Data Structures
Fall 2017, Prof. Bolton
Project

Name: _____
Net ID: _____

1 Heaps

1.1 Summary

You will design and implement a (min) heap data structure which operates as priority queue. The details of the design of the structure is largely at your discretion. Major factors to consider for the design should be theoretical efficiency (both time and space), practical efficiency, and appropriate memory management. Although some tips are provided below (and have been provided in class), the determination of *how* to make the structure efficient is largely your charge.

1.2 Programming Languages

I encourage you to submit your projects using C++. You may choose a different programming language with prior approval from me. However there are caveats as not all programming languages have the same characteristics. Also note, if choosing a language other than C++, you may by chance choose a language with which the TAs are not familiar, thus limiting the amount of assistance they might provide. Note one of the main goals of our class projects is for you to learn how to construct various data structures from the most *elemental programming constructs*. Thus you will not receive credit when using any pre-existing structures from programming libraries or code that has been created or designed by others. For example in C++ you cannot use the pre-existing vectors, stacks, lists, etc.

Note: There are many versions of C++. You must use the version that is currently running on the course server.

Please note that complex data structures (non-elemental constructs) are "built-into" some programming languages. If this is the case, you cannot use the built-in structure. For example, in Python, both list and "array" structures are fairly complex and not elemental programming constructs, e.g., they can change size dynamically. If you are using Python, you will not receive credit when using these structures.

If you have any questions as to what structures are permitted (and which are not permitted), given your language of choice, please ask me.

1.3 Planning and Design

Before implementation, you should plan and design using standard approaches, e.g. UML class diagrams, flow diagrams, etc. If you have questions pertaining to your project, I will first ask to see your designs. **I will not look at your code without first viewing your design documents.**

You will be faced with many design decisions during this project. It is best to spend the requisite time during the design stages to assure an appropriate and efficient implementation is built. Consider your options, perform a theoretical complexity analysis of the different options, and base your decision on the results of your analysis.

1.4 Input Requirements

The program will take no inputs and will be interactive. The user should be repeatedly prompted to input a command. Valid commands are add, remove and quit. The user will type in two tokens: the first is the command, the second is the priority or key. Use the following syntax "a 6" represents command add key 6 to the heap; "r" represents command remove root from the heap; and "q" represents command quit program. After each command, a string representation of the resulting heap should be printed to the screen – see details below.

1.5 Structural and Operational Requirements

1. A simple min-Heap class

    - the fundamental member variable will be a dynamic array – this is the heap. use a reasonable resizing scheme. (Note here the structure is largely defined for you.)
    - the data stored is assumed to be the priority. in practice, each node would also contain a reference to the item stored (and not just the priority of the item).
    - provide methods to add and remove an item. if the item is not in the heap, no changes are made.
    - provide a print method or overload the "<<" operator. print a display that indicates the structure of the heap. I encourage you to print a "bracketed" representation of the heap, which can be rendered using the the tool found at http://mshang.ca/syntree/ . Example min-heap: 1 [5 [9] [7]] [8 [10] [11]]. To construct such a print statement, perform a Depth First Traversal and 1) print an open bracket when traversing down 1 depth to a child node; 2) the print the priority representing that node; and 3) print a close bracket for each depth retreated.

Notes: To earn full marks it is expected that you will implement the structure very efficiently (in space and time). If you are faced with a space / time tradeoff, you will opt to improve time complexity (if the cost of space is relatively minor).

1.6 Output Requirements

The following steps should be executed by the main method:

1. Initialize min-Heap structure.

2. Repeatedly prompt user, add or remove items from the heap, and print string representation for updated heap.

No other outputs should be observed.

1.7 Submission and Compilation Requirements

Rather than fill out a cover letter, *please simply include comments when submitting in Canvas.* Submission Deadline – See Canvas. Budget your time well. Include significant time for design / planning and testing / debugging. Please submit early and often! Your last submission will be graded.

Your code must run on the class server.

To standardize submissions, you will submit a makefile, which will contain the necessary compilation commands for your code. The target executable will be named p3.

Thus, the following steps should run your code on the course server (CHECK IT).

```
make p3
./p3
```

If the program does not compile on the class server, or the program does not run, or the program crashes on an reasonable input the submission will not be accepted.

1.8 Testing and Debugging (Optional)

You may wish to construct an interactive interface to test the functionality of your structure at intermediate stages of development. This would likely be most efficient with an interactive interface that allowed you to interactively test various functionalities of your structure given different inputs. *If you do implement a testing interface, please be sure to comment it (so that it does NOT execute) before submission.*

1.9 Rubric

List of Requirements	Percentage
Heap Structure and Order Maintained	0.40
Dynamic Array and Memory Management	0.40
Add	0.10
Remove	0.10
TOTAL	**1.0**