Program 2 {August 18, 2014} 40 Points

Simulated Robot Movement in the RoboMall

Due: Monday, September 15, 2014 at 11:59 p.m.

This assignment emphasizes the use of ***arrays*** in C and provides practice with manipulating subscripts and abstracting with two and three-dimensional arrays. You may use ***structs*** in this assignment but it is possible to complete this program successfully only using ***arrays.*** **Note** – you may NOT use vectors to complete this assignment. However, as this assignment comes early in the course, students can keep the building array and the current robot location as **global variables**. However, this does **NOT** preclude the expectation that this program will maintain good control structure consisting of a number of small functions and routines where appropriate parameters are passed.

Your task is to create a simulator to simulate the motion of a **sequential** series of robots that move on two floors of **RoboMall** delivering objects to a series of stores. The last two pages of this assignment provide a diagram of the floor plans for the First and Second Floors of the RoboMall.

Stores are indicated with an '**S**' on the floor plans. ALL stores are identified by a three-dimensional coordinate location (**r, c, f**) where **r** indicates the row, **c** indicates the column and **f** indicates the floor where a store is located. Note – the first row, the first column and the First Floor are all located at 0 in their respective dimension of the coordinate system.

ALL robots enter and leave the RoboMall at entrance A1 (location (8, 16, 0) on the First Floor. Each subsequent robot enters the mall EXACTLY one time unit after the previous robot leaves the Mall. The simulation begins at time **0** with robot, R1, located at A1. Once a robot arrives at the location of its next targeted store, the robot remains at that spot **×** time units to simulate dropping off objects where **×** equals the floor number for the stores in the RoboMall. For example, if the robot's target store is located at (4, 6, 0), after the robot reaches the store, it remains at that spot one additional time unit because this store is on the First Floor. However, if the robot's target store is located at (12, 10, 1), after the robot reaches the store, it remains at that spot two additional units because this store is on the Second Floor.

Robot motion on both floors is quite restrictive. To reduce potential robot collisions, the floor plans indicate one-way paths (via an arrow) that a robot can travel on each floor. Two way paths have no arrows. Note, to reach internal mall stores, the robot must travel one way on a two-way path to get into the store and then travel the opposite way on a two-way path to leave that same store. Robots can ONLY move on the marked paths and into and out of stores.

This program provides a simulation of the time it takes all the robots to complete their assigned deliveries. It takes **1 time unit** for the robot to move one square on a given floor. It takes **1 time unit** to move from a square to the elevator square or from an elevator square to a normal square. It takes **3 time units** to ride the elevator between the two floors.

Assignment Input

The program will read in 'scripted' information for the robot simulations from an ASCII file. To keep it simple, do **NOT** use C file I/O commands. The file should be accessed by using a LINUX command line that **redirects** the input from a file. This enables you to test the program either using a file or using terminal input.

{Think of the scripted input as consisting of subsequent lines of input data}

The program begins by reading in from the first line of input script, the number of robots, n, that will be sequentially simulated. Assume the maximum number of robots per simulation is 9 and the maximum number of stores any robot will visit is 20. The number of robots determines how many more lines of input exist in the script file. The second line of input will read in data of the form:

```
s1  s2  s3 … sn
```

with each input data item corresponding to the number of stores R_i must visit.
For example, given the first two input lines:

```
4
3  8  4  11
```

The simulator would sequentially simulate four robots, R1 to R4 with R1 traveling to 3 stores; R2 going to 8 stores; R3 visiting 4 stores and R4 going to 11 stores.

Next, for each robot, for each store the robot will visit in sequential order, the program will read one line which corresponds to a single store's entry coordinates in the form:

```
xpos ypos zpos
```

where xpos ypos indicate the location on a given floor and zpos indicates the floor. {As in the figures, assume the upper left corner position on a floor is (0, 0).}

Continuing the example from above for R1 who must visit 3 stores, an example of three input lines is:

```
10   4  0
 6  12  1
 4   6  0
```

This defines the journey of R1 who must visit one store on the First Floor, one store on the Second floor and finally one more store back on the First Floor before exiting the Mall at A1. Note, each

robot's trip is strictly sequential based on the input lines. Namely, a robot may have to return to the First Floor after being on the Second Floor to visit the next store on its travel list.

Robot Motion Planning

Beginning from A1, its entry point to the RoboMall on the First Floor, each robot stops at its assigned stores in increasing sequential order to deliver objects by obeying the travel rules specified on the floor plans. Your program should use the **shortest path** for the robot between any two successive stores on its list. To go to a different floor, the robot uses the elevator located at the E square on each floor. Thus, a robot's life begins at its initial entry location, moves to deliver objects to each of the stores on its travel list. After the robot has delivered objects to the last store on its travel list, it finishes by exiting the RoboMall at A1. At that point, the i^{th} robot is **instantly eliminated** from the simulation. One time unit later, the $(i+1)^{th}$ robot is placed at A1 and the simulation continues. When the last robot completes its task and arrives at A1, the simulation ends.

Assignment Output

For each robot in the simulation, the program should print out a log of the robot's significant events that includes at a minimum the following type of information (but not necessarily in this format) :

robot i begins at entry location A1 at time t.
robot i arrives at store s (xpos, ypos) on the zpos+1 Floor at time t.
robot i got on the elevator at time t.
robot i got off the elevator at time t.
robot i left the simulation at time t.

Additionally, interleaved into the output is a **periodic** output identifying the current robot showing its current location on a two-dimensional map for the robot's current floor. The correct floor map should be printed out every 25 time units (i.e., for time = 0, 25, 50, 100, 125, …). You can earn one extra credit point, by having the map show ALL 25 robot locations since the last map printout.

After the simulation ends, print out a summary of each robot's time in the simulation.
For example:

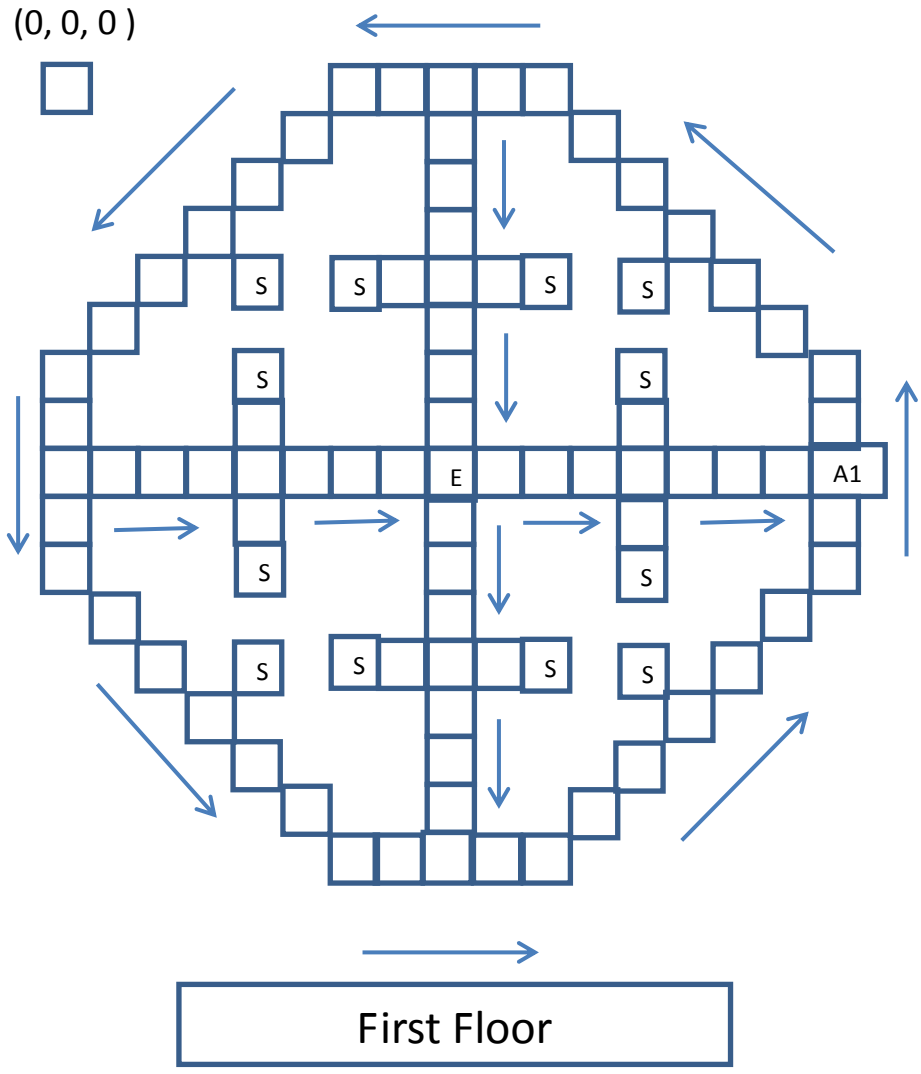
Robot	Start Time	Finish Time
1	0	65
2	66	134
…		
n	528	749

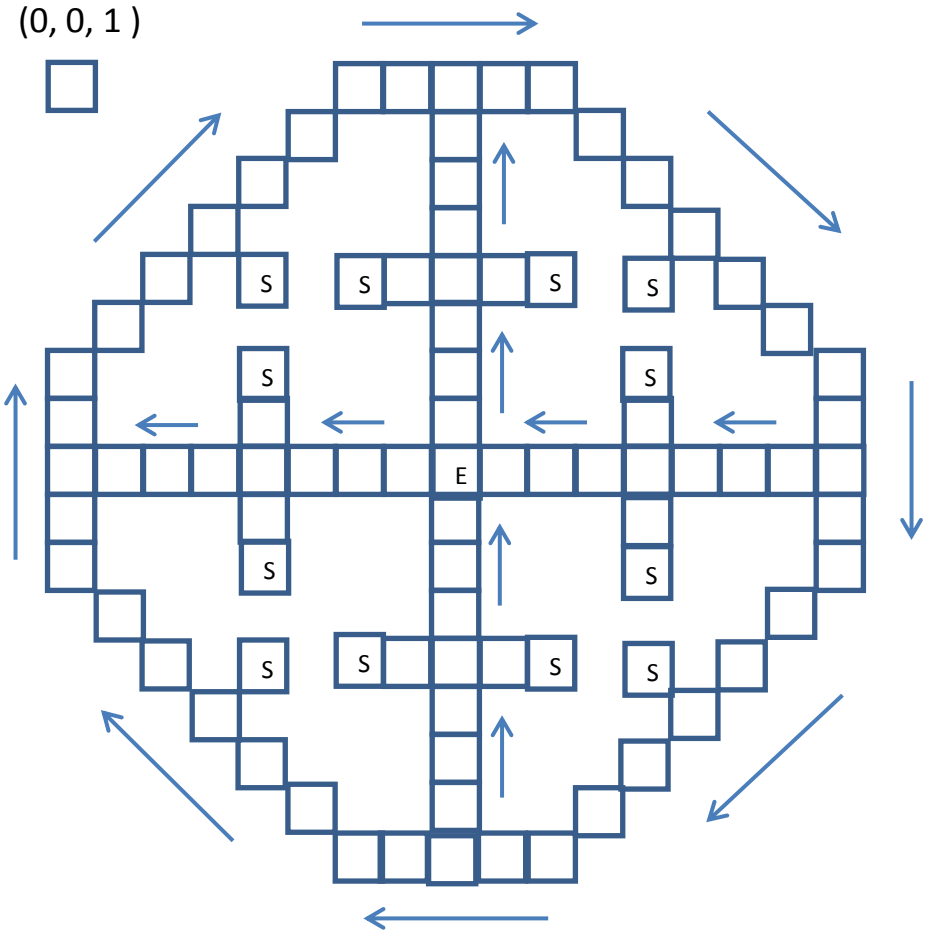
Program Development Hints

Remember this program will be graded taking into account good organization and the use of multiple functions and routines. It is highly recommended that students utilize a debugger and/or include a conditional define debug option that can easily be turned on and off. Writing a print current robot location function, should be one of the first things you do! While debugging your program, you should print out **other significant events** to assist you in tracking the robots' movements.

What to turn in for Program 2 (prog2)

Turn in your assignment using the ***turnin*** program on the CCC machines. You should turn in a tarred file that includes your source code, a make file, a README file and a sample output file. The make file should include the ability to cleanup leftover output and intermediate files between compile and execution cycles. The README file should provide any information to help the TA or SA test your program and evaluate your output files for grading purposes. If you are submitting a partially working program, it is CRITICAL that your README file indicate precisely (to the best your knowledge) what is currently working and not working in your program. Failure to do this can seriously limit the partial credit you will receive for this assignment. For example, if your program does not work successfully for a given floor, please indicate this in your README file.


(0, 0, 0)
S
S
S
S
S
S
E
A1
S
S
S
S
S
S
First Floor



Second Floor