**CSCI 2720**

**Project 3 – Binary Search Trees**

**This assignment is the property of CSCI 2720 course at University of Georgia. The assignment should not be copied, duplicated or distributed.**

In this assignment, you will create a Binary Search Tree (BST) to store and retrieve objects. You should use generics to make your programs support three different data types (int, double, and 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. BST 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 BST must maintain this property at all times. In this assignment, the BST should not accept duplicate values.

You may choose to implement the functions in the Binary Search 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 Search Tree class, you will create a Main application (driver file) that initializes a tree based on the file input and allows the user to interactively modify the tree based on the given commands. Finally, make sure that your output exactly matches the example output.

**Important Points**

1. You will use 3 input files: int-input.txt, double-input.txt, and string-input.txt to create three different types of BSTs.
2. Be sure to properly document your code with comments and add your name above functions that you implement if you are doing the assignment in a group.
3. You must follow the exact submission instructions given at the end of the document.
4. In the sample outputs we have covered all the important cases to test your implementation, so if 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.

**JAVA Generics**

You should make your BST and NodeType classes generic. For example, to make NodeType and BinarySearchTree class generic, use the following code snippet:

**public class** NodeType<T **extends** Comparable<T>> {

**public** T info;

**public** NodeType<T> left;

**public** NodeType<T> right;

} ;

**public class** BinarySearchTree<T **extends** Comparable<T>> {

**public void** deleteNode(T item)

{

……………………………

}

//other methods will go here

**private** NodeType<T> root;

};

Note that the “T” works just as the parameter for generics. During the runtime this “T” will get replaced by the datatype selected by the user for the BST. You need the Comparable class to compare generic objects.

Your program should support storing data of type **int**, **double**, and **string** depending on the input taken from the user at the very beginning of the program. The user should be able to enter “i” for **int**, “d” for **double** or “s” for **string**. Please see the following sample:

|  |
| --- |
| $ java BSTDriver string-input.txt |
| Enter list type (i - int, d - double, s - std:string): s |

In the above example, the user has provided “s” as the input. In this case, you are going to create a BST that is going to store string items. You will open the input file name provided from the command prompt to read string data and store each string input in the BST. The insert, delete, print and other commands should likewise support **string** inputs. If the user provides “i” or “d”, your program should be able to work with int or double 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 BST. You will be provided sample text files for each appropriate data type: one for **int**, one for **double**, and one for **string**.

Make sure your code does error check for file I/O operation, i.e., it should throw an error message if file is not present or unable to open the file correctly. You should not hardcode the input file names in the main.

**Project Files:**

* **NodeType.java:** A node will have left and right pointers for two children.
* **BinarySearchTree.java:**

○ Instance Variables:

**■ NodeType<T> root;**

○ Public member functions:

**■ BinarySearchTree();**

* + - * + Pre-Condition: None.
        + Post-Condition: Tree is initialized.

**■ 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 delete(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.

**■ boolean search(T item);**

* + - * + 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 should return true if *n* exists otherwise returns false.

**■ void inOrder();**

* + - * + Pre-Condition: The tree has been initialized.
        + Post-Condition: Print out the tree in in-order.
        + The function prototype does not include a parameter, so you can implement this by using as auxiliary function or using a getRoot function etc.

The following functions should be implemented. You can implement them however you like. Implement these functions as class functions using prototypes of your choice.

**getSingleParent** function: This function should print the nodes that have one child.

**getNumLeafNodes** function: This function should count the number of leaf nodes in the BST (nodes with no children) and then output the count.

**getCousins** function: This function should take in a node as input and prints the cousins of the given node. A cousin is a relative that is the child of a parent's sibling (source: Wikipedia).

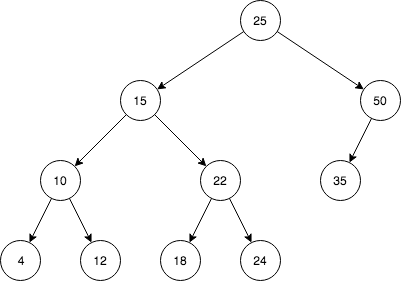
* Example:

○ In the figure below 12 has two cousins 18 and 24. Similarly 35 has two cousins 10 and 22 while 15 and 50 do not have any cousins.

**BinarySearchTreeDriver.java:** Create a main application that matches example output exactly.

**Example:**

Outputs are given for the starting BST shown in the diagram below:



The output of your program should match the sample output below. You don’t need to match the exact whitespaces, newline etc., but your output should look like the sample output below. You just need to test your code for the cases given below. If your code works correctly for the cases below than you don’t need to worry about testing anything else. Implement a basic error checking (giving a wrong choice for command, checking if filename provided, unable to open the text file).

## Standard Output I (int-input.txt)

Enter list type (i - int, d - double, s - string): i Commands:

(i) - Insert Item

(d) - Delete Item

(p) - Print Tree

(s) - Search Item

(l) - Count Leaf Nodes

(sp)- Find Single Parents

(c) - Find Cousins

(q) - Quit program

**//1. Print Tree(p) will do the in-order traversal and print the nodes**

# Enter a command: p

# In-order: 4 10 12 15 18 22 24 25 35 50

### //2. Insert(i)

//2a. Insert an item that is not present in the tree

# Enter a command: i

In-order: 4 10 12 15 18 22 24 25 35 50 Enter a number to insert: 20

In-order: 4 10 12 15 18 20 22 24 25 35 50

//2b. Insert an item that is already present in the tree

# Enter a command: i

In-order: 4 10 12 15 18 20 22 24 25 35 50 Enter a number to insert: 20

The item already exists in the tree.

In-order: 4 10 12 15 18 20 22 24 25 35 50

### //3. Search (s)

//3a. Search an item that is already present in the tree

# Enter a command: s

In-order: 4 10 12 15 18 20 22 24 25 35 50

Enter a number to search: 20

Item is present in the tree

//3b. Search an item that is not present in the tree

# Enter a command: s

In-order: 4 10 12 15 18 20 22 24 25 35 50

Enter a number to search: 67

Item is not present in the tree

### //4. Delete

//4a. Delete a leaf node

# Enter a command: d

# In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number to delete: 4

In-order: 10 12 15 18 22 24 25 35 50

Enter a command: d

In-order: 10 12 15 18 22 24 25 35 50

Enter a number to delete: 24

In-order: 10 12 15 18 22 25 35 50

//4b. Delete a node with one child

Enter a command: d

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number to delete: 50

In-order: 4 10 12 15 18 22 24 25 35

Enter a command: d

In-order: 4 10 12 15 18 22 24 25 35

Enter a number to delete: 18

In-order: 4 10 12 15 22 24 25 35

Enter a command: d

In-order: 4 10 12 15 22 24 25 35

Enter a number to delete: 22

In-order: 4 10 12 15 24 25 35

//4c. Delete a node with two children

Enter a command: d

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number to delete: 22

In-order: 4 10 12 15 18 24 25 35 50

Enter a command: d

In-order: 4 10 12 15 18 24 25 35 50

Enter a number to delete: 15

In-order: 4 10 12 18 24 25 35 50

Enter a command: d

In-order: 4 10 12 18 24 25 35 50

Enter a number to delete: 25

In-order: 4 10 12 18 24 35 50

//4d. Delete a node that is not present in the tree

Enter a command: d

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number to delete: 33

The number is not present in the tree

### //5. Count leaf nodes (it is tested by deleting some of the nodes in between)

### Enter a command: p //using given int-input file

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a command: l

The number of leaf nodes are 5

# Enter a command: d

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number to delete: 35

In-order: 4 10 12 15 18 22 24 25 50

Enter a command: l

# The number of leaf nodes are 5

Enter a command: d

In-order: 4 10 12 15 18 22 24 25 50

Enter a number to delete: 50

In-order: 4 10 12 15 18 22 24 25

Enter a command: l

The number of leaf nodes are 4

Enter a command: d

In-order: 4 10 12 15 18 22 24 25

Enter a number to delete: 18

In-order: 4 10 12 15 22 24 25

Enter a command: l

The number of leaf nodes are 3

### //6. Find single parents (it is tested by deleting some of the nodes in between)

Enter a command: p

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a command: sp

Single Parents: 50

Enter a command: d

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number to delete: 18

In-order: 4 10 12 15 22 24 25 35 50

Enter a command: sp

Single Parents: 22 50

Enter a command: d

In-order: 4 10 12 15 22 24 25 35 50

Enter a number to delete: 4

In-order: 10 12 15 22 24 25 35 50

Enter a command: sp

Single Parents: 10 22 50

Enter a command: d

In-order: 10 12 15 22 24 25 35 50

Enter a number to delete: 12

In-order: 10 15 22 24 25 35 50

Enter a command: sp

Single Parents: 22 50

Enter a command: d

In-order: 10 15 22 24 25 35 50

Enter a number to delete: 24

In-order: 10 15 22 25 35 50

Enter a command: sp

Single Parents: 50

Enter a command: d

In-order: 10 15 22 25 35 50

Enter a number to delete: 35

In-order: 10 15 22 25 50 Enter a command: sp

Single Parents: None

### //7. Find cousins

Enter a command: p

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a command: c

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number: 4

4 cousins: 18 24

Enter a command: c

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number: 10

10 cousins: 35

Enter a command: c

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number: 35

35 cousins: 10 22

Enter a command: c

In-order: 4 10 12 15 18 22 24 25 35 50 Enter a number: 15 15 cousins:

Enter a command: c

In-order: 4 10 12 15 18 22 24 25 35 50 Enter a number: 25

25 cousins:

Enter a command: c

In-order: 4 10 12 15 18 22 24 25 35 50

Enter a number: 22

22 cousins: 35

## Standard Output II (double-input.txt)

Enter list type (i - int, d - double, s - string): d

Commands:

(i) - Insert Item

(d) - Delete Item

(p) - Print Tree

(s) - Search Item

(l) - Count Leaf Nodes

(sp)- Find Single Parents

(c) - Find Cousins

(q) - Quit program

**//1. Print Tree(p) will do the in-order traversal and print the nodes**

Enter a command: p

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

### //2. Insert(i)

//2a. Insert an item that is not present in the tree

Enter a command: i

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number to insert: 12.9

In-order: 1.3 3.2 10.9 12.9 13.8 15.1 20.4 60.3 89.0

//2b. Insert an item that is already present in the tree

Enter a command: i

In-order: 1.3 3.2 10.9 12.9 13.8 15.1 20.4 60.3 89.0 Enter a number to insert: 3.2

The item already exists in the tree.

In-order: 1.3 3.2 10.9 12.9 13.8 15.1 20.4 60.3 89.0

### //3. Search (s)

//3a. Search an item that is already present in the tree

Enter a command: s

1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number to search: 13.8

Item is present in the tree

//3b. Search an item that is not present in the tree

Enter a command: s

1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number to search: 76.5

Item is not present in the tree

### //4. delete

//4a. Delete a leaf node

Enter a command: d

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0 Enter a number to delete: 1.3

In-order: 3.2 10.9 13.8 15.1 20.4 60.3 89.0

//4b. Delete a node with one child

Enter a command: d

In-order: 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number to delete: 15.1

In-order: 3.2 10.9 13.8 20.4 60.3 89.0

//4c. Delete a node with two children

Enter a command: d

In-order: 3.2 10.9 13.8 20.4 60.3 89.0

Enter a number to delete: 20.4

In-order: 3.2 10.9 13.8 60.3 89.0

//4d. Delete a node that is not present in the tree

Enter a command: d

In-order: 3.2 10.9 13.8 60.3 89.0

Enter a number to delete: 17

The number is not present in the tree

### //5. Count leaf nodes (it is tested by deleting some of the nodes in between)

### Enter a command: p //using given double-input file

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a command: l

The number of leaf nodes are 3

Enter a command: d

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number to delete: 89.0

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3

Enter a command: l

The number of leaf nodes are 3

Enter a command: d

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3

Enter a number to delete: 60.3

In-order: 1.3 3.2 10.9 13.8 15.1 20.4

Enter a command: l

The number of leaf nodes are 2

### //6. Find single parents (it is tested by deleting some of the nodes in between)

Enter a command: p

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a command: sp

Single Parents: 3.2 15.1 60.3

Enter a command: d

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number to delete: 1.3

In-order: 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a command: sp

Single Parents: 15.1 60.3

Enter a command: d

In-order: 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number to delete: 13.8

In-order: 3.2 10.9 15.1 20.4 60.3 89.0

Enter a command: sp

Single Parents: 60.3

Enter a command: d

In-order: 3.2 10.9 15.1 20.4 60.3 89.0

Enter a number to delete: 89.0

In-order: 3.2 10.9 15.1 20.4 60.3

Enter a command: sp Single Parents:

### //7. Find cousins

Enter a command: c

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number: 1.3

1.3 cousins: 13.8

Enter a command: c

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number: 15.1

15.1 cousins: 89.0

Enter a command: c

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number: 89.0

89.0 cousins: 3.2 15.1

Enter a command: c

In-order: 1.3 3.2 10.9 13.8 15.1 20.4 60.3 89.0

Enter a number: 10.9 10.9 cousins: None

## Standard Output III (string-input.txt)

Enter list type (i - int, d - double, s - string): s Commands:

(i) - Insert Item

(d) - Delete Item

(p) - Print Tree

(s) - Search Item

(l) - Count Leaf Nodes

(sp)- Find Single Parents

(c) - Find Cousins

(q) - Quit program

**//1. Print Tree(p) will do the in-order traversal and print the nodes**

Enter a command: p

In-order: Apple Igloo Jam Movie Party Zoo

### //2. Insert(i)

//2a. Insert an item that is not present in the tree

Enter a command: i

In-order: Apple Igloo Jam Movie Party Zoo

Enter a string to insert: Joey

In-order: Apple Igloo Jam Joey Movie Party Zoo

//2b. Insert an item that is already present in the tree

Enter a command: i

In-order: Apple Igloo Jam Joey Movie Party Zoo Enter a string to insert: Joey The item already exists in the tree.

In-order: Apple Igloo Jam Joey Movie Party Zoo

### //3. Search (s)

//3a. Search an item that is already present in the tree

Enter a command: s

In-order: Apple Igloo Jam Movie Party Zoo

Enter a string to search: Joey

Item is not present in the tree

//3b. Search an item that is not present in the tree

Enter a command: s

In-order: Apple Igloo Jam Movie Party Zoo

Enter a string to search: Jam

Item is present in the tree

### //4. Delete

//4a. Delete a leaf node

Enter a command: d

In-order: Apple Igloo Jam Movie Party Zoo Enter a string to delete: Apple

In-order: Igloo Jam Movie Party Zoo

//4b. Delete a node with one child

Enter a command: d

In-order: Igloo Jam Movie Party Zoo

Enter a string to delete: Igloo

In-order: Jam Movie Party Zoo

//4c. Delete a node with two children

Enter a command: d

In-order: Jam Movie Party Zoo

Enter a string to delete: Movie

In-order: Jam Party Zoo

//4d. Delete a node that is not present in the tree

Enter a command: d

In-order: Jam Party Zoo

Enter a string to delete: Joey

Item is not present in the tree

### //5. Count leaf nodes (it is tested by deleting some of the nodes in between)

Enter a command: p

In-order: Apple Igloo Jam Movie Party Zoo

Enter a command: l

The number of leaf nodes are 3

Enter a command: d

In-order: Apple Igloo Jam Movie Party Zoo Enter a string to delete: Zoo

In-order: Apple Igloo Jam Movie Party

Enter a command: l

The number of leaf nodes are 3

Enter a command: d

In-order: Apple Igloo Jam Movie Party

Enter a string to delete: Party

In-order: Apple Igloo Jam Movie

Enter a command: l

The number of leaf nodes are 2

### //6. Find single parents (it is tested by deleting some of the nodes in between)

Enter a command: p

In-order: Apple Igloo Jam Movie Party Zoo

Enter a command: sp

Single Parents: Party

Enter a command: d

In-order: Apple Igloo Jam Movie Party Zoo

Enter a string to delete: Zoo

In-order: Apple Igloo Jam Movie Party

Enter a command: sp Single Parents:

### //7. Find cousins

Enter a command: p

In-order: Apple Igloo Jam Movie Party Zoo

Enter a command: c

In-order: Apple Igloo Jam Movie Party Zoo

Enter a string: Apple

Apple cousins: Zoo

Enter a command: c

In-order: Apple Igloo Jam Movie Party Zoo

Enter a string: Jam

Jam cousins: Zoo

Enter a command: c

In-order: Apple Igloo Jam Movie Party Zoo

Enter a string: Zoo

Zoo cousins: Apple Jam

Enter a command: c

In-order: Apple Igloo Jam Movie Party Zoo

Enter a string: Party Party cousins: None

**Grading Rubric:**

|  |  |
| --- | --- |
| **Implementation** | **Grade** |
| **Binary Search Tree** |  |
| Insert | 10% |
| Delete | 15% |
| In-Order | 10% |
| Search | 10% |
| Get Single Parents | 10% |
| Get Num Leaf Nodes | 10% |
| Get Cousins | 20% |
| Using Generics | 10% |
| **Driver file** | 5% |
| **Readme, Comments, Specification Conformity** | 10% |
| **TOTAL (10 bonus points)** | **110%** |

Your program should run with the following command syntax:

$ java <driver file> <input file name>

**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 at ELC:

* NodeType.java
* BinaryTree.java
* BinaryTreeDriver.java
* int-input.txt
* double-input.txt
* string-input.txt
* Readme.txt (one Readme file with all necessary descriptions).

Please follow submission instructions for Odin (posted at ELC).