CSE 310 Assignment #5
(Max. Points: 30)

Due on: Wednesday, Nov. 8th, 2017, 11:59pm Arizona time

General Instructions:

- This is an individual assignment, please do not collaborate. *Make sure that you write every line of your own code. Using code written by someone else will be considered a violation of the academic integrity and will result in a report sent to the Dean's office.*
- It must be submitted online through Blackboard submission link! We don't accept submissions through emails.
- Submission link will be closed automatically once the due date/time is past and **no late assignment will be accepted**.
- You will be allowed 3 times to submit the assignment, but we will only grade your last submission.

Objectives

- Practice on some operations on Red-Black tree.

Assignment Description

This is a programming assignment. You are required to develop a red black tree with some of its operations. Follow below instructions closely.

1. Similar as what you did in Assignment #4, you need to implement a data structure to represent a tree node. Each node should have the following data fields:

- *discipline*
- *gender*
- *team_or_ind*
- *event*
- *venue*
- *medal*
- *athlete*
- *country*

The unique **key** to identify each node will be a combination of *discipline, gender, event,* and *athlete*.

Each node should also have its parent node called *parent*, its left child node called *left*, its right child node called *right*, and *color* which should be either **RED** or **BLACK**.

2. Using the nodes above, you need to build a red black tree by using their keys.

3. When you compare two nodes, compare their *discipline* first in alphabetical order; if *discipline* is the same, then compare their *event*; if *event* are the same, compare their *gender*; if *gender* are the same, then compare *athlete*. Note: for all these fields, they will be compared in alphabetical order according to each character's ASCII number.

4. Your red black tree needs to have operations called *preorderTreeWalk, inorderTreeWalk* and *postorderTreeWalk* which will print all tree node keys in a pre-, in- and post-order traversal respectively.

5. Your red black tree also needs to have operations called *treeSearch, treeMinimum, treeMaximum, treeSuccessor* and *treePredecessor*.

6. Your red black tree needs to have *treeInsert* operation and to implement this, first you need to define Binary Search Tree insertion, right rotate, and left rotate functions. They will be called in your red black tree insertion function.

7. Your red black tree class should have a constructor to initialize its root to be NULL, and a destructor that will delete all nodes in the tree and perform garbage collections. The destructor should also print **`The number of nodes deleted: X`**, where **X** is the number of nodes deleted and it should be called when a user chooses to exit the program.

8. You should implement a driver's program that takes the commands: *tree_inorder, tree_preorder, tree_postorder, tree_maximum, tree_minimum, tree_successor, tree_predecessor, tree_insert*, and *quit*. The program should print out each entered command before processing it. Below please find each command's detailed description:

tree_inorder

It prints all keys in the tree using the *in-order* traversal, using the following format:

```
discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            3. BRONZE
athlete:          Alexander Kristoff
country:          Norway
color:            BLACK

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            2. SILVER
athlete:          Rigoberto Uran
```

```
country:          Colombia
color:            RED
```

It should print `tree is empty` if the tree is empty.

tree_preorder

It prints all keys in the tree using the *pre-order* traversal. The format it prints should be similar to above **tree_inorder**. For example:

```
discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            3. BRONZE
athlete:          Alexander Kristoff
country:          Norway
color:            BLACK

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            2. SILVER
athlete:          Rigoberto Uran
country:          Colombia
color:            RED
```

It should print `tree is empty` if the tree is empty.

tree_postorder

It prints all keys in the tree using the *post-order* traversal. The format it prints should be similar to above **tree_inorder**. For example:

```
discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED
```

```
discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            2. SILVER
athlete:          Rigoberto Uran
country:          Colombia
color:            RED

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            3. BRONZE
athlete:          Alexander Kristoff
country:          Norway
color:            BLACK
```

It should print `tree is empty` if the tree is empty.

tree_minimum

It should call *treeMinimum* and print the **first** node in *in-order* traversal (*i.e.* the smallest node inside the tree) using the following format:

```
The first athlete is:
discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED
```

It should print `tree is empty` if the tree is empty.

tree_maximum

It should call *treeMaximum* and print the **last** node in *in-order* traversal *(i.e*. the largest node inside the tree) using the following format:

```
The last athlete is:

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
```

```
medal:            2. SILVER
athlete:          Rigoberto Uran
country:          Colombia
color:            RED
```

It should print `tree is empty` if the tree is empty.

tree_predecessor,discipline,gender,event,athlete

It should call *treePredecessor* and search for the predecessor of the given athlete. An example of this command is:

```
tree_predecessor,Cycling,M,Road,Alexander Kristoff
```

If it finds its predecessor, then it should print the result as:

```
The predecessor of the medal recipient Alexander Kristoff for Cycling with
event Road is:
discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED
```

If it cannot find its predecessor, then it should print:

```
The medal recipient Aleksander Winokurow for Cycling with event ROAD does not
have any predecessor
```

It should print `tree is empty` if the tree is empty.

tree_successor,discipline,gender,event,athlete

It should call *treeSuccessor* and search for the successor of the given athlete. An example of this command is:

```
tree_successor,Cycling,M,Road,Aleksander Winokurow
```

If it can find its successor, then it should print the result as:

```
The successor of the medal recipient Alexander Kristoff for Cycling with
event Road is:

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            3. BRONZE
athlete:          Alexander Kristoff
country:          Norway
color:            BLACK
```

If it cannot find its successor, then it should print:

```
The medal recipient Alexander Kristoff for Cycling with event ROAD does not
have any predecessor
```

It should print `tree is empty` if the tree is empty.

tree_search,discipline,gender,event,athlete

It should call *treeSearch* and search the given athlete with the given information. An example of this command is:

```
tree_search,Cycling,M,Road,Alexander Kristoff
```

If it can find the athlete specified, then it should print the result by showing its medal as:

```
The medal recipient Alexander Kristoff has the medal of 3. BRONZE
```

If it cannot find the athlete, then it should print:

```
Alexander Kristoff for Cycling with event Road not found
```

It should print `tree is empty` if the tree is empty.

tree_insert,discipline,gender,ind_team,event,venue,medal,athlete,country

It should call *treeInsert* and attempt to insert the given athlete. If there exists another athlete with the same discipline, gender, event, and athlete, then it should not be able to insert it. An example of this command is:

```
tree_insert,Cycling,M,IND,Road,Regent's Park,1. GOLD,Aleksander
Winokurow,Kazakhstan
```

If it can insert, then it should print:

```
The medal recipient Aleksander Winokurow for Cycling with event Road inserted
```

If it cannot insert, then it should print:

```
The medal recipient Aleksander Winokurow for Cycling with event Road NOT
inserted
```

It should print `tree is empty` if the tree is empty.

quit

The command quits the program and exit. The red black tree object should be deleted, and it should print how many nodes are deleted using the following format:

```
The number nodes deleted: 3
```

9. Sample Run

Below please find one sample run of the program (user input is in **bold**)

```
tree_inorder
next command: tree_inorder
```

```
tree is empty
```

tree_preorder
```
next command: tree_preorder

tree is empty
```

tree_postorder
```
next command: tree_postorder

tree is empty
```

tree_insert,Cycling,M,IND,Road,Regent's Park,1. GOLD,Aleksander Winokurow,Kazakhstan
```
next command: tree_insert,Cycling,M,IND,Road,Regent's Park,1. GOLD,Aleksander Winokurow,Kazakhstan

The medal recipient Aleksander Winokurow for Cycling with event Road inserted
```

tree_insert,Cycling,M,IND,Road,Regent's Park,2. SILVER,Rigoberto Uran,Colombia
```
next command: tree_insert,Cycling,M,IND,Road,Regent's Park,2.SILVER,Rigoberto Uran,Colombia

The medal recipient Rigoberto Uran for Cycling with event Road inserted
```

tree_insert,Cycling,M,IND,Road,Regent's Park,3. BRONZE,Alexander Kristoff,Norway
```
next command: tree_insert,Cycling,M,IND,Road,Regent's Park,3.BRONZE,Alexander Kristoff,Norway

The medal recipient Alexander Kristoff for Cycling with event Road inserted
```

tree_inorder
```
next command: tree_inorder


discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            3. BRONZE
athlete:          Alexander Kristoff
country:          Norway
color:            BLACK

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
```

```
medal:            2. SILVER
athlete:          Rigoberto Uran
country:          Colombia
color:            RED
```

tree_preorder

```
next command: tree_preorder

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            3. BRONZE
athlete:          Alexander Kristoff
country:          Norway
color:            BLACK


discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            2. SILVER
athlete:          Rigoberto Uran
country:          Colombia
color:            RED
```

tree_postorder

```
next command: tree_postorder


discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
```

```
venue:            Regent's Park
medal:            2. SILVER
athlete:          Rigoberto Uran
country:          Colombia
color:            RED

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            3. BRONZE
athlete:          Alexander Kristoff
country:          Norway
color:            BLACK
```

tree_maximum

```
next command: tree_maximum

The last athlete is:

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            2. SILVER
athlete:          Rigoberto Uran
country:          Colombia
color:            RED
```

tree_minimum

```
next command: tree_minimum

The first athlete is:

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED
```

tree_search,Cycling,M,Road,Alexander Kristoff

```
next command: tree_search,Cycling,M,Road,Alexander Kristoff

The medal recipient Alexander Kristoff has the medal of 3. BRONZE
```

tree_search,Archery,M,Archery,team USA

```
next command: tree_search,Archery,M,Archery,team USA

team USA for Archery with event Archery not found
```

```
tree_successor,Fencing,F,Foil,Arianna Errigo
next command: tree_successor,Fencing,F,Foil,Arianna Errigo

The medal recipient Arianna Errigo for Fencing with event Foil does not have
any successor

tree_predecessor,Cycling,M,Road,Alexander Kristoff
next command: tree_predecessor,Cycling,M,Road,Alexander Kristoff

The predecessor of the medal recipient Alexander Kristoff for Cycling with
event Road is:

discipline:       Cycling
gender:           M
team_or_ind:      IND
event:            Road
venue:            Regent's Park
medal:            1. GOLD
athlete:          Aleksander Winokurow
country:          Kazakhstan
color:            RED

quit
next command: quit

The number nodes deleted: 3
```

10. Implementation/Documentation Requirements

- You need implement this program using C++ and it has to read from the standard input (from a keyboard).
- Your program needs to compile and execute in *general.asu.edu*.
- You need to define *treeInorder, treePreorder, treePostorder, treeMinimum, treeMaximum, treePredecessor, treeSuccessor, treeInsert* for your red black tree, also the *constructor*, *destructor*, *leftRotate*, *rightRotate*, and binary search, insertion functions.
- Your code should be well documented and commented, and every file should have your name at its top.
- Also you are not allowed to use any predefined data structures (such as ones in the library in C++, etc.) except arrays and strings, you need to build your own data structures and operations associated with them (such as insert or search).

Copy any codes from other people's programs is considered to be cheating and will lead to a report to the Dean and you will be penalized. Also check the rules stated in the syllabus of the course regarding the academic integrity.

- At the top of each source code file, add the following information:

```
// Name of Author(s):
// ASU ID:
// Homework Number:
// Description:
```

What to turn in

Under the submission link on Blackboard, submit a zip file named ***youFirstNameyourLastName.zip*** containing the following:

1. One or more files of your well documented C++ source code (with a file extension *.cpp* or *.h*) implementing the commands required.

2. A Makefile that compiles your program to an executable named "***A5p1***" on the Linux machine *general.asu.edu*. Our TA will write a script to compile and run all student submissions on *general.asu.edu*; therefore executing the make command must produce the executable file "***A5p1***" in the same folder as your source code files. After such ***A5p1*** file is produced, then we should be able to execute your program using the command:

```
./A5p1 < sampleinput.txt
```

where "*sampleinput.txt*" is a file name, but we should be able to use a file with a different name with a content in the format specified in the assignment's statement.

Your program must read from a keyboard, not a file.

Error Handling

Your program will be tested with other input besides the sample input given, thus is expected to be robust.

Grading Rubric

- 2 pts – Documentation (correct and complete code description and comments, header for each function/method, each file should have your name at its top) and good indentation and spacing (easy to read)
- 1 pt – makefile is submitted and it works correctly to generate A5p1 executable file
- 13 pts – Each of the following functions are defined correctly: *inorder, preorder, postorder, treeMinimum, treeMaximum, treePredecessor, treeSucessor, treeSearch, constructor, destructor, treeInsert (for red black tree), rightRorate, and leftRotate*.
- 14 pts – Correct output for the test case(s)

Total points: 30