CSCI 310 – 02 (Fall 2019)

Programming Foundations

Programming Assignment 8

DUE: Sun, Nov 24, 11:59 PM (`turnin` time)

Specifications

In this programming assignment, you will be implementing a simple contact list. Most contact lists are typically implemented using a simple database; hence, in this program we will emulate a simple database using some data structures we have learned in class.

Our contact list will contain the following data about your contacts: their names and their birthdays. Your program should allow entry, removal, modification, or search of this data. For this version, you can assume that the names are unique. Your program should be able to save the data in a file for use later.

Design a class to represent the database and another class to represent the contacts. Use a balanced binary search tree as an index to the data. This index will be based on the contacts' last names.

Input

Your program will read in two input text files. The filenames for these files will be provided as *command–line arguments/parameters*; hence, given the name of the executable is `myContacts` then typing the command

```
myContacts  contacts.txt  commands.txt
```

would use `contacts.txt` and `commands.txt` as the filenames of the two input files. These files correspond to a *data file* and an *operations file* needed by your program.

1. **Data file** (*e.g.* `contacts.txt` in the example above)

   Overview This file contains the raw data that makes up the contact list. Each record contains information for one contact. The information for each contact includes: first name, last name, birth month, birth day, and birth year.

   File format This will be an ASCII text file. Each record will occupy a single line. Each line will contain all 5 attributes of a record with attributes separated by commas; hence, each line will have the form

   ```
   first_name,last_name,birth_month,birth_day,birth_year
   ```

   where 1≤`birth_month`≤12, 1≤`birth_day`≤31, and `birth_year` is a 4–digit value representing a calendar year.

2. **Operations file** (*e.g.* `commands.txt` in the example above)

   Overview This file contains a list of operations to be done on a contact list. Each operation is specified in a single line.

   File format This will be an ASCII text file. Each operation will have the form

   ```
   operation argument
   ```

   Here are the possible values for `operation`, listed alphabetically, and any `argument` it may need:

   `A` ***record*** Add a new record to the contact list. A new record will be provided using the same format as a record in the input data file (see above); for example:

   ```
   A Alice,Barber,3,14,1993
   ```

   `D` ***key*** Delete the unique record from the contact list with a key value (last name) indicated by *key*. Display the matching record that is deleted. If multiple records match the given *key*, display all such records but do not delete any of the records. Display an error message if the record is not found. Compare with `R`emove below.

   `F` ***key*** Find the record(s) from the contact list with a key value (last name) indicated by *key*. Display the record(s) that match the given *key*; otherwise, display an error message if no record(s) match the given key.

   `R` ***key*** Remove *all* records from the contact list with a key value (last name) indicated by *key*. Display all matching records that are deleted. Display an error message if no record matches the given *key*. Compare with `D`elete above.

   `S` Show all the contacts. The contacts will be displayed based on the ordering managed by the *binary search tree* index of the contact last names.

No error–checking will be done on these input files — assume they are correct.

Sample Input

1. Data file

```
Nicole,Martinez,10,26,1990
Peter,Butler,8,10,1991
Kelly,Carter,4,7,1998
Bobby,Garcia,2,27,1992
Alan,Sanchez,2,20,1999
Jeff,Powell,1,19,1999
Debra,Wright,1,23,1990
```

2. Operations file

```
S
F Barber
A Alice,Barber,3,14,1993
S
F Barber
F Martinez
D Martinez
D Smith
S
A Martin,Smith,3,21,1991
A Sam,Carter,10,11,1997
S
D Carter
R Carter
S
```

Output

The output consists of the results from each of the commands listed in the Operations file (see above). All output goes to standard output. When the program terminates, the database is stored using the same filename as the Data file (see above).

Sample Output

For the sample input given above, the program should produce the following output:

```
Loaded 7 records into contact list.
Showing contact list:
  Peter Butler, 8/10/1991
  Kelly Carter, 4/7/1998
  Bobby Garcia, 2/27/1992
  Nicole Martinez, 10/26/1990
  Jeff Powell, 1/19/1999
  Alan Sanchez, 2/20/1999
  Debra Wright, 1/23/1990
Find "Barber"
  Not found.
New contact added
  Alice Barber, 3/14/1993
Showing contact list:
  Alice Barber, 3/14/1993
  Peter Butler, 8/10/1991
  Kelly Carter, 4/7/1998
  Bobby Garcia, 2/27/1992
  Nicole Martinez, 10/26/1990
  Jeff Powell, 1/19/1999
  Alan Sanchez, 2/20/1999
  Debra Wright, 1/23/1990
Find "Barber"
```

```
  Found.
  Alice Barber, 3/14/1993
Find "Martinez"
  Found.
  Nicole Martinez, 10/26/1990
Delete "Martinez"
  Nicole Martinez, 10/26/1990
  Done.
Delete "Smith"
  Not found.
Showing contact list:
  Alice Barber, 3/14/1993
  Peter Butler, 8/10/1991
  Kelly Carter, 4/7/1998
  Bobby Garcia, 2/27/1992
  Jeff Powell, 1/19/1999
  Alan Sanchez, 2/20/1999
  Debra Wright, 1/23/1990
New contact added
  Martin Smith, 3/21/1991
New contact added
  Sam Carter, 10/11/1997
Showing contact list:
  Alice Barber, 3/14/1993
  Peter Butler, 8/10/1991
  Kelly Carter, 4/7/1998
  Sam Carter, 10/11/1997
  Bobby Garcia, 2/27/1992
  Jeff Powell, 1/19/1999
  Alan Sanchez, 2/20/1999
  Martin Smith, 3/21/1991
  Debra Wright, 1/23/1990
Delete "Carter"
  Multiple matches for "Carter"
  Kelly Carter, 4/7/1998
  Sam Carter, 10/11/1997
  Not done.
Remove "Carter"
  Kelly Carter, 4/7/1998
  Sam Carter, 10/11/1997
  Done.
Showing contact list:
  Alice Barber, 3/14/1993
  Peter Butler, 8/10/1991
  Bobby Garcia, 2/27/1992
  Jeff Powell, 1/19/1999
  Alan Sanchez, 2/20/1999
  Martin Smith, 3/21/1991
  Debra Wright, 1/23/1990
Wrote 7 records from contact list.
```

Additional Requirements

Implement a derived `ContactList` class based on the following `Database` abstract class:

```
// A simple database abstract class.
// bjuliano@csuchico.edu

/** @file Database.h */

#ifndef DATABASE_
#define DATABASE_

#include "NotFoundException.h"

template< class ItemType , class KeyType >
class Database
{
public:

    /** Tests whether this database is empty.
        @return True if the database is empty, or false if not. */
    virtual bool isEmpty() const = 0;

    /** Determines the number of entries (records) in the database.
        @return The number of entries/records in the database. */
    virtual unsigned getSize() const = 0;

    /** Adds a new record into the database.
        @param newData  The new record to add to the database.
        @post  The database contains the new record.
        @return  True if the addition is successful, or false if not. */
    virtual bool add( const ItemType& newData ) = 0;

    /** Removes the record with the given key from this database.
            Returns false if duplicate records matching the key exist.
                Default format for aKey is the value of the key (based on index);
        otherwise, two values may be provided, separated by commas,
        to identify a particular record that matches the extended key
        (done if there are duplicates).
        @param aKey  The key of the record to remove from the database.
        @return  True if the removal is successful, or false if not. */
    virtual bool remove( const KeyType& aKey ) = 0;

    /** Removes all record(s) with the given key from this database.
        @param aKey  The key of the record(s) to remove from the database.
        @return  True if the removal is successful, or false if not. */
    virtual bool removeAll( const KeyType& aKey ) = 0;

    /** Removes all records from this database. */
    virtual void clear() = 0;

    /** Gets an entry (or entries) with the matching key from this database.
        @post  The desired entry/entries has been returned in a set, and the
            database is unchanged. If no such entry was found, an exception
            is thrown.
        @param aKey  The key of the record to locate from the database.
        @return  The set containing the entry (or entries) in the database
            that matches the given search key.
        @throw  NotFoundException if the given entry is not in the database. */
    virtual set<ItemType> getEntry( const KeyType& aKey ) const
        throw(NotFoundException) = 0;
```

```
58
59      /** Tests if an entry matching the given key occurs in this database.
60          @post  The database is unchanged.
61          @param aKey  The search key of the entry to find.
62          @return  True if the entry occurs in the database, or false if not. */
63      virtual bool contains( const KeyType& aKey ) const = 0;
64
65      /** Destroys object and frees memory allocated by object. */
66      virtual ~Database() {  }
67
68  }; // end Database
69  #endif
```

Notice that the `Database` template class takes two parameters. The first one, `ItemType`, is the type of record data to store in the database. The second one, `KeyType`, is the type of the (primary) index used to organize the database. For `ItemType`, you would need to define a `Contact` class with the appropriate constructors, mutators, accessors, *etc.* Since the index is by the contacts' last name, the `Database` template parameter `KeyType` would be type `string`.

You will use the following built-in C++ *container classes* to assist you in solving this problem:

- `vector` to store the actual data records (contacts) of your contact list; and
- `multimap` to maintain an index to the last name attribute of a record.

Basically, the `multimap` contains `pair<string,unsigned>` elements where the `string` denotes the last name and the `unsigned` is the corresponding record's index location in the `vector`.

This is a fairly involved project, so plan on starting on it early. The most important part of successfully solving this problem is to make sure you understand all the various components and how they relate to each other. Be sure you can simulate and clearly identify and understand how the various components are updated and how they work together to make things happen **before** you start writing code. You will also need to invest time in understanding the C++ `multimap` container class and its functionality. You will also have to draw your own diagrams/pictures of your data structures to make sure you understand what is going on.

Deliverables

Your submission will consist of the following files, submitted using the Department of Computer Science's `turnin` facility:

- `Contact.h` – header file for the required `Contact` class
- `Contact.cpp` – implementation file for the required `Contact` class
- `ContactList.h` – header file for the required `ContactList` class
- `ContactList.cpp` – implementation file for the required `ContactList` class
- `myContacts.cpp` – driver code containing `main()` function

Note that the following files are available from our *CSCI 310 Lab Files folder* and on `turnin`:

- `Database.h`
- `NotFoundException.h`
- `NotFoundException.cpp`
- `string_tools.h`
- `string_tools.cpp`

We want you to develop good code documentation habits. Source code solutions submitted without any meaningful documentation will receive a total score of zero (0). You may refer to the *Google C++ Style Guide* section on source code comments as a guide.

Be sure to also review and adhere to the **Coding Standards** for this course.