

**Department of Electrical and Computer Engineering**  
**ECSE 202 – Introduction to Software Development**  
**Assignment 2**  
**Stacks, B-Trees, Sorting Data**

Due February 17, 2020 at 5:00 pm

**Problem Description**

In this assignment we will explore some of the data structures discussed in class, namely the **Stack and B-Tree classes**. You will use these in conjunction with a file reading program to sort a list of names in ascending and descending order by creating a B-Tree representation of the file and the performing an inorder traversal to sort the data. Finally, a **stack will be used to reverse the order** of the list for final display.

Here is an example run of the program you are to write, given a short file of names:

```
Assignment 2 - File Sorting Program
Enter name of file to sort: Names-short.txt
```

File in sort order:

```
Benes Dorethea
Britto Teisha
Freeze Clarisa
Galentine Dante
Woolery Ciera
```


File in reverse sort order:

```
Woolery Ciera
Galentine Dante
Freeze Clarisa
Britto Teisha
Benes Dorethea
```

Program terminated

To make this assignment tractable, you are provided with a listFile class that opens a text file for reading and displays it on the output device, line by line. The constructs used in this class are beyond where we are currently in the course, but you should have a sufficient knowledge of Java to figure out how to modify the code for Assignment 2.

The file containing the code, listFile.java, is available on myCourses and reproduced below:

```
package listFile; 
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;

/**
 * This program reads a specified text file and echos it to standard
 * display. You can use this as the basis of Assignment 2, adding the
 * necessary components to implement the program specifications.
 * @author ferrie
 *
 */

public class listFile {
    @SuppressWarnings("resource")
    public static void main(String args[]) {

        // Prompt user for a file name. If no name is entered, terminate
        // the program, otherwise attempt to open the file. If file open
        // is not successful, prompt again for a new name. Keep doing this
        // until successful open, or a blank line is entered.

        System.out.println("Simple File Listing Program");
        Scanner sc = new Scanner(System.in);
        BufferedReader rd = null;

        while(rd == null) {
            System.out.print("Enter name of file to list: ");
            String filename = sc.nextLine();
            if (filename.equals("")) {
                System.out.println("Program terminated");
                System.exit(0); // Exit
            }
        }
        // Try to open the specified file
        try {
            rd = new BufferedReader(new FileReader(filename));
        }
        catch (IOException ex) {
            System.out.println("Unable to open file, try again.");
        }
    }
}
```

```

// Read the file a line at a time into a string.  Print as read
// to the output display.  Modify the code below as necessary.

    System.out.println("");
    try {
        while (true) {
            String line = rd.readLine(); // Read a line of text
            if (line == null) break;      // Exit if at end of file
            System.out.println(line);     // Print (or do whatever)
                                         // to current line
        }
        catch (IOException ex) {
            System.out.println("I/O Error - program terminated");
            System.exit(-1);
        }

        System.out.println("\n\nProgram terminated");
    }
}

```

Pay particular attention to the last while loop in the program. Here you have access to each line of text in a file as a string. You should be able how to save each string in a B-Tree for further processing.

## Strategy

You have two essential problems to solve: i) **sort the file** and ii) **print the file sorted in both ascending and descending order**. As we have seen in class, sorting can be done easily by creating a B-Tree and then performing an inorder traversal to visit each node in sort order. This means that you need to **create a class, bTree, with addNode and traversal methods** (following the example in the lecture slides). The **bNode class**, which **holds the data**, must now **hold a String instead of an int for the payload**. This complicates the addNode method somewhat, as you now need a method to compare two strings (hint: [google java string compare method](#)). It is also a good idea to use the version of this method that is *case insensitive*, i.e., where smith john is equal to SMITH JOHN or Smith John. From here it should not be too difficult to figure out how write an inorder traversal method which prints out the nodes (lines of text) in ascending order.

The second component, is to **display the file again, but in reverse sort order**. You are not allowed to use any of the Java array types to do this. Instead, create a Stack class where the listNode class is modified to hold a string instead of an integer. Since a stack is accessed in last-in, first-out order, you can obtain the desired effect by creating a version of your inorder traversal that substitutes a PUSH for printing the value at the current node. Later, once you have output the list in ascending order, you can perform another while loop that pops the stack, displaying each node in descending order.

## Structure

Your program should contain the following **5 classes**:

1. **A2** The class corresponding to the main program derived from listFile.java.
2. **bTree** The B-Tree class containing the addNode() and in-order() methods.
3. **bNode** The B-Tree node class. This is defined as a helper class within the bTree.java file.
4. **Stack** The stack class containing push() and pop() methods.
5. **listNode** The listNode class. This is defined as a helper class within the Stack.java file.

## Approach

The A2 class provides a simple **user interface** which prompts for the name of a file containing a list of names to be sorted. It perhaps easiest to start with the listFile program and modify it as A2.java. The program begins by creating an **instance of a Stack** and an **instance of a B-Tree** – call these myStack and myTree for short. In the while loop, instead of printing the string immediately, call **myTree.addNode()** to save a reference to the current string to the B-Tree.

At this point all of the data read in is indexed in the B-Tree. Next step is to print the names in sorted order by performing an **inorder traversal**. The inorder method listed in the notes can be easily modified for this purpose, but it is also important to remember that we have to print this list in **reverse sort order afterwards**. So instead of simply printing the current node, we can also use **myStack.push()** in preparation for the reverse order list. Note: you will have to modify the inorder method to include a reference to myStack.

To list the file names in reverse sort order, all that is required is to pop the stack (while loop) until empty, printing each string on a separate line. It is strongly recommended that you develop and test each class independently before putting the program together. The bTree class can be derived from the example in the notes – *you must implement the non-recursive form of addNode() for this assignment*. Similarly, the Stack class requires only minor modifications to work for this assignment.

Just to underscore the point – the listNode and bNode helper classes do not store string data, only references to String objects elsewhere on the Heap. Memory for string objects is allocated in the second while loop of the main class, i.e., `String line = rd.readLine();`

## Instructions

Test your program using the Names.txt file. When inputting the file name, you must specify the complete name including path, e.g., `\Users\ferrie\assignments\A2\Names.txt`, unless the file resides in the *default directory*. If your project lives at `\Users\ferrie\workspace\A2`, then if you copy the Names.txt file to that location, then it suffices to type *Names.txt* when prompted for the file name.

If your program works correctly, it should produce a display similar to that shown on Page 1, with the exception of printing all the entries in Names.txt. Save this output by copying and pasting into a text file, e.g. into Notepad if you are using Windows, for example. Save this file as A2.txt. *n.b. your program is expected to replicate the example, exactly, except for the length of the list.*

Upload A2.txt along with all of your source (.java) files to myCourses as indicated.

## About Coding Assignments

We encourage students to work together and exchange ideas. However, when it comes to finally sitting down to write your code, this must be done *independently*. Detecting software plagiarism is pretty much automated these days with systems such as MOSS.

<https://www.quora.com/How-does-MOSS-Measure-Of-Software-Similarity-Stanford-detect-plagiarism>

Please make sure your work is your own. If you are having trouble, the Faculty provides a free tutoring service to help you along. You can also contact the course instructor or the tutor during office hours. There are also numerous online resources – Google is your friend. The point isn't simply to get the assignment out of the way, but to actually learn something in doing.

fpf/January 31, 2020