CSE225L – Data Structures and Algorithms Lab
Lab 14
Priority Queue

In today's lab we will design and implement the Priority Queue.

heaptype.h

```cpp
#ifndef HEAPTYPE_H
#define HEAPTYPE_H

template<class T>
struct HeapType
{
    void ReheapDown(int root, int bottom);
    void ReheapUp(int root, int bottom);
    T* elements;
};

#endif // HEAPTYPE_H
```

heaptype.cpp

```cpp
#include "heaptype.h"

template<class T>
void Swap(T& one, T& two)
{
    T temp;
    temp = one;
    one = two;
    two = temp;
}

template<class T>
void HeapType<T>::ReheapDown(int root, int bottom)
{
    int maxChild;
    int rightChild;
    int leftChild;

    leftChild = root*2+1;
    rightChild = root*2+2;

    if (leftChild <= bottom)
    {
        if (leftChild == bottom)
            maxChild = leftChild;
        else
        {
            if(elements[leftChild]<=elements[rightChild])
                maxChild = rightChild;
            else
                maxChild = leftChild;
        }
        if (elements[root] < elements[maxChild])
        {
            Swap(elements[root], elements[maxChild]);
            ReheapDown(maxChild, bottom);
        }
    }
}
```

```
template<class T>
void HeapType<T>::ReheapUp(int root, int bottom)
{
    int parent;
    if (bottom > root)
    {
        parent = (bottom-1) / 2;
        if (elements[parent] < elements[bottom])
        {
            Swap(elements[parent], elements[bottom]);
            ReheapUp(root, parent);
        }
    }
}
```

pqtype.h

```
#ifndef PQTYPE_H
#define PQTYPE_H
#include "heaptype.h"
#include "heaptype.cpp"

class FullPQ
{};

class EmptyPQ
{};

template<class T>
class PQType
{
private:
    int length;
    HeapType<T> data;
    int max;
public:
    PQType(int);
    ~PQType();
    void MakeEmpty();
    bool IsEmpty();
    bool IsFull();
    void Enqueue(T);
    void Dequeue(T&);
};
#endif // PQTYPE_H
```

pqtype.cpp

```
#include "pqtype.h"

template<class T>
PQType<T>::PQType(int max)
{
    this->max = max;
    data.elements = new T[max];
    length = 0;
}

template<class T>
PQType<T>::~PQType()
```

```
{
    delete [] data.elements;
}

template<class T>
void PQType<T>::MakeEmpty()
{
    length = 0;
}

template<class T>
bool PQType<T>::IsEmpty()
{
    return length == 0;
}

template<class T>
bool PQType<T>::IsFull()
{
    return length == max;
}

template<class T>
void PQType<T>::Enqueue(T newItem)
{
    if (length == max)
        throw FullPQ();
    else
    {
        length++;
        data.elements[length-1] = newItem;
        data.ReheapUp(0, length-1);
    }
}

template<class T>
void PQType<T>::Dequeue(T& item)
{
    if (length == 0)
        throw EmptyPQ();
    else
    {
        item = data.elements[0];
        data.elements[0] = data.elements[length-1];
        length--;
        data.ReheapDown(0, length-1);
    }
}
```

Generate the **driver file (main.cpp)** where you perform the following tasks. Note that you cannot make any change to the header file or the source file.

Operation to Be Tested and Description of Action	Input Values	Expected Output
Create a PQType object with size 15		
Print if the queue is empty or not		Queue is empty
Insert ten items, in the order they appear	`4 9 2 7 3 11 17 0 5 1`	
Print if the queue is empty or not		Queue is not empty
Dequeue one element and print the dequeued value		`17`
Dequeue one element and print the dequeued value		`11`