CBCS SCHEME

USN 1CR21IS043

21IS63

Sixth Semester B.E. Degree Examination, June/July 2024

Software Testing

Time: 3 hrs. Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and /or equations written eg, 42+8 = 50, will be treated as malpractice.

Module-1

1. a. What is Software Testing? Differentiate between functional testing and structural testing with an example. **(10 Marks)**

   b. Demonstrate the triangle problem statement along with a flowchart for traditional implementation. **(10 Marks)**

OR

2. a. With a neat diagram, explain the SATM. **(10 Marks)**

   b. Classify the types of faults and explain each with an example. **(10 Marks)**

Module-2

3. a. Examine boundary value analysis with the test cases using a triangle problem. **(10 Marks)**

   b. Examine the equivalence class testing. Examine the equivalence class test cases for the nextnate function. **(10 Marks)**

OR

4. a. What are the limitations of boundary value analysis and examine the test cases using boundary value analysis testing for commission problem. **(10 Marks)**

   b. Explain the format of the decision table. Build a decision table for a simple version of the triangle problem. **(10 Marks)**

Module-3

5. a. Define a program graph. Draw a program graph of the commission problem. **(10 Marks)**

   b. Define DD-path. Explain basis path testing with a suitable example. **(10 Marks)**

OR

6. a. Define predicate node, du-paths, dc-path. Give du-path for lock, stock and sales for commission problem. **(10 Marks)**

   b. Explain slice-based testing with an example. **(10 Marks)**

Module-4

7. a. Examine the traditional view of testing levels, alternate life cycle model. **(10 Marks)**

   b. Compare top-down and bottom-up integration strategies. **(10 Marks)**

OR

8. a. Formulate call graph based integration with the help of : i) Pairwise Integration ii) Neighborhood integration. **(10 Marks)**

   b. Define the SAJM system. Demonstrate the entity/relationship model of the SATM system. **(10 Marks)**

Module-5

9 a. Explain the basic concepts of requirement specification. **(10 Marks)**

b. Define the process of ASF testing and illustrate it with an example using the next date function. **(10 Marks)**

OR

10 a. Describe the context of interaction in software testing. **(10 Marks)**

b. What is the taxonomy of interaction? Explain the static interaction in a single process. **(10 Marks)**

* * * * *

USN

VTU Examination – JUN/JULY 2024
Scheme of Evaluation

Sub:	**SOFTWARE TESTING**			Sub Code:	**21IS63**	Branch:	**ISE**
Date:	**20/08/2024**	Duration:	**3 hrs**	Max Marks: **100**	Sem/Sec:	**VI/ A, B & C**	**OBE**

Answer any FIVE FULL Questions — MARKS | CO | RBT

MODULE -1

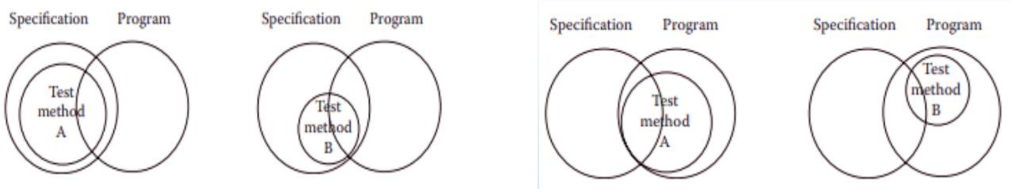
1. a. **What is Software Testing? Differentiate between functional testing and structural testing with an example.**

MARKS: 10 CO: 1 RBT: L2

Scheme: Definition + Differences + Diagram highlighting as an example – 2+5+3 marks

Solution: Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation.
Differentiate :

Functional Testing	Structural Testing
It is also known as black-box testing as no knowledge of the internal code is required.	It is also known as white-box or clear-box testing as thorough knowledge and access of the code is required.
It ensures that the system is error-free.	Finds errors in the internal code logic and data structure usage.
It is a quality assurance testing process ensuring the business requirements are met.	It does not ensure that the user requirements are met.
Functional testing checks that the output is given as per expected.	Performed the entire software in accordance with the system requirements.
Before writing a functional test case, a tester is required to understand the application's requirements.	Writing a structural test case requires understanding the coding aspects of the application.
It examines how well a system satisfies the business needs or the SRS.	It examines how well modules communicate with one another.



b. Demonstrate the triangle problem statement along with a flowchart for traditional implementation.

Scheme : Problem Statement + Code + Flowchart – 2+4+4 marks

Solution:

The Triangle Program accepts three integers as input.

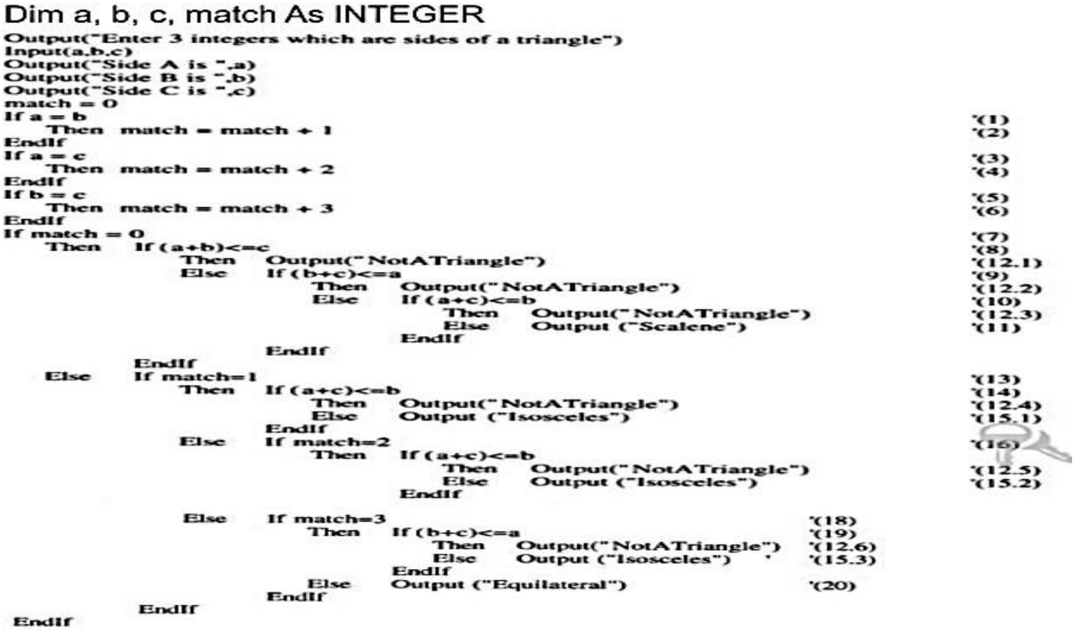
The output of the program is the type of triangle determined by the three sides: Equilateral, Isosceles, Scalene, or Not A Triangle.

The integers a, b, and c must satisfy the following conditions:

c1. $1 \leq a \leq 200$ c4. $a < b + c$

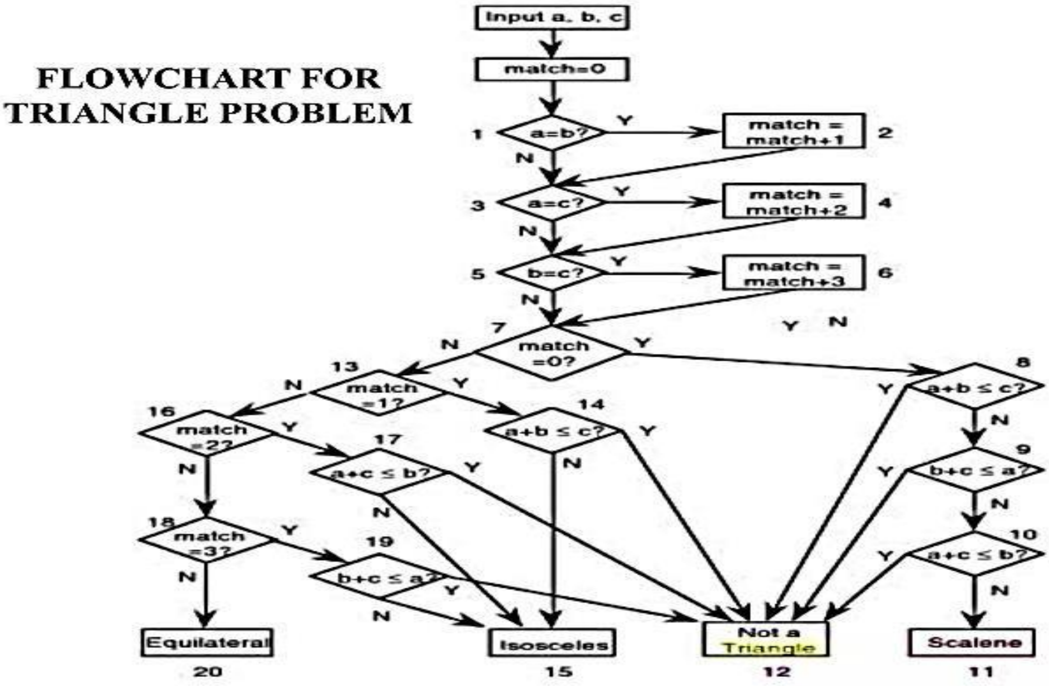
c2. $1 \leq b \leq 200$ c5. $b < a + c$

c3. $1 \leq c \leq 200$ c6. $c < a + b$



```
Dim a, b, c, match As INTEGER
Output("Enter 3 integers which are sides of a triangle")
Input(a,b,c)
Output("Side A is ",a)
Output("Side B is ",b)
Output("Side C is ",c)
match = 0
If a = b                                                           '(1)
    Then  match = match + 1                                        '(2)
EndIf
If a = c                                                           '(3)
    Then  match = match + 2                                        '(4)
EndIf
If b = c                                                           '(5)
    Then  match = match + 3                                        '(6)
EndIf
If match = 0                                                       '(7)
    Then    If (a+b)<=c                                            '(8)
                Then    Output("NotATriangle")                     '(12.1)
                Else    If (b+c)<=a                                '(9)
                            Then    Output("NotATriangle")         '(12.2)
                            Else    If (a+c)<=b                    '(10)
                                        Then    Output("NotATriangle")   '(12.3)
                                        Else    Output ("Scalene")       '(11)
                                    EndIf
                        EndIf
            EndIf
    Else    If match=1                                             '(13)
                Then    If (a+c)<=b                                '(14)
                            Then    Output("NotATriangle")         '(12.4)
                            Else    Output ("Isosceles")           '(15.1)
                        EndIf
                Else    If match=2                                 '(16)
                            Then    If (a+c)<=b
                                        Then    Output("NotATriangle")   '(12.5)
                                        Else    Output ("Isosceles")     '(15.2)
                                    EndIf

                Else    If match=3                                 '(18)
                            Then    If (b+c)<=a                    '(19)
                                        Then    Output("NotATriangle")   '(12.6)
                                        Else    Output ("Isosceles")  '  '(15.3)
                                    EndIf
                            Else    Output ("Equilateral")         '(20)
                        EndIf
            EndIf
EndIf
```



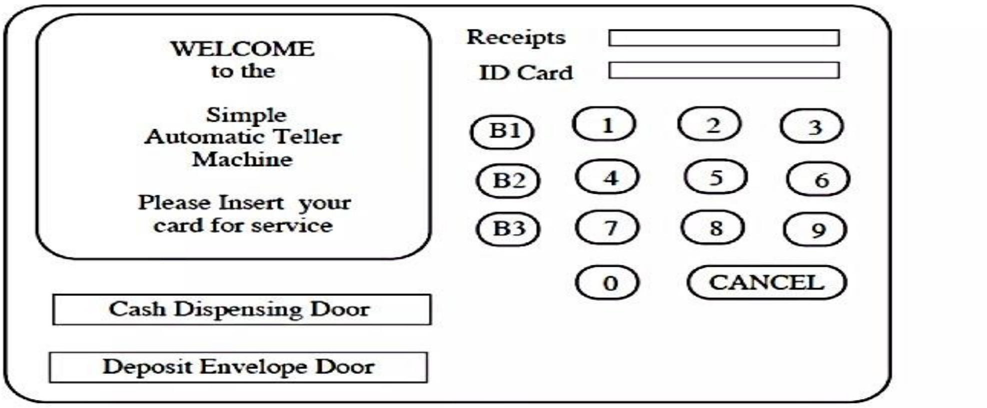
FLOWCHART FOR TRIANGLE PROBLEM

10

	(OR)			
2.	**a. With a neat diagram, explain the SATM.** **Scheme : Definition +Interface +Screens with Explanation – 2+3+5 Marks** **Solution:**	**10**	**1**	**L2**

THE SIMPLE AUTOMATIC TELLER MACHINE (SATM)

THE SATM TERMINAL

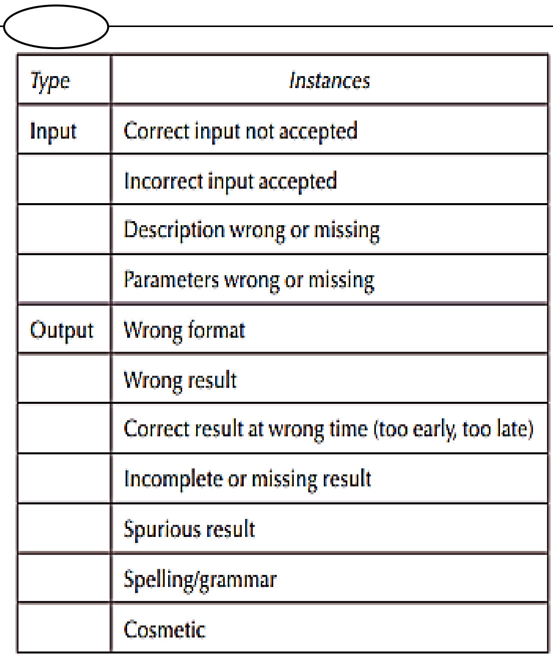


SATM PROBLEM STATEMENT CONT…

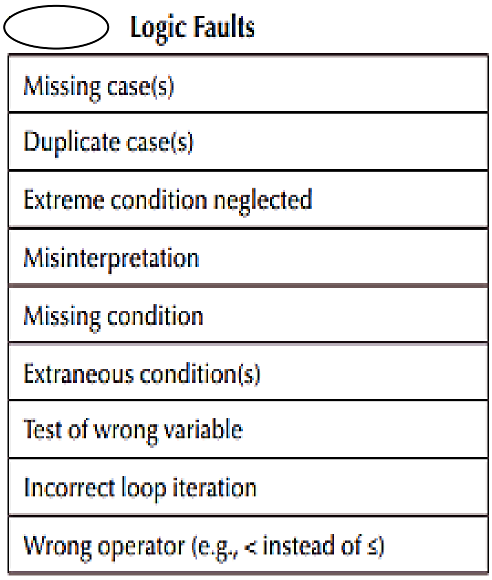
- Using a terminal with features, SATM customers can select any of the transaction types: **deposits, withdrawals, and balance inquiries**
- These transactions can be done on two types of accounts: checking and savings
- The SATM system communicates with bank customers via 15 screens (shown in figure)

Screen 1 Welcome please insert your ATM card	Screen 2 Please enter your PIN	Screen 3 Your PIN is incorrect. Please try again.
Screen 4 Invalid ATM card. It will be retained.	Screen 5 Select transaction: balance > deposit > withdrawal >	Screen 6 Balance is $dddd.dd
Screen 7 Enter amount. Withdrawals must be multiples of $10	Screen 8 Insufficient Funds! Please enter a new amount	Screen 9 Machine can only dispense $10 notes
Screen 10 Temporarily unable to process withdrawals. Another transaction?	Screen 11 Your balance is being updated. Please take cash from dispenser.	Screen 12 Temporarily unable to process deposits. Another transaction?
Screen 13 Please insert deposit into deposit slot.	Screen 14 Your new balance is being printed. Another transaction?	Screen 15 Please take your receipt and ATM card. Thank you.

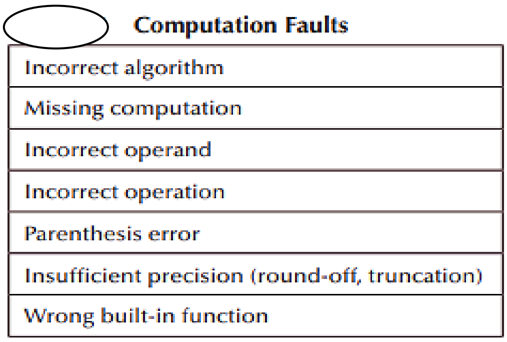
	b. **Classify the types of faults and explain each with an example.** **Scheme : Definition + Explanation – 2+2+2+2+2 marks** **Solution:**	10		



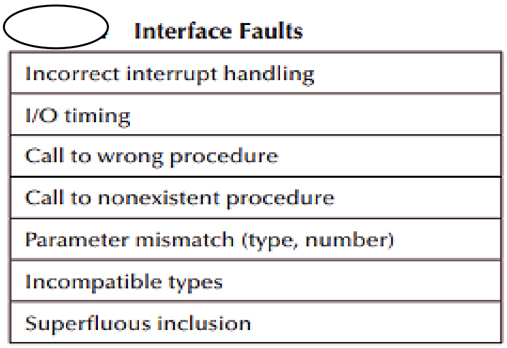
Type	Instances
Input	Correct input not accepted
	Incorrect input accepted
	Description wrong or missing
	Parameters wrong or missing
Output	Wrong format
	Wrong result
	Correct result at wrong time (too early, too late)
	Incomplete or missing result
	Spurious result
	Spelling/grammar
	Cosmetic



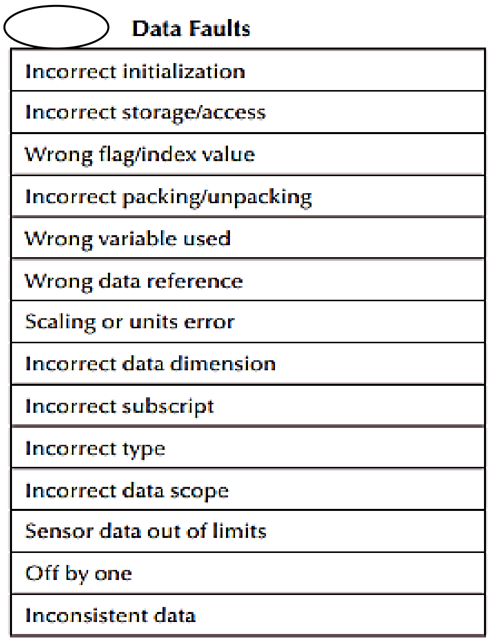
Logic Faults
Missing case(s)
Duplicate case(s)
Extreme condition neglected
Misinterpretation
Missing condition
Extraneous condition(s)
Test of wrong variable
Incorrect loop iteration
Wrong operator (e.g., < instead of ≤)



Computation Faults
Incorrect algorithm
Missing computation
Incorrect operand
Incorrect operation
Parenthesis error
Insufficient precision (round-off, truncation)
Wrong built-in function



Interface Faults
Incorrect interrupt handling
I/O timing
Call to wrong procedure
Call to nonexistent procedure
Parameter mismatch (type, number)
Incompatible types
Superfluous inclusion



Data Faults
Incorrect initialization
Incorrect storage/access
Wrong flag/index value
Incorrect packing/unpacking
Wrong variable used
Wrong data reference
Scaling or units error
Incorrect data dimension
Incorrect subscript
Incorrect type
Incorrect data scope
Sensor data out of limits
Off by one
Inconsistent data

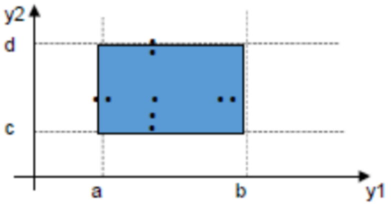
MODULE -2

3. **a. Examine boundary value analysis with the test cases using a triangle problem.**

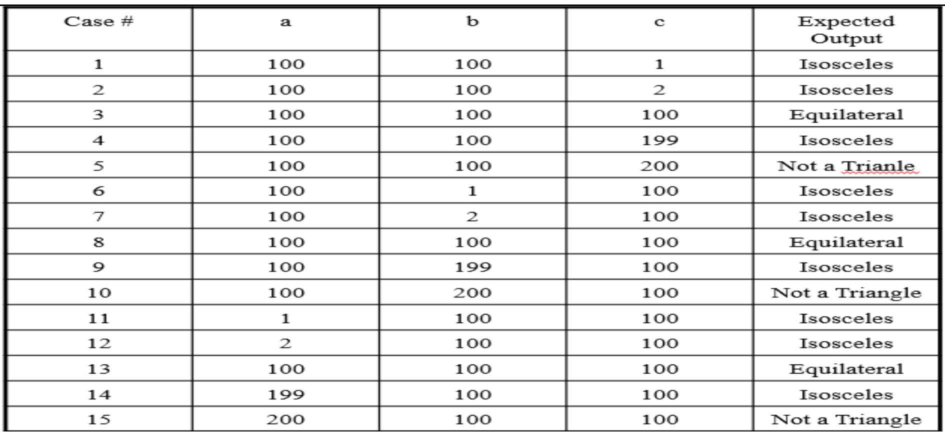
Scheme : Definition + Explanation + Test cases – 2+3+5 marks

Solution:

The basic idea in boundary value analysis is to select input variable values at their: Minimum, Just above the minimum, A nominal value, Just below the maximum and Maximum



10 **2** **L2**

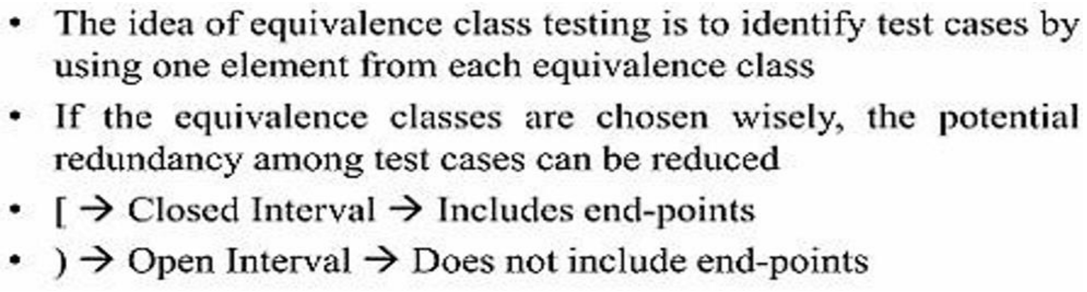


Case #	a	b	c	Expected Output
1	100	100	1	Isosceles
2	100	100	2	Isosceles
3	100	100	100	Equilateral
4	100	100	199	Isosceles
5	100	100	200	Not a Trianle
6	100	1	100	Isosceles
7	100	2	100	Isosceles
8	100	100	100	Equilateral
9	100	199	100	Isosceles
10	100	200	100	Not a Triangle
11	1	100	100	Isosceles
12	2	100	100	Isosceles
13	100	100	100	Equilateral
14	199	100	100	Isosceles
15	200	100	100	Not a Triangle

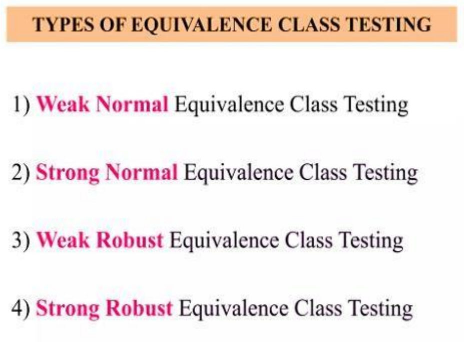
b. Examine the equivalence class testing. Examine the equivalence class test cases for the nextdate function.

Scheme : Definition + Explanation + Test cases – 2+3+5 marks

Solution:



- The idea of equivalence class testing is to identify test cases by using one element from each equivalence class
- If the equivalence classes are chosen wisely, the potential redundancy among test cases can be reduced
- [→ Closed Interval → Includes end-points
-) → Open Interval → Does not include end-points



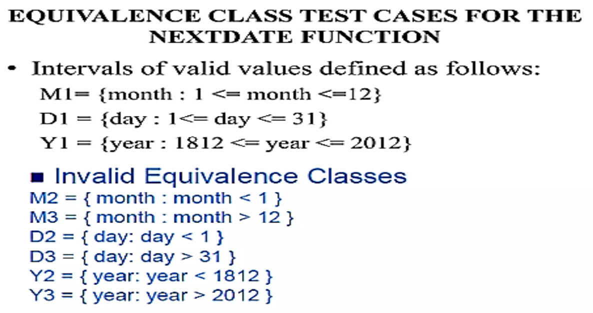
TYPES OF EQUIVALENCE CLASS TESTING

1) Weak Normal Equivalence Class Testing

2) Strong Normal Equivalence Class Testing

3) Weak Robust Equivalence Class Testing

4) Strong Robust Equivalence Class Testing



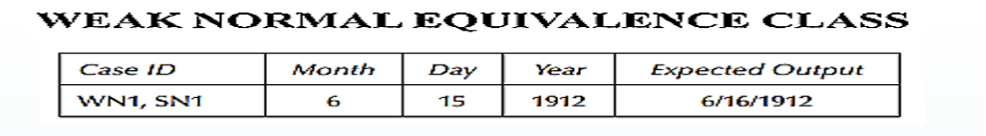
EQUIVALENCE CLASS TEST CASES FOR THE NEXTDATE FUNCTION

- Intervals of valid values defined as follows:

M1= {month : 1 <= month <=12}
D1 = {day : 1<= day <= 31}
Y1 = {year : 1812 <= year <= 2012}

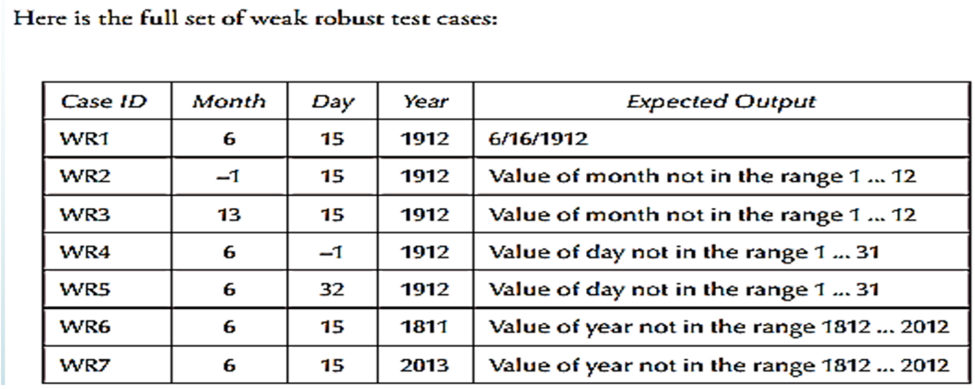
- Invalid Equivalence Classes

M2 = { month : month < 1 }
M3 = { month : month > 12 }
D2 = { day: day < 1 }
D3 = { day: day > 31 }
Y2 = { year: year < 1812 }
Y3 = { year: year > 2012 }



WEAK NORMAL EQUIVALENCE CLASS

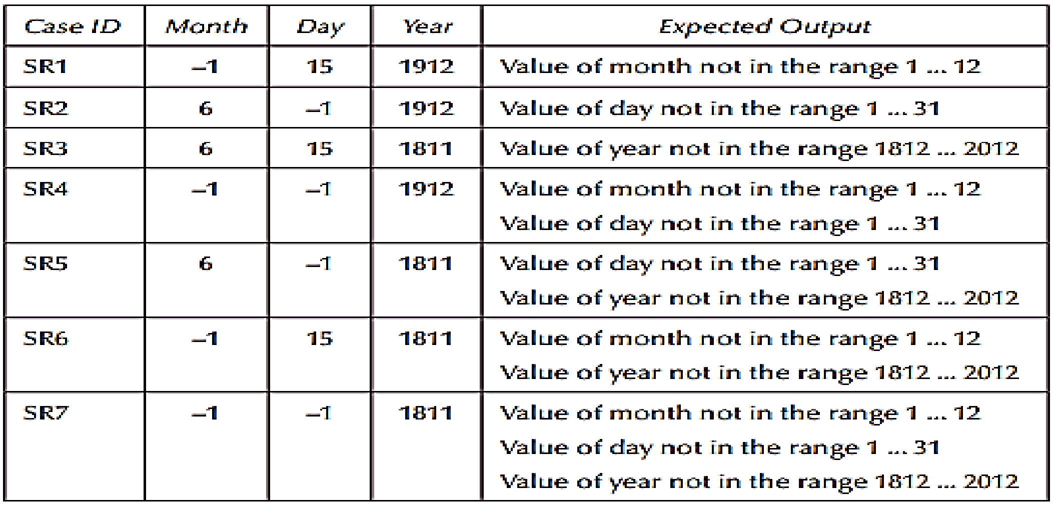
Case ID	Month	Day	Year	Expected Output
WN1, SN1	6	15	1912	6/16/1912



Here is the full set of weak robust test cases:

Case ID	Month	Day	Year	Expected Output
WR1	6	15	1912	6/16/1912
WR2	–1	15	1912	Value of month not in the range 1 ... 12
WR3	13	15	1912	Value of month not in the range 1 ... 12
WR4	6	–1	1912	Value of day not in the range 1 ... 31
WR5	6	32	1912	Value of day not in the range 1 ... 31
WR6	6	15	1811	Value of year not in the range 1812 ... 2012
WR7	6	15	2013	Value of year not in the range 1812 ... 2012

10 | 2 | L2



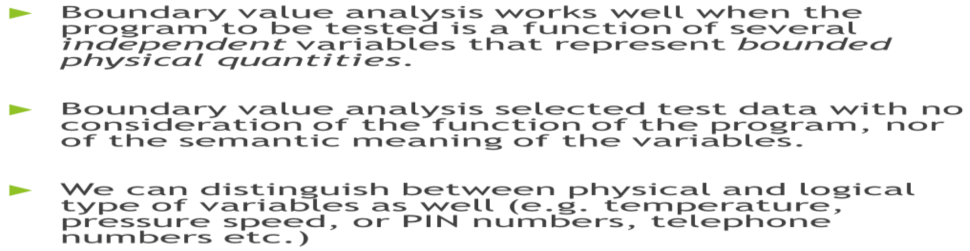
Case ID	Month	Day	Year	Expected Output
SR1	–1	15	1912	Value of month not in the range 1 ... 12
SR2	6	–1	1912	Value of day not in the range 1 ... 31
SR3	6	15	1811	Value of year not in the range 1812 ... 2012
SR4	–1	–1	1912	Value of month not in the range 1 ... 12 Value of day not in the range 1 ... 31
SR5	6	–1	1811	Value of day not in the range 1 ... 31 Value of year not in the range 1812 ... 2012
SR6	–1	15	1811	Value of month not in the range 1 ... 12 Value of year not in the range 1812 ... 2012
SR7	–1	–1	1811	Value of month not in the range 1 ... 12 Value of day not in the range 1 ... 31 Value of year not in the range 1812 ... 2012

(OR)

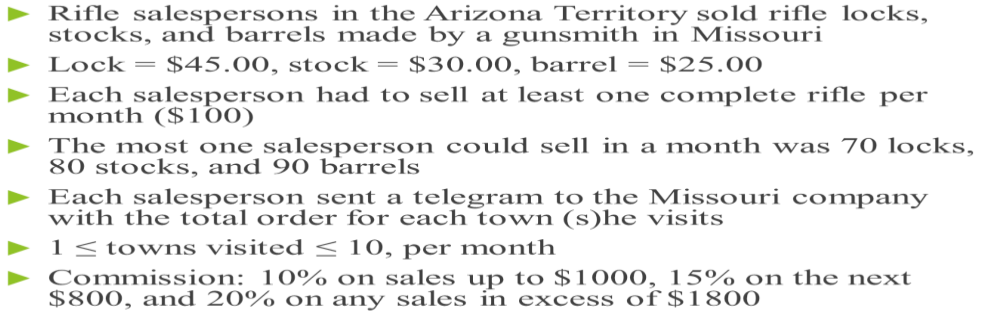
4. **a. What are the limitations of boundary value analysis and examine the test cases using boundary value analysis testing for commission problem.**

Scheme : Limitations + Explanation + Test cases – 2+3+5 marks

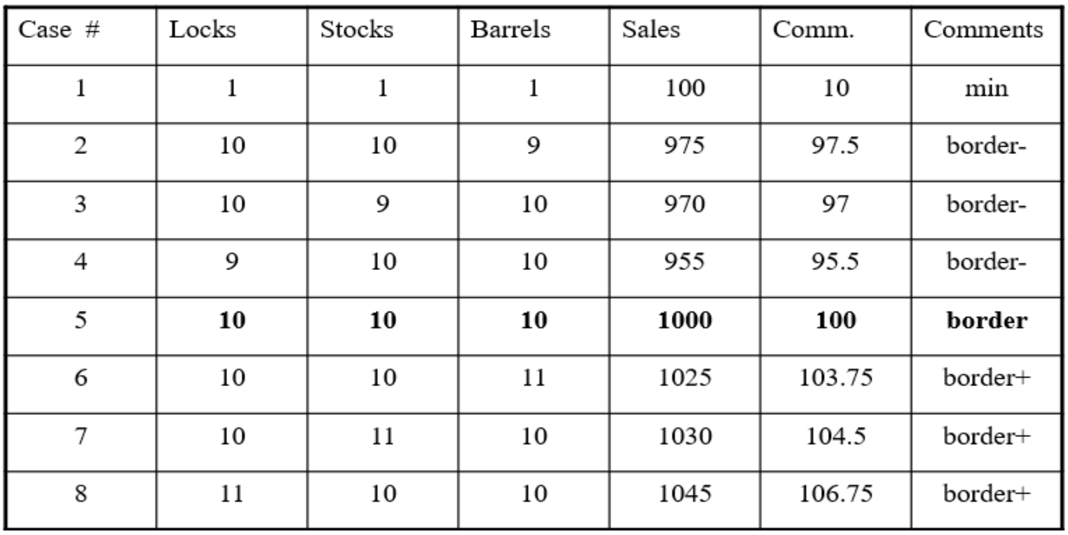
Solution:



- Boundary value analysis works well when the program to be tested is a function of several *independent* variables that represent *bounded physical quantities*.
- Boundary value analysis selected test data with no consideration of the function of the program, nor of the semantic meaning of the variables.
- We can distinguish between physical and logical type of variables as well (e.g. temperature, pressure speed, or PIN numbers, telephone numbers etc.)



- Rifle salespersons in the Arizona Territory sold rifle locks, stocks, and barrels made by a gunsmith in Missouri
- Lock = $45.00, stock = $30.00, barrel = $25.00
- Each salesperson had to sell at least one complete rifle per month ($100)
- The most one salesperson could sell in a month was 70 locks, 80 stocks, and 90 barrels
- Each salesperson sent a telegram to the Missouri company with the total order for each town (s)he visits
- 1 ≤ towns visited ≤ 10, per month
- Commission: 10% on sales up to $1000, 15% on the next $800, and 20% on any sales in excess of $1800



Case #	Locks	Stocks	Barrels	Sales	Comm.	Comments
1	1	1	1	100	10	min
2	10	10	9	975	97.5	border-
3	10	9	10	970	97	border-
4	9	10	10	955	95.5	border-
5	**10**	**10**	**10**	**1000**	**100**	**border**
6	10	10	11	1025	103.75	border+
7	10	11	10	1030	104.5	border+
8	11	10	10	1045	106.75	border+

10 **2** **L2**

b. Explain the format of the decision table. Build a decision table for a simple version of the triangle problem.

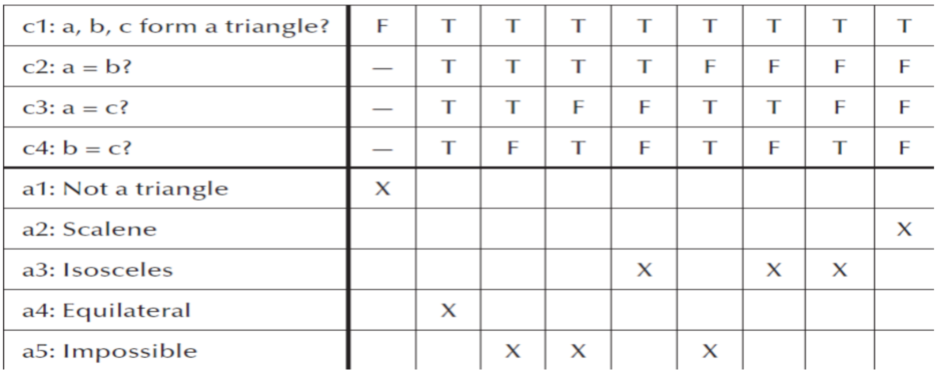
Scheme: Definition + Explanation + Test cases+ Decision Table – 2+3+3+2 marks

Solution:

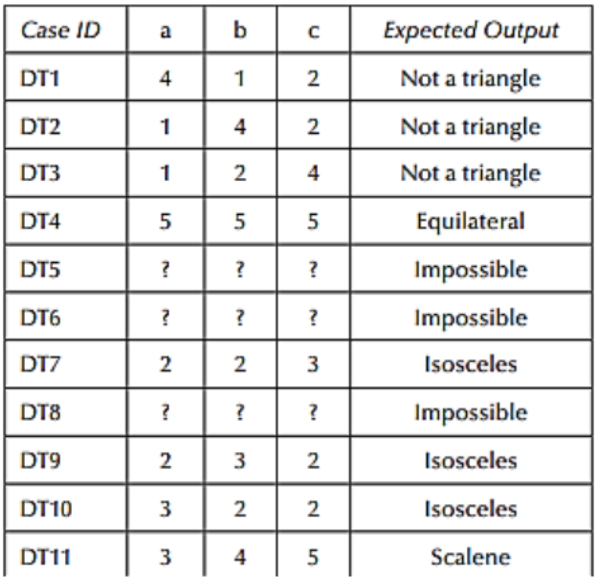
Decision Table Techniques

- To identify test cases with decision tables, we interpret conditions as inputs and actions as outputs. Sometimes conditions end up referring to equivalence classes of inputs, and actions refer to major functional processing portions of the item tested.
- The **rules are then interpreted as test cases**.
- Decision table have some assurance that we **will have a comprehensive set of test cases.** Several techniques that produce decision tables are more useful to testers.
- One helpful style is to add an action to show when a **rule is logically impossible**.
- In the decision table in Table 7.2, we see examples of don't care entries and impossible rule usage. If the integers a, b, and c do not constitute a triangle, we do not even care about possible equalities, as indicated in the first rule.
- In rules 3, 4, and 6, if two pairs of integers are equal, by transitivity, the third pair must be equal; thus, the negative entry makes these rules impossible.

Table 7.2 Decision Table for Triangle Problem



c1: a, b, c form a triangle?	F	T	T	T	T	T	T	T	T
c2: a = b?	—	T	T	T	T	F	F	F	F
c3: a = c?	—	T	T	F	F	T	T	F	F
c4: b = c?	—	T	F	T	F	T	F	T	F
a1: Not a triangle	X								
a2: Scalene									X
a3: Isosceles					X		X	X	
a4: Equilateral		X							
a5: Impossible			X	X		X			



Case ID	a	b	c	*Expected Output*
DT1	4	1	2	Not a triangle
DT2	1	4	2	Not a triangle
DT3	1	2	4	Not a triangle
DT4	5	5	5	Equilateral
DT5	?	?	?	Impossible
DT6	?	?	?	Impossible
DT7	2	2	3	Isosceles
DT8	?	?	?	Impossible
DT9	2	3	2	Isosceles
DT10	3	2	2	Isosceles
DT11	3	4	5	Scalene

10 **2** **L2**

MODULE -3

5. **a. Define a program graph. Draw a program graph of the commission problem.**

Scheme: Definition + Code + Program graph + Test Cases – 2+4+4 marks

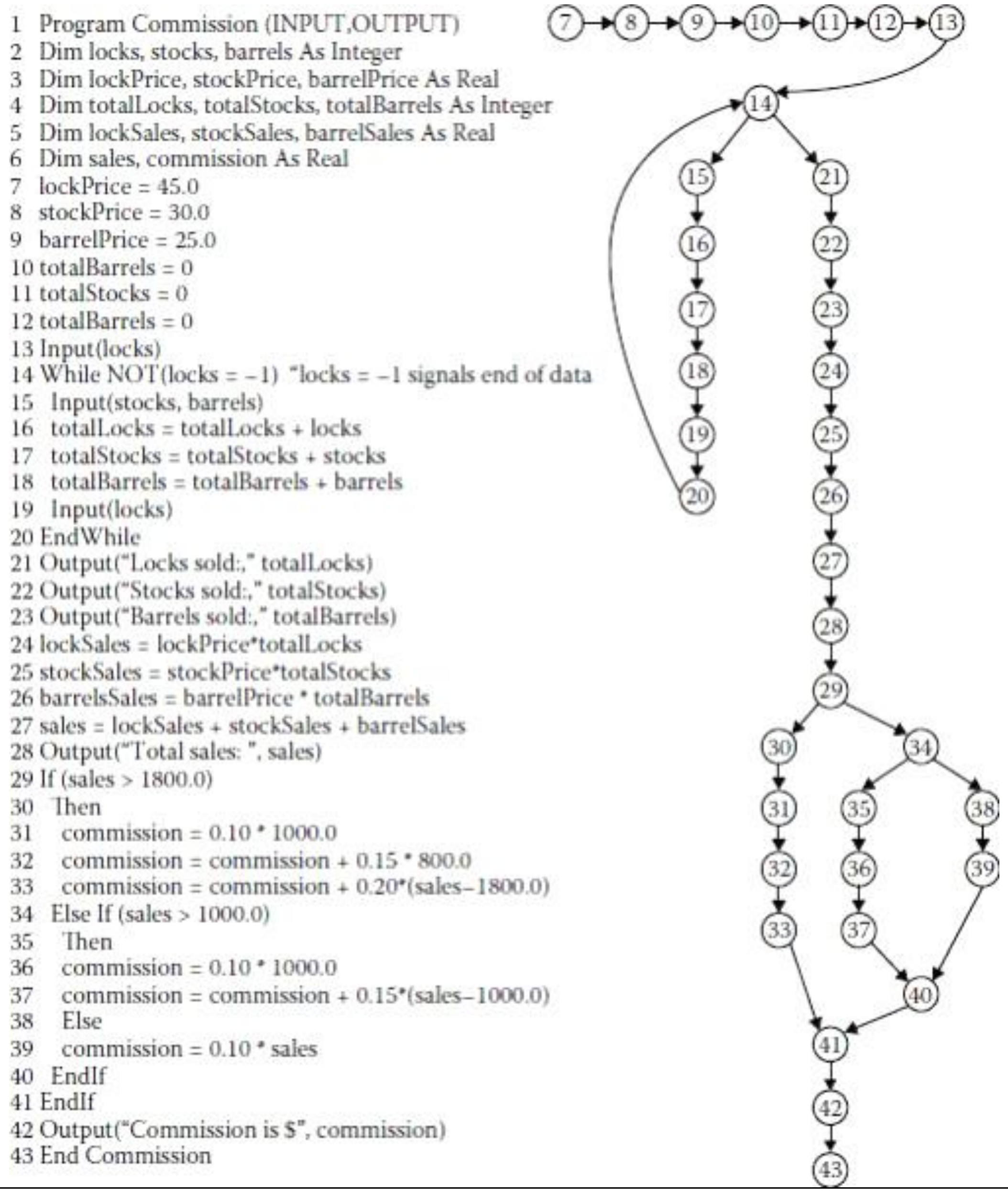
Solution:

The program graph G(P) is constructed with statement fragments as nodes and edges that represent node sequences.

- **Definition:**

  Node n ∈ G(P) is a <u>defining node</u> of the variable v ∈ V, written as DEF(v, n), if and only if the value of variable v is defined as the statement fragment corresponding to node n.
- Input statements, assignment statements, loop control statements, and procedure calls are all examples of statements that are defining nodes. When the code corresponding to such statements executes, the contents of the memory location(s) associated with the variables are changed.
- **Definition:**

  Node n ∈ G(P) is a <u>usage node</u> of the variable v ∈ V, written as USE(v, n), if and only if the value of the variable v is used as the statement fragment corresponding to node n.
- Output statements, assignment statements, conditional statements, loop control statements, and procedure calls are all examples of statements that are usage nodes. When the code corresponding to such statements executes, the contents of the memory location(s) associated with the variables remain unchanged.



```
1  Program Commission (INPUT,OUTPUT)
2  Dim locks, stocks, barrels As Integer
3  Dim lockPrice, stockPrice, barrelPrice As Real
4  Dim totalLocks, totalStocks, totalBarrels As Integer
5  Dim lockSales, stockSales, barrelSales As Real
6  Dim sales, commission As Real
7  lockPrice = 45.0
8  stockPrice = 30.0
9  barrelPrice = 25.0
10 totalBarrels = 0
11 totalStocks = 0
12 totalBarrels = 0
13 Input(locks)
14 While NOT(locks = −1)  ‘locks = −1 signals end of data
15   Input(stocks, barrels)
16   totalLocks = totalLocks + locks
17   totalStocks = totalStocks + stocks
18   totalBarrels = totalBarrels + barrels
19   Input(locks)
20 EndWhile
21 Output(“Locks sold:,” totalLocks)
22 Output(“Stocks sold:,” totalStocks)
23 Output(“Barrels sold:,” totalBarrels)
24 lockSales = lockPrice*totalLocks
25 stockSales = stockPrice*totalStocks
26 barrelsSales = barrelPrice * totalBarrels
27 sales = lockSales + stockSales + barrelSales
28 Output(“Total sales: ”, sales)
29 If (sales > 1800.0)
30   Then
31     commission = 0.10 * 1000.0
32     commission = commission + 0.15 * 800.0
33     commission = commission + 0.20*(sales−1800.0)
34   Else If (sales > 1000.0)
35     Then
36     commission = 0.10 * 1000.0
37     commission = commission + 0.15*(sales−1000.0)
38     Else
39     commission = 0.10 * sales
40   EndIf
41 EndIf
42 Output(“Commission is $”, commission)
43 End Commission
```

		10	3	L3

b. Define DD-path. Explain basis path testing with a suitable example.
Scheme: Definition + Basis path with example + Test Cases – 2+6+2 marks
Solution:

The reason that program graphs play such an important role in structural testingis due to the fact that they form the basis of a number of testing methods,including one based on a construct known as decision-to-decision paths (morecommonly referred to as DD-Paths).

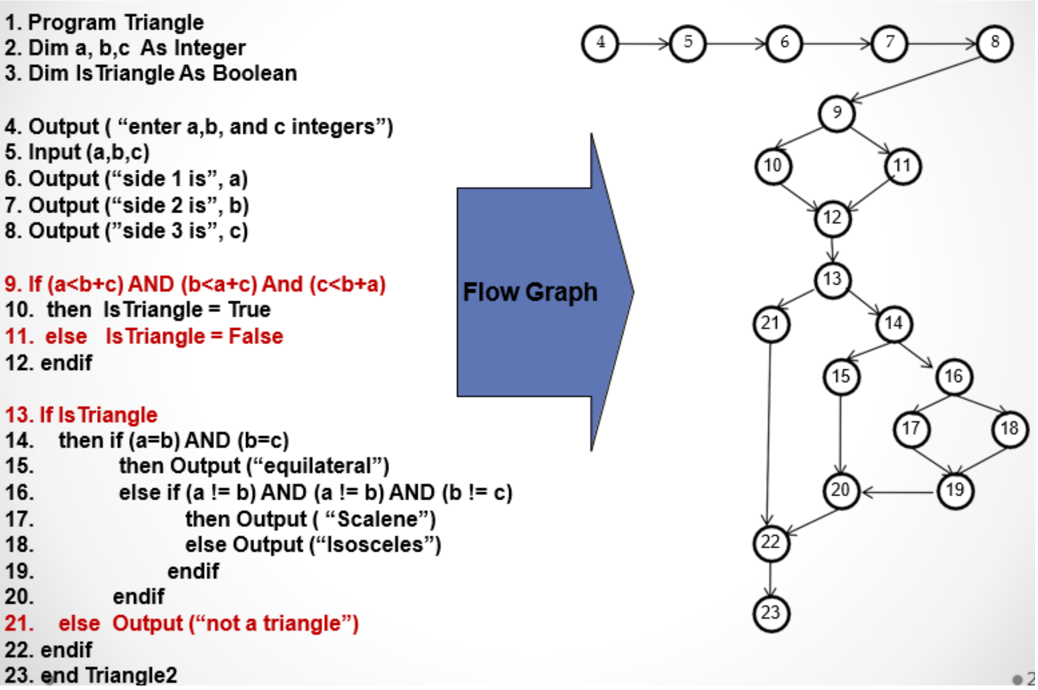
The idea is to use DD-Paths to create acondensation graph of a piece of software's program graph, in which a numberof constructs are collapsed into single nodes known as DD-Paths.

The definitions ofeach different type of DD-Path that a chain can be reduced to are given asfollows:

Type 1: A single node with an in-degree = 0.
Type 2: A single node with an out-degree = 0.
Type 3: A single node with in-degree >= 2 or out-degree >= 2.
Type 4: A single node with in-degree = 1 and out-degree = 1.
Type 5: The chain is of a maximal length >= 1.

All programs must have an entry and an exit and so every program graphmust have a source and sink node.

10 | 3 | L2



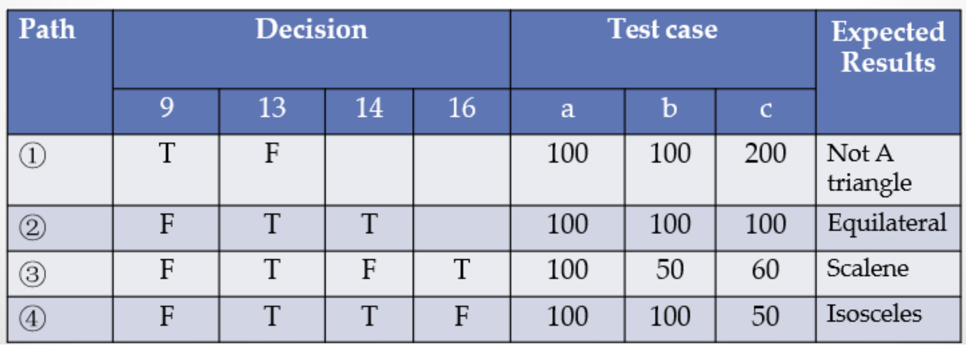
```
1. Program Triangle
2. Dim a, b,c  As Integer
3. Dim IsTriangle As Boolean

4. Output ( "enter a,b, and c integers")
5. Input (a,b,c)
6. Output ("side 1 is", a)
7. Output ("side 2 is", b)
8. Output ("side 3 is", c)

9. If (a<b+c) AND (b<a+c) And (c<b+a)
10.  then  IsTriangle = True
11.  else   IsTriangle = False
12. endif

13. If IsTriangle
14.    then if (a=b) AND (b=c)
15.             then Output ("equilateral")
16.             else if (a != b) AND (a != b) AND (b != c)
17.                        then Output ( "Scalene")
18.                        else Output ("Isosceles")
19.                  endif
20.          endif
21.    else  Output ("not a triangle")
22. endif
23. end Triangle2
```

Flow Graph
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
2



Path	Decision				Test case			Expected Results
	9	13	14	16	a	b	c	
①	T	F			100	100	200	Not A triangle
②	F	T	T		100	100	100	Equilateral
③	F	T	F	T	100	50	60	Scalene
④	F	T	T	F	100	100	50	Isosceles

(OR)

6. **a. Define predicate node, du-paths, dc-path. Give du-path for lock, stock and sales for commission problem.**
Scheme: Definition +DU path Table + Test Cases – 2+6+2 marks
Solution:

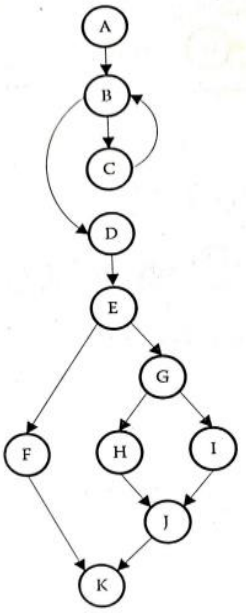
10 | 3 | L3

A usage node USE(v,n) is a ***predicate use*** (denoted as P-use), iff the statement n is a predicate statement; otherwise USE(v,n) is a ***computation use***, (denoted C-use)

- Nodes corresponding to predicate uses always have an outdegree ≥ 2
- Nodes corresponding to computation uses always have outdegree ≤ 1

A ***definition-clear (sub) path*** with respect to a variable v (denoted dc-path) is a definition-use(sub) path in PATHS(P) with initial and final nodes DEF(v,m) & USE(v,n) such that no other node in the (sub) path is a defining node of v

A ***definition-use (sub) path*** with respect to a variable v (denoted du-path) is a (sub) path in PATHS(P) such that for some v ∈ V, there are define and usage nodes DEF(v,m) & USE(v,n) such that m & n are the initial and final nodes of the (sub) path.



DD-Path graph of the commission program.

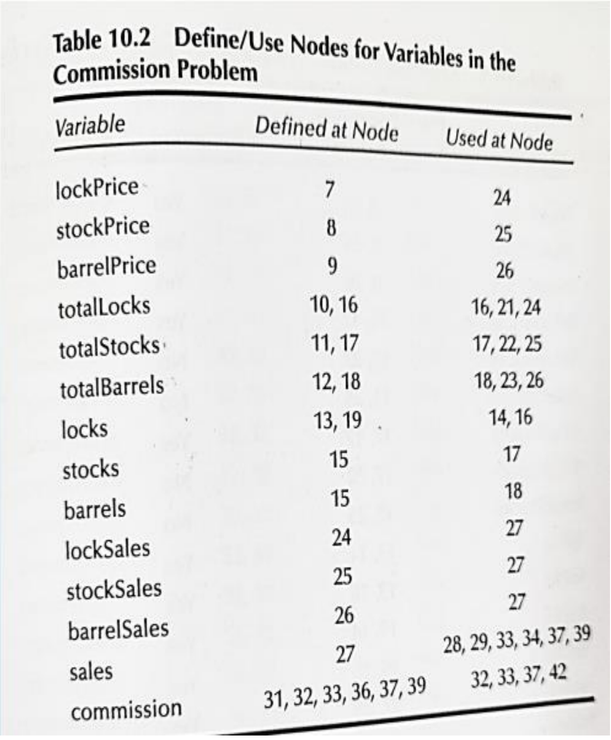
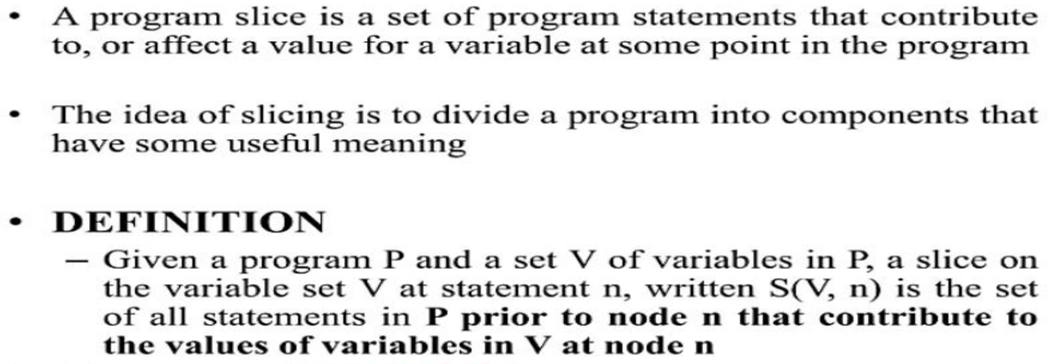


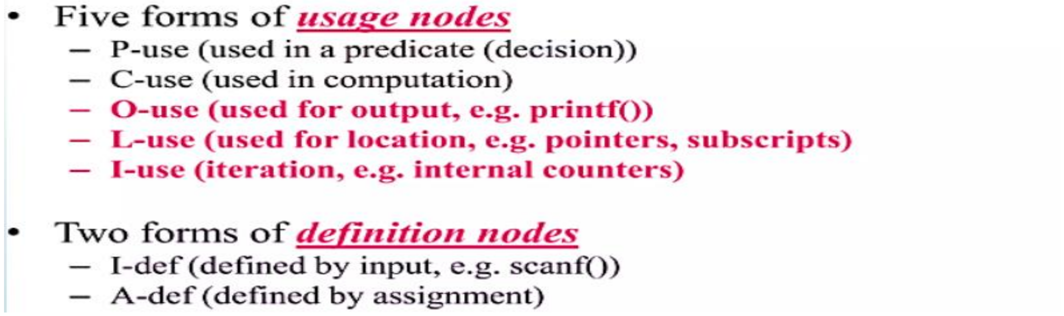
Table 10.2 Define/Use Nodes for Variables in the Commission Problem

Variable	*Defined at Node*	*Used at Node*
lockPrice	7	24
stockPrice	8	25
barrelPrice	9	26
totalLocks	10, 16	16, 21, 24
totalStocks	11, 17	17, 22, 25
totalBarrels	12, 18	18, 23, 26
locks	13, 19	14, 16
stocks	15	17
barrels	15	18
lockSales	24	27
stockSales	25	27
barrelSales	26	27
sales	27	28, 29, 33, 34, 37, 39
commission	31, 32, 33, 36, 37, 39	32, 33, 37, 42

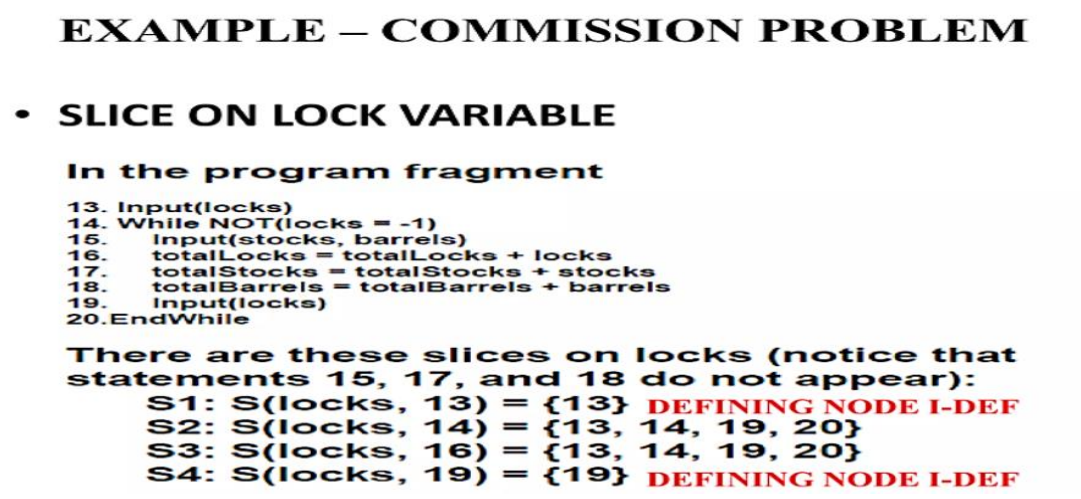
	b. Explain slice-based testing with an example. **Scheme : Definition + explanation + example – 2+5+3 marks** **Solution:**	**10**	**3**	**L2**



- A program slice is a set of program statements that contribute to, or affect a value for a variable at some point in the program
- The idea of slicing is to divide a program into components that have some useful meaning
- **DEFINITION**
  - Given a program P and a set V of variables in P, a slice on the variable set V at statement n, written S(V, n) is the set of all statements in **P prior to node n that contribute to the values of variables in V at node n**



- Five forms of ***usage nodes***
  - P-use (used in a predicate (decision))
  - C-use (used in computation)
  - **O-use (used for output, e.g. printf())**
  - **L-use (used for location, e.g. pointers, subscripts)**
  - **I-use (iteration, e.g. internal counters)**
- Two forms of ***definition nodes***
  - I-def (defined by input, e.g. scanf())
  - A-def (defined by assignment)



EXAMPLE – COMMISSION PROBLEM

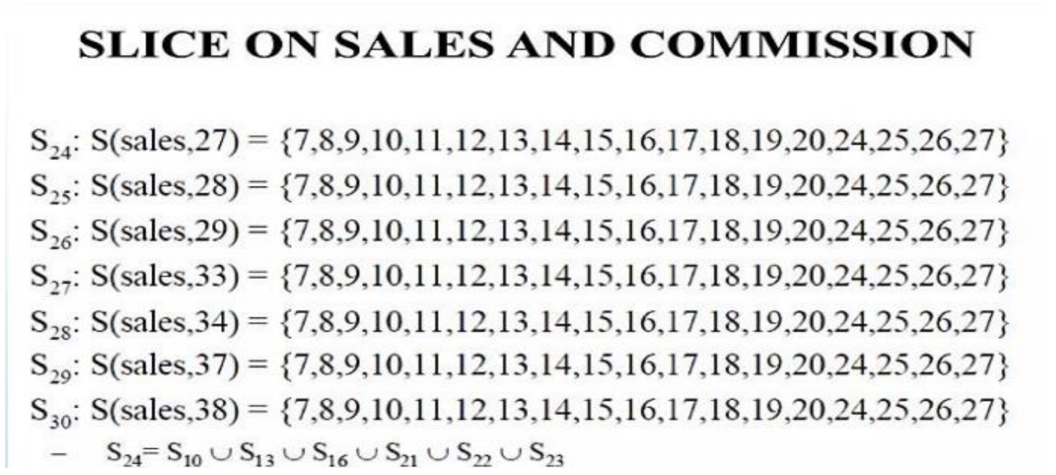
- **SLICE ON LOCK VARIABLE**

In the program fragment

```
13. Input(locks)
14. While NOT(locks = -1)
15.     Input(stocks, barrels)
16.     totalLocks = totalLocks + locks
17.     totalStocks = totalStocks + stocks
18.     totalBarrels = totalBarrels + barrels
19.     Input(locks)
20.EndWhile
```

There are these slices on locks (notice that statements 15, 17, and 18 do not appear):

S1: S(locks, 13) = {13} DEFINING NODE I-DEF
S2: S(locks, 14) = {13, 14, 19, 20}
S3: S(locks, 16) = {13, 14, 19, 20}
S4: S(locks, 19) = {19} DEFINING NODE I-DEF



SLICE ON SALES AND COMMISSION

S_{24}: S(sales,27) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{25}: S(sales,28) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{26}: S(sales,29) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{27}: S(sales,33) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{28}: S(sales,34) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{29}: S(sales,37) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{30}: S(sales,38) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

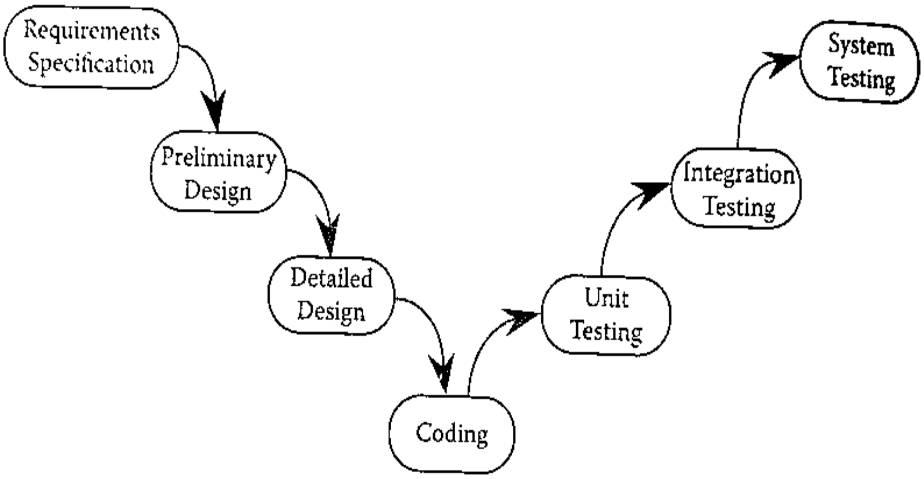
– $S_{24} = S_{10} \cup S_{13} \cup S_{16} \cup S_{21} \cup S_{22} \cup S_{23}$

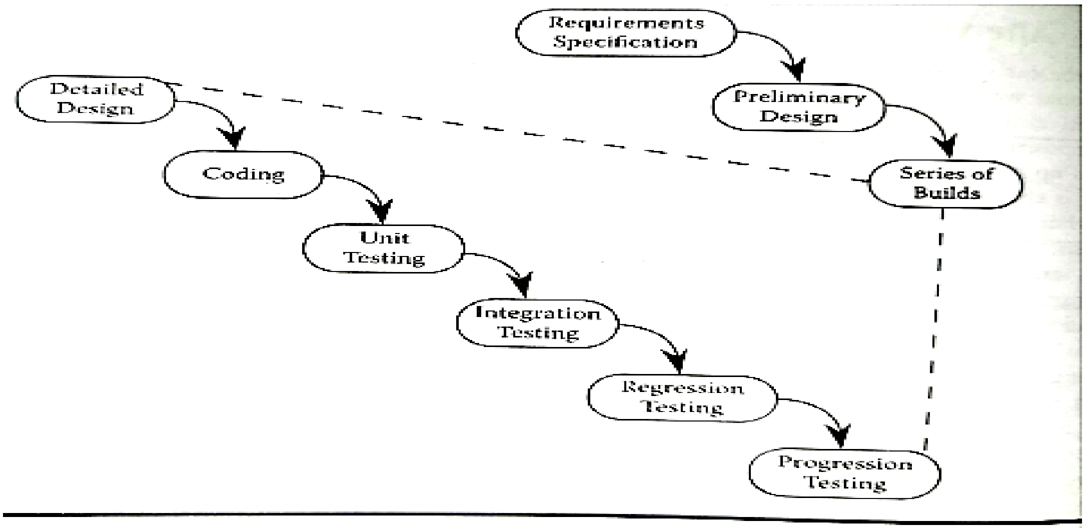
<table>
<tr><td colspan="5"><b><u>MODULE -4</u></b></td></tr>
<tr>
<td>7.</td>
<td>

a. Examine the traditional view of testing levels, alternate life cycle model.

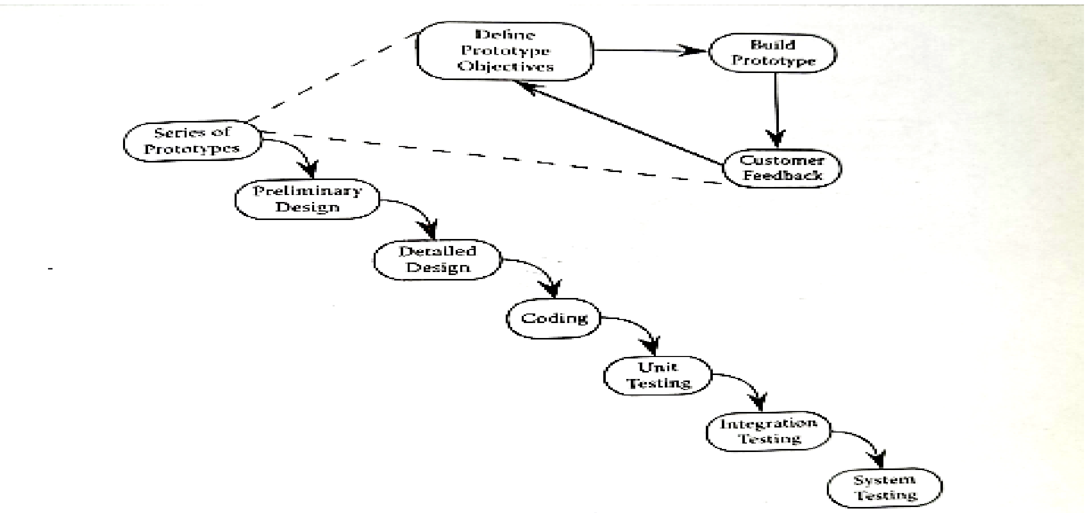
Scheme : Definition + diagram with explanation for each – 2+8 marks

Solution:

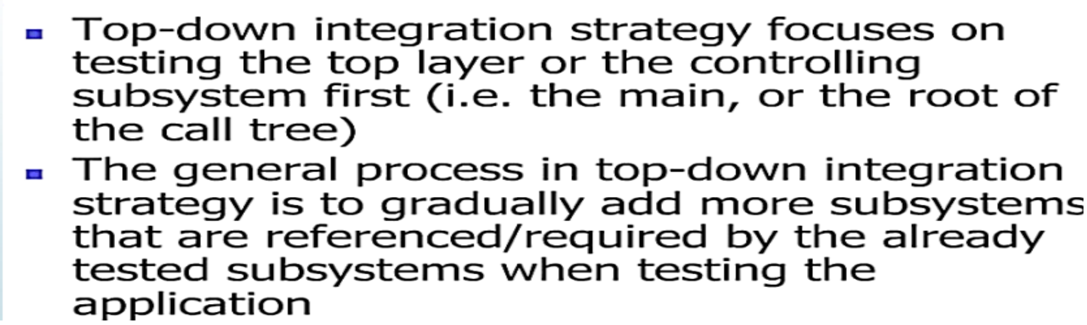
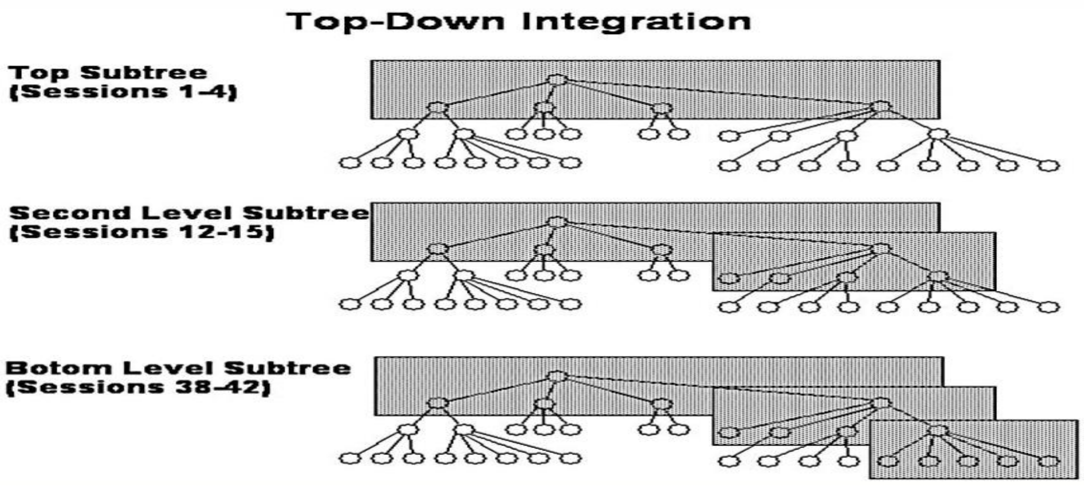
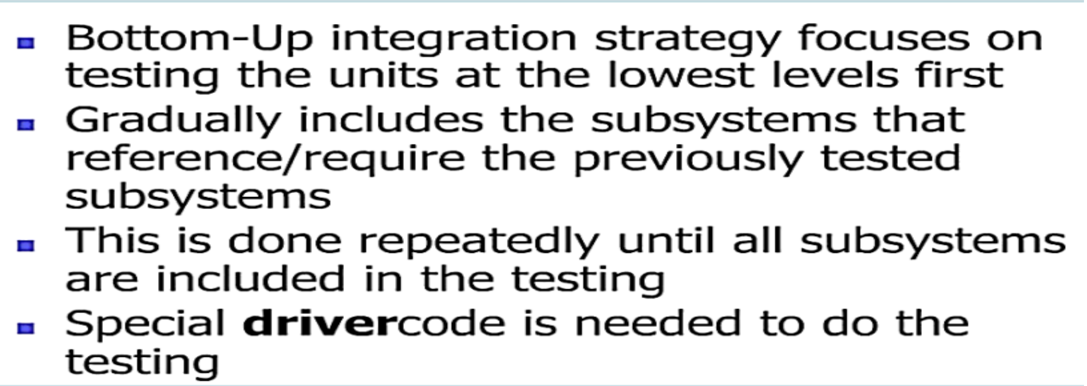
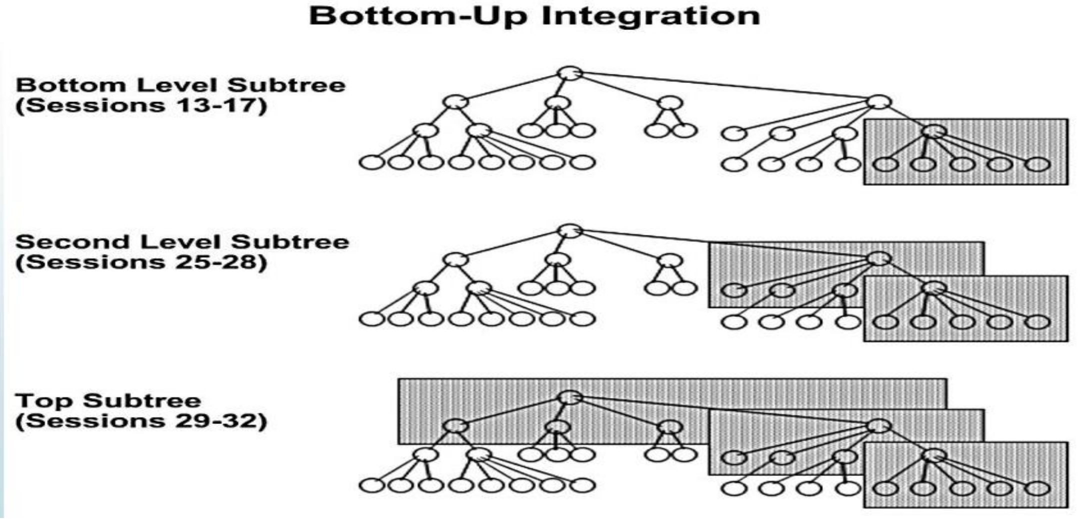


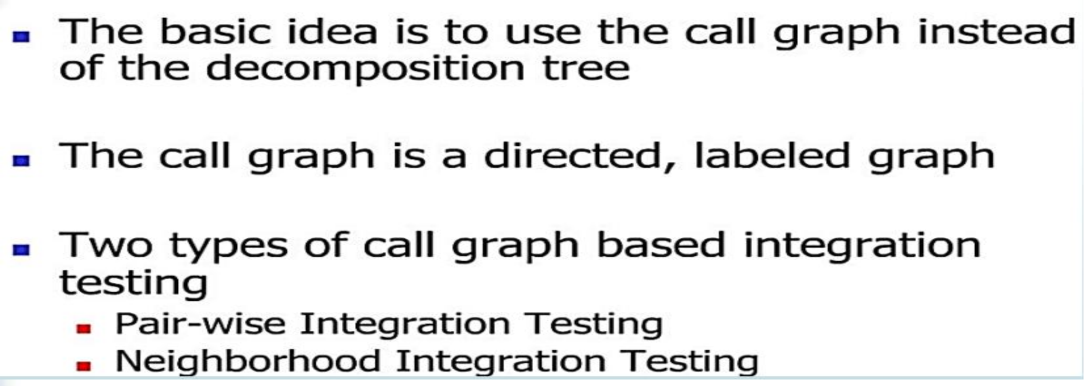
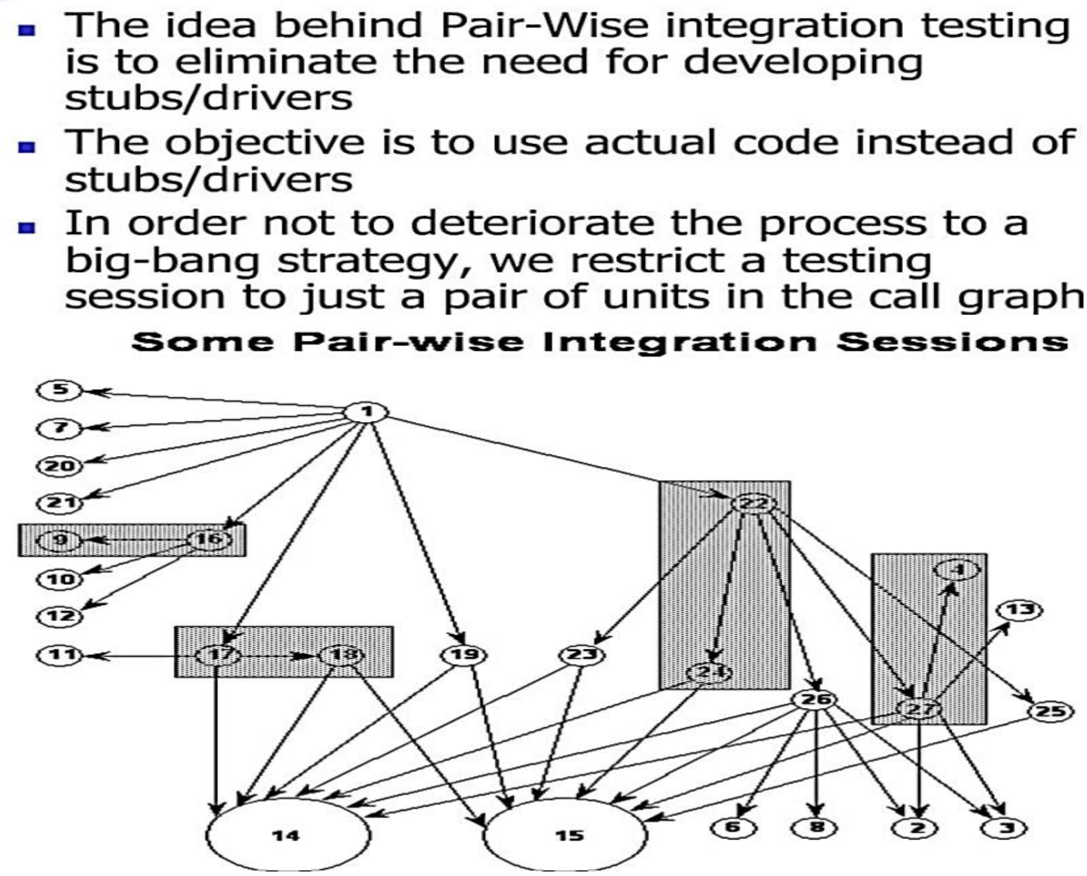
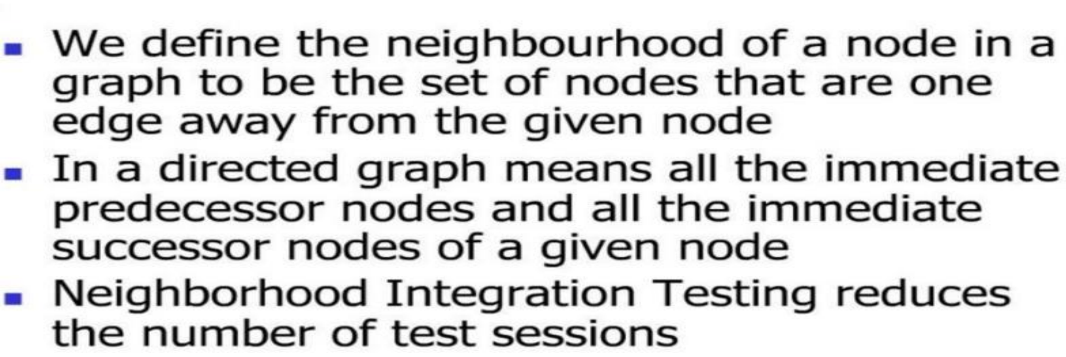


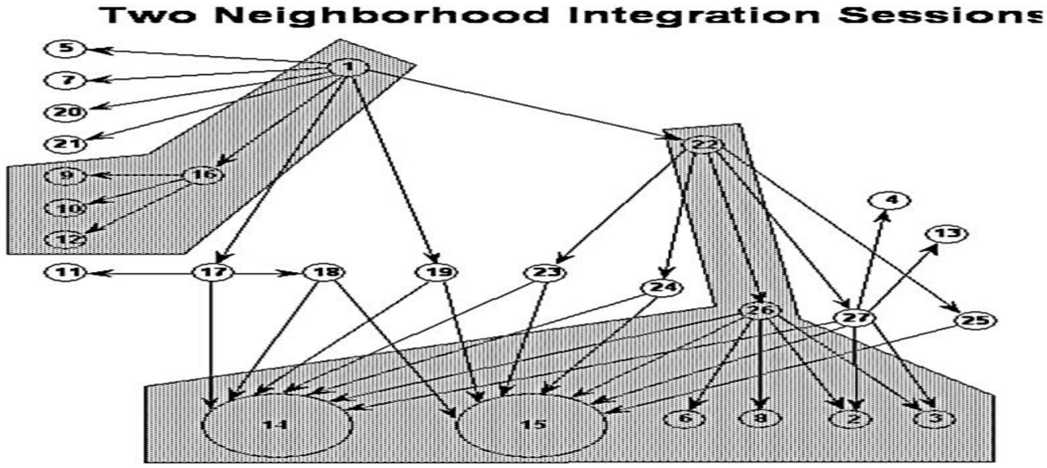
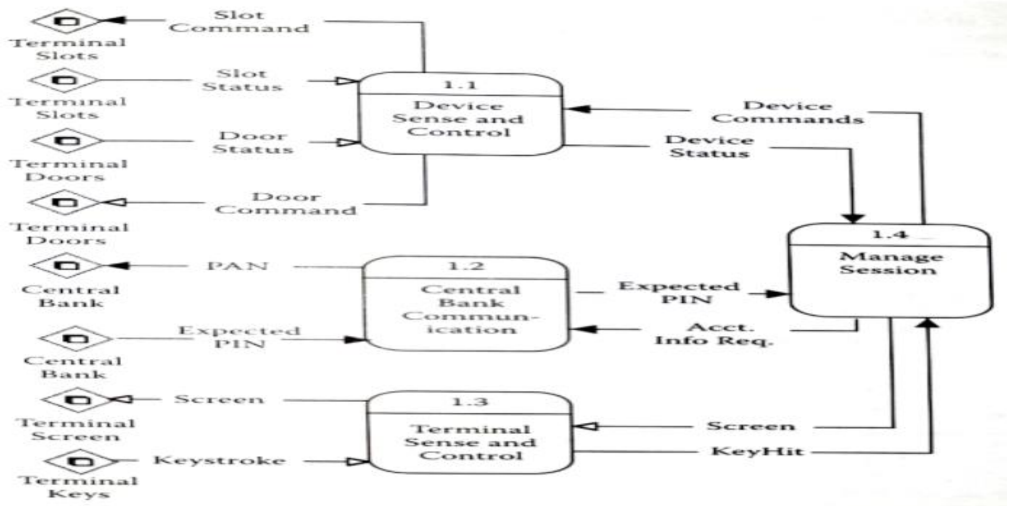
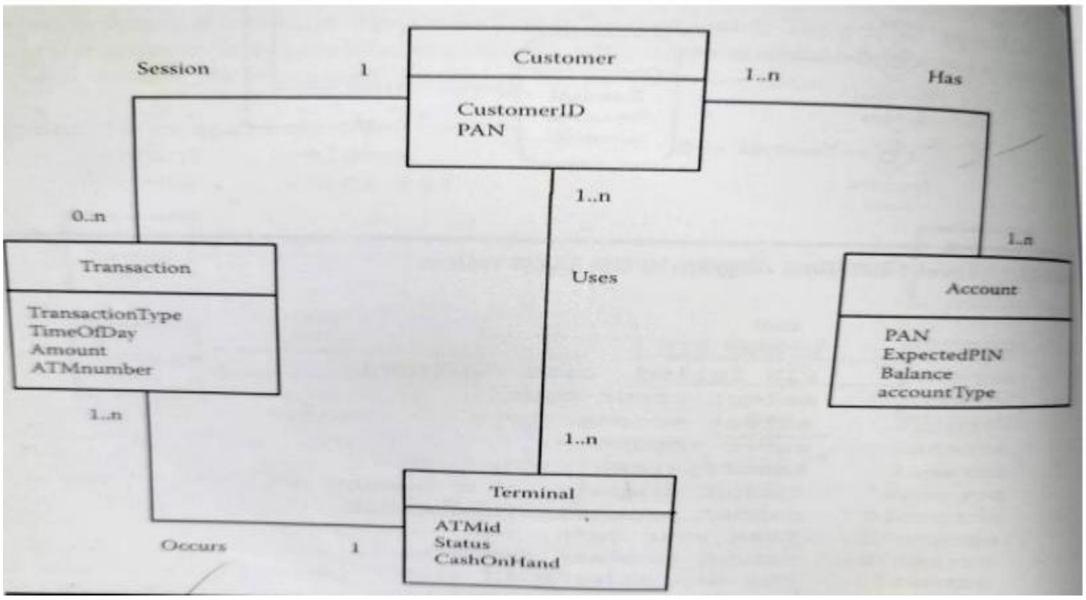
Life cycle with a build sequence.



</td>
<td><b>10</b></td>
<td><b>4</b></td>
<td><b>L2</b></td>
</tr>
</table>

	b. Compare top-down and bottom-up integration strategies. **Scheme : Definition with explanation and example for each – 5+5 marks** **Solution:**  - Top-down integration strategy focuses on testing the top layer or the controlling subsystem first (i.e. the main, or the root of the call tree) - The general process in top-down integration strategy is to gradually add more subsystems that are referenced/required by the already tested subsystems when testing the application  Top-Down Integration Top Subtree (Sessions 1-4) Second Level Subtree (Sessions 12-15) Botom Level Subtree (Sessions 38-42)  - Bottom-Up integration strategy focuses on testing the units at the lowest levels first - Gradually includes the subsystems that reference/require the previously tested subsystems - This is done repeatedly until all subsystems are included in the testing - Special **driver**code is needed to do the testing  Bottom-Up Integration Bottom Level Subtree (Sessions 13-17) Second Level Subtree (Sessions 25-28) Top Subtree (Sessions 29-32)	**10**	**4**	**L2**

(OR)				
8.	**a. Formulate call graph based integration with the help of i) Pairwise Integration ii) Neighborhood integration.** **Scheme : Definition with explanation and example for each – 5+5 marks** **Solution:**  - The basic idea is to use the call graph instead of the decomposition tree - The call graph is a directed, labeled graph - Two types of call graph based integration testing - Pair-wise Integration Testing - Neighborhood Integration Testing  - The idea behind Pair-Wise integration testing is to eliminate the need for developing stubs/drivers - The objective is to use actual code instead of stubs/drivers - In order not to deteriorate the process to a big-bang strategy, we restrict a testing session to just a pair of units in the call graph Some Pair-wise Integration Sessions 1 5 7 20 21 9 16 10 12 11 17 18 19 23 22 24 26 27 4 13 25 14 15 6 8 2 3  - We define the neighbourhood of a node in a graph to be the set of nodes that are one edge away from the given node - In a directed graph means all the immediate predecessor nodes and all the immediate successor nodes of a given node - Neighborhood Integration Testing reduces the number of test sessions	**10**	**4**	**L2**

	 Two Neighborhood Integration Sessions 1 5 7 20 21 9 16 10 12 11 17 18 19 23 22 24 4 13 26 27 25 14 15 6 8 2 3			
	b. Define the SATM system. Demonstrate the entity/relationship model of the SATM system. **Scheme : Definition with explanation and example for each – 5+5 marks** **Solution:**  Slot Command Terminal Slots Slot Status Terminal Slots Door Status Terminal Doors Door Command Terminal Doors PAN Central Bank Expected PIN Central Bank Screen Terminal Screen Keystroke Terminal Keys 1.1 Device Sense and Control Device Commands Device Status 1.4 Manage Session 1.2 Central Bank Commun-ication Expected PIN Acct. Info Req. 1.3 Terminal Sense and Control Screen KeyHit  Session 1 Customer CustomerID PAN 1..n Has 0..n 1..n Uses 1..n Account PAN ExpectedPIN Balance accountType Transaction TransactionType TimeOfDay Amount ATMnumber 1..n 1..n Terminal ATMid Status CashOnHand Occurs 1	10	4	L2

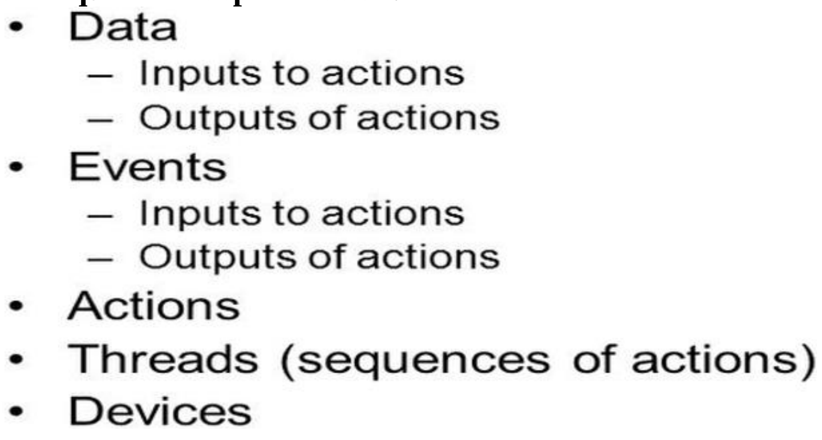
MODULE -5

9.	a. Explain the basic concepts of requirements specifications.	10	5	L2

Scheme : Definition with explanation and example for each – 5+5 marks

Solution:

Concepts of Requirement specification:

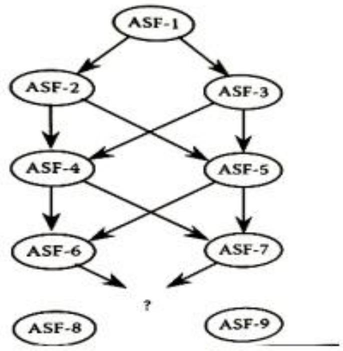


- Data
  - Inputs to actions
  - Outputs of actions
- Events
  - Inputs to actions
  - Outputs of actions
- Actions
- Threads (sequences of actions)
- Devices

	b. Define the process of ASF testing and illustrate it with an example using the next date function.	10	5	L2

Scheme: Diagram + 4 different sequences – 2+2+2+2+2 marks

Solution:



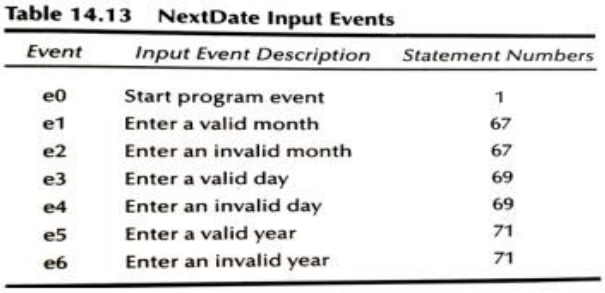


Table 14.13 NextDate Input Events

Event	*Input Event Description*	*Statement Numbers*
e0	Start program event	1
e1	Enter a valid month	67
e2	Enter an invalid month	67
e3	Enter a valid day	69
e4	Enter an invalid day	69
e5	Enter a valid year	71
e6	Enter an invalid year	71

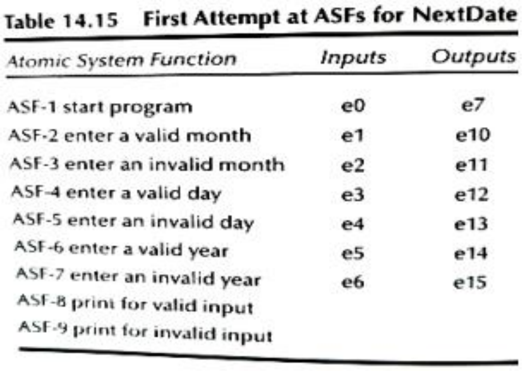


Table 14.15 First Attempt at ASFs for NextDate

Atomic System Function	*Inputs*	*Outputs*
ASF-1 start program	e0	e7
ASF-2 enter a valid month	e1	e10
ASF-3 enter an invalid month	e2	e11
ASF-4 enter a valid day	e3	e12
ASF-5 enter an invalid day	e4	e13
ASF-6 enter a valid year	e5	e14
ASF-7 enter an invalid year	e6	e15
ASF-8 print for valid input		
ASF-9 print for invalid input		

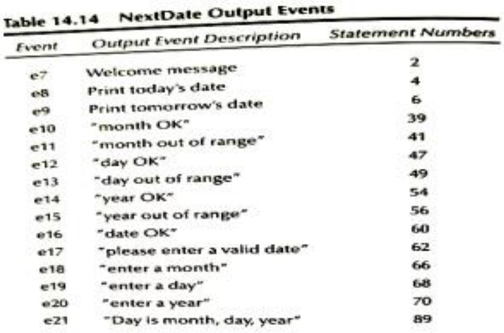


Table 14.14 NextDate Output Events

Event	*Output Event Description*	*Statement Numbers*
e7	Welcome message	2
e8	Print today's date	4
e9	Print tomorrow's date	6
e10	"month OK"	39
e11	"month out of range"	41
e12	"day OK"	47
e13	"day out of range"	49
e14	"year OK"	54
e15	"year out of range"	56
e16	"date OK"	60
e17	"please enter a valid date"	62
e18	"enter a month"	66
e19	"enter a day"	68
e20	"enter a year"	70
e21	"Day is month, day, year"	89

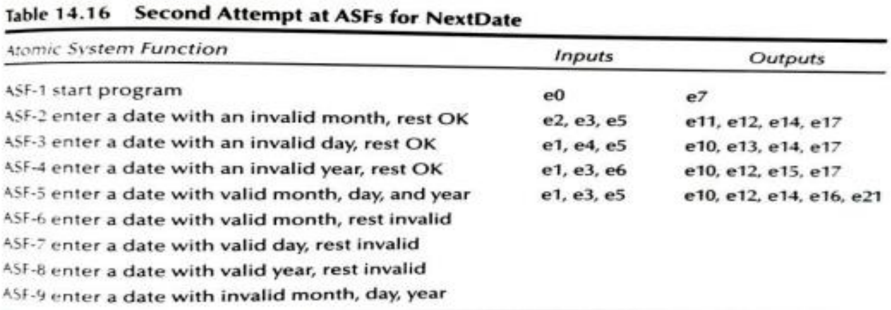
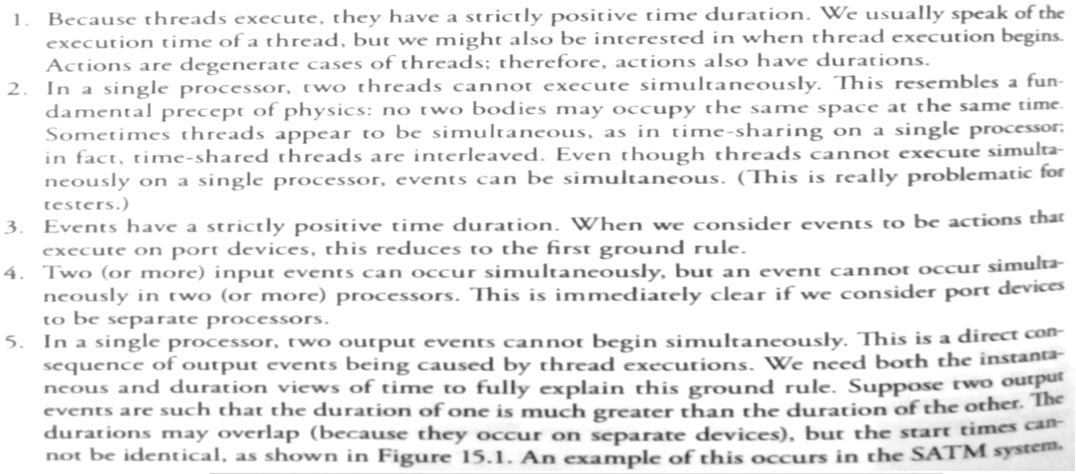
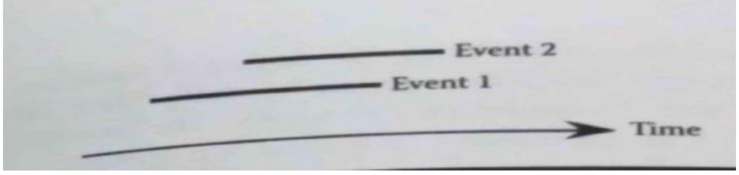
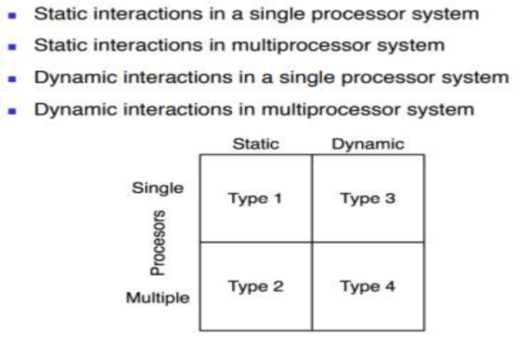
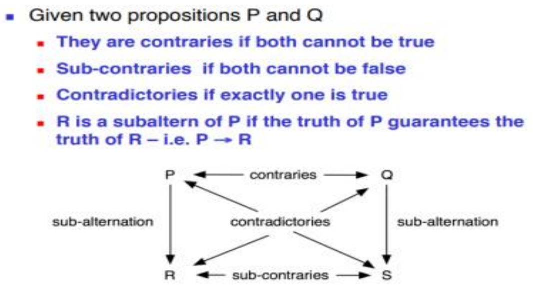
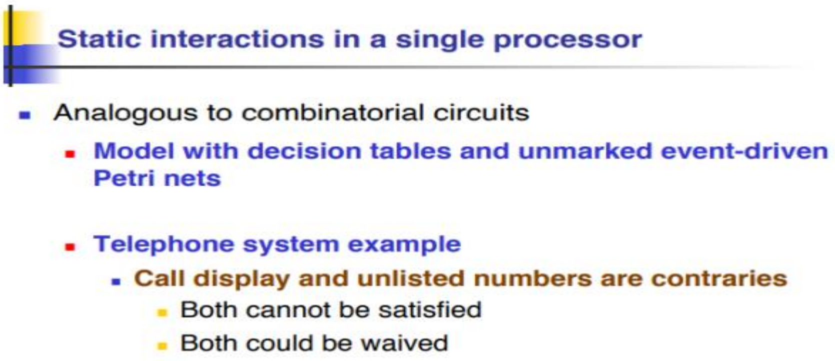


Table 14.16 Second Attempt at ASFs for NextDate

Atomic System Function	*Inputs*	*Outputs*
ASF-1 start program	e0	e7
ASF-2 enter a date with an invalid month, rest OK	e2, e3, e5	e11, e12, e14, e17
ASF-3 enter a date with an invalid day, rest OK	e1, e4, e5	e10, e13, e14, e17
ASF-4 enter a date with an invalid year, rest OK	e1, e3, e6	e10, e12, e15, e17
ASF-5 enter a date with valid month, day, and year	e1, e3, e5	e10, e12, e14, e16, e21
ASF-6 enter a date with valid month, rest invalid		
ASF-7 enter a date with valid day, rest invalid		
ASF-8 enter a date with valid year, rest invalid		
ASF-9 enter a date with invalid month, day, year		

(OR)				
10.	**a. Describe the context of interaction in Software Testing.** **Scheme: Definition with explanation and example for each – 5+5 marks** **Solution:**  1. Because threads execute, they have a strictly positive time duration. We usually speak of the execution time of a thread, but we might also be interested in when thread execution begins. Actions are degenerate cases of threads; therefore, actions also have durations. 2. In a single processor, two threads cannot execute simultaneously. This resembles a fundamental precept of physics: no two bodies may occupy the same space at the same time. Sometimes threads appear to be simultaneous, as in time-sharing on a single processor; in fact, time-shared threads are interleaved. Even though threads cannot execute simultaneously on a single processor, events can be simultaneous. (This is really problematic for testers.) 3. Events have a strictly positive time duration. When we consider events to be actions that execute on port devices, this reduces to the first ground rule. 4. Two (or more) input events can occur simultaneously, but an event cannot occur simultaneously in two (or more) processors. This is immediately clear if we consider port devices to be separate processors. 5. In a single processor, two output events cannot begin simultaneously. This is a direct consequence of output events being caused by thread executions. We need both the instantaneous and duration views of time to fully explain this ground rule. Suppose two output events are such that the duration of one is much greater than the duration of the other. The durations may overlap (because they occur on separate devices), but the start times cannot be identical, as shown in Figure 15.1. An example of this occurs in the SATM system.  	**10**	**5**	**L2**
	b. What is the taxonomy of interaction? Explain the static interaction in a single process. **Scheme : Definition with explanation and example for each – 3+5+2 marks** **Solution:**  ▪ Static interactions in a single processor system ▪ Static interactions in multiprocessor system ▪ Dynamic interactions in a single processor system ▪ Dynamic interactions in multiprocessor system Procesors — Static / Dynamic Single: Type 1 / Type 3 Multiple: Type 2 / Type 4  ▪ Given two propositions P and Q ▪ They are contraries if both cannot be true ▪ Sub-contraries if both cannot be false ▪ Contradictories if exactly one is true ▪ R is a subaltern of P if the truth of P guarantees the truth of R – i.e. P → R P contraries Q sub-alternation contradictories sub-alternation R sub-contraries S  **Static interactions in a single processor** ▪ Analogous to combinatorial circuits ▪ Model with decision tables and unmarked event-driven Petri nets ▪ Telephone system example ▪ Call display and unlisted numbers are contraries ▪ Both cannot be satisfied ▪ Both could be waived	**10**	**5**	**L2**

Faculty

Prof. Arvind R
Prof. Saba Tahseen