Project 2: Elevator Simulator

Project 2 is due in 2 parts.

Part A is due (Monday Class) 03-31-14 AND (Friday Class) 03-28-14.
Part B is due (Monday Class) 04-14-14 AND (Friday Class) 04-10-14

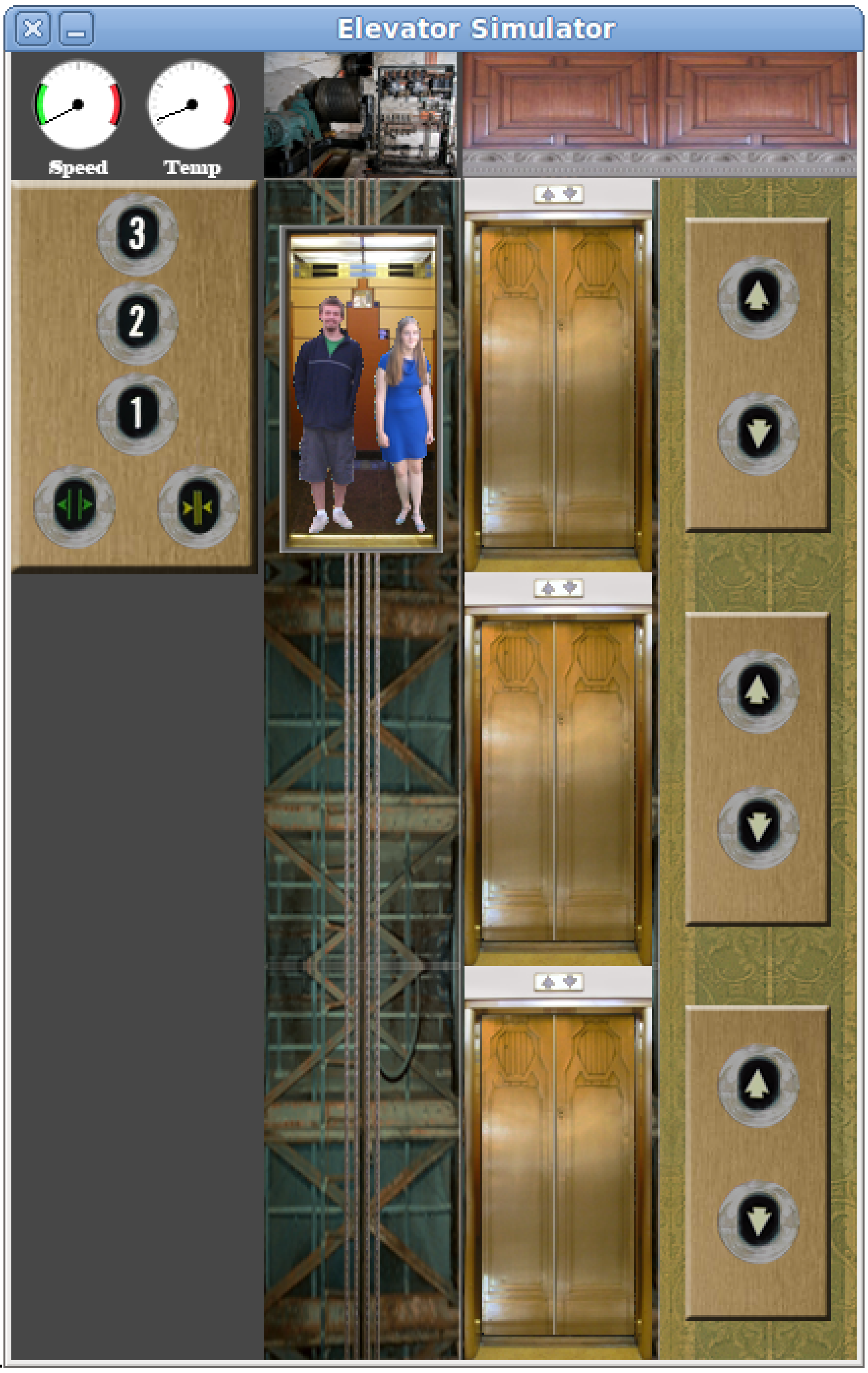
Resources

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

- The ElevatorLib Library Documentation
- Project Part A
- Project Part B

Obtaining the Project

For Project 2 you will be working with an Elevator Simulator. Below is an illustration of the Elevator Simulator:



.

It will be your responsibility to write the control software for the elevator. You need to control the motor, brakes, and doors. This is an example of a *controls application*, where you will be responsible for making a hardware system work correctly through code you write.

> Control systems like this control real-world system with real-world people involved. If your code does not work correctly, you may kill someone or even burn the building down.

This project must be compiled and executed on a machine with an X-Server and wxWidgets. This is true for the Linux lab machines. I make no guarantee about personal machines and it is unlikely that you can work on this project remotely given the animation it uses. It is best to work on this project in the lab.

First, create a directory named project2 in your Linux account. Remember, you use the mkdir command to create a directory. You can put this under cse251 or under the root directory, wherever you want to put it. Then download the file Elevator.tar.gz into that directory. You can right click on the link in the browser and choose Save Link As... to save the file.

Then, using a terminal window in the directory where you put Elevator.tar.gz, do this command:

```
tar xvzf Elevator.tar.gz
```

This will create a directory called Elevator. Change into that directory using cd.

Compiling and Running the Project

This project uses a make file to automate the building process. Thus, we do not use the gcc command directly. Instead, we use a Makefile to automate the process. Since some of the machines we use are still 32-bit machines, and others (namely the remote CSE machines arctic, pacific, black, etc) are 64-bit machines, there are two different commands that can be used to compile the project,

```
make elevator32
```

builds the 32-bit version of the code (this should be used in the lab, and possibly on personal computers), while

```
make elevator64
```

builds the 64-bit version of the code (this should be used on the remote logon machine and possibly on personal computers).

You should see output something like this:

```
cbowen@ubuntu:~/test2/Elevator$ make
gcc -c -o elevator.o elevator.c
g++ -o elevator elevator.o -lm libElevatorLib.a `wx-config --libs`
```

If your output does not look like this, look at the error messages. If the compiler is complaining about i686 or i386 code, then you are trying the 32-bit build on a 64-bit machine. If you get a significant amount of gibberish, then you are trying the 64-bit build on a 32-bit machine. As always, reading the error messages is helpful.

If you get an error about elevator.o being of the wrong type, run

```
make clean
```

which will remove any old compiled files that may be around when you switch machines. Also, running

```
make
```

with no arguments prints a summary of these commands for you

This is building your application for you. Whenever you need to recompile, just type: "make elevator32" or "make elevator64". We will be covering make files later.

To run the program for the first time, type:

```
./elevator
```

The program should display the elevator simulator. It will not be functional, because you have not written the code to make it functional, yet. You can exit the program by clicking the box with an X to close the window.

In the simulator there is one elevator. The section of the window with doors is the elevator doors for each of the three floors. To the right of the doors are the call buttons that call the elevator to that floor. A passenger presses the up button to indicate they want to ride the elevator up.

To the left of the doors we see the elevator in its shaft. There is only one elevator. This is the same elevator as the one you will see behind the doors. It is drawn twice so we can see both how it looks to users and what is going on inside the elevator shaft.

To the left of the elevator is the panel inside the elevator car. This is where you can press buttons to choose a floor to go do. There are also buttons to open and close the doors. The panel moves with the elevator car.

Above the elevator panel are two meters that indicate the speed of the elevator and the temperature of the motor. The speed meter has a green area. You can only apply the brake to stop the elevator if the speed is in this area (up to 0.33 meters per second). The red area indicates speeds too fast for the elevator design. Allowing the speed to go into the red area means your elevator is going too fast and may break.

The temperature meter tells the current temperature of the motor. The elevator is controlled by a DC motor. As you use it, it will heat up. If you put too much work on the motor it will get so hot it may catch file. Temperatures in the red zone indicate the motor is overheating and may burn your building down.

You can see the motor room in the small window above the elevator shaft.

There may occasionally be some jerkiness to the animation in the Elevator Simulator. This is mostly due to scheduling in the Linux operating system and is likely not the fault of your program.

Reference Solution

There is an example solution for the project included in executable form. If you run the program *elevator-solution* (*elevator-solution-64* on 64-bit machines), you can see how it is supposed to work when the entire project (Parts A and B) is complete. Run the reference solution with the command:

```
./elevator-solution
```

Please let me know if you find any bugs in the reference solution.

Some Exercises To Get Started

All of the code you write for this project will be contained in the file **elevator.c**. I have provided an initial version of this file to get you started. The initial version is:

```
#include <stdio.h>
#include <stdlib.h>

#include "ElevatorLib.h"

/*
 * Name :        <insert name here>
 * Description : Project 2 - The elevator controller
 */

/*
 * Name :        main()
 * Description : Program entry point.
 */
int main()
{
    /*
     * This call starts the elevator system running
     */
    printf("Elevator Startup\n");
    ElevatorStartup();

    /*
     * This loop runs until we shut the elevator system down
     * by closing the window it runs in.
     */
    while(IsElevatorRunning())
    {

    }

    /*
     * This call shuts down the elevator system
     */
    printf("Elevator Shutdown\n");
    ElevatorShutdown();
    return 0;
}
```

Please read the comments to tell what the functions all do in this code. Complete documentation for all of the functions available to use in the Elevator Simulator are available in the Resources section.

This program uses a *control loop*. This is a continuous loop that runs as long as your system is active. In the program above the control loop is this:

```
    /*
     * This loop runs until we shut the elevator system down
     * by closing the window it runs in.
     */
    while(IsElevatorRunning())
    {

    }
```

This loop runs at a rate of 1000 iterations per second. So, every time the body of the loop executes, 0.001 seconds have elapsed.

Do not put code in the main function before the call to ElevatorStartup or after the call to ElevatorShutdown. The simulation does not exist before started or after shutdown.

Turning Off the Brake

When you start the simulation, the elevator motor power is set to zero and the brake is on. The means the elevator stays on the first floor. To turn off the brake, we use the SetBrake function. Add this line of code before the control loop:

```
    SetBrake(false);
```

Compile and run the program (remember, type "make" to compile). You may be surprised to see that the elevator goes up. Elevators have counterweights. When the elevator goes up, the counterweight goes down. To be efficient, the counterweight weights the same as the car plus some average number of passengers. The car masses 453 kilograms. The counterweight is 679 kilograms. So, with no brake the car goes up.

The simulator has the ability to change the number of passengers randomly (up to 3). Add this line of code right after the SetBrake call:

```
    ChangeLoading();
```

Now, each time you run the program you will get a different loading. Try several runs to see what happens with different numbers of passengers. It's a good idea to leave this call in so you can test with different passenger loads.

You can also use this call to put the maximum load in the elevator (3 passengers):

```
    SetLoading(7);
```

Turning On the Motor

Now add this line of code right after the SetBreak call to turn on the motor with maximum power in the up direction:

```
    SetMotorPower(1);
```

When you run this you will see that the speed is off the scale. We're clearly running the elevator too fast. When the elevator gets to the top, it can't go anywhere, so it stops. The motor is then stalled. It quickly heats up and eventually catches on fire. Note that the same thing will happen if you start the motor with the brake on.

Going Up and Going Down

Generally elevators travel up until they have serviced all passengers in that direction, then they go down. We are going to add some code to handle this decision. Add this variable to the main function:

```
    bool goingUp = true;
```

Now, put this code inside the control loop:

```
        if(goingUp && GetFloor() > 3)
        {
            /* We are above the third floor. Reverse direction */
            goingUp = false;
            SetMotorPower(-1);
            ChangeLoading();
        }
```

This will go to the top of the shaft and then go back down. It will then get stuck, of course. I'll let you add the code for the other direction so the elevator goes back and forth. You need to start looking in the documentation for ElevatorLib to understand what each of the functions do and find functions that do what you need to do for the project.

I added the call to change the loading so we can tell what the elevator does with different loadings. This will be useful for Part A of the project.

Why do we test for goingUp and GetFloor() > 3? Try changing the test to this:

```
        if(GetFloor() > 3)
```

Do you understand what it is doing different? Remember: a) the control loop runs continuously and b) the elevator does not instantly change direction when we change the motor power. It has mass and has to be deccelerated to zero and accelerated in the other direction.

After you have seen what this does, restore the line to what it was before:

```
        if(goingUp && GetFloor() > 3)
```

You may now proceed to Project 2 Part A.

CSE 251