CS6140: Advanced Programming Lab
Assignment 3

Lead TA: Rahul C. S.

August 14, 2014

Description

The goal of this assignment is to re-engineer your previous assignment code into a well-structured and modularized piece of code. The assignment has two parts. The first part is simply to implement the heap sort (which was already done in assignment 2). But this time the code should follow the given class structure. Additionally you need to use make files in order to specify dependencies while compilation. The first part of the assignment has to be submitted in class. The second part of the assignment is to implement a binary search tree reusing some portions of your heapsort code. The second part can be done later and has to be submitted by 17th August midnight. In each of the parts to be submitted, make a tar.gz file named as *roll-number*.tar.gz and upload it to moodle.

Part 1

Implement heap sort using pointer based binary heaps. The input (set of jobs), the input file format and the ordering rules are exactly the same as in assignment 2. Your program should implement the following two classes: 1) class Node, and 2) class Heap. The public functions of both the classes are specified below. Make sure that the data for each of these classes are private. You are free to implement any other private member functions for the respective classes.

```
class Node {
public:
        int getJobId();
        int getPriority();
        int getDuration();
        void setJobId(int);
        void setPriority(int);
        void setDuration(int);
        Node* getLeftPtr();
        Node* getRightPtr();
        Node* getParentPtr();
        void setLeftPtr(Node*);
        void setRightPtr(Node*);
        void setParentPtr(Node*);
};
```

```
class Heap {
public:
        Heap();
        ~Heap();
        void insert(Node*);
        Node* extractMin();
        void printHeap();
};
```

The Node class should be present in *Node.h* file and the public interfaces have to be implemented in *Node.cpp* file. Similarly, the Heap class definition should be present in *Heap.h* file and the functions have to be implemented in *Heap.cpp* file. You will need to define the private data members and private functions for

both these classes. You are also allowed to define additional public functions if absolutely required. Finally, the main function should be in a file *main.cpp* which implements heapsort. Thus your whole code of heapsort is split into 5 files – Node.h, Node.cpp, Heap.h, Heap.cpp, main.cpp.

Write a make file with correct dependencies to support the following:

1. make objects // creates object files Node.o, Heap.o, main.o
2. make heapsort // creates an executable named 'heapsort' from the *.o files
3. make target // creates a *roll-number*.tar.gz file containing the *.h *.cpp and makefile. Make sure that your tar.gz file does not contain the *.o files and the executable.

Part 2:

Implement a pointer based binary search tree using the Node class defined earlier. The key for every data item is the *jobId* and we will assume that jobIds are unique. The binary search tree should support insertion, deletion, and search operations. In addition, there should be a print function that prints the preorder traversal of the tree. The public interface for the class is as given below:

```
class BST{
public:
    BST();
    ~BST();
    void insert(Node*);
    void remove(int);
    Node* search(int);
    printTree();
};
```

Split the class definition and the functions into BST.h and BST.cpp files respectively. Finally, have a main function (implemented in main.cpp) which accepts 2 files – (1) input file (containing a list of jobs as in the case of heap sort), (2) a test cases file which contains the following types of lines.

- I jobId duration priority
  Insert the above job in the BST. In case of inserting a job with duplicate jobId, flag an error message: "Duplicate jobId. Insert unsuccessful."
- S jobId
  Search the job with id as jobId and print the details (duration and priority).
- R jobId
  Remove the job with id as jobId from the BST. In case of removing a jobId which is not present, flag an error message:
  "JobId not found. Remove unsuccessful."
- P
  Print the current BST.

Write a make file with correct dependencies to support the following:

1. make objects // creates object files Node.o, BST.o, main.o
2. make BST // creates an executable named 'binTree' from *.o files.
3. make target // creates a *roll-number*.tar.gz file containing the *.h *.cpp and makefile.