BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

Second Semester 2013-2014

CS /IS F211 Data Structures and Algorithms

Lab Sheet – 5 [Duration: 150 minutes]

General Instructions for Programming

1. All inputs to the program must be either (a) command line arguments (b) or read from a file (other than stdin). DO NOT READ anything from stdin and DO NOT USE PROMPTS like "Please enter a number …".
2. You are required to write the output to a file (other than stdout) and errors if any to a different output channel (stderr or another file).
3. Use standard C coding conventions for multi-file programs. Separate the following: interfaces of functions (use a ".h" file), data type definitions (use another ".h" file), ADT / algorithm implementation (use a ".c" file), and driver/test code (use another ".c" code). In general, each module has to be written in **separate** c files.
4. All files related to a lab **must** be put inside a single directory by the name of the lab (lab1, lab2, etc.).
5. Valid makefile must be present in the directory.
6. Ensure that all the code written by you are compiling correctly. Preferably use gcc with the options **`-W -Wall -O2`**, while compiling your code.
7. Instructions for uploading the files shall be provided separately.

Problem 1

In an advanced country, a point system is maintained to keep track of erring drivers, and vehicle owners. The objective for this lab is to comeup with a software to keep track of penalties incurred by the drivers, and help decide those drivers whose driving license should be revoked.

Each driver is assigned an initial score of ten. Offenses and their corresponding penalty are : (parking ticket, -1), (overspeeding,-2), (underspeeding-2), (drunk driving-5), (reckless driving,-7), (accidents, -9). If a driver's score becomes 0 or negative, his/her license will be revoked. Whenever an offense involving a vehicle takes place, the owner of the vehicle will be charged a penalty of half the penalty awarded to driver rounded to the nearest integer; irrespective of whether the owner or a different person was driving the vehicle.

The list of offenses is provided as a text file containing the vehicle number and the nature of offense. The list of drivers along with their UID and license number is available as input in a file named drivers.txt. A file named owners.txt maps the vehicle number to the UID of the owner. For the sake of simplicity, license number is a 7-digit integer, and vehicle number contains 4 digits. UID is a unique identifier provided to every citizen in the country, and it contains 7 digits.

The data structures to be used, the file format for various files and the general step-by-step process for this lab's task are being stated in the remainder of this document.

Data structures to be used:

1) **Vehicle**: This contains the list of all vehicles along with the vehicle number and UID of owners, with UID as the key. Vehicle is maintained as a dynamic list.
2) **Drivers**: This represents the dynamic list of the tuples (license number, UID, score).
3) **revokeList**: This list contains the tuple (license number, UID, score).

File Formats:

ALL input files are named as #name.txt; where # indicates the number of entries in the files and name.txt is the file's name as mentioned in the format below/.

The file format for the input files is as follows:
drivers.txt:

<UID>,<License Number>
Note: There is no space between number of items and item name. End Note.
For ex:

```
4drivers.txt
5998589
,0075512

5175406
,4907122

8594013
,1071786

6385096
,8988772
```

The file format for owner.txt is as follows:
UID,vehicle number
Note: There is no space between number of items and item name. End Note.
For ex:
1owner.txt

```
1829133,1512
```

The file format for offenses.txt is as follows
<license number> <offence penalty>
For ex:

```
0075512
 1
8988772
 7
1071786
 7
8988772
 5
0075512
 8
```

Random number generation

```
for i in $(seq 10000)
 do
   dd if=/dev/urandom count=1 2> /dev/null | cksum | cut -f1 -d" " | cut -b2-8
 done
```

The above script generates 10000 random numbers of 7 digits. Adjust the result to generate requisite data.

for i in $(seq 10000)

```
do
  dd if=/dev/urandom count=1 2> /dev/null | cksum | cut -f1 -d" " | cut -b2-5
done
```

The above script generates 10000 random numbers of 4 digits for vehicle number. Adjust the result to generate requisite data.

Step 1:
Identify appropriate data structures relevant for the problem and write them in **Offense.h**

Step 2:
Design **`Offense`** ADT with the following operations (**Offense`Ops.h`**)

(a) Identify the relevant operations
(b) Add prototypes in OffensesOps.h

Step 3: Implement the above operations in the corresponding c files
a) populateVehicles(in **populateVehicles.c):** This takes the owner file name as input and returns the Vehicle list in sorted order.
b) populateDrivers (in **populateDrivers):** This takes the drivers file name as input and returns the Driver list in sorted order.
c) makeRevokeList (in **makeRevokeList.c):** This takes Vehicles, Drivers, and file of offenses as input; and updates the revokelist and drivers list and returns it.
d) printRevoke(in **printRevoke.c):** this reads the revoke list and prints the revoked licenses.

All list creations must have filename as a parameter.

Step 4: driver file.
Create a simple driver file (**`revoke.c`**) that can read multiple input files corresponding to the various input files. The names of input files are passed as command line parameters. Note that if no input file is passed, the program should attempt to run with drivers.txt, owners.txt and offenses.txt for the relevant input set.
The executable **`revoke`** generates output files **revoke.txt** that will contain the list of drivers (license number, UID, vehicle number) whose license is to be revoked.

Tasks:

a) Implement the functions using normal lists (dynamic array or linked list)
b) Implement Vehicles and Drivers as Hashtable.
    a. Vehicle registration numbers mapped to table length; Drivers as dynamic array
    b. Driver license numbers mapped to table length; Vehicle as dynamic array
    c. Both as 2 different hash tables. Consider suitable hash functions.
c) Vary the size of hashtable (m)
d) Plot time taken for the various techniques, and perform comparative analysis.
e) Argue whether RevokeList should be a hashtable or not.

Deliverables: All relevant header files, relevant c files**, `makefile.`**