**CSE225L – Data Structures and Algorithms Lab**

**Lab 14**

**Priority Queue**

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

|  |  |
| --- | --- |
| **heaptype.h**  #ifndef HEAPTYPE\_H\_INCLUDED  #define HEAPTYPE\_H\_INCLUDED  template<class ItemType>  struct HeapType  {  void ReheapDown(int root, int bottom);  void ReheapUp(int root, int bottom);  ItemType\* elements;  int numElements;  };  #endif // HEAPTYPE\_H\_INCLUDED  **heaptype.cpp**  #include "heaptype.h"  template<class ItemType>  void Swap(ItemType& one, ItemType& two)  {  ItemType temp;  temp = one;  one = two;  two = temp;  }  template<class ItemType>  void HeapType<ItemType>::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 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);  }  }  } | **pqtype.h**  #ifndef PQTYPE\_H\_INCLUDED  #define PQTYPE\_H\_INCLUDED  #include "heaptype.h"  #include "heaptype.cpp"  class FullPQ  {};  class EmptyPQ  {};  template<class ItemType>  class PQType  {  public:  PQType(int);  ~PQType();  void MakeEmpty();  bool IsEmpty();  bool IsFull();  void Enqueue(ItemType);  void Dequeue(ItemType&);  private:  int length;  HeapType<ItemType> items;  int maxItems;  };  #endif // PQTYPE\_H\_INCLUDED  **pqtype.cpp**  #include "pqtype.h"  template<class ItemType>  PQType<ItemType>::PQType(int max)  {  maxItems = max;  items.elements=new ItemType[max];  length = 0;  }  template<class ItemType>  PQType<ItemType>::~PQType()  {  delete [] items.elements;  }  template<class ItemType>  void PQType<ItemType>::MakeEmpty()  {  length = 0;  }  template<class ItemType>  bool PQType<ItemType>::IsEmpty()  {  return length == 0;  }  template<class ItemType>  bool PQType<ItemType>::IsFull()  {  return length == maxItems;  } |

|  |  |
| --- | --- |
| template<class ItemType>  void PQType<ItemType>::Enqueue(ItemType newItem)  {  if (length == maxItems)  throw FullPQ();  else  {  length++;  items.elements[length-1] = newItem;  items.ReheapUp(0, length-1);  }  } | template<class ItemType>  void PQType<ItemType>::Dequeue(ItemType& item)  {  if (length == 0)  throw EmptyPQ();  else  {  item = items.elements[0];  items.elements[0] = items.elements[length-1];  length--;  items.ReheapDown(0, length-1);  }  } |

Now generate the **Driver file (main.cpp)** where you perform the following tasks:

| **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 |
| * You have ***N*** magical bags of candies in front of you. The *ith* bag has ***Ai*** candies in it. It takes you one minute to finish a bag of candies, no matter how many candies in it. Every time you finish a bag with ***X*** candies in it, the bag is magically replenished with ***X/2*** (rounded down to the nearest integer) more candies. Write a program that determines the maximum number of candies you can eat in K minutes.   The input is a sequence of integers. The first integer ***N*** is the number of bags. The next integer ***K*** is the number of minutes you have. The next ***N*** integers is the number of candies in the bags. The output of your program is a single integer which represents the maximum number of candies you can eat. | 5 3 2 1 7 4 2 | 14 |