Project 2: Elevator Simulator Part B Help Page

This page gives suggestions and help with Project 2 Part B. It is meant to answer some of the questions I have gotten repeatedly during the project.

Resources

Please refer to this section for resources to help complete the project.

- Project Main Page
- Project Part B
- The ElevatorLib Library Documentation

Global Variables

I have no problem with using global variables in this program. If you declare a variable before the main function, it is global and available to all functions in your program. This makes it easier in this project to move code to functions. For example, you can make state global like this:

```
int state = Idle;     /* state is global since it is before main */

int main()
{
    printf("Elevator Startup\n");
    ElevatorStartup();
```

If you do so, state is available to every function, so your control loop can be as simple as:

```
    while(IsElevatorRunning())
    {
        switch(state)
        {
        case Idle:
            StateIdle();
            break;

        case Moving:
            StateMoving();
            break;

        case OpenDoor:
            StateOpenDoor();
            break;

        case ClosingDoor:
            StateClosingDoor();
            break;
        }

    }
```

Because state is global, I do not have to pass it to each of the state functions.

States and State Machines

I have seen several people who have decided to try to complete the project without using states or a state machine. Let me make it clear: you will not get it done! I repeat: YOU WILL NOT GET IT DONE! I will not be sympathetic and we surely won't give you much credit when grading. This project is not that difficult if you just think about the four states (you may use more if you wish) and what you should do in each state.

In my solution I have four states: Idle, Moving, OpenDoor, and ClosingDoor. Some have a state for moving up and another for moving down. That is reasonable. My entire control loop becomes just this code:

```
    while(IsElevatorRunning())
    {
        switch(state)
        {
        case Idle:
            StateIdle();
            break;

        case Moving:
            StateMoving();
            break;

        case OpenDoor:
            StateOpenDoor();
            break;

        case ClosingDoor:
            StateClosingDoor();
            break;
        }

    }
```

If you are wondering where the PID controller is, it's in StateMoving(). After all, it only applies when the elevator is moving, right?

ButtonPressed is NOT a state. It's an event.

WhatFloorToGoTo and *WhatFloorToGoToInThisDirection*

I've gotten lots of questions about *WhatFloorToGoTo* and *WhatFloorToGoToInThisDirection*. I provided these two functions to make the project easy! They do nearly all of the logic for you. I have told you what function to use in each of the two situations. It should be clear.

So, what does WhatFloorToGoToInThisDirection do? It should be completely clear from the documentation, but let me put it this way: You are in the elevator and controlling it manually. You need to know what floor to slow down and stop at. So, as the elevator is moving, you repeatedly yell: "I'm going up. What floor am I going to?" Someone looks at the state of the buttons and yells back: "3". You look as where you are: are you within 1/2 a floor of floor 3? If so, you need to slow down. Have you reached floor 3? If so, you need to stop. That's all there is to. You just keep yelling that question over and over again. If someone presses a button, the answer may change to 2 instead of 3.

So, what does WhatFloorToGoTo do? The elevator is stopped with the doors closed. You yell "WhatFloorToGoTo?" If no button has been pressed, someone yells back: -1. You go "that's not a floor, I'm not going there" and you do nothing. If they yell "floor 2", you decide, should I go up or should I go down and you turn off the brake, turn on the motor and go.

Someone asked for code examples. Here are a couple:

```
void StateIdle()
{
    int gotoFloor;

    gotoFloor = WhatFloorToGoTo(goingUp);
    if(gotoFloor > 0)
    {
        // We need to go to the requesting floor
        printf("Request for floor %d\n", gotoFloor);
```

```
void StateMoving()
{
    // Where are we going?
    int gotoFloor = WhatFloorToGoToInThisDirection(goingUp);

    // How far are we away from the floor we are going to?
    double distance = fabs(GetFloor() - gotoFloor);
```

These examples just about give away the entire project. There is not much more code than this.

The Control Loop

The control loop is this code:

```
    while(IsElevatorRunning())
    {
        switch(state)
        {
        case Idle:
            StateIdle();
            break;

        case Moving:
            StateMoving();
            break;

        case OpenDoor:
            StateOpenDoor();
            break;

        case ClosingDoor:
            StateClosingDoor();
            break;
        }

    }
```

Each of these functions must do what it needs to do and return. Do not write code in a control loop that stops and waits for something. For example, don't do this:

```
    /* Don't do this: it will not work!!! */
    while(WhatFloorToGoTo(goingUp) == -1)
    {
        /* Waiting for a button to be pressed */

    }
```

In any control system the control loop needs to run continuously. In this program the function IsElevatorRunning() has to be called regularly or the program does not work. In the Garage assignment, did we wait for the button to be pressed? Of course not. We had a state that continuously checked to see if the button was pressed.

Wrong Things To Do

This is not a difficult project. If you are having difficulty, we will try to help all we can. I do not feel the class should have difficulty with this project.

But, it is very easy to make this project just about impossible to complete. And, for some reason many students seem to want to do that even when I warn them of the difficulties. Here are some things you can do that will cost you enormous amounts of time and will likely lead to a project that does not work:

a) Not using a state machine. This project is a direct application of Lecture 10. It is meant to be an application of that material. It is a simple state machine problem with only four states. If you try to do it with "if" statements and try to think of a bunch of cases you will spend a lot of time, generate a lot of code, and not get done.

b) Putting all of the code in one function. Breaking a program into functions makes it simpler. Putting all of the code in one function makes it harder to program and understand. If you don't believe me, that's fine. It's your time to waste.

CSE 251