**Programming Project Report**

Name: Riley Nadwodny

Date: 4/30/2020

**Academic Integrity Statement:** I pledge that I have neither given nor received unauthorized help on this programming assignment.

**Problem Statement:**

The goals of this programming assignment were to create a binary search tree (BST) class containing the information about the 1000 most popular names in the us by male, female, or last name. The BST uses the name string for the search key. Each node in the tree contains the following information as required by the Project 6 guide from John Gauch:

* Name – first name or last name read from input file
* Count – number of times the name is used in the US
* Percentage – percentage of people in the US with that name
* Rank – the rank of the name in the US top 1000 names
* Left – pointer to left subtree
* Right – pointer to right subtree

Another goal was to create a main program that reads in an ASCII input file to initialize the BST using any of the three provided ASCII files: last.txt, girls.txt, or boys.txt. For this project, it was not required to prompt the user for which input file to use. Because of this, “boys.txt” was used but with the name “Male.txt” instead. This file contains the most 1000 popular boys names in the US, with 1000 lines that contain the name, the percentage, the count, and the rank all separated by spaces so the information is easy to read in. The program prompts the user for a name to search, and informs them to type “QUIT” whenever they wish to exit the program. The program outputs the name if found including the name, count, percentage, and rank information, and outputs an error if the name is not found. There was no error handling necessary for this program.

**Design:**

For this project, two classes are necessary. The first class is the Node class which contains the individual nodes that hold the information within the greater BST. Each node contains a string, two ints, a float, and node pointers to the left and right.. The string contains the name read in from the ASCII file, one int contains the count while the other contains the rank, and the float contains the percentage. Left and Right are both set to NULL.

The tree class contains a constructor and destructor, along with both public and private functions. The private functions are helper functions to the public functions, and the public functions merely call the private functions to do the heavy work. There is also a ReadData function that reads in the data from the male.txt input file to initialize the BST. If the file fails to open, an error message is output and the program ends. Otherwise, the BST is created and filled in with the information from the input file.

This setup works well because the rather than calling the gross private functions in the main program, the public functions can be called instead which then call the private functions to do the work. The only public function that does not call a private function is the ReadData function.

**Implementation:**

To start, a large chunk of sample code was provided by John Gauch. This sample code was adapted as necessary to implement this program. The sample code worked with data type int. To make this BST work with strings and contain the info necessary, all the ‘int Value’s had to be changed to ‘string Name’. Next, the Node class needed to contain a few extra data types to include the necessary information read in from the ASCII file. These are declared as ‘int Count’, ‘float Percentage’, and ‘int Rank’. A new function was created for reading in the ASCII file. While ‘!din.eof()’, the program reads in the name, count, percentage, and rank, and then inserts the information into the BST.

In the main program, a BST is declared and ‘string name’ in order to store the user’s name to search. The ReadData function is called in order to read in the data. A while loop starts that prompts the user for a name to search for. If the name is found, the name and it’s node information is printed out to the user, and then they are prompted for a new name. Whenever they desire to quit the program, they can type “QUIT” and the program will terminate. Overall, this project took about four hours to design and implement, given the large amount of code already provided to build the program off of.

**Testing:**

This program was relatively easy to test. Because the program reads in a string, any inputs with numbers or special characters and symbols is searched for just like a name, but is never found. The program works as expected. If the name is found, the node is printed out to the user and if the node is not found, the error message is printed out as necessary. The program was tested on the UARK CSCE turing server and the typescript file is included in the zip folder.

**Conclusions:**

Overall, this project was a success, and a relatively easy one at that. Modifying the provided sample code was not too difficult, and the functions work as intended. The program as a whole performs without a hitch and functions as intended. I do not believe that I would do anything differently next time, but rather I would do everything the same again. This project took roughly six hours total to complete, as I found some last-minute changes such as finding an unused variable and other tiny things to adjust within the program.