Assignment 09 – Heaps

CS 244

Due: December 09, 2014, 8:00 AM

Late penalties will be as described in the syllabus.

Overview
Create a C++ template class Heap implemented using a std::vector.

General Objectives:

Explore: Priority Queues, Heaps, Arrays, Complete Binary Trees.
Specifically: Discover how to implement a template class Heap using the std::vector

First Step

In Linux in your …/Documents/Programs folder
Create a folder named ***A09***

This is necessary as you will compress the folder and its contents for submission.

Second Step Check D2L

There will be starter code for this assignment. Likely posted near where this document was found. It may make things easier, and some of the files are explicitly required.

The files you will need for this assignment MUST be named:

Heap.h	**HeapTester.cpp**	
makefile	**randomNumber.h**	**randomNumber.cpp**

with the corresponding classes properly defined and declared in each and your main() function located in the tester file. The makefile and randomNumber.h/.cpp will be provided for you and ***should not*** require any changes. But include them in your turn-in submission.

All files (excluding makefile and randomNumber.h/.cpp) should have comments with the file name, your name, the last modified date, and a description of the file contents. Each function should also be appropriately commented in the header and implementation files. The body of each function may need comments for the less than obvious parts of the code (if any).

-=-=-

Assignment 09 – Heaps

CS 244

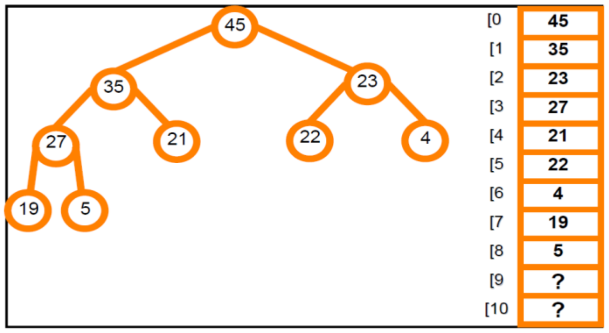
Building new Implementations using Existing Classes

One of the principles of good programming is to *reuse* existing code whenever practical. If you can reuse existing code, you don't need to spend the time to rewrite it. Code used previously has also been debugged, and will likely contain fewer errors. One of the easiest ways to create a container is to leverage an existing data type to build a new abstraction. In this lesson we will illustrate this process by building a **Heap** that will use a **std::vector**. Along with this we will also be examining the idea of priority queues.

Priority Queues, Heaps, Arrays and Complete Binary Trees

A priority queue is a collection of data designed to make it easy to find the element with highest priority. Such a data structure might be used, for example, in a hospital emergency room to schedule an operating table.

A heap is an efficient technique to implement a priority queue. The priority queue data structure is characterized by the header file shown at the bottom of this page. A heap stores values in a complete binary tree. For this assignment, the value of each node in the heap is greater than or equal to each of its child nodes. Notice that a heap is partially ordered, but not completely. In particular, the largest element is always at the root. Although we will continue to think of the heap as a tree, the internal representation will be a vector.



Why use a vector? Using a vector is a good choice because a complete binary tree can be efficiently stored as a vector. The children of the node at index ***i*** are found at positions 2***i***+1 and 2***i***+2 in the vector, the parent is stored at (***i***-1)/2.

```
template <typename T>
class heap
{
public:
   heap(); // Constructor: creates an empty heap.
   // add an item to the heap and re-adjust tree accordingly
   void enqueue(const T& entry);
   // remove an item from the heap and re-adjust tree accordingly
   T dequeue();
   // return the current size of the heap (size of m_vect)
   int size() const{ return m_vect.size(); }
   // create a heap from a specified vector
   void makeHeap(vector<T> source);
   // return a vector of all the elements in the heap from greatest to least
   vector<T> getMaxToMin();

private:
   // restore heap by swapping the element up the heap
   void reheapifyUp(size_t curSpot);
   // restore heap by swapping the element down the heap
   void reheapifyDown(size_t curSpot);
   // elements are stored in a member vector
   vector<T> m_vect;
};
```

Starter code should match this code. It may have additional code or comments

Ask the instructor for clarification if any differences are encountered. Starter code typically is the correct way to go… but ask anyway.

Assignment 09 – Heaps CS 244

The two primary functions of a heap are enqueue and dequeue. They are described as follows:

Insert a new element into a Priority Queue (enqueue)
To insert a new value into a heap the value is pushed onto the back of the vector v. This preserves the complete binary tree property, but not the heap ordering. To fix the ordering, the new value is swapped (reheapify up) into its new position. It is compared to its parent node. If larger, the node and the parent are exchanged. This continues until either the root is reached, or the new value finds its correct position less than its parent. Because this process follows a path up a complete binary tree, it is limited in steps to the height of the complete binary tree.

Remove Highest Priority Element (dequeue)
The largest (highest priority) value is always found at the root. But when this value is removed it leaves a "hole." Filling this hole with the last element in the heap restores the complete binary tree property, but not the heap order property. To restore the heap order the element brought up to replace the root must reheapify down into position by swapping down the tree with its maximum child to maintain the heap property.

What you must do:

Notice you are given starter code for this assignment.
The above header file and some of the corresponding implementation is provided for you.
Review the code provided.
Find and fix any code as indicated by TODO statements (if any).
Update comments to include your name, due date and meaningful descriptions of files and functions.

Implement the enqueue function:
Implement the enqueue function as described above (so max value of heap stays root)

Implement the dequeue function:
Implement the dequeue function as described above (so max value of heap stays root)

Implement the makeHeap function:
Delete any existing heap elements and Enqueue all of the elements of the array sent into your heap.

Implement the getMaxToMin function:
Return a vector v, such that v[0] is the maximum value found in the heap and v[n-1] is the minimum and if printed the elements would be in decreasing (non-increasing) order. The heap may be destroyed in creating the return vector, but should be rebuilt before returning.

Setup your tester file main() function
In your tester file, use the randomNumber class to create a vector of one hundred random integers.
Print the (unsorted) vector.
Create a variable of the class heap<int>
Send your vector of random numbers to the heap to create a heap from them (call makeHeap)
Call the heap function: getMaxToMin
Print the results and verify they are indeed in the correct order.
Perform additional testing as you see appropriate.

-=-=-=

Assignment 09 – Heaps

CS 244

Grading

100 points possible

For the program, at least 70 of the points will be determined by an automatic grading program. The remaining will be determined from code examination and/or further automated grading.

Turn-In Directions

Correctly submitting your work is worth 0 points,
but if not done correctly will likely result in nothing to grade.

Preparation

In Ubuntu Linux browse to your A09 folder
Make sure your source code files are in the folder (and any related worksheets)
Right click on the A09 folder,
Set the file name to be A09_*yourlastname*.tar.gz
where *yourlastname* is your last name
Example: if your last name is Gollygee
then the filename would be A09_Gollygee.tar.gz
Select compress
This should create the file named A09_ *yourlastname*.tar.gz

Submit

Submit the A09_ *yourlastname*.tar.gz file
to the correct course drop box in D2L