Assignment 1

Elevator Pseudocode

Due: February 1, 2015
before midnight

Purpose:

In this assignment you will use pseudocode (or PDL—Program Description Language) to map out the logic for the operation of an elevator. Most programs in the real world *do not* have solutions that can be found on the internet, nor are there code recipes available for every possible situation you'd ever encounter: someone has to figure out how the pieces of the program will fit together *before* that code can be written. This assignment will expose you to the side of programming in which virtually none of the work takes place in the computer; almost all of it happens inside your head, or on a piece of scrap paper.

(Welcome to the true 'problem-solving' side of programming.)

You will have completed this assignment when you have:

- ☐ Submitted your pseudocode (via Canvas) according to the guidelines provided in this document, in PDF format;
- ☐ Desk-checked your pseudocode using the various testing scenarios described, by walking through your pseudocode and checking its response;
- ☐ Used the 'Algonquin Pseudocode Standard' to format your submission.

Caution:

Do not assume that this assignment is 'easy' and therefore can be left to the last moment. While you may not lose marks now if your algorithm is flawed (I won't be testing the correctness of your logic, just your ability to use pseudocode properly) you will lose marks on Assignment 4, when you will have to implement your Assignment 1 pseudocode in C, according to the specifications given at that time.

Proper program design takes time, most of it in the planning stage (with or without pseudocode!). Do NOT leave this assignment to the last minute.

Assignment 1

Elevator Pseudocode

I. Introduction

The correct operation of an elevator is something we all take for granted. However, elevators do not program themselves; someone has to figure out what happens when a button—either a button inside the elevator itself or one of the buttons in the hall way in front of the elevator doors on each floor—gets pushed.

The consequences of faulty elevator design can be fatal: a man was killed in the Lord Elgin hotel on Elgin St. about 25 years ago by faulty elevator operation. Moreover, the companies that manufacture elevators cannot afford to test them using poorly-designed logic: the code has to work right the first time, therefore it needs to be properly simulated in software before it is used on a functioning piece of hardware. (See: http://hardware.slashdot.org/story/12/12/01/1953200/one-cool-day-job-building-algorithms-for-elevators).

In this assignment you are asked to provide the pseudocode that will control the operation of a working elevator. This pseudocode is independent of any language, therefore the amount of C (or Java, C++, Cobol, etc.) that you know at this point is not relevant to the actual completion of this project. Note however that while the algorithm to control the operation of a single elevator is not a lengthy one, that doesn't mean it doesn't require some thought to implement correctly. Therefore, you cannot afford to wait to the last minute to start this assignment; you need to begin thinking about this well in advance of any actual programming stage.

II. Description of the problem to be solved

In Assignment 4 you will be provided with a small library of C code that simulates the hardware side of an elevator operation. This will take the form of functions that your code will call that indicate if a button has been pushed either inside the elevator by a passenger or outside the elevator door on each floor (see next page). Additionally, this library will contain a simple text-based simulator designed to graphically display the operation of an elevator, including the number of people inside the elevator and the number of people waiting on each floor. (Note that this information will be provided to assist you in determining if something is wrong with your C code; as with a real elevator, the number of people waiting to either take or exit the elevator is not relevant to its actual operation—the elevator neither knows nor cares how many people are waiting inside or outside the elevator.)

In the current assignment, you are responsible for writing the actual pseudocode that determines the operation of the elevator in a building. You should assume the following:

- There is only one elevator in operation in this building;

- The number of floors in the building in which the elevator operates is equal to a value defined as FLOORS. (You can assume this value is preset; as with the other constant values described below in capitals, you do not need to define this value in your pseudocode.);

- All of the floors are identified by number. You should assume that the following integer constants have been defined:
    - BASEMENT is always floor 0;
    - GROUND is floor `1`;
    - TOP = FLOOR-`1`;
    - All other floors are identified by an integer value

- Additionally, you should assume that the UP constant is numerically equal to `1`, the DOWN constant is numerically equal to -`1`, and STOP is equal to 0.

- On each floor, there are two buttons: UP and DOWN (…with two exceptions. There's no UP button on the TOP floor, and no DOWN button in the BASEMENT.)

- Inside each elevator, there are FLOORS number of buttons, everything from BASEMENT to TOP.

At each floor, your code (optionally) can call on any of the following functions. These represent the I/O instructions passed to the elevator hardware by the passengers that have pushed buttons either inside or outside the elevator:

- `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;

- `buttonPressedOutsideGoingUp()`, which simulates someone pressing the UP button on a particular floor. The function returns an array consisting of FLOOR number of 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 returned by this function will be set to a non-zero value;

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

- The function `openDoorCloseDoor()` effectively does exactly what it's name suggests. The function takes no values and returns no values. But it is necessary that the code representing the elevator hardware knows that the passengers can get on to and off of the elevator. Therefore, you must call this function to allow the simulation to keep proper track of the number of passengers waiting for the elevator on each floor, as well as inside the elevator.

One final function needs to the called, not optionally, *but on each floor*. This function is called `Go()`, and you will pass the value UP , DOWN, or STOP into it. The value returned is the actual floor number, an integer. While the first four functions are optional—you decide if one, two, three or all four need to be called at each floor—the `Go()` function is not optional; it must be called at each floor.

Note: YOU DO NOT NEED TO WRITE THE PSEUDOCODE FOR ANY OF THESE FUNCTIONS: ASSUME THEY ALREADY EXIT AND RETURN THE VALUES STATED ABOVE

Your pseudocode must be written such that, given the information that has been made available via the above functions, *on each floor your elevator 'decides' on one of following three courses of action*:

1. It goes up a floor

2. It goes down one floor
3. It stays put, i.e. STOPs on the current floor

To start your pseudocode, assume that all the logic happens inside the following loop:

```
WHILE (POWERGOOD)

   …

END WHILE
```

where you may assume that POWERGOOD is a constant always equal to a true value. For the purpose of your pseudocode, you may need to *preset* certain values (before the WHILE loop) that determine, e.g. which default direction the elevator is heading at startup; that's okay.

III. Testing

Your code should be properly desk-checked—checked on paper by walking through the following scenarios—to ensure proper operation. In other words, you need to walk through your pseudocode 'in your head' to test that it functions as planned. (Most of the following should be obvious from your everyday experience using elevators.) The following represent the minimum number of situations which you need to test for:

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.

Furthermore, note that elevators do not keep track of the number of occupants (although they may keep track of the maximum weight of all the passengers—something you need *not* be concerned with). Therefore, your code should not attempt to keep track of the number of passengers, as this is immaterial to the actual functioning of the device.

IV. Marks/Evaluation

The purpose of this assignment is (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. (Note that you may also use a flowchart to help you organize your thinking about this problem. However, you will not be marked on any flowcharts which you submit.)

Your pseudocode will be marked as follows:

Requirement	Mark
Used pseudocode according to the 'Algonquin Standard'	9
Spacing and indentation for pseudocode follows the documentation standard, i.e. single line spacing between related functional blocks, indentation to signal start of a new code block, comment blocks, etc.	7
Used functions described above correctly, i.e. for the purpose described	4
Total:	20

You will *not* be marked on the logical correctness of your code at this stage; that will be done in Assignment 4, when you will need to implement the pseudocode from this assignment in C. At that point, your C code *must* correspond to the pseudocode you have done in this assignment. Each point of departure between your pseudocode and the actual C code you present for Assignment 4 will result in a 10% deduction from your overall Assignment 4 mark.

This assignment is worth 2.5% of your total grade (out of 20% total available for assignments). However, since the results of this assignment feed directly into Assignment 4, failure to supply Assignment 1 pseudocode prior to the February 1st deadline will also affect your mark for Assignment 4. Specifically, assignment 4 will be worth 6.5% of the total assignment grade. Pseudocode handed in prior to the February 1st deadline accounts for 2% of that mark--almost 1/3 of the assignment 4 mark. Hence failure to supply elevator pseudocode before February 1st midnight will not only cost you 2.5% for Assignment 1, but it will also cost you an additional 2% for Assignment 4. So think of Assignments 1 and 4 as being two pieces of the same puzzle: together they are worth 9% of the total mark. *Half of that mark is based on having the pseudocode supplied prior to the deadline.*

Therefore, it is in your best interest to put considerable forethought into this project; the effort you put in to Assignment 1 gets you marks both now, in February, and in the future, at the start of April when Assignment 4 will be due (and when, incidentally, you will be *very* busy, and very regretful if you didn't spend enough time desk-checking your pseudocode at the start of the semester, when your time was a bit freer.)

Note that pseudocode examples may be found at the start of Module 3. Knowledge of C code—or any specific programming language for that matter— is <u>not</u> required to understand how pseudocode is actually used in these Examples.

V. Submission Guidelines

Your pseudocode should be submitted in PDF format on or before the deadline, February 1st, before midnight, to the dropbox set up in Canvas. 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 all files (even if there's only one PDF file in your zip). Your zip file *must* have the following name:

`Assignment1_YourLastName_YourFirstName.zip`

Note that you are responsible for keeping a copy of your Assignment 1 files on hand, as you'll be using this pseudocode again in Assignment 4.

Pseudocode summary, from Module 3, Slide 15.

Input/Output	**Decision Statement**	**Loop Statement**
GET (i.e. input)	**IF…ENDIF**	**FOR…ENDFOR**
PUT (i.e. output)	**IF…ELSE…ENDIF**	**WHILE ()…ENDWHILE**
Functions	**SELECT CASE… CASE…ENDCASE… DEFAULT… ENDSELECT**	**DO … UNTIL ()**
FUNCTION Name **(…)** …**RETURN**… …**ENDFUNCTION** **CALL** Name		**Structure**
		STRUCT… ENDSTRUCT
Assignment	**Comparison**	**Logical Operators**
← (i.e. <-)	**=** **<>** (i.e. not equals) **>=** **<=** **>** **<**	**AND** **OR** **XOR** **NOT**
Mathematical Operators		
+ - / * MOD SIN() etc.		

Note that Module 3 contains pseudocode examples which do not require *any* knowledge of C. Therefore, you should feel free to explore these slides and use them regardless of whether or not we've covered the C code developed as a result of the pseudocode given in these examples.

To Repeat: pseudocode is just a rough way to write out your ideas about how code will be sequenced. C depends very heavily on this thinking, but the opposite is not true: thinking about code and determining the correct sequence is *not* a function of how much C you know.

Addenda (added Jan 19th)

1. The `Go()` function returns the floor number *after* the elevator has moved UP or DOWN, but before the doors open.
2. You may create any variables in your pseudocode you find necessary, e.g. `direction`, `floor`, etc.
3. There is only one 'instance' of an elevator, hence your pseudocode should *not* contain any references like `elevator1.Go(UP)`. This is C, so you can drop the *object.method* notation. Just write `Go(UP)` and skip the object, which is implicit anyway.
4. Your pseudocode is one giant algorithm, and so, while you *are* required to follow the documentation standard in your submission for Assignment 1—including adding a header box containing the PURPOSE, DATE, etc.—you are not required to list an ALGORITHM, since the pseudocode itself *is* that algorithm.