IGNOU MCA MCS-44 Solved Assignment 2011

Course Code	:	MCS-044
Course Title	:	Mini Project
Assignment Number	:	MCA (4)/044/Assign/2010-11
Assignment Marks	:	100
Maximum Marks	:	25%
Last Date of Submission	**:**	**31st October, 2010 (for July, 2010 session)** 30th April, 2011 (for January, 2011 session)

Question 1:

Describe the Systems Development Life Cycle (SDLC) that suits the above specifications. Also, evaluate the systems requirements. (10 Marks)

Ans 1

1. **Definition:** The system is required to implement the online Ticket Booking system for the common wealth games 2010 for booking & cancellation. The Tickets are available in the different categories ordinary, deluxe & cabin.
2. **Analysis Phase** is consists of ER & DFD is solved in question No3. & database design is given in Q No 7.
3. **Coding:** we will implement the coding modules which will consist of the following Menu.

    1. Book Ticket
    2. Cancel Ticket
    3. Add Game event
    4. Add Game Venue

Feasibility & Cast Analysis is given I question No. 2.

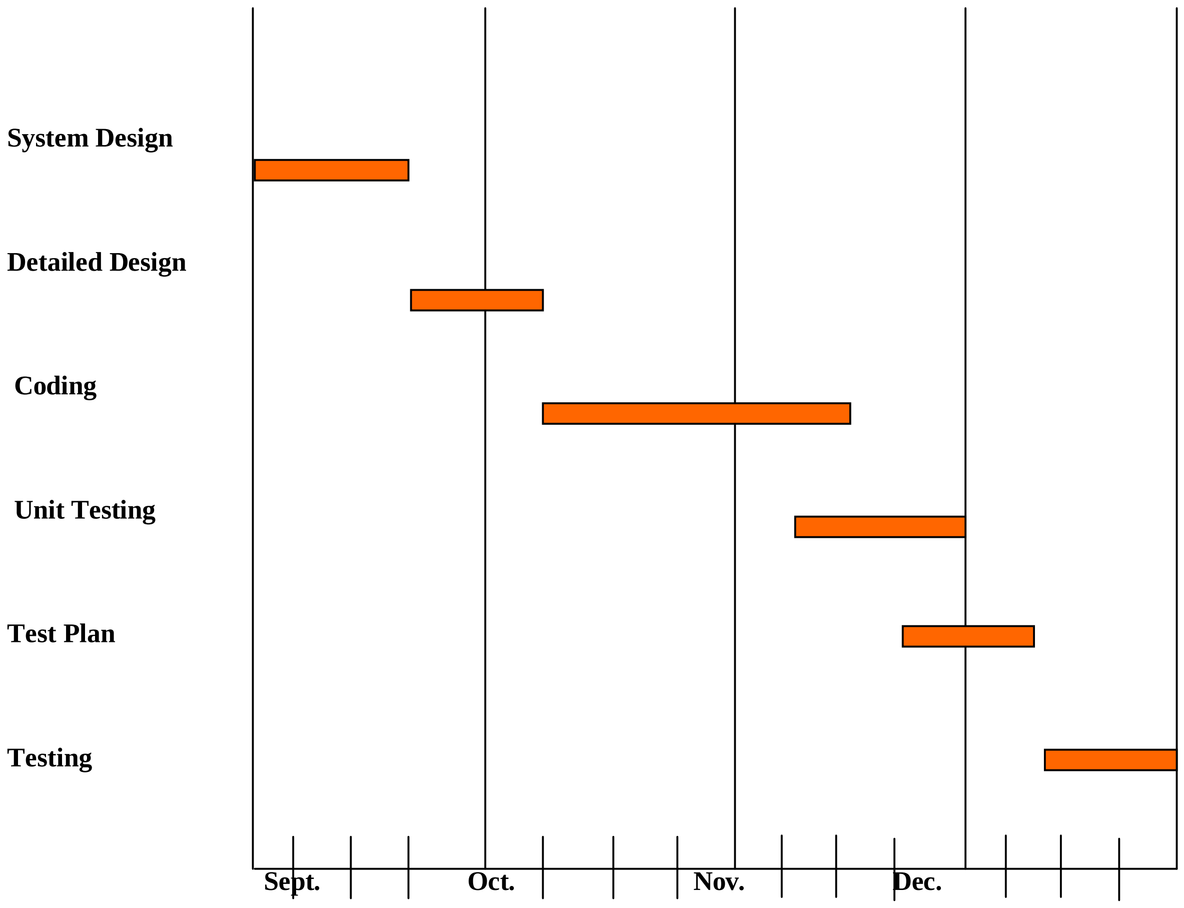
4 . Testing: Testing is defined in question No. 8.

Ans2.

GANT CHART

Gantt charts mainly used to allocate resources to activities. The resources allocated to activities include staff, hardware, and software. Gantt charts (named after its developer Henry Gantt) are useful for resource planning. A Gantt chart is special type of bar chart where each bar represents an activity. The bars are drawn along a timeline. The length of each bar is proportional to the duration of the time planned for the corresponding activity.

Gantt chart is a project scheduling technique. Progress can be represented easily in a Gantt chart, by coloring each milestone when completed. The project will start in the month of January and end after 4 months at the end of April.



Pert Chart

PERT (Project Evaluation and Review Technique) charts consist of a network of boxes and arrows. The boxes represent activities and the arrows represent task dependencies.

PERT chart represents the statistical variations in the project estimates assuming a normal distribution. Thus in a PERT chart instead of making a single estimate for each task, *pessimistic, likely,* and *optimistic* estimates are also made. The boxes of PERT charts are usually annotated with the pessimistic, likely, and optimistic estimates for every task. Since all possible completion times between the minimum and maximum durations for every task have to be considered, there are many critical paths, depending on the permutations of the estimates for each task. This makes critical path analysis in PERT charts very complex. A critical path in a PERT chart is shown by using thicker arrows. The PERT chart representation of the buses scheduling problem of Figure A. is shown in Figure B.

Task	ES	EF	LS	LF	ST
Specification Part	0	15	0	15	0
Design Database Part	15	60	15	60	0
Design GUI Part	15	45	90	120	75
Code Database Part	60	165	60	165	0
Code GUI Part	45	90	120	165	75
Integrate and Test	165	285	165	285	0
Write User Manual	15	75	225	285	210

Figure A : Different Tasks for the Bus traveling management system are shown in above table.

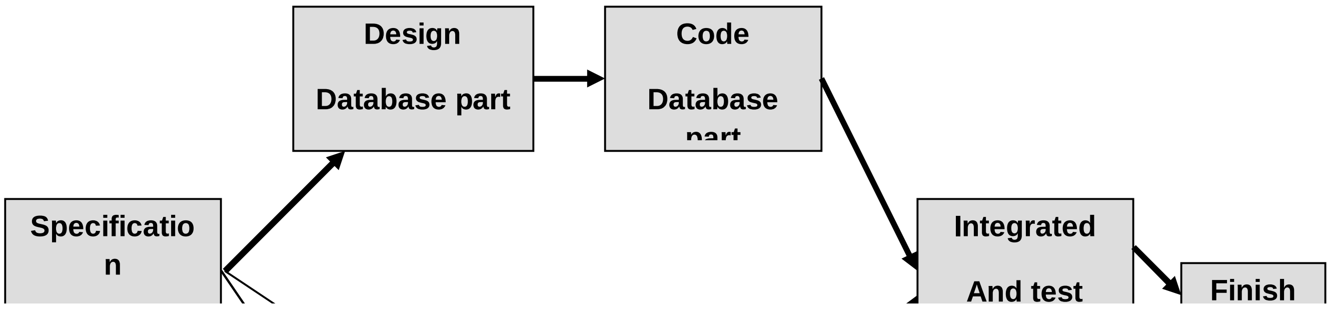


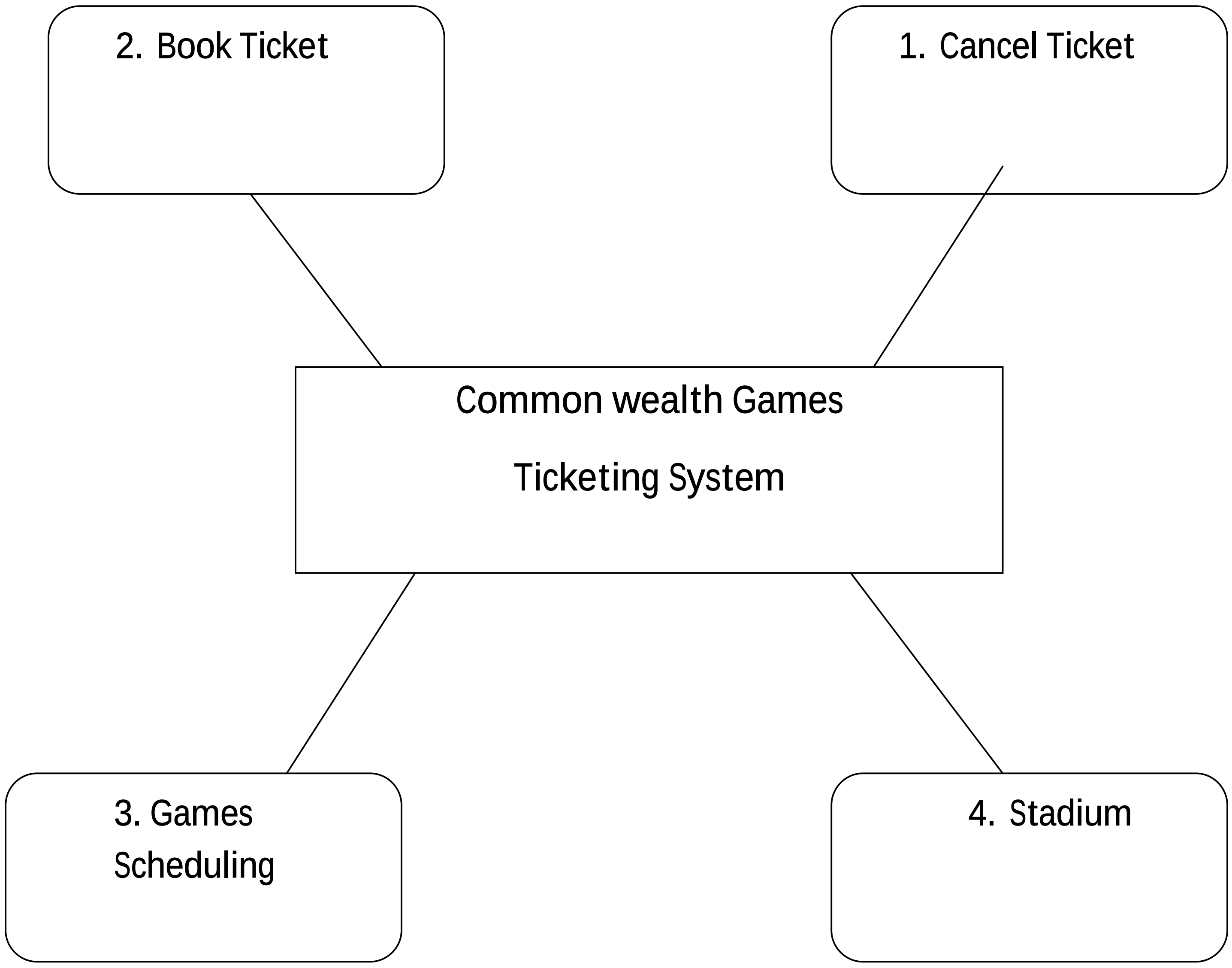
FIGURE B: PERT chart representation of the Bus Traveling Management **System**.

PERT charts are a more sophisticated form of activity chart. In activity diagrams only the estimated task durations are represented. Since the actual durations might vary from the estimated durations, the utility of the activity diagrams is limited.

Ans3. The different phases of software development lie cycle are shown below.

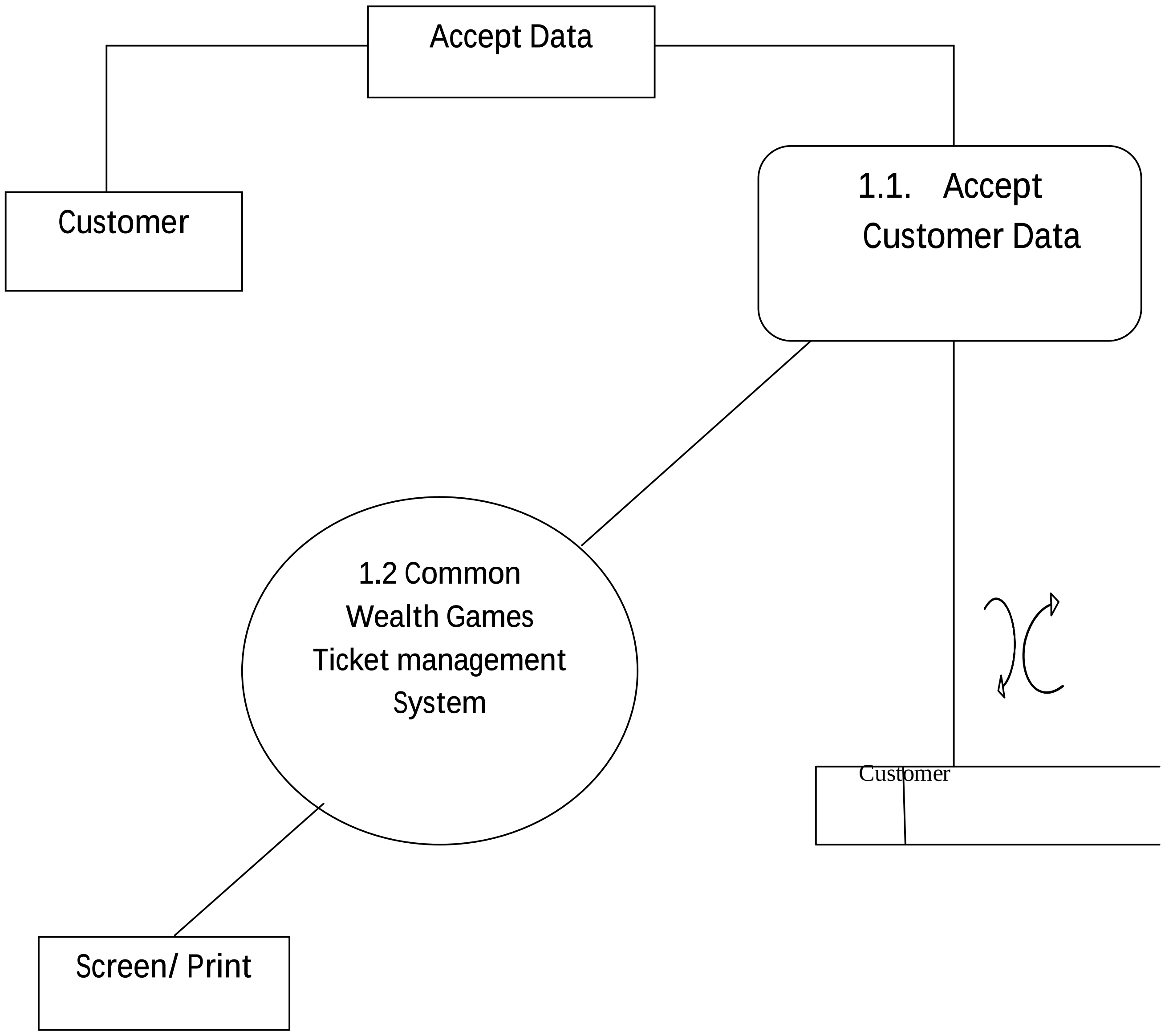
E-r Diagram

Context Level DFD

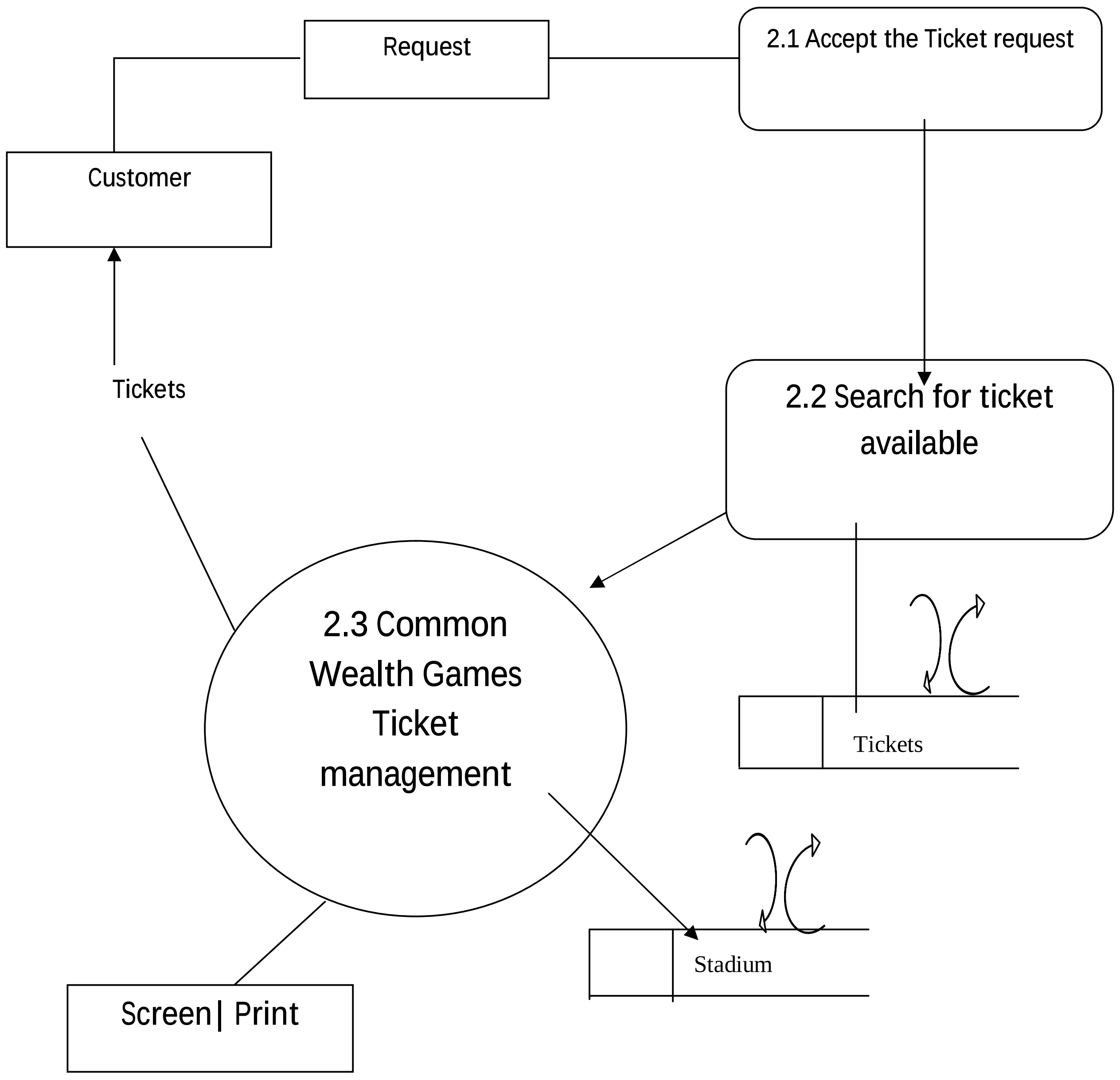


C)

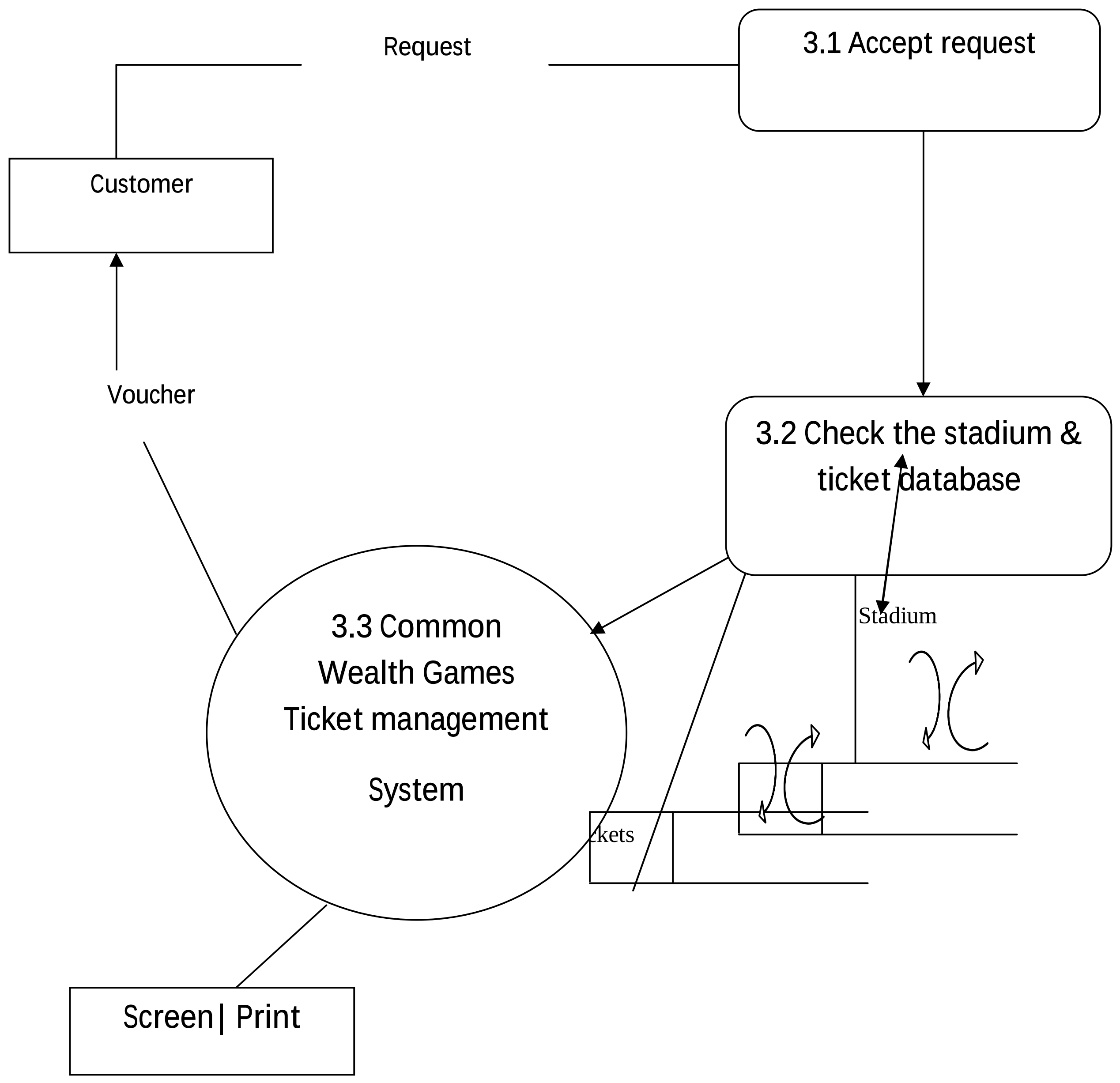
DFD for Customer Registration


Accept Data
Customer
1.1. Accept Customer Data
1.2 Common Wealth Games Ticket management System
Customer
Screen/ Print

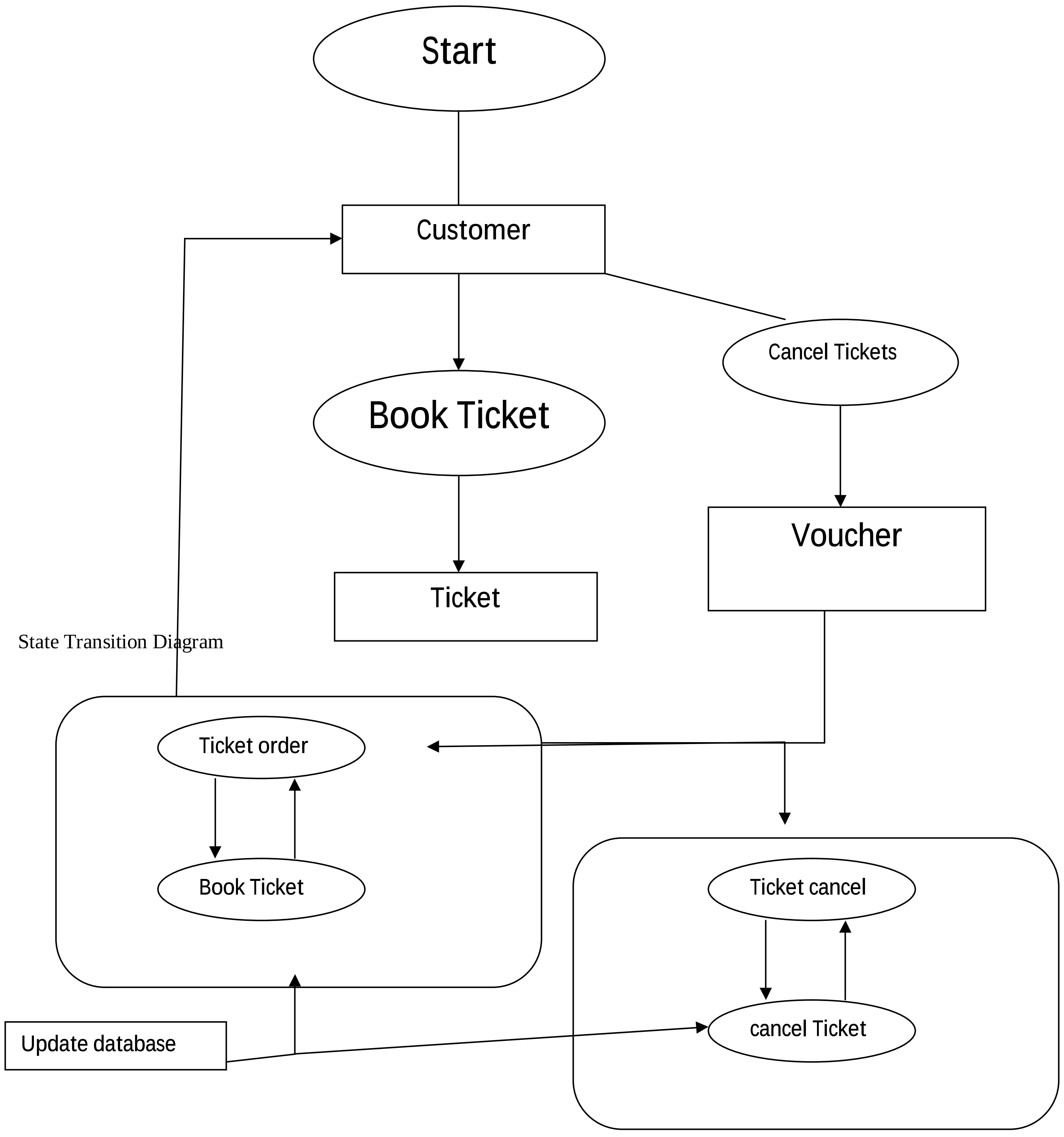
DFD For Ticket Booking


Request
2.1 Accept the Ticket request
Customer
Tickets
2.2 Search for ticket available
2.3 Common Wealth Games Ticket management
Tickets
Stadium
Screen| Print

DFD For Cancellation Of Booking


Request
3.1 Accept request
Customer
Voucher
3.2 Check the stadium & ticket database
3.3 Common Wealth Games Ticket management System
Stadium
kets
Screen| Print

Ans4. System Flow Chart



State Transition Diagram

Ans5. **HARDWARE AND SOFTWARE REQUIREMENT SPECIFICATION**

a. **Software Requirement**

Platform	:	Windows
The Operating System	:	Windows XP Professional sp2
Server :		
Front-End Tool	:	Java ServerPages, JSF
Editing tool	:	NetBeans IDE 6.5.1
Browser	:	Internet Explorer
Database	:	MySQL Community Server

b. **Hardware Used**

Processor	:	Intel Pentium 4 (3.06 GHz)
Memory	:	1 GB RAM.
Network Adapter	:	Ethernet Adaptor
Modem	:	56kbps Voice Fax Data
Secondary Storage	:	Seagate Hard disk (80 GB)

Ans6. System design phase which require for commonwealth game management system will at the first stage consist of analysis & scheduling the different by ER Diagram & then reducing by it followed by ER diagram to the database & then Normalizing the database to give the data Flow Diagram for each one of them.

In Summay we can say

1. E-R diagram is ceated [Q. No3].
2. Data dictionary is created [Q. No3].
3. DFD is created [Q. No4].
4. Normalisation of data is done[Q. No7].

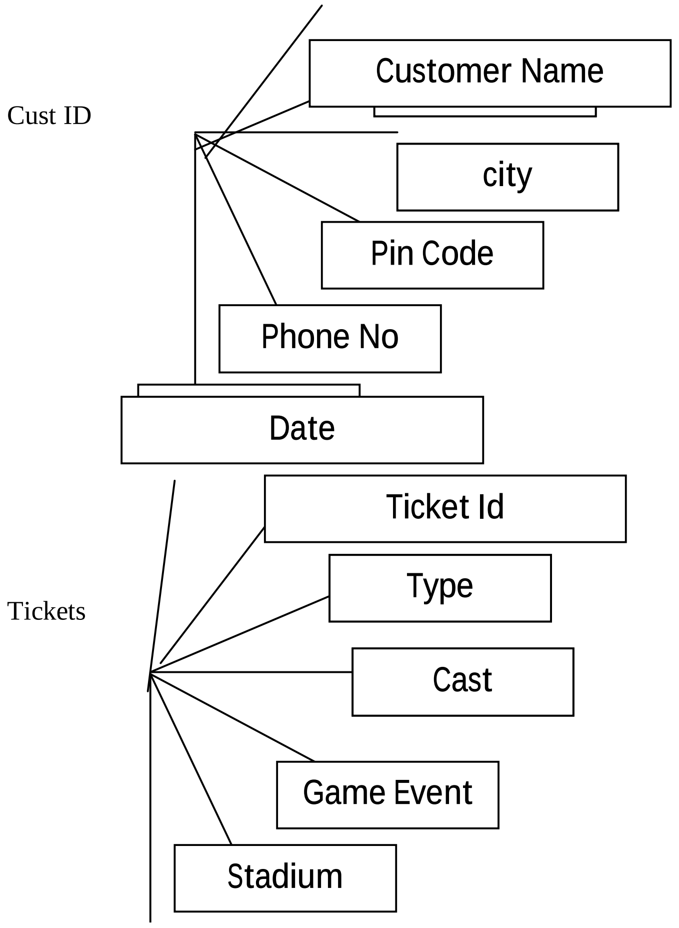
Ans7. Unnormalised data is consist of the following fields

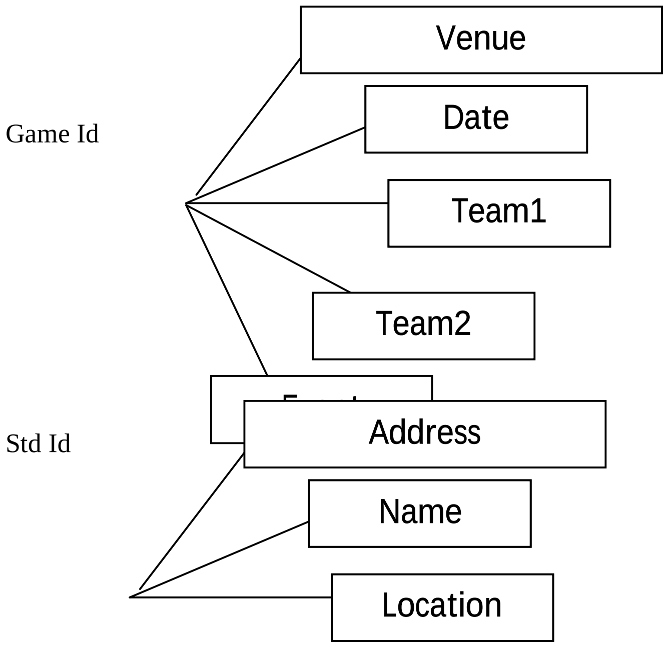
1. Custom Id
2. Name
3. Address
4. City

5. PIN Code
6. Email
7. Phone
8. Id proof
9. Type
10. Cast
11. Ticket Id
12. Game Event
13. Stadium
14. Timing
15. Price
16. Date of Ticket
17. Game Id
18. Venue
19. Date of event
20. Team 1
21. Team2
22. Std Id
23. Address
24. Name
25. Location
26. Event

In the first Normal from every Field should have a atomic value fro that we will fill each field.

For the second Normal form we will find the primary key & then the data which is functionally dependent on them





IN the third Normal form finaaly without the transitional dependency we find the following Normalised tables.

1 Customer (Customer Id, Name, Address, City, Pin code, email, phone, Id proof).

2 Tickets (Type, cost, Ticket Id, GameEvent, stadium, Timming, date)

3 Games (games Id, venue, date, team1, team2, Event)

4 Stadium (std Id, address, Name, Location).

Ans8. **Testing**

Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test , with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding software bugs. It can also be stated as the process of validating and verifying that a software program/application/product meets the business and technical requirements that guided its design and development, so that it works as expected and can be implemented with the same characteristics.

A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions.[11] The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

Defects and failures

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirements gaps, e.g., unrecognized requirements, that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

Software faults occur through the following process. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure.[12] Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new hardware platform, alterations in source data or interacting with different software.[12] A single defect may result in a wide range of failure symptoms.

Compatibility

A frequent cause of software failure is compatibility with another application, a new operating system, or, increasingly, web browser version. In the case of lack of backward compatibility, this can occur (for example...) because the programmers have only considered coding their programs for, or testing the software upon, "the *latest* version of" this-or-that operating system. The unintended consequence of this fact is that: their latest work might not be fully compatible with earlier mixtures of software/hardware, or it might not be fully compatible with *another* important operating system. In any case, these differences, whatever they might be, may have resulted in (unintended...) software failures, as witnessed by some significant population of computer users.

This could be considered a "prevention oriented strategy" that fits well with the latest testing phase suggested by Dave Gelperin and William C. Hetzel, as cited below [13].

Input combinations and preconditions

A very fundamental problem with software testing is that testing under *all* combinations of inputs and preconditions (initial state) is not feasible, even with a simple product. This means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, non-functional dimensions of quality (how it is supposed to *be* versus what it is supposed to *do*) -- for example, usability, scalability, performance, compatibility, reliability -- can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another.

Static vs. dynamic testing

There are many approaches to software testing. Reviews, walkthroughs or inspections are considered as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. The former can be, (and unfortunately in practice often is...) omitted, whereas the latter takes place when programs begin to be used for the first time - which is normally considered the beginning of the testing stage. This may actually begin before the program is 100% complete in order to test particular sections of code (modules or discrete functions). For example, Spreadsheet programs are, by their very nature, tested to a large extent "on the fly" during the build process as the result of some calculation or text manipulation is shown interactively immediately after each formula is entered

1. UNIT TESTING:

This is the smallest testable unit of a computer system and is normally tested using the white box testing. The author of the programs usually carries out unit tests.

2. INTEGRATION TESTING:

In integration testing, the different units of the system are integrated together to form the complete system and this type of testing checks the system as whole to ensure that it is doing what is supposed to do. The testing of an integrated system can be carried out top-down, bottom-up, or big-bang. In this type of testing, some parts will be tested with white box testing and some with black box testing techniques. This type of testing plays very important role in increasing the systems productivity. We have checked our system by using the integration testing techniques.

3. SYSTEM TESTING:

A part from testing the system to validate the functionality of software against the requirements, it is also necessary to test the non-functional aspect of the system. Some examples of non-functional tools include tests to check performance, data security, usability/user friendliness, volume, load/stress that we have used in our project to test the various modules.

System testing consists of the following steps:

1. Program(s) testing.
2. String testing.
3. System testing.
4. System documentation.
5. User acceptance testing.

4. FIELD TESTING:

This is a special type of testing that may be very important in some projects. Here the system is tested in the actual operational surroundings. The interfaces with other systems and the real world are checked. This type of testing is very rarely used. So far our project is concerned; we haven't tested our project using the field testing.

5. ACCEPTANCE TESTING:

After the developer has completed all rounds of testing and he is satisfied with the system, then the user takes over and re-tests the system from his point of view to judge whether it is acceptable according to some previously identified criteria. This is almost always a tricky situation in the project because of the inherent conflict between the developer and the user. In this project, it is the job of the bookstores to check the system that whether the made system fulfills the goals or not.

Test Cases

Impact	Test Description	Test Result	Result
Book Ticket	Accept Valid data	Ticket Booked	Pass
Book Ticket	Accept Invalid data	Ticket not booked	Fail
Cancel Ticket	Accept Ticket Id [valid]	Ticket Cancelled	Pass
Cancel Ticket	Accept ticket Id [Invalid]	Ticket Not Cancelled	fail

Course Code : MCSL-045

Course Title : UNIX and DBMS Lab

Part – I : MCS – 041

Question1.

(a) $ find / -name ‘program.c’ 2>/dev/Null
$find / -name

(b) Who : sort

(c) $ quep –c ‘IGNOU’ Ignou project.txt

(d) Cat > Assignment.txt

This assignment is quite tough & Lengthy
Ctr+d will save the file assignment

(e) Chmod + R Myfile.txt
Chmod - R Myfile.txt

Chmod +w Myfile.txt

Chmod - w Myfile.txt

(f) $ quep -n ‘ab’ assignment.txt

(g) Cmp file1 file2

(h) Step 1 : Cat / etc/ passed
Step 2 : Cat / etc/ passed / qrep “/have”

Step 3 : Now we will get all the user account which have their have
There have share in / home
Step 4 : Now we will modify an
Cat / etc/ passed / qrep “/have” / cut – d : -f

(i) Cat > Ignou _ jalad

1. I Initially
2. Was happy
3. To take
4. Admission
5. In Ignou
6. Now I
7. See the
8. Attitude
9. Of My
10. Teacher
11. At study
12. Center they
13. Don't teach
14. & I
15. Am really
16. Looted &
17. Find cheated
18. To take
19. Admission in
20. IGNOU.

    Ctrl + d
    $ Split -b 10 Ignou_jalad would
    Split the file Ignou_jalad into
    Two pieces 'Ignou_jalad 1, Ignou_jalad 2.

(j) Tr –Delete ' =;:'" & ( ) []'
Delete Specified set of characters
Defined in set 1 but do not translate

Question 2

(a)

```
Filename  =  “ignou.txt”
  File = open (filename ,  ‘r’)
Obj = file.read (filename)
Echo “obj.Account No.”
Echo “obj.Date”.
Echo “obj. Credit”.
Echo “obj.Amount”.
```

(b)

```
          File  = $1

          Echo –n              “enter a file name:”

          Read file

          If [! –f $ file]

          Then

          Echo “ $ file not a file!”

          Exit!

fi

          Echo –n “enter a password:”

          Read password.

# do encryphon any UNIX crypt Cammand

# this command will proper for a password
```

```
Crypt $ password <$file> $ file.cpy

Echo " $file.cpy created as encrypted file"


(c)       a= $1

          b=$2

          c=$3

          if [$ # -lt 3]

          then

          echo "$0 n1 n2 n3"

          exit1

fi

if [$ a –gt  $b-a  $a –gt $c]

then

echo "$ a is largest integer"

elif [ $b – gt  $  a  -a  $b – gt $ c]

then

echo " $b is largest integer'

elsif [ $c –gt  $ a –a $ c –gt $ b];

then

echo $c is largest interg"

else

echo "sorry cannot given number"
```

fi

PART-II: MCS-043

Question No. (1)

(a)
```
Create database University
Use University
Create Table student

(Id int primary key,
Name char (25), Not Null,
Programme char (25), Not Null,
Total_semester in not null
Semester_completed  int not null
Semester _Registered int not null
)
Creat Table fees
(
Id int primary key,
Semester int Not Null,
Subject char(20) Not Null,
Total_fee int Not Null,
Fee_paid int not null,
{_registered char (20) Not Null
)
Create Table Department
(Id int primary key ,
Name char (20) Not Null,
Programme char (20) Not Null
)
```

(b) (i)

```
Select student_Id, student_Name,Student_Programme

From student inner join fees on

Student:Id = fees.Id where Fees_Register='No'
```

(ii)

```
Select count (subject) from Department

Group by Department having

Count (Subject) < 50
```

(iii)

```
Select count (semester) from student

Group by semester having count (semester) > 4
```

(iv)

```
Select count (Semester) from student group by semester having

Count (semester) =8
```

(c) Create view data entry

```
 As
Insert into student values (1,'join', 'MCA', 2, 2, 3)
```

```
Create view show
As
Select * from student where Id=@Id
```

(d)
```
Create Procedure pro()
As
Select sum (fee paid) from fees
Group by Id having
Sum (fee paid) <8000
```

(e)
```
Begin Transaction
Insert into student values
(2, ‘Albert’, ‘BCA’, 3.34)
Commit Transaction
```

(f) (i)
```
Create or Replace Trigger “CCAPADM”
                     TRG_PRINT_INDEX
          AFTER REGISTRATION
          On “CCAPADM”
```