CSC 2430 Spring 2019

LAB 2 – Call Analysis I

Due: See Canvas

Goal: Your assignment is to write a C++ program to read in a list of phone call records from a file into a "call database.' Your program will then allow the user to enter a prefix for a phone number in E164 format (refer to the previous lab for E164 details). Your program will list out all calls to phone number which have that prefix. The assignment introduces you to building a program using multiple files, implementing an ADT (Abstract Data Type), and C++ structs.

Call Database Interface: You will use an ADT for the call database. In a software development team, developers will agree ahead of time on component interfaces (e.g. an ADT interface), so they can work in parallel. If someone doesn't follow the agreement, there will be a build errors and an angry development manager. We too are going to agree on a common definition for call database interface. You must use the definitions describe below exactly as specified.

The first part defines the CALL ADT, which stores one call record:

```
// Stores raw information about one call
struct CALL {
	string start;
	string phonenum;
	string duration;
};

// Get Duration of call as an int
unsigned int GetDurationCall(CALL& call);
// Get formatted start time of call
string GetStartCall(CALL& call);
// Get country code of call
string GetCountryCodeCall(CALL& call);
// Get local phone number of call
string GetPhoneNumberCall(CALL& call);
```

It should not take you long to implement the interface functions relating to CALL. Why? Because you wrote the logic for all of them in Lab 1. The only difference is that you have to use the specified names for the functions, and the parameter is a CALL data structure rather than a string parameter.

The second part defines the CALLDB ADT, which stores a list of call records:

```
// Maximum number calls in call database
const int MAXCALLS = 15;

// Call database
struct CALLDB {
	CALL callLog[MAXCALLS];		// Stores calls in database
	unsigned int  numCalls;		// Number of calls stored
};

// Initializes call database to have store no calls
void InitCallDB(CALLDB& calldb);
// Load database with calls stored in a file
unsigned int LoadCallDB(CALLDB& calldb, istream& fin);
// Get count of call records
unsigned int GetCountCallDB(CALLDB& calldb);
// Retrieve call records
bool GetCallInCallDB(CALLDB& calldb, unsigned int index, CALL& call);
// Return first index >= of call w/ matching phone number prefix, or -1
int FindByPhonePrefixInCallDB(CALLDB& calldb, unsigned int startIndex, string prefix);
```

You will need to write these five functions:

1) *InitCallDB* initializes the call database to have zero CALL records. This is a one liner.

2) *LoadCallDB* reads call records from an open `ifstream` (`istream` to be general) until there is no more data or an error occurs. This function uses a loop. On each iteration, it will read the three fields as strings, call the validate functions you wrote in Lab 1, and if the data passes validation and there is room left in the array, fill in the first unused CALL record with the data you read. This function will return the count of records that failed validation or which would not fit in the array because there was no room left. NOTE: the type of the second parameter is actually `istream`. Using this type instead of ifstream makes writing a unit test program easier.

3) *GetCountCallDB* returns the number of call records in the database at the current time. This is another one line.

4) *GetCallInCallDB* checks whether the specified index is within range of the CALL records currently stored in the callLog array. If it is not in range the function returns false. Otherwise it will return true and set the call reference parameter to the CALL record at the specified index in the callLog array.

5) *FindByPhonePrefixInCallDB* will scan the database of call records looking for the first record at or after the specified index with phone number field having the specified prefix. It returns the index of the first record it finds, or else -1 if it can't find one.

Filenames you must use: Your main program will be built from 3 source files:

- callanalysis.cpp – main program
- calldb.h – header file for call database ADT
- calldb.cpp – implementation of call database ADT

Getting Started:
Developers often set some milestones where they do a partial implementation of their changes, and stabilize the code, and then continue onto the next milestone. I think it would help you if you schedule 3 milestones for this assignment.

Milestone 0:

- Create calldb.h and calldb.cpp.
- Copy in the definition of the ADT into calldb.h.
    - Make sure to write include guard statements beginning with `#ifndef _CALLDB_H_` (see other zyBooks 1.21 (week4-5 workout) for an example).
- Add stubs (i.e., barebone function definitions) for the four interface functions in calldb.cpp.
- Download "`lab3unittest.cpp`".
- Try compiling using the command "`g++ -std=c++11 lab3unittest.cpp calldb.cpp`".
    - **Ensure the code builds with no errors before moving on.**

Milestone 1:

- First implement *GetDurationCall*, *GetStartCall*, *GetCountryCodeCall*, and *GetPhoneNumberCall*.
    - This should not take long since you can adapt the code you wrote in Lab 1.
- Copy these previous validation functions you wrote to calldb.cpp as well.
    - These functions are not part of the ADT interface, but they are internal functions that your ADT implementation will use.

- Next implement *InitCallDB*, *GetCountCallDB,* and *GetCallInCallDB.*
    - The first two are one line, the third takes a bit more code, but still is quite simple.
- Now it's time to write *LoadCallDB* so you can read data into the call database.
- Finally implement *FindByPhonePrefixInCallDB.*

Now try compiling and running unit test program. If the program terminates with an assert, look for the line number of the assert. It should refer to a line in lab3unittest.cpp. Check what is wrong, and fix your code. When it finally passes the unit test, then your ADT implementation is probably right.

After finishing Milestone 1, **submit your calldb.cpp and calldb.h file to zyBooks Autograder Lab 2-1**. If it passes your local unit test, then your code is very highly likely to pass zyBooks autograder. I will be ready to help you otherwise.

Milestone 2: Now it's time to take your main program from Lab 1 and try to update it to run against your ADT. You will want to have a main program and an additional function for printing the record that match a specified phone number prefix. This print function will take the call database as a reference parameter and a string parameter specifying the prefix to match on.

Here's an outline of what your main program should do:

1) Get a filename via cin with prompt `File Name:`
2) Print message and exit if file couldn't be successfully opened
    a. `Can't open '<filename>'`
3) Initialize your call database
4) Load the records from the file into the call database
5) Close your file
6) Print a message indicating how many records were added (X), and how many dropped (Y)
    a. `Log successfully read into database, X records added, Y records dropped`
7) Print out records in the call database
    a. `Contents of Call Database`
    b. If there is no records, Print `No records`
    c. If there's any, print with the same format to the Lab 1 one.
8) Go into a loop, where you
    a. prompt the user to enter a phone number prefix to query on
        i. `E164 prefix for query:`
    b. print out the records that match the query
        i. If there is no records, Print `No records`
        ii. If there's any, print with the same format to the Lab 1 one.
9) If the user simply hits ENTER to specify the query string, your program terminates.
    a. You can detect this by checking if the input string is empty.

NOTE: you will use your print function to print out all the records in the call database in step 7), and also when you print the records that match the query in 8a).

The following screenshots illustrate the operation of the program:

Empty File:

```
File Name:empty.txt
Log successfully read into database, 0 records added, 0 dropped

Contents of Call Database
No records

E164 prefix for query:+3
No records

E164 prefix for query:
```

Nonexistent file:

```
File Name:phonephone.txt
Can't open 'phonephone.txt'
```

File with 3 records, all correctly formatted:

```
File Name:phoneLog1.txt
Log successfully read into database, 3 records added, 0 dropped

Contents of Call Database
Time      Country  Phone Number    Duration
11:00:54  NA       (206)-281-2000  98:32
09:12:33  FR       01 40 20 59 90  6:04
14:53:16  HK       2367 7065       3:00

E164 prefix for query:+3
Time      Country  Phone Number    Duration
09:12:33  FR       01 40 20 59 90  6:04

E164 prefix for query:
```

File with two incorrectly formatted records (and no good records):

```
File Name:phoneLog2.txt
Log successfully read into database, 0 records added, 2 dropped

Contents of Call Database
No records

E164 prefix for query:+3
No records

E164 prefix for query:
```

File with 16 records (1 more than will fit):

```
File Name:phoneLog3.txt
Log successfully read into database, 15 records added, 1 dropped

Contents of Call Database
Time      Country  Phone Number    Duration
11:00:54  NA       (206)-281-2000  0:33
14:02:11  NA       (800)-642-7676  10:01
08:21:33  FR       01 40 20 59 90  6:04
13:32:10  NA       (800)-642-7676  5:24
08:21:33  FR       08 92 70 12 39  4:16
07:04:53  HK       2367 7065       3:00
08:53:16  HK       2508 1234       3:35
12:22:04  NA       (206)-281-2000  2:30
15:41:10  NA       (855)-836-3987  5:05
20:03:11  FR       01 42 34 56 10  2:03
06:02:09  NA       (800)-642-7676  5:24
08:14:18  NA       (425)-519-0080  2:35
10:09:57  HK       2112 0099       1:41
08:17:15  HK       2508 1234       3:35
09:00:28  NA       (206)-281-2000  8:25
16:20:13  NA       (800)-642-7676  10:01

E164 prefix for query:+3
08:21:33  FR       01 40 20 59 90  6:04
08:21:33  FR       08 92 70 12 39  4:16
20:03:11  FR       01 42 34 56 10  2:03

E164 prefix for query:+31
No records

E164 prefix for query:+338
08:21:33  FR       08 92 70 12 39  4:16

E164 prefix for query:
```

As described earlier, your solution will consist of three files: "callanalysis.cpp", "calldb.h", and "calldb.cpp". You must use these filenames.

If you think you have finished the main source code, before submitting, you can check if these files would work correctly.

1) Create a new directory
2) Copy the three files you have written to this new directory
3) Bring up a Command Prompt or Terminal.
4) Use "cd" to position yourself into this directory
5) Run the command "g++ -std=c++11 callanalysis.cpp calldb.cpp"
6) Make sure it compiles with no errors, and links. "a.exe" or "a.out" should be created.
7) Download the test files "empty.txt", "phoneLog1.txt", "phoneLog2.txt", and "phoneLog3.txt", into this directory
8) Run your program on all four files, do some queries. Make sure the results look right and match the screenshots. Your program will be run and the results visually compared with what is expected.

When you are able to pass all these tests, submit "callanalysis.cpp", "calldb.h", and "calldb.cpp." Submit these three files to **zyBooks Autograder Lab 2-2. It will do input/output tests and check if your code generates the correct result.**

Program Style
Please refer to the Lab 1 assignment document for appropriate style.

In addition, there are some more style rules for working with ADT's:

1) You must not access the ADT in any other way than calling the interface functions defined in the .h file.
2) Adding new ADT interface functions is not allowed.

Finally, always pass struct's as reference parameters to avoid needless copying of information onto the stack.

Grading
Correctness is essential. Make sure your solution builds as specified above and correctly handles all the input files described earlier in this document. In addition, the unit test must pass all tests.

You should submit your codes to zyBooks Autograder Lab 2-1 and 2-2. See the above instructions carefully.

Even if your solution operates correctly, points will be taken off for:

- Not following the design described above
- Not adhering to style guidelines described above
- Using techniques not presented in class
- Programming error not caught by other testing

Academic Integrity

This programming assignment is to be done on an individual basis. At the same time, it is understood that learning from your peers is valid and you are encouraged to talk among yourselves about programming in general and current assignments in particular. Keep in mind, however, that each individual student must do the work in order to learn. Hence, the following guidelines are established:

- Feel free to discuss any and all programming assignments but do not allow other students to look at or copy your code. Do not give any student an electronic or printed copy of any program you write for this class.
- Gaining the ability to properly analyze common programming errors is an important experience. Do not deprive a fellow student of his/her opportunity to practice problem solving: control the urge to show them what to do by writing the code for them.
- If you've given the assignment a fair effort and still need help, see the instructor or a lab assistant.
- **If there is any evidence that a program or other written assignment was copied from another student, neither student will receive any credit for it. This rule will be enforced.**
- Protect yourself: Handle throw-away program listings carefully.

Final Note
In the later labs, we may reuse the code you write in this assignment, so don't lose track of it.