USN		

Internal Assessment Test 2 – MAY 2023

Scheme of Evaluation

Sub:	SOFTWARE TESTING					Sub Code:	18IS62	Branch:	ISE
Date:	23/05/2023	Duration:	90 min	Max Marks:	50	Sem/Sec:	VI/ A, B & C		OBE

<u>Answer any FIVE FULL Questions</u> | MARKS | CO | RBT

1. **Explain McCabe's basis path testing for Triangle problem.** [10] CO3 L2

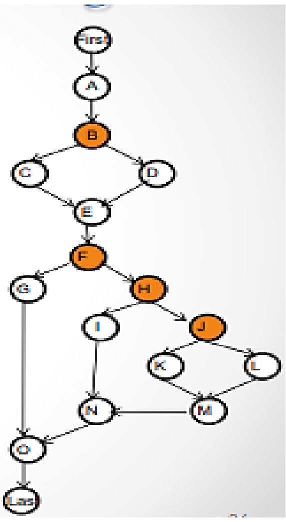
```
Program Triangle
Dim a, b,c  As  Integer
Dim IsTriangle As Boolean

Output ( "enter a,b, and c integers")
Input (a,b,c)
Output ("side 1 is", a)
Output ("side 2 is", b)
Output ("side 3 is", c)

If (a<b+c) AND (b<a+c) And (c<b+a)
 then  IsTriangle = True
 else  IsTriangle = False
endif

If IsTriangle
  then if (a=b) AND (b=c)
         then Output ("equilateral")
         else if (a != b) AND (a != b) AND (b != c)
                then Output ( "Scalene")
                else Output ("Isosceles")
              endif
       endif
  else  Output ("not a triangle")
endif
end Triangle2
```

McCabe	Paths	Expected Results
Original	P1: First-A-B-C-E-F-H-J-K-M-N-O-Last	Scalene
Flip P1 at B	P2: First-A-B-D-E-F-H-J-K-M-N-O-Last	Infeasible path
Flip P1 at F	P3: First-A-B-C-E-F-G-O-Last	Infeasible path
Flip P1 at H	P4: First-A-B-C-E-F-H-I-N-O-Last	Equilateral
Flip P1 at J	P5: First-A-B-C-E-F-H-J-L-M-N-O-Last	Isosceles



Test Case	a	b	c	Expected Results	From Path
1	3	4	5	Scalene	P1
2	4	1	2	Not a Triangle	P6
3	5	5	5	Equilateral	P4
4	3	2	2	Isosceles	P5

2(a) **Write Second Try Decision Table for NextDate Problem.** [05] CO2 L1

	1	2	3	4	5	6	7	8
c1: Month in	M1	M1	M1	M1	M2	M2	M2	M2
c2: Day in	D1	D2	D3	D4	D1	D2	D3	D4
c3: Year in	—	—	—	—	—	—	—	—
Rule count	3	3	3	3	3	3	3	3
Actions								
a1: Impossible				X				
a2: Increment day	X	X			X	X	X	
a3: Reset day			X					X
a4: Increment month			X					?
a5: Reset month								?
a6: Increment year								?
	9	10	11	12	13	14	15	16
c1: Month in	M3	M3	M3	M3	M3	M3	M3	M3
c2: Day in	D1	D1	D1	D2	D2	D2	D3	D4
c3: Year in	Y1	Y2	Y3	Y1	Y2	Y3	—	—
Rule count	1	1	1	1	1	1	3	3
Actions								
a1: Impossible						X	X	X
a2: Increment day	X	X	?					
a3: Reset day			?	X	X			
a4: Increment month	X		X	X	X			
a5: Reset month								
a6: Increment year								

(b) **Explain fault based testing.** [05] CO2 L2

Fault-based testing uses a fault model directly to hypothesize potential faults in a program under test, as well as to create or evaluate test suites based on its efficacy in detecting those hypothetical faults.

Original program: The program unit (e.g., C function or Java class) to be tested.

Program location: A region in the source code. The precise definition is defined relative to the syntax of a particular programming language. Typical locations are statements, arithmetic and Boolean expressions, and procedure calls.

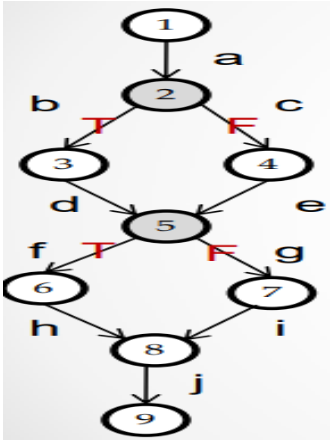
Alternate expression: Source code text that can be legally substituted for the text at a program location. A substitution is legal if the resulting program is syntactically correct.

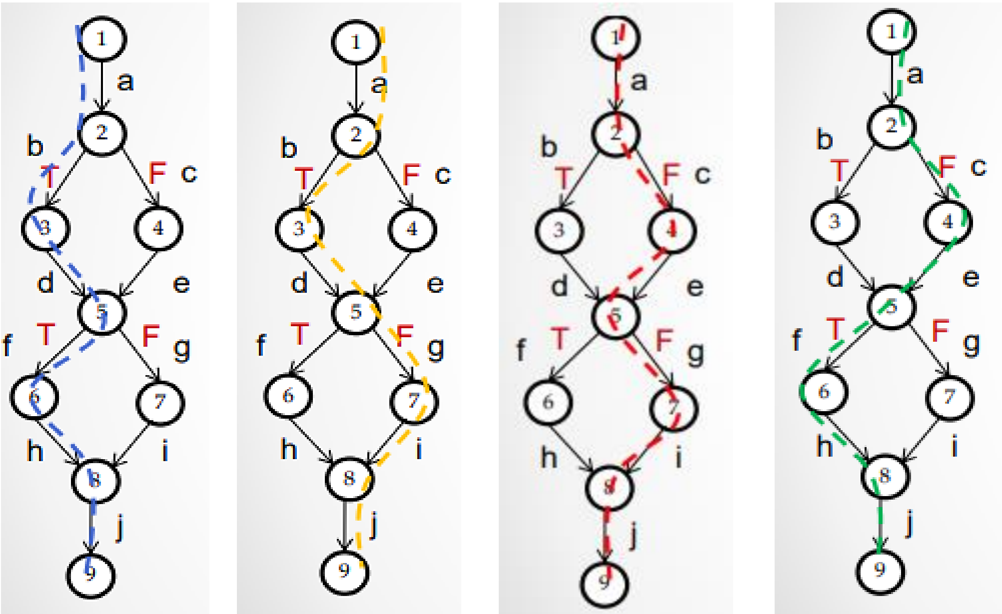
Alternate program A program obtained from the original program by substituting an alternate expression for the text at some program location.

Distinct behaviour of an alternate program R for a test t The behaviour of an alternate program R is distinct from the behaviour of the original program P for a test t, if R and P produce a different result for t, or if the output of R is not defined for t.

For example, failure of the Tacoma Narrows Bridge in 1940 led to new understanding of oscillation in high wind and to the introduction of analyses to predict and prevent such destructive oscillation in subsequent bridge design. The causes of an airline crash are likewise extensively studied, and when traced to a structural failure they frequently result in a directive to apply diagnostic tests to all aircraft considered potentially vulnerable to similar failures.

3	**Apply Path testing Strategy and generate path testing coverage table for the given flow graph:**	[10]	CO3	L3





Paths	Decisions		Process-Link									
	2	5	a	b	c	d	e	f	g	h	i	j
1-2-3-5-6-8-9 (a-b-d-f-h-j)	T	T	✓	✓		✓		✓		✓		✓
1-2-4-5-7-8-9 (a-c-e-g-i-j)	F	F	✓		✓		✓		✓		✓	✓
1-2-3-5-7-8-9 (a-b-d-g-i-j)	T	F	✓	✓		✓			✓		✓	✓
1-2-4-5-6-8-9 (a-c-e-f-h-j)	F	T	✓		✓		✓	✓		✓		✓

4 Compute the following for the given source code:

A. Flow Graph B. Cyclomatic Complexity C. Determine the basis set of independent paths. D. Test Cases.

[10] CO3 L3

```
1  program Example()
2  var staffDiscount, totalPrice, finalPrice, discount, price
3  staffDiscount = 0.1
4  totalPrice = 0
5  input(price)
6  while(price != -1) do
7      totalPrice = totalPrice + price
8      input(price)
9  od
10 print("Total price: " + totalPrice)
11 if(totalPrice > 15.00) then
12     discount = (staffDiscount * totalPrice) + 0.50
13 else
14     discount = staffDiscount * totalPrice
15 fi
16 print("Discount: " + discount)
17 finalPrice = totalPrice - discount
```

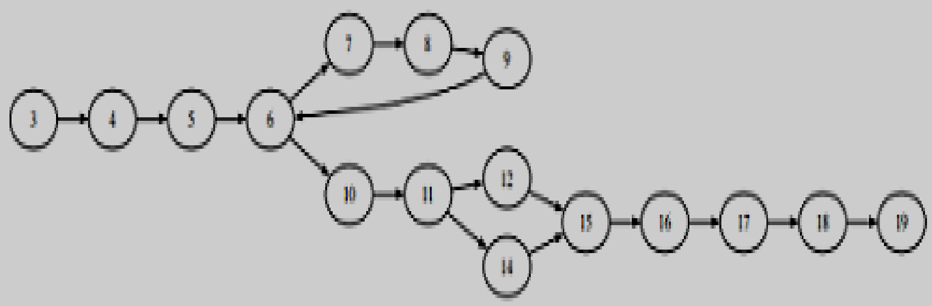
DU path for staff discount
P1 (3, 12) = <3, 4, 5, 6, 7, 8, 9, 10, 11, 12> is definition clear
P2 (3, 14) = <3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14> is not definition clear

DU path for total price
P3 (4, 7) = <4, 5, 6, 7> is definition clear
P4 (4, 10) = <4, 5, 6, 7, 8, 9, 10> is not definition clear
P5 (7, 6) = <7, 8, 9, 6> is definition clear
P6 (7, 7) = <7, 8, 9, 6, 7> is not definition clear
P7 (7, 10) = <7, 8, 9, 6, 10> is definition clear
P8 (7, 11) = <7, 8, 9, 6, 10, 11> is definition clear
P9 (7, 12) = <7, 8, 9, 6, 10, 11, 12> is definition clear
P10 (7, 14) = <7, 8, 9, 6, 10, 11, 12, 13, 14> is definition clear

DU path for final price
P11 (17, 17) = <17, 17> is definition clear

DU path for discount
P12 (12, 16) = <12, 13, 14, 15, 16> is not definition clear
P13 (12, 17) = <12, 13, 14, 15, 16, 17> is not definition clear
P14 (12, 16) = <12, 13, 14, 15, 16> is not definition clear
P15 (14, 16) = <14, 15, 16> is definition clear
P16 (14, 17) = <14, 15, 16, 17> is definition clear

DU path for price
P17 (5, 6) = <5, 6> is definition clear
P18 (5, 7) = <5, 6, 7> is definition clear
P19 (8,6) = <8, 9, 6> is definition clear
P20 (8, 7) = <8, 9, 6, 7> is definition clear



The program graph for the example code

Cyclomatic Complexity :**4**

5 (a)	Explain mutation analysis in testing. Mutation analysis is the most common form of software fault-based testing. A fault model is used to produce hypothetical faulty programs by creating variants of the program under test. Variants are created by "seeding" faults, that is, by making a small change to the program under test following a pattern in the fault model. The patterns for changing program text are called mutation operators, and each variant program is called a mutant. Original program under test: The program or procedure (function) to be tested. Mutant A program differs the original program for one syntactic element (e.g., a statement, a condition, a variable, a label). Distinguished mutant A mutant that can be distinguished for the original program by executing at least one test case. Equivalent mutant A mutant that cannot be distinguished from the original program. Mutation operator A rule for producing a mutant program by syntactically modifying the original program. Mutants should be plausible as faulty programs. Mutant programs that are rejected by a compiler, or that fail almost all tests, are not good models of the faults we seek to uncover with systematic testing. The mutation analysis process described in the preceding sections, which kills mutants based on the outputs produced by execution of test cases, is known as strong mutation. It can generate a number of mutants quadratic in the size of the program. Each mutant must be compiled and executed with each test case until it is killed. The time and space required for compiling all mutants and for executing all test cases for each mutant may be impractical.	[05]	CO2	L2
(b)	Write Valid and Invalid Classes of Variables for Commission and NextDate Problem • 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 }	[05]	CO2	L1

The valid classes of the input variables are

L1 = {locks: 1 ≤ locks ≤ 70}
L2 = {locks = −1} (occurs if locks = −1 is used to control input iteration)
S1 = {stocks: 1 ≤ stocks ≤ 80}
B1 = {barrels: 1 ≤ barrels ≤ 90}

The corresponding invalid classes of the input variables are

L3 = {locks: locks = 0 OR locks < −1}
L4 = {locks: locks > 70}
S2 = {stocks: stocks < 1}
S3 = {stocks: stocks > 80}
B2 = {barrels: barrels < 1}
B3 = {barrels: barrels > 90}

Q.No	Question	Marks	CO	Level
6	Write the source code of triangle problem in Fortran Style and compute DD path graph, path testing for the same.	[10]	CO3	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
```

① 4-5-6-7-8-9-10-12-13-21-22-23
② 4-5-6-7-8-9-11-12-13-14-15-20-22-23
③ 4-5-6-7-8-9-11-12-13-14-16-17-19-20-22-23
④ 4-5-6-7-8-9-11-12-13-14-16-18-19-20-22-23

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

code statement	Path/node name	DD-path Case
Skip 1- 3 (or w/4)		
4	first	1
5 – 8	A	5
9	B	3
10	C	4
11	D	4
12	E	3
13	F	3
14	H	3
15	I	4
16	J	3
17	K	4
18	L	4
19	M	3
20	N	3
21	G	4
22	O	3
23	last	2

Faculty Signature CCI Signature HOD Signature