Programming Project 3: Linked Lists

Summary

In this assignment, you will use linked lists to create a simple contact list. The objectives of the assignment are

- to give you experience in designing an ADT;
- to improve your C/C++ programing and software design skills; and
- to give you experience in programming linked lists and dynamically allocated data.

The project is to be completed in two phases:

1. In the first phase, you will create and submit an ADT for the contact list described below.
2. In the second phase, you will write an implementation of an ADT that I post, as well as a main program that uses it. ***The ADT can not be the one you submitted***.

The ADT is worth 25% of the grade and is due on November 7. The second part is worth 75% of the grade and is due on November 17.

Requirements

The program must implement a contact list. Specifically, when the program starts up, it looks for a file named `contactlist` in the current working directory. If the program finds the file and opens it successfully, it enters an interactive mode, which is described below; otherwise, it displays a message informing the user that it could not open the contact list file, after which it exits.

Contact List

A ***contact list*** consists of an unlimited number of records, each of which has three members:

- *name*, which consists in turn of two members:
  - *first name*
  - *last name*
- *telephone number*,
- *email address*.

Names are not necessarily unique; there may be multiple records with the same name. In this way multiple telephone numbers and/or email addresses can be stored for the same person. Names and email addresses are represented case-sensitively. In other words, Hunter and hunter are different strings as far as this application is concerned. First names and last names may each contain up to 32 characters, which may be letters, hyphens (-), or apostrophes ('). Telephone numbers are strings that may contain up to 16 digits. They cannot contain anything else. The application may display them with hyphens and parentheses, but these are not stored in the record. Email addresses may be up to 127 characters, and must contain exactly one '@' character. All characters to the right of the '@' must be letters, digits, or periods. The characters to the left may be anything except a comma. The telephone number and/or email address field can be a null string (a string with nothing in it.)

Contact List File

The `contactlist` file must be in a specific format in order for the program to open it correctly. It must be a comma-separated-values file, with one record per line and individual members within the records separated by commas. A record consists of the strings that are values of the *first name, last name, telephone, email address.* For example, a record might look like

```
anthony,rocco,2121234567,rocco.cannon@blasting.com
```

No white space is allowed before or after the commas. If white space is found, this is an error. *However, the program is not required to validate the contactlist file; it may assume it is in the proper form.*

The Contact List ADT

The operations supported by the contact list ADT are stated informally below. Your first task will be to refine them and create an ADT that supports them.

Operation and Arguments	**Description**
display(ostream)	The contact list is written to the given ostream in sorted order, by last name as primary key and by first name as a secondary key. If there are two records with identical primary and secondary key, the telephone number is used as a tertiary key, and if need be the email address is the quaternary key. This operations returns the number of records written.
insert(record)	A single record is added to the contact list, provided that it is not an exact duplicate of an existing record (same first and last name, same telephone and same email address. This should return the number of records inserted. Again, the keys are last name, first name, telephone number, then email address.
insert(contact_list)	Every record in the given contact list should be inserted to the existing contact list, provided that none are exact duplicates. Those that are duplicates are not inserted, and those that are not are inserted using the same ordering as described for the insert operation above. This should return the number of records successfully inserted into the list.
remove(record)	The supplied record must contain at least a non-null last name and first name. If the supplied record has either a null last name or a null first name, this operation does nothing. Every record in the contact list whose non-null members match the corresponding non-null members of the given record is deleted from the list. For example, if the given record is `smith,john,2127725000,`"" then all records with the name `john smith` with telephone number `2127725000` are deleted. In other words, the email address is ignored in this case. This returns the number of records deleted.
size()	This returns the number of records in the list.
save()	This saves a copy of the current `contactlist` by writing it to a file named `contactlist.bkp` in the current working directory, overwriting any such file if it already exists. It returns the number of records written to the `contactlist.bkp` file. If it does not have permission to create the file, or if the file cannot be written for any other reason, it returns -1.

Operation and Arguments	Description
find(ostream, lastname, firstname)	Searches the list for all records whose last and first names match the given names and displays them on the given (open) stream in the order in which they occur in the contact list. It returns the number of records written to the stream.
find(ostream, record)	Displays all records whose non-null members match the non-null members of the given record, in the order in which they occur in the contact list, on the open stream. It returns the number of records written to the stream.
make_empty()	This deletes all of the records in the list and returns the number of records deleted.

ADT Rubric and Submission

The first phase requires that you submit the ADT that you design. The ADT must be written using the operation contract model with pre- and post-conditions and UML syntax. It must list all parameters and their types and what the return values are. It will be graded out of 25 points on its correctness (10 pts) (does it describe the class accurately), its completeness (5 pts) (does it include everything that the class interface should have), and its clarity (5 pts) (is it ambiguous or unclear). It will also be graded on the degree to which it adheres to UML and the documentation standards (5 pts).

The ADT should be written as a plain text file, not as a formatted document, nor as a PDF, and should be named `hwk3_adt_`*`username`* , where *`username`* is your username on the Computer Science Department's UNIX network. It should be submitted by running the command

```
/data/biocs/b/student.accounts/cs235_sw/bin/submitadt 3 hwk3_adt_username
```

The ADT part of the assignment is due by the end of the day (i.e. 11:59PM, EST) on November 7, 2016.

Implementation Requirements

Contact List Class

The contact list must be kept in sorted order of increasing last name as primary key, first name as secondary key, telephone number as tertiary key, and email address as quaternary key. The `display()` operation will make no more than a single pass through the list. The list must use a linked representation, either singly- or doubly-linked. The implementations of insertions and deletions should make a single pass through the list in the worst case.

Note. The list that you implement must not use any public functions other than those specified in the ADT, and those functions must have the exact signatures specified in the ADT.

Main Program Requirements

After the contact list has been loaded into memory, the main program enters interactive mode, repeatedly displaying a prompt and waiting for the user to enter a command. After a command is entered and the program responds to it, the prompt is displayed again, unless the command was the `quit` command. The set of commands that the application must support is listed below. The commands are case-sensitive; they must be entered in lower case only. Words in *italic* are placeholders for user-supplied data.

Command	Description
`list`	List the entire contact list on standard output (the terminal window) followed by the number of entries listed.

Command	Description
insert	Prompt the user to enter contact info: last name, first name, telephone number, and email address. The user should be allowed to omit the telephone number and/or the email address. It will display a simple message indicating whether the contact was inserted or not. (E.g. record inserted/record not inserted.)
delete	Prompt the user to enter information to delete a record. The user must supply a last name and a first name, and optionally, a telephone number and/or email address. It will search for entries that match all of the non-null data supplied by the user and delete them. It displays how many were deleted.
find	Prompt the user to enter information to find a record. The user can leave any member blank. This will display on the standard output all records whose non-null members match the non-null data items entered by the user.
save	Save the current state of the contact list to a file named `contactlist.bkp` in the current working directory, replacing that file if it already exists.
quit	Terminate the application.

You are free to decide how to allow the user to enter the various pieces of data for the commands, but you must document your method well. For example, you can prompt for each item or use something like "lastname=smith firstname=john" etc. Your program is expected to ensure that only valid names are stored in the contact list. You may assume that the user enters at least one correct character, so that a name is never an empty string. The program should use the longest valid prefix of the entered text as the name. For example, all of these entered strings should be stored as the name "`john`": `john653`, `john`, `john$%^`, and `john___`.

Project Organization

Your project must consist of three separate files, *named exactly as follows*, case-sensitively:

`contactlist.h` The class interface alone, with the exact set of public member functions specified in the ADT I provided, unmodified. If you choose to create other classes that your contact list class will use, then their interfaces should be placed into this file.

`contactlist.cpp` The implementation of the class interface. If you define any other classes in the `contactlist.h` file, then their implementations should be placed into this file.

`main.cpp` This file should contain the main program and all functions that it uses other than the member functions of the contact list class.

Implementation Grading Rubric

The program will be graded based on the following rubric out of 100 points.

- A program that cannot run because it fails to compile or link on a `cslab` host loses 80%. The remaining 20% will be assessed using the rest of the rubric below.
- Meeting the requirements of the assignment: 50%
- Performance 10%
- Design (modularity and organization) 15%

- Documentation: 20%
- Style and proper naming: 5%

This implies that a program that does not compile on a lab computer cannot receive more than 20 points. Some implementations might be more efficient in terms of running time than others. Although you will not have learned enough to write this program in a very efficient way, there are certain programming decisions that can lead to very poor performance or to acceptable performance. The performance component of 5% will be given if the program is reasonably efficient.

Submitting the Source Code

I will post the ADT you are to use for the implementation on November 8. You will have to implement that ADT in the next phase. This assignment is due by the end of the day (i.e. 11:59PM, EST) on November 17, 2016. Once your program is finished, you are to create a directory named `project3_username`, where `username` is the same as above. Put all source code files into that directory. Do not put executables, data files, or test files in it. If I find any, you will lose three points for each file that does not belong there. Zip up this directory using the UNIX zip command, i.e.,

```
zip -r project3_username.zip ./project3_username
```

This will compress all of your files into the file named `project3_username.zip`.

Before you submit the assignment, make sure that it compiles and runs correctly on one of the `cslab` machines. Do not enhance your program beyond this specification. Do not make it do anything except what is written above.

You are to submit the zip file by running the program `submit235project,` which requires two arguments: the number of the project and the pathname of your zip file:

```
/data/biocs/b/student.accounts/cs235_sw/bin/submit235project 3 project3_username.zip
```

where `project3_username.zip` is replaced by the name of your zip file. If you decide to make any changes and resubmit, just run the command again and it will replace the old file with the new one. Do not try to put your file into this directory in any other way - you will be unable to do this. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.