

DMV TRANSACTION PROCESSING

For this assignment, you will write a C++ program that simulates transaction processing and queue management at a typical DMV (Department of Motor Vehicles) office. Assume, for simplicity, that only the following transactions are carried out at the office: new driver's license application, driver's license renewal, new vehicle registration application, and vehicle registration renewal. Priority is given to transactions that take less time to process; as such, priority levels will be assigned to each transaction type. If a window is available to process a transaction and several people are waiting in line, the person with the highest-priority transaction will be served first; within the same priority level, transactions are served on a first-come-first-served basis.

There are other assumptions that apply to this assignment and they are listed below:

- There is one dispatcher window that screens incoming transactions. The screening step takes exactly one minute and does not consider priority level. After screening has been performed, the person waits for an available transaction window. In effect, a person will need to line up twice.
- There are eight transaction windows, which may be opened or closed at any time during the simulation. Initially, all windows are closed.
- Driver's license renewals and registration renewals are transactions that have priority level 1 (highest). New vehicle registration applications have priority level 2, while new driver's license applications have priority level 3 (lowest).
- It takes exactly 5 minutes to process a vehicle registration renewal, 7 minutes to process a driver's license renewal, 10 minutes to process a new vehicle registration, and 20 minutes to process an application for a driver's license.
- If multiple windows are available to carry out a transaction, choose the lowest numbered window.
- A driver may have at most one vehicle registered under his name.

The program will first load from a text file an initial database of licensed drivers and registered vehicles (to be checked against for renewal requests). It will then simulate, in sequence, the commands that appear in another input text file. The commands indicate the entry of people into the office with their respective transactions and the opening and closing of transaction windows. The program will be invoked as follows:

DMV <databasefilename> <commandfilename> <outputfilename>

The simulation program will report (write to the output file), among other things, the time when a transaction was entertained or completed as well as how long a person needed to wait in line before the transaction was completed.

Six types of commands are possible, and these are enumerated below:

```
<time> window open <i>
<time> window close <i>
<time> license apply <soc-security> <name> <birthdate>
<time> license renew <soc-security>
<time> vehicle apply <soc-security> <vehicle-id> <car-details>
<time> vehicle renew <soc-security> <vehicle-id>
```

Single spaces separate each of the elements in a command line. <time> is a non-negative integer; in the command file, the <time> field will occur in sequence (non-decreasing order). <i> is a window number from 1 to 8. <soc-security> is a nine-digit string. <name>, <birthdate>, <vehicle-id> and <car-details> are variable-length strings that may contain letters, numbers, commas, and hyphens.

Note that it is possible to have two commands that contain the same <time> value. In the case where two or more incoming transactions arrive at the same time, the dispatcher window will serve these transactions according to their order in the input file.

The database file is simply a sequence of lines that contain driver information. Two types are possible: lines that describe driver information, and lines that describe both driver and registered vehicle information. Formats for these two possibilities follow:

```
<soc-security> <name> <birthdate>
<soc-security> <name> <birthdate> <vehicle-id> <car-details>
```

Single spaces separate the commands in the database file and the field types and formats are the same as for the command file.

You will need to output the following events as they arise, in the proper sequence:

- When a window is opened (1st command)
- When a window is closed (2nd command)
- When a customer enters the DMV (3rd through 6th commands)
- When a transaction has been screened (include wait time in dispatcher line)
- When a transaction has been entertained (include wait time from time of dispatch)
- When a transaction has been completed (include total wait time from time of entry)

At the end of the simulation, the following summary information should be printed, for each transaction type (license application, license renewal, vehicle application, vehicle renewal):

- Total number of transactions processed
- Average waiting time
- Maximum waiting time

The same information for ALL transactions should also be reported.

Input and output format details will be demonstrated in an example to be posted on the website and discussed in class.

You may assume that no parsing errors exist in the input files. Furthermore, only the following logical errors need to be detected:

- Attempt to renew a license that does not exist in the database
- Attempt to renew vehicle registration for a driver that does not exist in the database
- Attempt to renew vehicle registration for a vehicle that is not associated with the indicated driver

All three errors indicated above occur in the screening step. For each of these cases, the transaction is cancelled but one minute is consumed at the dispatcher window. These transactions should not be included in the statistics shown at the end of the simulation.

This assignment is an exercise in simple discrete event simulation and in implementing queues and binary trees. You are required to use a pointer-based binary search tree to maintain licensed drivers (use social security number as the search key) and an array-based priority queue (using a min-heap) for transaction processing. For the screening window, a regular FIFO queue will suffice. Finally, you will need a separate, time-ordered queue for event processing to carry out the simulation. For details on events and simulation, refer to the handout posted on the course website.

You may implement the screening window queue and the event queue using any method you prefer, as long as they are your own code. You may pattern your code after the code described in the textbook but you may not use any of the standard C++ templates.

A test input file (longer than the example) as well as the corresponding output file will be available on the course website by Monday, 21 July 2003. The program is due on 25 July 2003 11:59 pm. You are to submit a tarred, gzipped file (.tar.gz) containing all source code and a makefile. Make sure your project compiles in the Mandrake Linux environment. **This program will be demonstrated in the presence of one of your TAs.** A sign-up sheet will be available before the due date. As before, refer to the course web site for Curator system instructions, program documentation guidelines, honor code policies, and the pledge sheet that you must include in your main source file.