C programming assignments (for 10 Marks)

Student projects

Project# 1

Simulation of an Airport

You need to simulate the working of an airport.

Let us consider a small but busy airport with only one runway. In each unit of time (minute, hour), one plane can land or one plane can take off, but not both. Planes arrive ready to land or ready to take off at random times, so at any given unit of time, the runway may be idle or a plane may be landing or taking off and there may be several planes waiting either to land or take off. We therefore need two queues, called landing and takeoff, to hold these planes. It is better to keep a plane waiting on the ground than in the air, so a small airport allows a plane to take off only if there are no planes waiting to land. Hence, after receiving requests from new planes to land or take off, our simulation will first service the head of the queue of planes waiting to land, and only if the landing queue is empty, will it allow a plane to take off. The simulation should run through many units of time.

The expected output from all the simulation projects should be on the lines of what is shown in the next 3 pages.

```
One plane can land or depart in each
Up to 5 planes can be waiting to land or take off at any time.
How many units of time will the simulation run ? 30
Expected number of arrivals per unit time (real number) ? 0.47
Expected number of departures per unit time? 0.47
Plane 1 ready to land.
1:  Plane 1 landed; in queue 0 units.
2:  Runway is idle.
    Plane 2 ready to land.
    Plane 3 ready to land.
3:  Plane 3 landed; in queue 0 units.
4:  Plane 3 landed; in queue 1 units.
    Plane 4 ready to land.
    Plane 5 ready to land.
    Plane 6 ready to take off.
    Plane 7 ready to take off.
5   Plane 4 landed; in queue 0 units.
    Plane 8 ready to take off.
6:  Plane 5 landed; in queue 1 units.
    Plane 9 ready to take off.
    Plane 10 ready to take off.
7:  Plane 6 took off; in queue 2 units.
8:  Plane 7 took off; in queue 3 units.
9.  Plane 8 took off; in queue 3 units.
    Plane 11 ready to land.
10: Plane 11 landed; in queue 0 units.
    Plane 12 ready to take off.
11: Plane 9 took off; in queue 4 units.
    Plane 13 ready to land.
    Plane 14 ready to land.
12: Plane 13 landed; in queue 4 units.
13. Plane 14 landed; in queue 1 units.
14: Plane 10 took off; in queue 7 units.
    Plane 15 ready to land.
    Plane 16 ready to take off.
    Plane 17 ready to take off.
15: Plane 15 landed; in queue 0 units.
    Plane 18 ready to land.
    Plane 19 ready to land.
    Plane 20 ready to take off.
    Plane 21
```

Both queues are empty

landing queue is empty

16: Plane 18 landed; in queue 0 units.
Plane 22 ready to land.
17: Plane 19 landed; in queue 1 units.
Plane 23 ready to take off.
Plane 23 told to try later.
18. Plane 22 landed; in queue 1 units.
Plane 24 ready to land.
Plane 25 ready to land.
Plane 26 ready to land.
Plane 27 ready to take off.
Plane 27 told to try later.
19. Plane 24 landed; in queue 0 units.
Plane 28 ready to land.
Plane 29 ready to land.
Plane 30 ready to land.
Plane 31 ready to land.
Plane 31 directed o another airport.
20: Plane 25 landed; in queue 1 units.
Plane 32 ready to land.
Plane 33 ready to take off.
Plane 33 told to try later.
21: Plane 26 landed; in queue 2 units.
22: Plane 28 landed; in queue 2 units.
23: Plane 29 landed; in queue 3 units.
Plane 34 ready to take off.
Plane 34 to try later.
24: Plane 30 landed; in queue 4 units.
Plane 35 ready to take off.
Plane 35 told to try later.
Plane 36 ready to take off.
Plane 36 told to try later.
25: Plane 32 landed; in queue 4 units.
Plane 37 ready to take off.
Plane 37 told to try later.
26: Plane 12 took off; in queue 15 units.
27: Plane 16 took off; in queue 12 units.
28: Plane 17 took off; in queue 13 units.
29: Plane 20 took off; in queue 13 units.
Plane 38 ready to take off.
30: Plane 21 took off; in queue 14 units.

summary

```
Simulation has concluded after 30 units.
Total number of planes processed:        38
    Number of planes landed:             19
    Number of planes taken off:          10
    Number of planes refused use:         8
    Number left ready to land:            0
    Number left ready to take off:        1
    Percentage of time runway idle:    3.33
    Average wait time to land:         1.11
    Average wait time to take off:     8.60
```

Project #2

You need to simulate the working of an airport.

Let us consider a small but busy airport with two runways, one always used for landings and the other always used for takeoffs. Planes arrive ready to land or ready to take off at random times, so at any given unit of time, the runways may be idle or a plane may be landing or taking off and there may be several planes waiting either to land or take off. We therefore need two queues, called landing and takeoff, to hold these planes. It is better to keep a plane waiting on the ground than in the air, so a small airport allows a plane to take off only if there are no planes waiting to land. Hence, after receiving requests from new planes to land or take off, our simulation will first service the head of the queue of planes waiting to land, and only if the landing queue is empty, will it allow a plane to take off. The simulation should run through many units of time

Project #3

You need to simulate the working of an airport.

Let us consider a small but busy airport with two runways, one usually used for landings and one usually used for takeoffs. Planes arrive ready to land or ready to take off at random times, so at any given unit of time, the runways may be idle or a plane may be landing or taking off and there may be several planes waiting either to land or take off. If one of the queues is empty, then both runways can be used for the other queue. Also, if the landing queue is full and another plane arrives to land, then takeoffs will be stopped and both runways used to clear the backlog of landing planes. The simulation should run through many units of time

Project #4

You need to simulate the working of an airport.

Let us consider a small but busy airport with three runways, one always reserved for landing, one for takeoff and the third used for landings unless the landing queue is empty, in which case it can be used for takeoffs. Planes arrive ready to land or ready to take off at random times, so at any given unit of time, the runways may be idle or a plane may be landing or taking off and there may be several planes waiting either to land or take off. The simulation should run through many units of time

Project #5

You need to simulate the working of an airport.

Let us consider a small but busy airport with only one runway. In each unit of time (minute, hour), one plane can land or one plane can take off, but not both. Planes arrive ready to land or ready to take off at random times, so at any given unit of time, the runway may be idle or a plane may be landing or taking off and there may be several planes waiting either to land or take off. We therefore need two queues, called landing and takeoff, to hold these planes. It is better to keep a plane waiting on the ground than in the air, so a small airport allows a plane to take off only if there are no planes waiting to land. When each plane arrives to land, it will (as part of its data) have a (randomly generated) fuel level, measured in units of time remaining. If the plane does not have enough fuel to wait in the queue, it is allowed to land immediately. Hence the planes in the landing queue may be kept waiting additional units and so may run out of fuel themselves. Check this out as part of the landing function and find about how busy the airport can become before planes start to crash from running out of fuel.

Project #6

Write a program in C to mimic the “adduser” command on Linux. This command will add either an ordinary user or a system user. It has to handle 2 files “passwd” and “shadow”. Both these files will be in some folder specified by an environment variable PFILE. The program has to take all arguments as command line arguments (Refer man pages for the command line arguments)

Project #7

Write a program in C to mimic the “addgroup” command on Linux. This command will add a new group. It has to handle 2 files “passwd” and “shadow”. Both these files will be in some folder specified by an environment variable PFILE. The program has to take all arguments as command line arguments (Refer man pages for the command line arguments)

Project #8

Write a program in C to mimic the “deluser” command on Linux. This command will delete a user. It has to handle 2 files “passwd” and “shadow”. Both these files will be in some folder specified by an environment variable PFILE. The program has to take all arguments as command line arguments (Refer man pages for the command line arguments)

Project #9

Write a program in C to mimic the “delgroup” command on Linux. This command will delete a group. It has to handle 2 files “passwd” and “shadow”. Both these files will be in some folder specified by an environment variable PFILE. The program has to take all arguments as command line arguments (Refer man pages for the command line arguments)

Project #10

Write a program in C to mimic the “passwd” command on Linux. This command will help the user to change the password. Refer man pages to get details on the command line arguments to be passed. The encryption and decryption of the password is part of this project. You can use any simple encryption and decryption mechanism.

Project #11

Write a program in C that verifies the integrity of the users and authentication information. It checks that all entries in /etc/passwd and /etc/shadow have the proper format and contain valid data. The user is prompted to delete entries that are improperly formatted or which have other uncorrectable errors. Refer man pages for the command “pwck”.

Project #12

Write a program in C to mimic the “finger” command (user information lookup program) on Linux. Refer man pages for the command “finger”.

Project #13

Write a program in C to mimic the “cmp” command (compare two files byte by byte) on Linux. Refer man pages for the command “cmp”.

Project #14

Write a program in C to mimic the “diff” command (compare files line by line) on Linux. Refer man pages for the command “diff”.

Project #15

Write a program in C to mimic the “grep” command (**grep** searches the named input *FILE*s for lines containing a match to the given *PATTERN*. By default, **grep** prints the matching lines.) on Linux. Refer man pages for the command “grep”.

Project #16

Write a program in C to mimic the “sort” command (sorts lines of text files) on Linux. Refer man pages for the command “sort”.

Project #17

Bug tracking software

This program should help a user file a bug (with all details – a unique ID which is generated by the program, Type of the bug, a brief description, Priority for the bug, time at which it was filed, status of the bug (“Not yet assigned”, “In process”, “Fixed”, “Delivered”, name of the person who filed the bug, etc). There should be a provision to change the status of the bug, get a report on bug statistics) all of these using file operations.

Project #18

Resume scanning software; Given a number of text files and the patterns to look for, separate out those files which have these patterns in them.

Example; All the files which have Java & Python, Java or Python, “Project Management”, etc should be shortlisted.

Project #19

Write a program in C which has functions to encrypt and decrypt a given text using the Caesar cipher.

It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is named after Julius Caesar, who apparently used it to communicate with his generals. The shift should be user defined.

Project #20

Develop a mini employee leave management system; You need to have the employee details in a file, allow the employee to log in using his employee ID (which is unique); allow employees to avail a certain number of leaves per year (which is split into x casual leaves, y medical leaves and z earned leaves); if the employee is well within the allowed limits, automatically approve the leave; else reject and provide the reason.