Program 2 – SMU Phone Book

Due Monday, February 25 at 11:59pm

Background

According to the YellowPagesGoesGreen.org, over 500 million phone directories are printed and distributed in the United States every year. The paper used to create each of these books make up about 19 million trees. In order to reduce the amount of felled trees, SMU Lyle is taking an initiative to move all of their contacts online to an SMU Phone Book. You have been hired to create this online phone book so that users can quickly look up contacts by name or by phone number.

The Task

Reading in the Phone Book

Your task is to create a program that reads in an input file *phoneBook.txt* that contains a comprehensive list of contacts. The contacts names and phone numbers should be read into 2 vectors:

```
vector<string> names;
vector<string> phoneNumbers;
```

An example of the input file is provided below:

phoneBook.txt

```
Laura Bush,2142842934
Kathy Bates,2143954218
Lauren Graham,7130385534
Kourtney Kardasian,2149492789
Doak Walker,5127834500
```

Each line of this file contains the following information separated by a comma:

1. Full Name (String)
2. Phone Number (String)

Searching the Phone Book

Once the phone book file has been read, the user should be allowed to begin searching the phone book directory for specific contacts by name or by phone number. Your interface should begin by prompting the user with a list of options that they may select:

1. **Find By Name (Recursive) –** this will use a recursive function to find a contact by name
2. **Find By Name (Iterative) –** this will use an iterative function to find a contact by name
3. **Find By Phone Number (Recursive) –** this will use a recursive function to find a contact by phone number
4. **Find By Phone Number (Iterative) –** this will use an iterative function to find a contact by phone number
5. **Quit –** will terminate the program

When the user selects an option, they will be asked to enter a search query that will be used to search through the list of contacts for a matching name or phone number. If a contact that matches that query is found, the program will return that result to the user providing the name, phone number and the entry that the contact appeared in. An example result will look like the following:

2. Kathy Bates - 2143954218

If no contact matches that query your program should say "No Results Found". After the results have been returned the user may make another selection from the options listed above. They may search again or they may select (5) which will quit and terminate the program.

Implementation Details

The core of your program will be the 2 search functions that are used to find results. One will be a recursive search (using recursion) and the other will be an iterative search (using a loop). You are to implement these functions in a separate file from your main.cpp. You will have one header file (.h) with the function definitions and an implementation file (.cpp). You **MUST** use the following function prototypes to complete this program. You may call one or more functions from these methods but these must be the ones your search engine calls after the user enters a query.

The function definitions are listed below:

```
int searchRecursive(std::vector<std::string> &data, std::string query);

int searchIterative(std::vector<std::string> &data, std::string query);
```

The first function *searchRecursive* will recursively search through a vector to find the index of the contact that matches the string *query.* The index value returned by the function will be used

to find the name and phone number of that contact. If a contact is not found the function will return **-1**.

The second function *searchIterative* will iteratively search through the vector to find the index that matches the string *query.* Similarly to the other function, the index value returned by the function will be used to find the name and phone number of that contact. If a contact is not found the function will return **-1**.

NOTE: A query must be an EXACT match:

query == “Lauren Graham”
or
query == “7130385534”

Test Program

SMU would like to be sure that the software solution you have provided will get results as expected. Therefore, in addition to your main **program2.cpp** file, you will also have a **program2test.cpp** file that will use the *cassert* methods to test the functionality of your 2 search functions. You must have at least 5 asserts or test cases per function, so 10 in total.

If your test.cpp program is successful. It should print “Success” at the end of your test.

Program Architecture

The architecture for your program should look like the following:

program2.cpp

```
#include “search_functions.h”

int main() {
     // Read in File

     do {
          // prompt user for options

          // prompt user for query

          // search

          // display results
     } while (option != 5)
}
```

search_functions.h

```
#ifndef SEARCH_FUNCTIONS
#define SEARCH_FUNCTIONS

int searchRecursive(std::vector<std::string> &data, std::string query);

int searchIterative(std::vector<std::string> &data, std::string query);


#endif
```

search_functions.cpp

```
#include <vector>
#include <string>

int searchRecursive(std::vector<std::string> &data, std::string query)
{
     // implementation
}

int searchIterative(std::vector<std::string> &data, std::string query)
{
     // implementation
}
```

program2test.cpp

```
#include "search_functions.h"

int main() {
      vector<string> test;

      // define test cases here
}
```

Example Output

The following example assumes the input file matches the one provided in this assignment.

```
┌─[erikgabrielsen][Eriks-MBP-2][~/Desktop/CSE1342_Spring2019/programming_assignments/Program2]
└─▪ ./a.out
Welcome to SMU's Phone Book

-------------------------
(1) - Find By Name (recursive)
(2) - Find By Name (iterative)
(3) - Find By Phone (recursion)
(4) - Find By Phone (iterative)
(5) - Quit

Please Select an Option:
```

Next a user enters **1** to search the phone book recursively.

```
┌─[erikgabrielsen][Eriks-MBP-2][~/Desktop/CSE1342_Spring2019/programming_assignments/Program2]
└─▪ ./a.out
Welcome to SMU's Phone Book

-------------------------
(1) - Find By Name (recursive)
(2) - Find By Name (iterative)
(3) - Find By Phone (recursion)
(4) - Find By Phone (iterative)
(5) - Quit

Please Select an Option: 1
Enter a name:
```

User Enters in the query “Kathy Bates”

```
Welcome to SMU's Phone Book

-------------------------
(1) - Find By Name (recursive)
(2) - Find By Name (iterative)
(3) - Find By Phone (recursion)
(4) - Find By Phone (iterative)
(5) - Quit

Please Select an Option: 1
Enter a name: Kathy Bates

2. Kathy Bates - 2143954218

Please Select an Option:
```

Full Example Output

```
Welcome to SMU's Phone Book

--------------------------
(1) - Find By Name (recursive)
(2) - Find By Name (iterative)
(3) - Find By Phone (recursion)
(4) - Find By Phone (iterative)
(5) - Quit

Please Select an Option: 1
Enter a name: Kathy Bates

2. Kathy Bates - 2143954218

Please Select an Option: 3
Enter a phone number: 2143954218

2. Kathy Bates - 2143954218

Please Select an Option: 2
Enter a name: Erik Gabrielsen

No Results Found

Please Select an Option: 5
┌─[erikgabrielsen][Eriks-MBP-2][~/Desktop/CSE1342_Spring2019/programming_assignments/Program2]
└─▪
```

What to Turn In

You will upload a zip file called <LastName_FirstInitial>Program2.zip containing your **program2.cpp** file, **search_functions.h** file, **search_functions.cpp** file, **test.cpp** file, **phoneBook.txt** file, your program executable **program2.out** file, and your test executable **program2test.out** file.

To compile your main program:

```
g++ -o program2.out program2.cpp search_functions.cpp
```

To compile your test program:

```
g++ -o program2test.out test.cpp search_functions.cpp
```

Tips For Success

1. START EARLY!!!! There are a lot of parts to this program so it is essential that you start early to finish the assignment on time.

2. BEGIN SMALL – Start working one small piece of functionality at a time. This will help you not be so overwhelmed by trying to solve the entire thing at once. A possible work flow would be:
    - Implement the search functions first and make sure they work as expected with a hard coded vector (Implement the test program to help with this!)
    - Implement the different states of your input (I recommend using ENUMs to keep track of the programs current state)
    - Implement the file I/O – read in the file line by line and add the string data to each vector.
    - Clean up the code and comment
3. BREAK UP YOUR WORK – like starting small, break up your work into smaller chunks that you can tackle every day. Think about the different parts of this program and how you can solve each part rather than thinking about the task as a whole
4. DO NOT PROCRASTINATE!