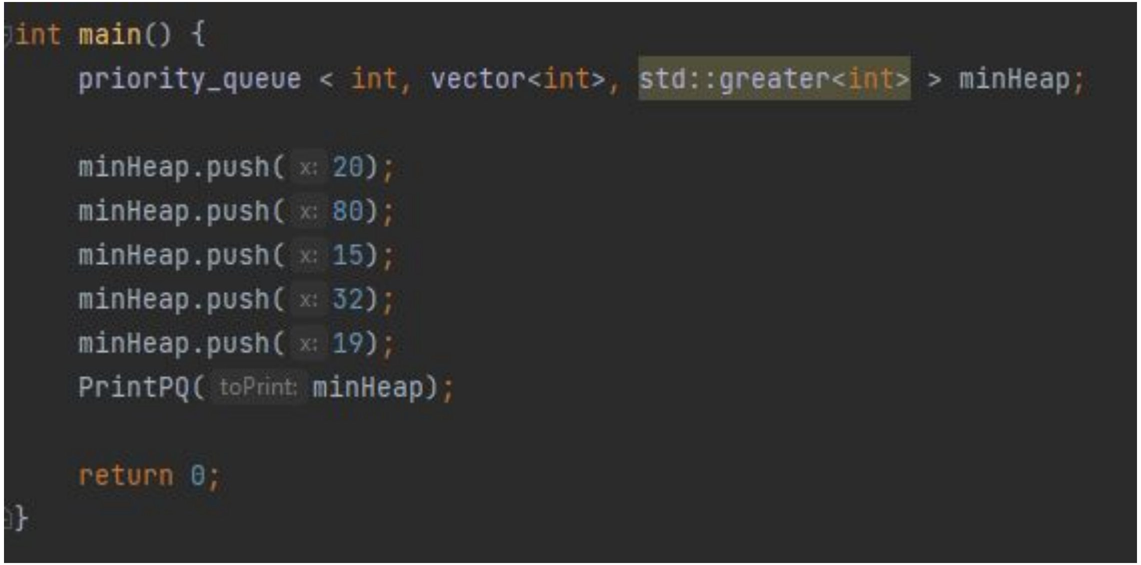


```
int main() {
    priority_queue < int, vector<int>, ComparatorExample > maxHeap;

    maxHeap.push( x: 20);
    maxHeap.push( x: 80);
    maxHeap.push( x: 15);
    maxHeap.push( x: 32);
    maxHeap.push( x: 19);
    PrintPQ( toPrint: maxHeap);

    return 0;
}
```

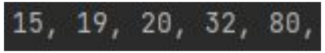


```
80, 32, 20, 19, 15,
```

Priority Queues



```
int main() {
    priority_queue < int, vector<int>, std::greater<int> > minHeap;

    minHeap.push( x: 20);
    minHeap.push( x: 80);
    minHeap.push( x: 15);
    minHeap.push( x: 32);
    minHeap.push( x: 19);
    PrintPQ( toPrint: minHeap);

    return 0;
}
```



```
15, 19, 20, 32, 80,
```

Announcements

Resources for this lecture are posted on Gauchospace

Quiz 4 is next week: time complexity, stacks/queues, anything prior

Lab 04 and Lab 05 due next week, autograder problems

Up next

Sorting!