Project Design Requirements

Specification Report: Due the March 31.

- Description of the project design:
  - Draft description of the algorithm to solve the problem.
  - Detail flowcharted design.
  - Submit names of team members.
  - Describe the demonstration, the user interface and the expected outputs.

Formal Design

- GUI Design
  - User-friendly data input.
  - Data fields properly initialized?
  - Well formatted output.
- Exception Handling
  - Data edit and validation.
  - Program notifies user if errors occur, re-prompts or terminates under control.
  - File handling
- Code Structure
  - Program should be well commented and easy for humans to read.
  - Organization and Flow of the methods/classes.
  - Method argument signatures and use of return codes.
- Numerical Routines
  - Compartmentalization of the methods.
  - Efficiency of operation and use of system resources.
- Documentation
  - What does this do and how does it accomplish its function.

Project Grading (Shared grade points are to be distributed by team members' consensus)

- Program Correctness – 60%
  - All methods perform to specifications.
  - Adherence to instructed theory.
  - Compiles error free and executes without errors that cause abnormal termination.
  - Handling of error conditions.
  - Efficiency of Algorithm.
- Proper implementation of design – 20%
- Coding Style – 10%
  - Program should be well documented.
  - Code should be clear and read well.
  - Proper use of indentation to highlight flow control
- Project Report and Presentation – 10%

Completed Project – Due on the class meeting prior to the final exam.

- Complete electronic copies only -- including all source code and documentation required to successfully compile, link and execute programming.

Choose one of the following projects:

Elevator Simulation

Problem Statement:
You are to simulate a single car elevator in a 10 floor office building. The purpose of this simulator is to estimate how long it would take to evacuate people from the building under some policy that we have the elevator implement.

Constraints:

- Requests to go DOWN occur at the rate of between 0 and 2 riders per minute per floor. (The Random class and the Math class have methods that will be helpful in generating random numbers.)
- The elevator starts at floor 1 and returns there when there are no riders or requests.
- The elevator must pick up only requesters going in its present direction, unless there are no occupants, in which case it must pick up the longest waiting request in the queue.
- Riders going in the same direction as the elevator but not at the head of the queue can be re-prioritized to the the front of the queue.
- There is no practical limit to the number of riders that can fit into the elevator car.

Initial Conditions:
The elevator rests at base floor 1.
There are no occupants.
These are no requests.

Operation:
Use a random number generator to generate the number of people requesting elevator service. Run for a 1000 requests. A request is the up-down button outside of the elevator.

Data Displays:
For each event and each floor visited show (print):

- the current floor,
- the number of occupants in the car,
- the destination floor,
- the number of riders alighting on a given floor,
- then number of riders debarking at the ground floor,
- the station arrival event (doors opening),
- the station departure event (doors closing),
- the number of pending requests, from people waiting at some floor.

At the end of the simulation show (print):

- The average wait time between requesting an elevator and its arrival to pickup.
- The total number of riders that evacuated the building.
- The total time to evacuate building.

Strategy

You want to create a double-linked queue to maintain the requests to go DOWN. This queue should contain the originating floor (the destination floor is always 1).

Since I am not asking for an application using multiple threads, multiple elevator cars or asynchronous interrupts, you can use a list of requests in which to populate the queue. More requests can be added to the queue when all requests are exhausted. But feel free to create a queue to works asynchronously with interrupts so that it can take requests at any time, like in the real world.

When you pickup a floor of riders, the request is removed from the request queue.

After you have run this simulation, try different strategies and alter the constraints in an attempt to make the elevator run more efficiently.

Restaurant Simulation

Problem Statement:
You are to simulate an N-table restaurant. The purpose of this simulator is to estimate how to best serve your patrons waiting for dinner.

Constraints:
- Patrons arrive at the rate of between 0 and 5 per 15-minute period. (The Random class and the Math class have methods that will be helpful in generating random numbers.)
- The restaurant can seat 20 for dinner at any time.
- Each arriving party is a party of 1 needing 1 table.
- Each diner takes 30 minutes to eat before leaving the table available.
- Patrons seeing a line greater that 15 people leave without waiting for dinner.
- VIPs may arrive for dinner with a 5 percent probability, and are given priority.

Initial Conditions:
All tables are available.
There is no waiting line of customers.

Operation:
Use a random number generator to generate the number of patrons requesting seating.
Run for a 200 arriving patrons.

Data Displays:
For each event show (print):
- the current queue size of waiting patrons,
- the number of seats taken,
- the number of patrons being seated each interval.
- the number of patrons finishing dining each interval.
- the number of patrons leaving before getting in queue.

At the end of the simulation show (print):
- The average wait time between requesting a seat and being seated.
- The total number of patrons that had dinner.
- The total time needed to serve dinner to all.

Strategy

You want to create a double-linked queue to maintain the requests of arriving patrons. This queue should contain the requester-id and the time of arrival.

If at VIP arrives you must prioritize them to the front of the queue.

After you have run this simulation, try different strategies and alter the constraints in an attempt to make the restaurant run more efficiently. What would be your suggestions?