Project 1

Binary Search Tree Implementation

CS 241 Winter 2017, Dr. Hao Ji

Taylor Thurlow

tjthurlow@cpp.edu

30 January 2017

## Section 1: Project description

The program is an implementation of the Binary Search Tree. Using general types, it supports any comparable data type. The program will provide a simple command-line based interface for constructing and manipulating a binary search tree.

## Section 2: Project specification

Your program should read from the standard input a sequence of integer values, with each value separated by a space. Your task is to:

i. Build a binary search tree using these values in the order they are entered.
ii. Print 3 traversals: pre-, in-, and post-order.
iii. Allow the user to insert/delete a value. Once a new tree is generated, print it in-order.
iv. Find predecessor of a given value. The predecessor is the node that appears right before the given value in an in-order traversal.
v. Find successor of a given value. The successor is the node that appears right after the given value in an in-order traversal.

In your BST implementation, the add and delete methods must be implemented using recursion. Note that no duplicates are allowed in this BST.

## Section 3: Testing methodology

The most important, and clearly the most complicated part of testing an implementation of a Binary Search Tree is how many different edge cases there are. Handling single-node trees and two-node trees combines a lot of these cases. For ease of printing and accessing adjacent values, I stored the traversals in ArrayLists, which made it necessary to do bounds checking when looking for predecessors and successors. It's also important that misuse of the command prompt is handled. Checking for number of arguments has been implemented, and if the code is unsure of any input, the user will be redirected to the help text. When possible, the prompt will tell you what you did wrong.

## Section 4: Lessons learned

This project was my first real use of general class (templating) in Java. All of my previous experience had been in C++. It does function quite similarly, but it took me a while to remember an issue I had a long time ago before I really understood templating - you can't use primitive types. Additionally, while I did understand how recursion worked, I had never used it as extensively as I did in this project, and I think it definitely was a great exercise in that regard, particularly with the code for removing a node. A pen and paper can be a huge help when mentally working through edge cases and different scenarios. This particular project also reminded me that often times things appear to be working properly when in reality there is an edge case that has not been considered. By testing varied data sets, I was able to recognize a few cases that I had not considered when writing the code for the first time. I think it would be interesting to work out an automated, timed method for manipulating a tree, to find out how efficient (or inefficient) my implementation is. If I have some spare time I may look into a way to do that - though I may have to use something other than integers.

```
Please enter the initial sequence of values:
51 29 68 90 36 40 22 59 44 99 77 60 27 83 15 75 3

Pre-order: 51 29 22 15 3 27 36 40 44 68 59 60 90 77 75 83 99
In-order: 3 15 22 27 29 36 40 44 51 59 60 68 75 77 83 90 99
Post-order: 3 15 27 22 44 40 36 29 60 59 75 83 77 99 90 68 51

Command: h
   i <val> - insert a value
   d <val> - delete a value
   p <val> - find predecessor
   s <val> - find successor
         e - exit the program
    h or ? - display this message
Command: i 88
   Result: 3 15 22 27 29 36 40 44 51 59 60 68 75 77 83 88 90 99
Command: i 42
   Result: 3 15 22 27 29 36 40 42 44 51 59 60 68 75 77 83 88 90 99
Command: i 22
   22 already exists, ignoring.
Command: d 44
   Result: 3 15 22 27 29 36 40 42 51 59 60 68 75 77 83 88 90 99
Command: d 90
   Result: 3 15 22 27 29 36 40 42 51 59 60 68 75 77 83 88 99
Command: d 70
   70 not found in BST, ignoring.
Command: d 68
   Result: 3 15 22 27 29 36 40 42 51 59 60 75 77 83 88 99
Command: s 75
   Successor: 77
Command: p 99
   Predecessor: 88
Command: e

Process finished with exit code 0
```