CSE225L – Data Structures and Algorithms Lab
Lab 16
Heap

In today's lab we will design and implement the Heap ADT.

HeapType.h

```
#ifndef HEAP_TYPE_H
#define HEAP_TYPE_H

class FullHeap{};
class EmptyHeap{};

template<class ItemType>
class HeapType {
    public:
      HeapType(int);
      ~HeapType();
      void MakeEmpty();
      bool IsEmpty();
      bool IsFull();
      void Insert(ItemType newItem);
      void Delete(ItemType &item);
      void Print();
    private:
      int length;
      ItemType* elements;
      int maxItems;
      void ReheapDown(int root, int
bottom);
      void ReheapUp(int root, int
bottom);
};

#endif
```

HeapType.cpp

```
#include "HeapType.h"
#include <iostream>

using namespace std;

template<class ItemType>
HeapType<ItemType>::HeapType(int max)
{
      maxItems = max;
      elements = new ItemType[max];
      length = 0;
}

template<class ItemType>
HeapType<ItemType>::~HeapType()
{
    delete [] elements;
}
```

```
template<class ItemType>
void HeapType<ItemType>::MakeEmpty()
{
    length = 0;
}

template<class ItemType>
bool HeapType<ItemType>::IsFull()
{
    return length == maxItems;
}

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

template<class ItemType>
void HeapType<ItemType>::Insert(ItemType
newItem)
{
    if (length == maxItems)
        throw FullHeap();
    else {
        length++;
        elements[length-1] = newItem;
        ReheapUp(0, length-1);
    }
}

template<class ItemType>
void HeapType<ItemType>::Delete(ItemType&
item)
{
  if (length == 0)
    throw EmptyHeap();
  else {
    item = elements[0];
    elements[0] = elements[length-1];
    length--;
    ReheapDown(0, length-1);
  }
}

template<class ItemType>
void HeapType<ItemType>::Print()
{
    for (int i=0; i<length; i++) {
        cout << elements[i] << endl;
    }
}
```

```
template <class ItemType>
void Swap(ItemType &one, ItemType &two)
{
    ItemType temp;
    temp = one;
    one = two;
    two = temp;
}
template<class ItemType>
void HeapType<ItemType>::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);
        }
    }
}
```

```
template<class ItemType>
void HeapType<ItemType>::ReheapDown(int
root, int bottom)
{
    int maxChild, rightChild = root*2+2,
leftChild = root*2+1;
    if (leftChild <= bottom) { //there is
at least one child
        if (leftChild == bottom) //it is
the only child
            maxChild = leftChild;
        else { //there are two children
            if (elements[leftChild] <=
elements[rightChild])
                maxChild = rightChild;
            else
                maxChild = leftChild;
        }
        if (elements[root] <
elements[maxChild]) {
            Swap(elements[root],
elements[maxChild]);
            ReheapDown(maxChild, bottom);
        }
    }
}
```

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 Heap object with size 10		
• Print if the heap is empty or not		Heap is Empty
• Insert six items, in the order they appear	4 9 11 17 0 1	
• Print if the Heap is empty or not		Heap is not Empty
• Check if the Heap is full		Heap is not full
• Print the values in the Heap		17 11 9 4 0 1
• Delete one item and print the deleted value		17
• Delete one item and print the deleted value		11