CSE30: Lab #6 – Structures

Overview

This lab will give you some insight of the power associated with Object Oriented Programming by exploring the concept of structures. In addition, you will have to perform simple string operations, which are essential tools for programmers dealing with text. As will become clear in this lab, structures allow programmers to define their own "data type" by defining objects. In this lab, you will define a Time structure that represents hours, minutes, and seconds of the day, along with a Course structure that represents a course that you can take at a University. Together, the structures can be used to create a schedule of course for students.

Note: You need to have a separate program for each part of this lab. When you submit your assignment through CatCourses, make sure that ALL PARTS are included.

Getting started

Create a new directory in your main development directory (probably on Desktop/CSE30) called Lab_06. Try to use the terminal on your own, without getting help from the TA to setup the new directories (try to learn/remember the terminal commands).

Reminder: Once you have created a file and understood its content, you want to compile the source file (ex. main.cpp) into an executable so that you can run it on your computer. Note: EVERYTIME you modify your source file, you MUST re-compile your source file for the changes to take effect. You will compile from the command line by running:

```
g++ <source> -o <executable>
```

where g++ is a program (already installed on your Linux system) that compiles C++ source files, **source** is the source file for your program (main.cpp in this example), **-o** tells the compiler that you want to give the executable its own name, and **executable** is the name you want to give your program. In order to compile your first program, you would have to run:

```
g++ main.cpp -o main
```

Assuming that your program compiled successfully (i.e. no errors were found), you can run your program by typing **"./main"** in the terminal/console.

Good coding practices (worth 2 points!)

Writing code that is understandable by humans is as important as being correct for compilers. Writing good code will help you complete the code, debug it and … get good grades. It is very important to learn as soon as possible, because bad habits are hard to get rid of and good habits become effortless. Someone (guess who) reads your code will be in a better mood if it is easy to understand … leading to better grades! This lab will include 2 points (10% for code quality):

- Explanations with comments
- Meaningful names
- Indenting of blocks { } and nesting …
- Proper use of spaces, parentheses, etc. to
- Visible, clear logic

- One / simple statements per line
- Anything that keeps your style consistent

(Exercise 1) Create – timeInput.cpp

In this part of the lab, you will be creating your own **Time** structure to keep track of various times of the day. Your ***Time*** structure should contain 3 **integer** variables:

- one for the hour
- one for the minutes
- one for the seconds

Your program will setup a start & end time for a lecture at the University. It will ask the user to enter the **start time** and the **end time** for the lecture, using the 24 hour format (HH:MM:SS). You need to check that the input produces a valid time:

- check if the string contains the right characters (i.e. two characters [0-9], a colon, etc.)
- check if the time entered makes sense (i.e. HR is 0-23, etc. ... for instance, 33:79:99 does NOT make sense).
- If the time is incorrect, output an error message and exit the program.

Once the user has entered two valid times, your program will output the start and end times in a 24 hour format. For those who are unfamiliar with 24 hour format, please see http://en.wikipedia.org/wiki/12-hour_clock.

Note: in order to receive full credit for this part of the lab, you MUST create a main program that defined the ***Time*** structure and the following two functions:

- ***getTimeFromUser***: takes in a ***Time*** structure variable as a parameter, reads a string from user, checks to see if it is a valid 24 hour format, stores it inside the ***Time*** structure variable, and return true or false depending on whether or not the time is valid. **Since the Time structure is not returned at the end of this function, you need to pass the reference (&) of the Time structure instead of the identifier (name) as the parameter. This way, the contents of the Time structure will be updated after the function is executed and goes back to the main function.**
    - Write a pseudocode of ***getTimeFromUser*** function before writing it in C++ code.

- ***print24Hour***: takes in a ***Time*** structure variable as a parameter and prints the time using the 24 hour format.

Hints:
- see the following examples for the output structure
- use ***getline*** with ***cin*** for all of the user input i.e. getline(cin, my_line) instead of cin >> my_line in order to read symbols (":") from user.
- you will need to read the entire string for the time (i.e. 12:30:00), and process it to extract the numbers. You should extract the numbers one by one, convert them to integers, and store them in your structure. The string functions ***substr*** and ***find*** will be especially useful.

- ***my_line.substr(x, n)*** will return a string of ***n*** characters starting at position ***x*** of ***my_line***. Find more information about ***substr*** function at http://www.cplusplus.com/reference/string/string/substr/

- ***my_line.find("ABC", x)*** will search for ***"ABC"*** starting at the position ***x*** of ***my_line*** and return the position of the first occurrence of ***"ABC"*** (at or after position ***x***). Find more information about ***find*** function at http://www.cplusplus.com/reference/string/string/find/

- as you have done in previous labs, you will need to convert from strings to integers. The function you can use to do so is ***atoi***, which takes in an ***array of characters as a parameter (NOT a string).*** To convert a string to an array of characters, use the function ***c_str()***. i.e. ***atoi(my_line.c_str())***
- in some cases, you will need to add a 0 in front of single-digit numbers (e.g. 15:02:00, as opposed to 15:2:0). The functions ***setw*** and ***setfill*** will allow you to do that.
  - ***cout << setfill ('x') << setw (10);*** will set 'x' to fill up 10 character spaces.
  - ***cout << 77 << endl;*** will output **xxxxxxxx77** when the previous line was used.
  - Find more information about ***setfill*** and ***setw*** functions at http://www.cplusplus.com/reference/iomanip/setfill/

Example 1:
Enter the start time for the lecture (format is HH:MM:SS): ***12:22PM***
The start time entered is invalid!

Example 2:
Enter the start time for the lecture (format is HH:MM:SS): ***23:59:59***
Enter the end time for the lecture (format is HH:MM:SS): ***23:85:00***
The end time entered is invalid!

Example 3:
Enter the start time for the lecture (format is HH:MM:SS): ***09:00:00***
Enter the end time for the lecture (format is HH:MM:SS): ***10:15:00***
The lecture starts at 09:00:00 and ends at 10:15:00

Example 4:
Enter the start time for the lecture (format is HH:MM:SS): ***13:00:00***
Enter the end time for the lecture (format is HH:MM:SS): ***13:15:30***
The lecture starts at 13:00:00 and ends at 13:15:30

(Exercise 2) Create – courseInfo.cpp

In this exercise you will expand your timeInput.cpp, which only deals with lecture times, so that you can have more information about each lecture, and print a schedule to the student. More specifically, you should keep your ***Time*** structure from timeInput.cpp and create a new ***Course*** structure consisted of the following variables:

1. **name**, type string, which stores the name of the course
2. **credits**, type int, which stores the number of credits for the course

3. **majorRequirement**, type bool, which tells if the course is a requirement for your major

4. **avgGrade**, type double, which gives the average grade (in percentage) that past students received in the course
5. **days**, type string, which stores the days that lectures are held
6. **startTime**, type Time (from exercise 1), which is the start time of the lecture
7. **endTime**, type Time (from exercise 1), which is the end time of the lecture

Your program should read the course information from a text file called "in.txt". This file should contain the following pieces of information on the lines specified below:

Line 1: Course number: How many courses the user wants to sign up for.
Line 2: Name of the course
Line 3: Credits
Line 4: Is the course a major requirement? (1=yes, 0=no)
Line 5: Average grade for the course
Line 6: Lecture days
Line 7: Start time of the lecture
Line 8: End time of the lecture
Line 9-15: Repeat same information for additional course, if it exists.

Note:
- In order to receive full credit for this part of the lab, you MUST create the two structures, ***Time*** and ***Course***, consisted of the proper variables.
- In order to read the **avgGrade**, you will need to convert from strings to floats. The function you can use to do so is ***atof***, which takes in an array of characters as a parameter **(again, NOT a string)**. To convert a string to an array of characters, use the function ***c_str()***.

Example 1:

in.txt:

```
1
cse20
4
1
89
Mondays
16:30:00
17:20:00
```

Your output should be:

```
--------------------
SCHEDULE OF STUDENT
--------------------
COURSE 1: cse20
Note: this course is a major requirement!
Number of Credits: 4
Days of Lectures: Mondays
Lecture Time: 4:30:00pm -5:20:00pm
Stat: on average students get 89% in this course.
```

−−−−−−−−−−−−−−−−−−

Example 2:

in.txt:

```
3
cse20
2
0
75
Mondays
16:30:00
17:20:00
cse30
4
1
78
Mondays and Wednesdays
16:30:00
17:45:00
math5
3
0
98
Tuesdays
09:00:00
09:50:00
```

Your output should be:

```
−−−−−−−−−−−−−−−−−−
SCHEDULE OF STUDENT
−−−−−−−−−−−−−−−−−−
COURSE 1: cse20
Note: this course is not a major requirement...
Number of Credits: 2
Days of Lectures: Mondays
Lecture Time: 4:30:00pm -5:20:00pm
Stat: on average students get 75% in this course.
−−−−−−−−−−−−−−−−−−
COURSE 2: cse30
Note: this course is a major requirement!
Number of Credits: 4
Days of Lectures: Mondays and Wednesdays
Lecture Time: 4:30:00pm -5:45:00pm
Stat: on average students get 78% in this course.
−−−−−−−−−−−−−−−−−−
COURSE 3: math5
Note: this course is not a major requirement...
Number of Credits: 3
Days of Lectures: Tuesdays
Lecture Time: 9:00:00am -9:50:00am
Stat: on average students get 98% in this course.
−−−−−−−−−−−−−−−−−−
```

Example 3:

in.txt:

0

Your output should be:

```
--------------------
SCHEDULE OF STUDENT
--------------------
```

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have included the following ***before*** you press Submit:

- Your **timeInput.cpp**, **courseInfo.cpp**, and a list of Collaborators.
- File attachments of the pseudocode you create in this lab. If you write your pseudocode on papers, scan them as images and attached the images. If you write your pseudocode in text editor, save them as text files and attach them in your submission.
- Documentation (in a text file) of code you used from the internet. You may want to cut-and-paste the original code as well.