A Heap implemented as an array in C++

Header file for a heap item

```
//----------------------------------------------------------------
// HeapItem.h
// Simple class with which to build the heap demonstration.
//
// Author: Dr. Rick Coleman
//----------------------------------------------------------------
#ifndef HEAPITEM_H
#define HEAPITEM_H

class HeapItem
{
     private:
          int m_iKey;                              // Heap item priority key
          double m_dData;                         // Dummy data value

     public:
          HeapItem();                              // Default constructor
          HeapItem(int key, double data);     // Constructor
          ~HeapItem();                         // Destructor
          int getKey();                         // Return item priority
          void setKey(int key);               // Set the priority key value
          double getData();                    // Return data item
          void setData(double data);          // Set the data item value
};

#endif
```

Implementation (.cpp) file for a heap item

```
//----------------------------------------------------------------
// HeapItem.cpp
// Implementation file for a simple class with which to build
//          the heap demonstration.
//
// Author: Dr. Rick Coleman
//----------------------------------------------------------------
#include "HeapItem.h"

//-----------------------------------
// Default constructor
//-----------------------------------
HeapItem::HeapItem()
{
     m_iKey = 0;
     m_dData = 0.0;
}

//-----------------------------------
// Constructor
//-----------------------------------
```

```
HeapItem::HeapItem(int key, double data)
{
     m_iKey = key;
     m_dData = data;
}

//-----------------------------------
// Destructor
//-----------------------------------
HeapItem::~HeapItem()
{
}

//-----------------------------------
// Return item priority
//-----------------------------------
int HeapItem::getKey()
{
     return m_iKey;
}

//-----------------------------------
// Set the priority key value
//-----------------------------------
void HeapItem::setKey(int key)
{
     m_iKey = key;
}

//-----------------------------------
// Return data item
//-----------------------------------
double HeapItem::getData()
{
     return m_dData;
}

//-----------------------------------
// Set the data item value
//-----------------------------------
void HeapItem::setData(double data)
{
     m_dData = data;
}
```

Header file for a class implementing a heap as an array.

```
//----------------------------------------------------------------
// Heap.h
// Demonstration of a heap implemented as an array.  Adapted from
//   sample code in C++ Plus Data Structures, 4th ed. by
//   Nell Dale.
// Author: Dr. Rick Coleman
//----------------------------------------------------------------
#ifndef HEAP_H
#define HEAP_H

#include "HeapItem.h"

class Heap
```

```
{
     private:
          HeapItem     *m_Elements;                 // Pointer to dynamically allocated array
          int          m_iNumElements;              // Number of elements in the heap
          int          m_iHeapLength;               // Size of the array

     public:
          Heap(int size = 10);                     // Parameterized constructor
          ~Heap();                                 // Destructor
          void ReheapDown(int root, int bottom);   // Reheap after removing item
          void ReheapUp(int root, int bottom);     // Reheap after inserting item
          bool Enqueue(HeapItem *item);            // Add an item to the heap
          bool Enqueue(int key, double data);      // Add an item to the heap
          HeapItem *Dequeue();                     // Get item at the root
          int getNumElements();                    // Return number of elements in the heap
          void printAll();                         // Print all the elements in the heap
};

#endif
```

Implementation file for a class implementing a heap as an array.

```
//---------------------------------------------------------------
// Heap.cpp
// Demonstration of a heap implemented as an array.  Adapted from
//   sample code in C++ Plus Data Structures, 4th ed. by
//   Nell Dale.
// Author: Dr. Rick Coleman
//---------------------------------------------------------------
#pragma warning(disable:4996) // Tell Microsoft to not give warnings when
                                    // I use K&R char arrays as strings.  I know
                                    // what I'm doing and don't need MS to protect me.
#include <iostream>
#include "Heap.h"

using namespace std;

//-----------------------------------------
// Parameterized default constructor
//-----------------------------------------
Heap::Heap(int size)
{
     // Create heap of given size
     m_Elements = new HeapItem[size];
     m_iNumElements = 0;
     m_iHeapLength = size;
}

//-----------------------------------------
// Destructor
//-----------------------------------------
Heap::~Heap()
{
     delete[] m_Elements;
}

//-----------------------------------------
// Reheap after removing item
//-----------------------------------------
void Heap::ReheapDown(int root, int bottom)
```

```
{
     int maxChild;
     int rightChild;
     int leftChild;
     HeapItem temp;

     leftChild = root * 2 + 1;          // Get index of root's left child
     rightChild = root * 2 + 2;          // Get index of root's right child

     // Check base case in recursive calls.  If leftChild's index is less
     // than or equal to the bottom index we have not finished recursively
     // reheaping.
     if(leftChild <= bottom)
     {
          if(leftChild == bottom)          // If this root has no right child then
          {
               maxChild = leftChild;     //     leftChild must hold max key
          }
          else
          {     // Get the one lowest in the tree (highest index in the array)
               if(m_Elements[leftChild].getKey() <= m_Elements[rightChild].getKey())
                    maxChild = rightChild;
               else
                    maxChild = leftChild;
          }
          if(m_Elements[root].getKey() < m_Elements[maxChild].getKey())
          {
               // Swap these two elements
               temp = m_Elements[root];
               m_Elements[root] = m_Elements[maxChild];
               m_Elements[maxChild] = temp;
               // Make recursive call till reheaping completed
               ReheapDown(maxChild, bottom);
          }
     }
}

//---------------------------------------
// Reheap after inserting item
//---------------------------------------
void Heap::ReheapUp(int root, int bottom)
{
     int parent;
     HeapItem temp;

     // Check base case in recursive calls.  If bottom's index is greater
     // than the root index we have not finished recursively reheaping.
     if(bottom > root)
     {
          parent = (bottom -1) / 2;
          if(m_Elements[parent].getKey() < m_Elements[bottom].getKey())
          {
               // Swap these two elements
               temp = m_Elements[parent];
               m_Elements[parent] = m_Elements[bottom];
               m_Elements[bottom] = temp;
               // Make recursive call till reheaping completed
               ReheapUp(root, parent);
          }
     }
}
```

```
//---------------------------------------
// Add an item to the heap
//---------------------------------------
bool Heap::Enqueue(HeapItem *item)
{
     if(m_iNumElements < m_iHeapLength)
     {
          m_Elements[m_iNumElements] = *item; // Copy item into array
          ReheapUp(0, m_iNumElements);
          m_iNumElements++;
          return true;
     }
     return false;
}

//---------------------------------------
// Add an item to the heap
//---------------------------------------
bool Heap::Enqueue(int key, double data)
{
     bool retVal;
     HeapItem *temp = new HeapItem(key, data);
     retVal = Enqueue(temp);
     delete temp;  // Delete this dynamically created one
     return retVal;
}

//---------------------------------------
// Get item at the root
//---------------------------------------
HeapItem *Heap::Dequeue()
{
     HeapItem *temp = new HeapItem(m_Elements[0].getKey(), m_Elements[0].getData());
     m_iNumElements--;
     // Copy last item into root
     m_Elements[0] = m_Elements[m_iNumElements];
     // Reheap the tree
     ReheapDown(0, m_iNumElements - 1);
     if(m_iNumElements == 0)
         return NULL;
     else
         return temp;
}

//---------------------------------------
// Return number of elements in the heap
//---------------------------------------
int Heap::getNumElements()
{
     return m_iNumElements;
}

//---------------------------------------
// Print all the elements in the heap
//---------------------------------------
void Heap::printAll()
{
     for(int i=0; i<m_iNumElements; i++)
     {
          cout << "Heap element " << i << ". key=" << m_Elements[i].getKey() << "  data=" <<
```

```
               m_Elements[i].getData() << endl;
     }
}
```

Main file used to test the heap

```
//=================================================================
// Code211_Heap.cpp
// Demonstration of a heap implemented as an array.  Adapted from
//   sample code in C++ Plus Data Structures, 4th ed. by
//   Nell Dale.
// Note: Even though we think of a heap as a tree-like structure
//       it is very difficult to implement a heap as a linked
//       data type.  Since a heap must always be a Complete
//       binary tree it is actually rather easy to implement
//       such a structure in an array.
// Author: Dr. Rick Coleman
//=================================================================
#pragma warning(disable:4996) // Tell Microsoft to not give warnings when
                                     // I use K&R char arrays as strings.  I know
                                     // what I'm doing and don't need MS to protect me.

#include "Heap.h"
#include "HeapItem.h"
#include <iostream>

using namespace std;

void main()
{
     Heap *theHeap = new Heap(10);  // Create a heap of the default size

     cout << "Building the heap and adding items\n\n";

     // Add some items
     theHeap->addItem(123, 1.23);
     theHeap->addItem(345, 3.45);
     theHeap->addItem(234, 2.34);
     theHeap->addItem(678, 6.78);
     theHeap->addItem(456, 4.56);
     theHeap->addItem(567, 5.67);
     theHeap->addItem(789, 7.89);

     // This will build a heap that looks like this
     //                    789
     //                   /   \
     //                456     678
     //                / \     / \
     //              123 345 234 567

     // See what we got
     cout << "Elements in the heap.\n";
     theHeap->printAll();

     cout << "Dequeuing items from the heap.\n\n";

     while((temp = theHeap->Dequeue()) != NULL)
     {
                cout << "Dequeueing " << temp->getKey() << endl;
                delete temp; // delete this one
                // See what we have left
                cout << "Elements in the heap.\n";
                theHeap->printAll();
                cout << endl;
     }
}
```

Results from Testing the Heap

```
Building the heap and adding items

Elements in the heap.
Heap element 0. key=789  data=7.89
Heap element 1. key=456  data=4.56
Heap element 2. key=678  data=6.78
Heap element 3. key=123  data=1.23
Heap element 4. key=345  data=3.45
Heap element 5. key=234  data=2.34
Heap element 6. key=567  data=5.67

Dequeuing items from the heap.

Dequeueing 789
Elements in the heap.
Heap element 0. key=678  data=6.78
Heap element 1. key=456  data=4.56
Heap element 2. key=567  data=5.67
Heap element 3. key=123  data=1.23
Heap element 4. key=345  data=3.45
Heap element 5. key=234  data=2.34

Dequeueing 678
Elements in the heap.
Heap element 0. key=567  data=5.67
Heap element 1. key=456  data=4.56
Heap element 2. key=234  data=2.34
Heap element 3. key=123  data=1.23
Heap element 4. key=345  data=3.45

Dequeueing 567
Elements in the heap.
Heap element 0. key=456  data=4.56
Heap element 1. key=345  data=3.45
Heap element 2. key=234  data=2.34
Heap element 3. key=123  data=1.23

Dequeueing 456
Elements in the heap.
Heap element 0. key=345  data=3.45
Heap element 1. key=123  data=1.23
Heap element 2. key=234  data=2.34

Dequeueing 345
Elements in the heap.
Heap element 0. key=234  data=2.34
Heap element 1. key=123  data=1.23

Dequeueing 234
Elements in the heap.
Heap element 0. key=123  data=1.23
```

```
Dequeueing 123
Elements in the heap.
```