Assignment 4

Elevator C Code

Due: April 17, 2015
before midnight

Purpose:

In this assignment you will learn:

- how to translate pseudocode into working C code, and the pitfalls associated with not having thought through your programming logic in enough detail;
- how to link separate code modules, including the pre-compiled object code located in the elevator simulator software.

You will have completed this assignment when you have:

- Implemented the pseudocode you originally submitted in Assignment 1 as working C code;
- Linked your object code with the elevator simulator object code to create an executable file;
- Used your code to simulate the operation of a working elevator, which should behave in every way just as an elevator would in the real world.

Assignment 4

Elevator C Code

I. Introduction

In Assignment 1 you were tasked with constructing the pseudocode designed to control the operation of an elevator. Now the time has come to implement that pseudocode in C. In this assignment you'll use the logic you devised in that first assignment to control the operation of a 'real' elevator, or at least one manipulated in code using an elevator simulator.

II. The Elevator Simulator Software

The Elevator Simulator Software (or ESS) displays the presumed operation of a working elevator that behaves according to your instructions.

Sample output for the ESS is shown in Figure 1 at right. The output can be divided into three columns. The leftmost column displays the floors in a building, from the TOP floor (the 10th floor in this example) down to the BASEMENT. So the total number of FLOORS is `11`.

The middle column shows the building itself, with each box representing one floor. The actual location of the elevator during each cycle of the software is indicated by the box with numbers and arrows in it. In Figure 1, the elevator is on the 2nd floor, and it contains two people who wish to go up and no one who wished to go down.

The right column indicates the number of potential passengers waiting for the elevator who have pressed the UP and DOWN buttons on each floor. Figure 1 indicates that on the 3rd floor one person has pressed the UP button; on the sixth floor one person is waiting to catch the elevator going DOWN.

```
Elevator Simulator Software Version 1.0

Floor:TOP
Floor: 9
Floor: 8
Floor: 7
Floor: 6                                  ▲0 ▼1
Floor: 5
Floor: 4
Floor: 3                                  ▲1 ▼0
Floor: 2                  ↑2 ↓0
Floor:GROUND
Floor:BASEMENT

COMMAND ISSUED: Go(UP)
```

Figure 1

The last lines of the simulator indicate the command that has just been executed by your code, along with any messages output that may be displayed by the simulator itself (none are shown in the example above.)

Each time you call the `Go()` function, the old screen is cleared and a fresh output is generated. Note that you can use `ddd` to interact with the ESS functions one line at a time. One thing you *cannot* do is use `printf()` to output messages to the console. This is because the ESS software uses Unicode to output its display, and this requires that you *must* use the `wprintf()`

statement for your output instead: `printf()` and `wprintf()` are mutually exclusive in the same program. To use Unicode with your output, see, for example, http://www.cplusplus.com/reference/cwchar/wprintf/. (Note that C and C++ use Unicode in the same way, so time spent learning Unicode now will potentially save you time in the future.)

Your code should allow the user to step through each cycle of your code by entering a character, such as the space bar, to allow for proper testing; do not execute your code on 'autopilot', running it repetitively without the possibility of human intervention.

As described in Assignment `1`, there are five functions you can call on to manipulate the functioning of the elevator. These five functions are:

- `Go()`, which takes as arguments `UP`, `DOWN`, or `STOP` to tell the elevator to go up one floor, down one floor, or stop at the current floor. You must call this function each time through your loop, or the ESS output will not be displayed. Note that the output shown in Figure 1 is assumed to appear *after* the `Go()` command has been issued, and the elevator has just arrived at the floor whose value has been returned by the `Go()` function.

  Also note that the three parameters of the `Go()` function, `UP`, `STOP`, and `DOWN`, are enumerated data types declared in ESS.h having the tagName `nextFloor`. An enumerated data in C is just a constant integer, so while you should use the enumeration type given in your code, you can safely debug your code by passing variables declared as `const int` to this function, rather than using the three 'official' enumerated data type.

- `buttonPressedInside()`, which simulates a button pressed by a passenger inside the elevator. The value returned is the numerical value of the floor number the passenger wants to stop at. In the ESS display, passengers exit when they reach the floor they selected after this button was pushed. So if one of the people in the elevator shown in Figure 1 wished to exit at the 3rd floor, then the output would indicate this by displaying `↑1 ↓0` when the elevator reached that floor—one of the two people in the elevator has exited.

- `buttonPressedOutsideGoingUp()`, simulates someone pressing the `UP` button on a particular floor. This function returns a pointer to an array of `FLOORS` elements. For each floor on which the `UP` button has been pushed, that element will be non-zero. So, for example, if someone in the basement pushes the `UP` button, then the 0th element of the array pointed to by this function will be set to a non-zero value;

- `buttonPressedOutsideGoingDown()` performs the same operation as the previous function, but returns a pointer to an array indicating the floors on which a potential passenger has pressed the `DOWN` button;

- `openDoorCloseDoor()` does exactly what it's name suggests. The function takes no parameters and returns no value. But it is necessary that this be called on each floor during which passengers need to get on or off of the elevator so that the simulator software works correctly. Therefore, you

must call this function appropriately to allow the simulation to keep proper track of the number of passengers waiting for the elevator on each floor, as well as the number of passengers inside the elevator.

Additionally, to use the ESS software, there is a sixth function, `initialize()`, which you must call upon before the simulation can be run.

Note that there is one minor change from the functions originally described in Assignment 1. Originally, the `buttonPressedInside()` function returned only a single integer value representing the floor number requested by a passenger in the elevator. In the slightly-modified version of this code, `buttonPressedInside()` can be called repeatedly to return as many button pushes as have been made inside the elevator. Each integer value returned still represents one floor request. But since there may be more than one request, the function can now be called repeatedly. A `-1` returned by this function indicates that there are no more outstanding floor requests. Therefore, where your pseudocode originally indicated a single call to `buttonPressedInside()`, you can now replace this with a loop that downloads integers from this function until a `-1` is returned.

Each of these six functions is listed as `extern` in the `ESS.h` file, which you should `#include` at the top of the program that implements your pseudocode as C code.

Note that the ESS software comes in 32- and 64-bit flavours, depending on which version of Ubuntu you had loaded at the start of the semester. The `ESS.h` file works with either version, of course, since it is just a text file. But you must choose which of the two versions of `ESSxx` will work with your OS/architecture (where *xx* is 32 or 64 depending on which compiler you selected when you installed Ubuntu.)

One additional feature is available for your use as part of the ESS simulator. `ESS.h` contains an `extern` integer value, `repeat`. If set to a non-zero value, this disables the `srand()` call in `ESSxx.c`, so that the elevator simulator generates the same output each time it runs. Use this for debugging purposes, and set it to a non-zero value when you feel your code is ready to handle 'real-world situations', where the actual signals generated by passengers are somewhat more irregular and unpredictable.

Note that the ESS software has been tested under a very limited set of circumstances, those which I consider to be appropriate use, but which you may not; therefore, treat this as a 'Beta' version which may exhibit unexpected bugs/features depending on how you use/abuse it. Please report any truly unexpected behaviours which you encounter (i.e. anything not described in the description of the five standard functions listed above) and check Canvas for periodic updates to this software, which will be corrected as *legitimate* problems arise; it is expected that there will be a few. The first version of `ESSxx` will be released April 2nd, and you can expect updates if and when bugs are encountered.

There is one feature of the `ESSxx` software which you may think is a bug, but which is not. If your elevator fails for some reason, then an ERROR is flagged (a message is printed at the bottom of the elevator simulator.) At this point the code locks up, and you must use *ctl-C* to exit the program. This is a legitimate feature designed to prevent you from continuing execution after your simulation has failed.

The ESSxx software is provided as a (.o) file that you can link with your elevator C code separately. To compile your C code into a working executable:

a. Ensure that ESS.h is included with your elevator C code, which is here referred to as Elevator.c

b. Compile your code as you normally would, but stop short of creating an executable file by using the –c option with gcc. i.e.

   gcc –c Elevator.c -g

   (Do not use the –o option at this stage.) This command has the effect of creating an object file called Elevator.o, which can be linked with the ESS.o file in the next step. (The –g is for debugging purposes only, and can be neglected if you're not using ddd.)

c. Link the ESS object file with your compiled C code (above) using

   gcc –o Elev Elevator.o ESS*xx*.o

   This should produce an executable output file called Elev that can be used with ddd, or can simply be executed at the command line.

When you run this file, you should get output that looks something like the ESS simulation shown above in Figure 1. Please contact me if you have any problems with the output.

III. Documentation

For this assignment you are NOT required to comment your code according to the documentation standard. However, you should follow the guidelines for spacing and indentation: make a proper job of it. *More importantly, you MUST include each line of your pseudocode from Assignment 1 with each equivalent line of your C code. The pseudocode should be made inline with the C code. For example:*

```
if thisfloor != TOP{  //IF FL <> TOP
   thisfloor = Go(UP); // FL<-Go(UP)
   openDoorCloseDoor();//CALL openD…
}  etc.
```

There may be places where several lines of pseudocode can be implemented in one line of C code, and vice versa: that's okay. However, you must list each of the lines of pseudocode that you originally submitted in Assignment 1 in your Assignment 4 comments. Note any discrepancies, and the reasons for these additions/deletions.

IV. Marks/Evaluation

The purpose of this assignment, in conjunction with Assignment 1, was (1) to test your ability to think through a problem and sequence instructions in the correct order, *in advance of actual coding*; and (2) use pseudocode for that purpose. For each significant departure from your original plan, you will be docked 10%. Since the ultimate standard by which your finished code is judged is its ability to mimic the actual operation of an elevator, you will be docked marks for what is effectively poor elevator operation. In other words, if your C code follows your pseudocode exactly but produces unusual behavior, you will lose marks. But if you correct this behavior to make the elevator function the way an elevator should function, you'll lose 10% for each major departure from your pseudocode.

As a reminder, here's the description of what constitutes 'proper elevator operating behavior', which your pseudocode was supposed to anticipate, according to Assignment 1:

"1. If your elevator is on the TOP floor, it cannot go UP under any circumstances;

2. If your elevator is on the BASEMENT floor, it cannot go down under any circumstances;

3. The only time the elevator STOPs is when there are no button presses inside the elevator and no UP or DOWN requests from people waiting on the various floors of the building;

4. The default 'location' for the elevator, when there are no requests, should be the BASEMENT;

5. An elevator going UP from, say, the 5th floor to the 9th floor does not stop at the 6th floor just because there is a DOWN request at that floor;

6. Similarly, when the elevator goes DOWN from say the 4th floor to the GROUND floor, it does not stop for an UP request on the 3rd floor;

7. Passengers of the elevator do not always need to go to or from the GROUND floor; passengers may travel from one floor to another within the building;

8. Passengers make mistakes; they get on an UP-going elevator and press the button for a lower floor even though the elevator will continue to rise. Similarly, someone waiting on a floor may press the DOWN button, and then decide to take the staircase instead."

If your code was poorly planned, you'll need to decide how best to fix its operation so as to minimize the impact on your overall mark. Again, you must note any differences between the code you planned in Assignment 1 and the code you've actually produced in this assignment, in your comments.

As stated in Assignment 1, this assignment is worth 6.5% of your total mark, with almost one third of that mark being due to the pseudocode you submitted at the start of the semester.

Requirement	Mark
Used the pseudocode you submitted February 1st. * This must be submitted inline with your C program code.	8
Code implements the actual functioning of an elevator, according to the description given in Assignment 1.	11
Formatted correctly, i.e. spaced and indented properly according to the documentation standard	4
Clarity of code: terse is good, but not if applied to the point that your code becomes obscure	3
Total:	26

***REMINDER: If you did not submit pseudocode for this assignment prior to the start of February as required, then you automatically receive '0' for the pseudocode portion of this assignment.**

V. Submission Guidelines

Your C code should be submitted online to the dropbox in Canvas, on or before midnight, April 17th. Late submissions will be penalized according to the schedule outlined in Module 0, Slide 20, 'Late Assignments and Labs.' If there are any problems with Canvas, please ship your code to me at houtmad@algonquincollege.com.

When submitting, please be sure to zip your C file with the following name:

`Assignment4_YourLastName_YourFirstName.zip`