Final Report

Bryant Wong 107571009

Due Date: 4/26/2017

**Status of program:**

The program is complete, compiles and runs on Visual Studio Community 2015 perfectly.

The program compiles on csegrid properly. However, there was a problem while compiling while passing a string by reference. The compiler kept thinking that the string was supposed to be a pointer (i.e. No reference to the function 'functionName(std::string&\*) exists'. I fixed this by passing by value, which greatly affected the efficiency of reading in all 10,000 entries in the large data file. The version of code that passed by reference compiled and ran on Visual Studio fine. The program runs on csegrid; all menus and submenus function properly.

**Summary of provided functions.**

In Addressbook.cpp

addEntry : Calls BST::addNode(std::string, Entry)

deleteEntry : Calls BST::deleteNode(Node\*)

displayEntry : Displays the contents of the passed node's data to screen.

searchEntry : Ports the contents of the search tree to a vector, then searches it for user input field and data, returns a vector of node pointers to the entries that the user selected.

inline checkPhoneNumber : Checks if user entered field for phone number is a 10 digit number.

editEntry : Changes the field selected by the user to the data input by the user

readFile : Reads the entire contents of the passed ifstream from a text doc.

sortEntries : Used when the user wants to sort by a particular field when writing entries to file. Can perform single sort for single field or, if there are several repeated values in one field, will sort by that field, then maintaining fidelity of that field, sort by another field. If there are no repeated values in the first field, will sort by second field.

In BST.cpp

Contains functions from CSCI 2421 Homework 8 BSTree.cpp modified to accept the parameters I pass from the other parts of the program.

In Menu.cpp

inline isInteger : For checking the ID, returns false if the user entered string is not an integer.

runMenu : The main menu function, asks for user input for which action they wish to take, loops if invalid entry.

inline checkPhoneNumber : Checks if phone number entered is a valid 10 digit phone number, returns true if it is, ignoring ()- and space characters.

addEntry : The user interface for manually adding an entry. Asks for each field with error checking and sends the entry data to AddressBook::addEntry.

deleteEntry : Calls the searchEntry function to have the user search for, and select, an entry they wish to delete, then sends the node pointer associated with the entry to AddressBook::deleteEntry.

editEntry : Calls the searchEntry function to have the user search and select an entry they wish to edit, asks which field they want to edit, then sends the associated pointer to AddressBook::editEntry.

searchEntry : Asks the user which field they want to use to perform a search, passes that field to AddressBook::searchEntry.

readFile : Opens an ifstream for reading in a file, asks if user wants to use the small or large input file. Calls AddressBook::readFile.

writeFile : Opens an ofstream for write to a file and calls searchEntry to find the entries they wish to write to file. After the list of entries is populated, asks the user if they wish to sort the entries, if so, calls the sortEntries function to sort the entries according to user input, asks the user which fields they wish to be written to the file, then asks the user what they wish to name the file to be written. If they enter a file name that already exists, it will overwrite that file, if they enter a file name that does not exist, the file will be created. The data will be written according to the user's parameters.

sortEntries : User interface to ask if the user wishes to sort by one field or two fields, calls the AddressBook::sortEntries function to sort according to how they wish to sort. If they select sort by two fields, if the first field has any repeated data, for example people sharing the same first name, it will sort by that field, and then secondary sort by the second field. If the first field does not have any repeated data, it will sort by the second field only.

In AddressBook.h

Inline functions equals and less than for comparing the correct field according to user input.

**Design for Final Database Project**

Program Description:

**Changes from original design document: None**

Original text from design document:

This program will be a database system that stores 15 different fields of contact information per entry found in an address book. Users will be able to search the database using any of the 15 fields as well as the search mode and field, and also run a secondary search, read data from a file, write data to a file, delete a record, modify a record, insert new records, and overwrite the read file or save a new file. Users should be able to create an output in text format of specifically chosen contacts, the specific fields they wish to show about those contacts, and to sort the final output based on a key and to sort the output on a secondary key.

Input Requirements: **Changes marked in bold**

For menu: a char to select menu choice of:

1. Read File

2. Search Entries: a char to select which field.

2a. A char to select "exact" or "contains"

2b. A string to enter what to search for

2c. A char to select if they wish to run a secondary search or new search

2d. A char to select "exact" or "contains"

2e. A string to enter what to search for

**2f. A char to select if they want to view what they found**

**2g. A char to select whether or not the selection is correct**

3. Edit entry: (runs search function first to find entry to be edited)

3a. a char to select which field to edit

3b. a string to change the data in the field. All names must be composed of letters, all phone numbers must be 10 digits and numbers, all emails must include @ and .

4. Delete entry: (runs search function first to find entry to be edited).

**4a. A char to select whether or not the selected field is correct to be deleted.**

5. Add entry: 14 strings for the first 14 fields, plus 4 more strings for affiliates information.

**~~5a. All names must be composed of letters, using isalpha()~~**

5b. All phone numbers must be 10 digits and numbers **~~using isdigit()~~**

5c. All email must include @ and .

6. Output to file**~~: an int to represent how many contacts the output file will have.~~**

**Runs the search function to select entries they wish to write to file.**

6a. a char for each field they wish to include.

6b. a char to select which field to sort entries by.

6c. a char to select whether they want to sort entries by another field.

6d. a char to select which field to sort entries by.

**~~6e. a char to select overwrite read in file or save as file~~**

6f. a string to enter file name, **entering an existing file name to overwrite and a new file name to generate a new file.**

7. Exit

Output Requirements: **Changes marked in bold**

Maximum 14 strings + 4 strings(for affiliates). **~~All names must be composed of letters,~~** all phone numbers must be 10 digits and numbers, all emails must include @ and .

Problem solution discussion and Data Structures: **Changes marked in bold**

Menu while loop for choosing add, delete, edit, search entry **(log n for properly balanced tree),** read from file, print to file, exit. Additional menus in search entry for secondary searches, in edit entry for which field to edit, and in print to file to sort (n^2) or secondary sort (n^2) and which field to sort for. **~~Searching for "contains" will probably use a recursive function to step through the data of the field one character at a time to find the user input with complexity of (n^n).~~ "Contains" search will use std::find function included in string.**

**~~The searching and secondary searching should be done in the primary binary search tree and a secondary binary search tree because searching a search tree has much lower complexity (log n) than sorting something like a vector and then searching it, despite the fact that creating the secondary tree would have complexity of (n^2).~~ The searching and secondary searching was all done in a vector with complexity n because the tree is only sorted by the key (the entry's ID number) and is only complexity logn if the tree is balanced and searching only for the ID. Searching by any other field would have to traverse every node, resulting in complexity n. Porting to a vector is complexity n, searching a vector is complexity n so searching for an entry is O(2n)=O(n).** The sorting and secondary sorting will be done in a vector I believe because, while sorting an array does have a complexity of (n^2), the trade off of using a binary search tree to sort, which is essentially already sorted, actually requires more time because populating the tree in the first place has a complexity of (**~~n^2~~** **nlogn**) and if the user wants to apply a secondary sort, the result of the first sort would have to be added to a secondary binary tree with the same complexity. It really depends on how much the user wants to search, sort, add, edit, or delete, so in the cases where the user would be searching more, a binary search tree will be used while the cases where the user would be sorting more, a vector will be used.

User Interface: **Changes marked in bold**

Menu

1. Read File

2. Add entry

3. Search Entry

3a. Enter 1-15 for which field you would like to search for

3b. Choose "exact" or "contains"

3c. Enter data to search for

3d. Would you like to refine your search?

3e. Enter 1-15 for which field you would like to search for

3f. Choose "exact" or "contains".

3g. Enter data to search for

4. Edit entry

4a. Enter field to search for (runs through menus in choice 3)

4b. Choose the item you wish to edit

4c. Enter field to edit

4d. Enter data you would like to change that field to.

4e. Would you like to edit another field in this entry?

5. Delete entry

5a. Enter field to search for (runs through menus in choice 3)

5b. Choose the item you wish to delete

6. Write to file

**Calls search entry function with associated User Interface options to choose which entries are to be written to file**.

**~~6a. Enter the number of contacts you would like to write to file~~**

6b. Choose the fields you would like have displayed (enter some char when finished)

6c. Would you like to sort them?

6d. (If yes), Enter field you would like to sort by

6e. Would you like to sort as another field?

6f. (If yes), Enter field you would like to sort by

7. Exit

Data Flow Chart: No changes from original document.

Data Flow Chart.emf

UML Class Diagram:

**Some of the passed parameters have changed, the overall class diagram is still the same.**

