CS162 Project 2
Physical Exercise Recorder
With File Read and Array of Structs

For this assignment, you will build on project 1 to add new functionality to your exercise recorder program. Specifically, you will load previous exercise sessions from a text file when the program starts, and store the information in an array of structs. Use a function for this (see below for function prototype suggestions). Take a look at the struct in the textbox below to get an idea of the information that needs to be read from the file and stored for each exercise session. The items are: name, date, note, time, calories, and maximum heartrate. When the user adds new activities, add them to the array of structs also. There is no need to write the data back to a text file with this project. In other words, you only need to read data from a file, not write data to a file. Also add the ability to list all exercises in memory and to search for a specific exercise. Create separate functions for this functionality also. Notice in the example output below that there might be more than match, so make sure to output all of the entries that match the given name.

```
struct exerciseData {
  char name[strSize];
  char date[strSize];
  char note[strSize];
  int time;
  int calories;
  int maxHeartRate;
};
```

Example File

There will be an example text file in the assignment area along with this assignment, but you may create your own. If you create your own file, do not add headers or other file details, and do not use a different format. Do not assume that you know how many items are in the text file. This example has 4, but you must be able to handle a file with any number of items (within reason; the size of your array will be the limit, see below). The example file uses one line per workout, and it uses commas to separate the values:

```
Elliptical,06/10/17,great workout,40,400,135
Treadmill,06/12/17,doggin it,20,150,120
Stationary Bike,06/15/17,felt good,30,200,130
Elliptical,06/20/17,great-worked out with Mike,45,350,140
```

Example Program Output

```
Welcome to the exercise tracking program.
What is the name of the exercise data text file to load? exercise.txt
Successfully loaded 4 activities.
What would you like to do: (l)ist all, (s)earch by name, (a)dd an exercise, or (q)uit? : l
The exercises are as follows:
Name             Date      Time  Calories   Max Heartrate  Note
Elliptical       06/10/17  40    400        135            great workout
Treadmill        06/12/17  20    150        120            doggin it
Stationary Bike  06/15/17  30    200        130            felt good
Elliptical       06/20/17  45    350        140            great-worked out with Mike
What would you like to do: (l)ist all, (s)earch by name, (a)dd an exercise, or (q)uit? : a
What exercise activity did you do? Free Weights
What was the date (mm/dd/yy): 06/21/17
How many minutes? 60
How many calories did you burn? 160
What was you max heart rate? 110
Do you have any notes to add? Felt the burn
OK, Free Weights on 06/21/17. Time: 60, Calories: 160, Max heartrate: 110: Note: felt the burn.
Record the activity time and calories (y/n)?  y
Your activity info has been recorded.
What would you like to do: (l)ist all, (s)earch by name, (a)dd an exercise, or (q)uit? : l
The exercises are as follows:
Name             Date      Time  Calories   Max Heartrate  Note
Elliptical       06/10/17  40    400        135            great workout
Treadmill        06/12/17  20    150        120            doggin it
Stationary Bike  06/15/17  30    200        130            felt good
Elliptical       06/20/17  45    350        140            great-worked out with Mike
Free Weights     06/21/17  60    160        110            felt the burn
What would you like to do: (l)ist all, (s)earch by name, (a)dd an exercise, or (q)uit? : s
What activity would you like to search for? Elliptical
Here are the activities matching Elliptical:
Date      Time  Calories   Max Heartrate  Note
06/10/17  40    400        135            great workout
```

```
06/20/17  45    350        140            great-worked out with Mike
Found a total of 2 entries.
What would you like to do: (l)ist all, (s)earch by name, (a)dd an exercise, or (q)uit? : q
Thank you for using the exercise tracking program.
```

Program Functions

You must write at least 3 functions to implement this project, in addition to main(). The 3 functions are:

```
int loadData(exerciseData exerciseList[]); // pass the array of structs, return total exercises read.
int search(exerciseData exerciseList[], int count); // pass the array and the number of items in the array.
void list(exerciseData exerciseList[], int count); // Print all data in the list.
```

Program Requirements

- In addition to the requirements listed below, follow all of the requirements for project 1: No literals, no globals, must compile with g++, no string objects or stl containers, no c-style input or output, guard against bad data (for cin only; no need to guard against bad data from the file).
- You must use the <iomanip> library and setw() to output the data in column-formatted output.
- Guard against index out of bounds / buffer overflow. See below in strategies.
- Do not use cin extraction (>>) to read data into c-strings, because this is a security issue. Use cin.getline() instead.
- The array of exerciseData structs must be declared in main, and the struct must be defined globally. This means that you must pass the array of structs to your functions as an argument.
- To-do items for loadData:
    - Set up a local c-string to store the filename and a local input file object, ask the user what the file name is, and use the c-string to open the input file. Set up an eof-controlled while loop to read all data from the file. After the loop, return the number of exercises read. Remember that arrays are passed by reference, so loading data into the exerciseList array parameter will change the actual array passed from main. Suppose you also create a local index variable and a temporary c-string:

```
index = 0;
char tempStr[SIZE];
while (!inFile.eof()) {
    // use tempString to store c-strings from the file: cin.getline(tempStr, SIZE, ',');
    // see below, programming strategies, for an explanation.
    // read the data into the array at index: strcpy(exerciseList[index].name, tempStr);
    // add one to index.
}
// close the input file.
return index;
```

- To-do items for search
    - Set up a c-string to ask the user what exercise they would like to search for, maybe searchName, then create a for loop to iterate through all of the items in the array. If you find a match, output the data for the exercise, and increase the count of the items found. At the end of the function, return the number of items found. Do not output the number of items found, only output the data for each exercise. Output the number of items found back in main, after the function returns. Suppose count is a formal parameter and specifies the number of items in the array. Then set up a local variable to store the number of items found, maybe exCount:

```
for(i = 0; i < count; i++) {
    // use strcmp to compare the search string. Remember that you can't do direct compare.
    if(strcmp(searchName, exerciseList[i].name) == 0) {
        // found a match, output the data and increment exCount.
    }
}
return exCount;
```

- To-do items for list
    - This function is straight forward compared to the other functions. Just create a for loop that stops at the formal parameter count, and output all data in the array.

Programming Strategies

- Remember that newline and extraction don't get along very well, as explained in the last project.

- When the program starts, you will have an empty array of structs, so the index of the first item will be zero. Then when you call loadData, the array will be partially filled up with the data from the file, and you will know how many items were read, because loadData will return that information. Suppose you set up a variable, indexCount, in main. Then you could call loadData using indexCount:
  ```
  indexCount = loadData(exerciseList);
  ```
  Now we know how far we need to search for items, or how many items to list, and this is the value passed to list() and search:
  ```
  count = search(exerciseList, indexCount);
  ```
  We need to use count to stop the loops, because if you loop all the way up to ARRAY_SIZE, most of the items in the array will be empty.
- indexCount serves a dual purpose here. It tells us how many items are in the array, and the index of the next item to go into the array. Suppose loadData() returns 4. Then we know that locations 0, 1, 2, and 3 contain data, and that index location 4 is the next available location in the array.
- Let's look at how we can read words and numbers from a line, separated by a delimiter, in this case, the comma. Suppose we have a line such as:
  ```
  the words,5,some more words,7
  ```
  Don't forget that there's a newline at the end that must be removed, either with getline() or ignore(). So we only want to extract "the words" into a c-string. There is a 3-argument version of getline that will do that:
  ```
  inFile.getline(theString, SIZE, ',');
  ```
  The delimiter is the third argument, so it will stop reading when it reaches the comma. Since this is getline, it throws away the comma (cin.get() will stop at the delimiter, but it leaves it in the buffer). Now we need to get the number from the stream. There are two ways to do it: read it into a string (just like we did with getline above) and then use atoi() to convert it, or use regular extraction. If you use regular extraction, remember to get rid of the delimiter (the comma):
  ```
  inFile >> numberVar;
  inFile.ignore();  // get rid of the comma.
  ```
- When you create your array of exerciseData structs, there are only a limited number of places to store information. Set up a global called SIZE or ARRAY_SIZE or something like that, and set it to a reasonable number, like 100 or 128 or something like that. Your program will be continually adding data to the array, and if it adds too many items, it might have an index out of bounds. An index out of bounds could crash your program, give you strange results, or allow a hacker to take over your computer. So always check for index out of bounds:
  ```
  if(userChoice == 'a') {
      if(index >= ARRAY_SIZE) { // Suppose ARRAY_SIZE is 100. Then 99 is the last index.
          cout << "Sorry, the array is full." << endl;
      } else {
          // add the items to the array at index like normal.
      }
  }
  ```
  This is just example code. You don't have to use if statements (you could use a switch) and you can set up your logic in a way that makes more sense to you.

Place your name, assignment number, and a sources line at the top of the code file that lists any sources of help that you may have received.

How to submit:
Submit your source code file via email attachment on the PCC Linux system. Use mailx to do so. Make sure that your name is part of the source code file name. For me it could be something like MartinA02.cpp. Suppose that your source code file is called NameProject2.cpp, and you would like the subject line to be "162 A02, Name," which stands for assignment 2, and replace "Name" with your actual name. Then the mailx command to submit your assignment would be:

```
mailx -a NameProject2.cpp -s "162 A02, Name" robert.martin4@localhost < /dev/null
```

You can mail the message to yourself as well. Just put your email address after (or before) my email address, separated by a comma:

```
mailx –a NameProject2.cpp –s "162 A02, Name"
robert.martin4@localhost,yourLoginName@localhost < /dev/null
```

After that, start up the alpine program and see if you have a new message from yourself with your source code file attached. If you are really worried about your submission going through, you may also submit the source code file to the D2L assignments area, but I must receive the file through the Linux mail system for you to get credit. Don't modify your files after you submit them to preserve the date information.
You may submit as many versions of your homework as you would like up to the due date. If you find an error in your code after you have made a submission, then you may send me a revised version, but only if the due date has not passed.
If your submission is late, then you may only submit once. Please see the syllabus for late submission details.