CSCI 2720
Assignment 4 – Binary Search Trees
Due: November 5, Friday 11:59 PM

In this assignment, you will create a Binary Search Tree to store and retrieve objects. Unlike with Assignment 2, you will not need ItemType for this assignment. Instead, you will use C++ templates to make your program support three different data types (int, float, and std::string). The data type will be specified by the user before any operations are run, as described later in the document. The purpose of this assignment is for you to become familiar with basic tree operations, and understand the efficiency of trees compared to previously studied data structures. Binary Tree nodes have only two children, left and right. Nodes are compared based on their Key instance variable. All elements in the left subtree of a given node have key values less than the given node and all elements in the right subtree have key values greater than the given node. A Binary tree must maintain this sorted property at all times. In this assignment, the binary tree should not accept duplicate values.

You may choose to implement the functions in the Binary Tree class iteratively or recursively. As with the previous assignments, you may create additional functions to implement the features asked in the document. Once all functions have been implemented for the Binary Tree class, you will create a Main application that initializes a tree based on file input and allows the user to interactively modify the tree based on the given commands. Finally, **make sure to properly document all the code with comments**, **and make sure that your output exactly matches the example output**.

Important Points

1. In this assignment, place your NodeType definition in BinaryTree.h.

2. You will not need ItemType, and will instead make your program “generic” using templates.

3. You will use 3 input files: eg: int-input1.txt, float-input1.txt, and string-input1.txt.

4. Be sure to properly document your code with comments, and add your name above functions that you implement if you are in a group.

5. You must follow the exact submission instructions given at the end of the document.
6. In the sample outputs we have covered all the important cases to test your implementation so as long as your code is able to generate outputs same as the sample outputs in the document then you don’t need to worry about testing any other cases.

Templates

You should make your BST class templated, as well as the NodeType struct, by adding `template<class T>` before the class/struct definition. For example, to make NodeType and BST templated, use the following code snippet in BinaryTree.h:

```
template<class T>
struct NodeType {
      // NodeType members
};

template<class T>
class BinaryTree {
      // BinaryTree members
};
```

Note that the "T" works just as the parameter for generics in Java, and may be replaced by something else, such as "U". However, unlike with Java, you must specify what data types you are planning to support, which can be done in your .cpp files. For example, to do this for BinaryTree, add the following lines of code at the bottom of BinaryTree.cpp, below all your method implementations:

```
template class BinaryTree<int>;
template class BinaryTree<float>;
template class BinaryTree<std::string>;
```

Your program should support storing data of type **`int`**, **`float`**, and **`std::string`** depending on input taken from the user at the very beginning of the program. The user should be able to enter "i" for **`int`**, "f" for **`float`** or "s" for **`std::string`**. Please see the following sample output:

```
      ./main input.txt
      Enter tree type (i - int, f - float, s - std:string): s
```

In the above example, the user has provided "s" as the input. In this case, you should initialize your generic tree to store **`std::string`** values. This means you should also read the values in the given text file as **`std::string`** values and enter them to the tree as **`std::string`** objects. The insert, delete and print commands should likewise support **`std::string`** inputs. If the user provides "i" or "f", your program should be able to work with ints or floats respectively.

The user is responsible for selecting an appropriate type for a given program run depending on the provided input file and the values that are expected to be stored in the tree. You will be provided sample text files for each appropriate data type: one for **`int`**, one for **`float`**, and one

for `std::string`.

Project Files:

- **`BinaryTree.h`:**
    - Structures
        - Node structure that has the following members
            - `T key`
            - `Node<T> *left`
            - `Node<T> *right`
    - Instance Variables:
        - `Node *root;`
    - Public member functions:
        - `BinaryTree();`
            - Pre-Condition: None.
            - Post-Condition: Tree is initialized.
        - `~BinaryTree();`
            - Pre-Condition: Tree has been initialized.
            - Post-Condition: All node pointers freed and root points to null
        - `void insert(T &key);`
            - Pre-Condition: Tree and parameter key initialized.
            - Post-Condition: Insert a node with the value of key into the tree. **No duplicates are allowed**.
        - `void deleteItem(T &key);`
            - Pre-Condition: Tree and parameter key initialized.
            - Post-Condition: Remove a node with a key value equal to the parameter key's value otherwise leave the tree unchanged (if the key is not present). In situations, where the node to be deleted has two children, replace the deleted node with its immediate predecessor or successor.
        - `void retrieve(T &item, bool &found) const;`
            - Pre-Condition: Tree, item, and found are all initialized.
            - Post-Condition: item should refer to a key of a Node *n* in the tree where the value of *n.key* is equal to the value of *item* and *found* should be equal to true if *n* exists. Otherwise *found* should be equal to false.
        - `void preOrder() const;`

- Pre-Condition: The tree has been initialized.
- Post-Condition: Print out the tree in pre-order.

- `void inOrder() const;`
  - Pre-Condition: The tree has been initialized.
  - Post-Condition: Print out the tree in in-order.
- `void postOrder() const;`
  - Pre-Condition: The tree has been initialized.
  - Post-Condition: Print out the tree in post-order.
- `int getLength() const;`
  - Pre-Condition: The tree has been initialized.
  - Post-Condition: Return the value of length of the BST (same as the method discussed in the class).

The following function can be implemented however you like, just make sure their input and output formatting matches the sample output below. Implement these functions as a class function using prototypes of your choice. For above functions like preorder the function prototype does not include a parameter so you can implement this by using as auxiliary function (as discussed in lecture) or using a getRoot function etc.

`getNumSingleParent` function - This function should return the number of nodes that have one child

- Example:
  - If the function getNumSingleParent is called on the BST below in the example output, the function should return 1.

`getNumLeafNodes` function - This function should count the number of leaf nodes in the BST (Nodes with no children) and then output the count

- Example:
  - If the function getNumLeafNodes is called on the BST shown below in the sample output, the function should return 5.

`getSumOfSubtrees` function - This function should take in a node as input, search for that node and then return the sum of both subtrees of the given node.

- Note: Do not include the node that you call this function on in the sum.
- Example:
  - If the function getSumOfSubtrees(10) is called on the BST shown below in the sample output, the function should return 16.
  - If the function getSumOfSubtrees(50) is called on the BST shown below in the sample output, the function should return 35.
  - If the function getSumOfSubtrees(4) is called on the BST shown below in the sample output, the function should return 0.
  - If the function getSumOfSubtrees(15) is called on the BST shown below in the sample output, the function should return 90.
  - If the function getSumOfSubtrees(60) is called on the BST shown below in the sample output, the program should print out "Item not in tree".

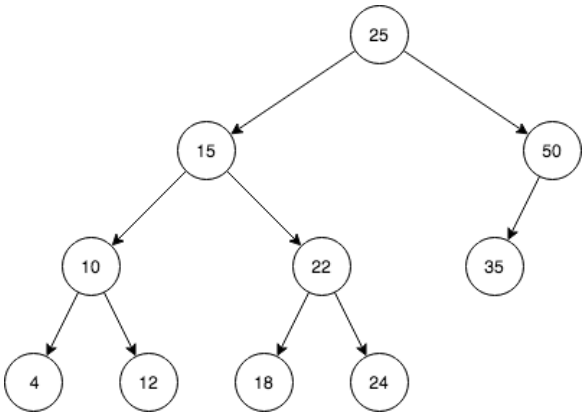
In the readme file give the pseudo code (steps) for the aboves 3 operations. Using this pseudo-code to explain the complexity (big O) of your 3 operations. To compute the complexity of above functions, write a recurrence relation and then solve that recurrence relation using the Master method (again as discussed in the lecture slides).

- **`BinaryTree.cpp`:**
    - Implement all the functions defined in the header file.
- **`Main.cpp`:**
    - Create a main application that matches example output exactly.

You can introduce other functions that are not specifically mentioned in the guidelines when necessary.

Example Output:

Outputs are given for the starting binary tree shown in the diagram.



Standard Output I

```
./main int-input1.txt
Enter tree type (i - int, f - float, s - std:string): i
Commands:
insert (i), delete (d), retrieve (r), length (l), in-order (n),
pre-order (p), post-order (o), getNumSingleParent (s),
getNumLeafNodes
(f), getSumOfSubtrees (t),  quit (q)
```

//1. pre-order(p)

```
Enter a command: p
Pre-Order: 25 15 10 4 12 22 18 24 50 35
```

//2. in-order(n)

```
Enter a command: n
In-Order: 4 10 12 15 18 22 24 25 35 50
```

//3. post-order(o)

```
Enter a command: o
Post-Order: 4 12 10 18 24 22 15 35 50 25
```

//4. length(l)

```
Enter a command: l
Tree Length: 10

Enter a command: q
Quitting program…
```

Standard Output II

```
./main int-input1.txt
Enter tree type (i - int, f - float, s - std:string): i
Commands:
insert (i), delete (d), retrieve (r), length (l), in-order (n),
pre-order (p), post-order (o), getNumSingleParent (s),
getNumLeafNodes
(f), getSumOfSubtrees (t),  quit (q)
```

//5a. insert(i) - insert an item

```
Enter a command: i
Item to insert: 13
In-Order: 4 10 12 13 15 18 22 24 25 35 50

Enter a command: p
Pre-Order: 25 15 10 4 12 13 22 18 24 50 35
```

//5b. insert(i) - insert an item that exists in tree

```
Enter a command: i
Item to insert: 25
Item already in tree.
In-Order: 4 10 12 13 15 18 22 24 25 35 50

Enter a command: q
Quitting program…
```

Standard Output III

```
./main int-input1.txt
Enter tree type (i - int, f - float, s - std:string): i
Commands:
insert (i), delete (d), retrieve (r), length (l), in-order (n),
pre-order (p), post-order (o), getNumSingleParent (s),
getNumLeafNodes
(f), getSumOfSubtrees (t),  quit (q)
```

//6a. delete(d)

```
Enter a command: d
Item to delete: 50
In-Order: 4 10 12 15 18 22 24 25 35

Enter a command: o
Post-Order: 4 12 10 18 24 22 15 35 25

Enter a command: p
Pre-Order: 25 15 10 4 12 22 18 24 35
```

//6b. delete(d)

```
Enter a command: d
Item to delete: 10
In-Order: 4 12 15 18 22 24 25 35

Enter a command: p
Pre-Order: 25 15 4 12 22 18 24 35
```

//6c. delete(d) - delete a non-existent item

```
Enter a command: d
Item to delete: 100
Item not in tree.
In-Order: 4 12 15 18 22 24 25 35

Enter a command: q
Quitting program…
```

Standard Output IV

```
./main int-input1.txt
Enter tree type (i - int, f - float, s - std:string): i
Commands:
insert (i), delete (d), retrieve (r), length (l), in-order (n),
pre-order (p), post-order (o), getNumSingleParent (s),
getNumLeafNodes \
(f), getSumOfSubtrees (t),  quit (q)
```

//7a. retrieve(r) - retrieve an existent item

```
Enter a command: r
Item to be retrieved: 25
Item found in tree.
```

//7b. retrieve(r) - retrieve an non-existent item

```
Enter a command: r
Item to be retrieved: 101
Item not in tree.
```

```
Enter a command: q
Quitting program…
```

Standard Output V

```
./main int-input1.txt
Enter tree type (i - int, f - float, s - std:string): i
Commands:
insert (i), delete (d), retrieve (r), length (l), in-order (n),
pre-order (p), post-order (o), getNumSingleParent (s),
getNumLeafNodes
(f), getSumOfSubtrees (t),  quit (q)
```

//8.invalid command

```
Enter a command: k
Command not recognized. Try again

Enter a command: b
Command not recognized. Try again

Enter a command: q
Quitting program…
```

Standard Output VI

```
./main int-input1.txt
Enter tree type (i - int, f - float, s - std:string): i
Commands:
insert (i), delete (d), retrieve (r), length (l), in-order (n),
pre-order (p), post-order (o), getNumSingleParent (s),
getNumLeafNodes
(f), getSumOfSubtrees (t),  quit (q)
```

//9. getNumSingleParent (s)

```
Enter a command: s
Number of Single Parents: 1
```

//10. getNumLeafNodes (f)

```
Enter a command: f
Number of leaf nodes: 5
```

//11. getSumOfSubtrees (t)

```
Enter a command: t
Item to get sum of subtrees: 10
Sum of Subtrees: 16

Enter a command: t
```

```
  Item to get sum of subtrees: 99
  Item not in tree.

  Enter a command: q
  Quitting program…
```

Standard Output VII - float-input1

```
  ./main float-input1.txt
  Enter tree type (i - int, f - float, s - std:string): f
  Commands:
  insert (i), delete (d), retrieve (r), length (l), in-order (n),
  pre-order (p), post-order (o), getNumSingleParent (s),
getNumLeafNodes
  (f), getSumOfSubtrees (t),  quit (q)
```

```
//1. pre-order(p)
  Enter a command: p
  Pre-Order: 20.4 10.9 3.2 15.1 60.3 89.0
```

```
//2. insert(i) - insert an item
  Enter a command: i
  Item to insert: 40.4
  In-Order: 3.2 10.9 15.1 20.4 40.4 60.3 89.0

  Enter a command: p
  Pre-Order: 20.4 10.9 3.2 15.1 60.3 40.4 89.0
```

```
//3. delete(d)
  Enter a command: d
  Item to delete: 60.3
  In-Order: 3.2 10.9 15.1 20.4 40.4 89.0

  Enter a command: p
  Pre-Order: 20.4 10.9 3.2 15.1 40.4 89.0

  Enter a command: q
  Quitting program…
```

Standard Output VIII - string-input1

```
  ./main string-input1.txt
  Enter tree type (i - int, f - float, s - std:string): f
  Commands:
  insert (i), delete (d), retrieve (r), length (l), in-order (n),
  pre-order (p), post-order (o), getNumSingleParent (s),
getNumLeafNodes
  (f), getSumOfSubtrees (t),  quit (q)
```

//1. pre-order(p)

```
Enter a command: p
Pre-Order: Movie Igloo Apple Jam Party Zoo
```

//2. insert(i) - insert an item

```
Enter a command: i
Item to insert: Neck
In-Order: Apple Igloo Jam Movie Neck Party Zoo

Enter a command: p
Pre-Order: Movie Igloo Apple Jam Party Neck Zoo
```

//3. delete(d)

```
Enter a command: d
Item to delete: Igloo
In-Order: Apple Jam Movie Neck Party Zoo

Enter a command: p
Pre-Order: Movie Jam Apple Party Neck Zoo

Enter a command: q
Quitting program…
```

Grading Rubric:

Implementation	**Grade**
Binary Tree	**85%**
Insert	10%
Delete	10%
Pre-Order	5%
In-Order	5%
Post-Order	5%
Retrieve	5%
Get Length	5%
Get Num Single Parents	10%
Get Num Leaf Nodes	10%
Get Sum of Subtrees	10%
Templates	10%
Main Application	**5%**
Readme, Comments, makefile & Spec conformity	**10%**
TOTAL	**100%**

Compiling the Program (Same as assignment - 1):

A Makefile should compile your code into a program called "main". Your program should run with the following command syntax:

```
./main <input file name>
```

Commands to run and compile the code should be documented in the Readme File clearly.

Code that fails to compile or the code that compiles but fails to run will receive a grade of zero.

Late Submission Policy:

Except in the cases of serious illness or emergencies, projects must be submitted before the specified deadline in order to receive full credit. Projects submitted late will be subject to the following penalties:

- If submitted 0–24 hours after the deadline 20% will be deducted from the project score
- If submitted 24–48 hours after the deadline 40% points will be deducted from the project score.
- If submitted more than 48 hours after the deadline a score of 0 will be given for the project. Students unable to submit a project due to a serious illness or other emergencies should contact the instructor as soon as possible before a project's deadline. Based on the circumstances, the instructor will decide on an appropriate course of action.

Submission Notes:

You must **include your full name** and **university email address** in your **Readme.txt** file. If you are doing the project in a **group of two**, list the **full names and the email addresses of all two group members**. If you are in a group you must also **describe the contributions of each group member** in the assignment.

Contribution is expected to be 50% + 50%.

Submit the following files on Odin:

A folder named BinarySearchTree containing

- **BinaryTree.h**
- **BinaryTree.cpp**
- **Main.cpp**
- **Makefile**
- **int-input1.txt**
- **float-input1.txt**
- **string-input1.txt**
- **Readme.txt (**One Readme file with all necessary descriptions).

Odin Login Instructions

This semester we are switching from Nike to a new server called Odin. Unlike with Nike, to use Odin, you must use the UGA VPN first. Please visit remote.uga.edu if you haven't already and install the VPN client for your OS. Be sure to have an SSH client like PuTTY or MobaXterm and use that to connect. The server's name is odin.cs.uga.edu and you will log in with your MyID credentials. For those that have used Nike before, the login process is almost the same, except that nike.cs.uga.edu is replaced by odin.cs.uga.edu, and, as stated before, you'll log in with your MyID credentials instead of your regular Nike credentials.

Group Submissions:

****Note: Assignment can be done individually or in a group of two students. The members of the group should be from the same section. Submit a readme file with your names, UGA emails and any other information that you want to give the TA to run your code. If you are doing it in a group of two make sure to include the contribution of each student in the readme file.**

If you are working in a group of two, your submission folder must be named as below:

```
LastName1_LastName2_assignment4
```

When you are submitting the folder you should use the submit command as below:

```
$ submit LastName1_LastName2_assignment4 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

If you are working in a group, you must **submit the assignment from the Odin account of only one of the members** in the group.

Individual Submissions:

If you are working individually, your submission folder must be named as below:

```
LastName_assignment4
```

Submission command should be used as below:

```
$ submit LastName_assignment4 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

****For all the submissions, the folder naming and the readme is very important in order to be able to locate your assignment and grade it.**

****Note: Make sure you submit your assignment to the right folder to avoid grading delays.**

When using the submit command, first set your current directory of the terminal to the immediate parent of the assignment folder.

You can check your current working directory by typing:

```
$ pwd
```

For example if your assignment is in the following directory,

```
/documents/csci2720/assignments/LastName_assignment4
```

Before executing the submit command, change to the assignment's immediate parent directory:

```
$ cd /documents/csci2720/assignments/
```

Then execute the submit command:

```
$ submit LastName_assignment4 <odin-submission-destination>
```

Make sure to replace <odin-submission-destination> with the correct destination from the table on the bottom of this document.

To check if your submission command was valid, there should be a receipt file.

```
$ ls LastName_assignment4
// should return rec##### like 'rec54321'
```

Selecting the Odin Submission Destination

You must select your Odin submission destination using the following mapping based on the section of the course you are registered to.

Section	Class Time	Odin Submission Destination
CRN 17504	M 10:20 AM	csci-2720a
CRN 42634	M 04:10 PM	csci-2720c