1. **Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume that the upper limit for the size of any side is 10. Derive test cases for your program based on boundary-value analysis, execute the test cases and discuss the results.**

```
#include<stdio.h>
#include<conio.h>
void main()
{

int a,b,c,c1,c2,c3;
char istriangle;
do
{
printf("\nenter 3 integers which are sides of triangle\n");
scanf("%d%d%d",&a,&b,&c);
printf("\na=%d\tb=%d\tc=%d",a,b,c);

c1 = a>=1 && a<=10;
c2= b>=1 && b<=10;
c3= c>=1 && c<=10;
if (!c1)

printf("\n the value of a=%d is not the range of permitted value",a);

if (!c2)
printf("\n the value of b=%d is not the range of permitted value",b);

if (!c3)

printf("\n the value of c=%d is not the range of permitted value",c); } while(!(c1 && c2 &&
c3));

// to check is it a triangle or not

if( a<b+c && b<a+c && c<a+b )
istriangle='y';

else

istriangle ='n';

if (istriangle=='y')
if ((a==b) && (b==c))
printf("equilateral triangle\n");
```

```
else if ((a!=b) && (a!=c) && (b!=c))
printf("scalene triangle\n");

else
printf("isosceles triangle\n");
else

printf("Not a triangle\n");
getch();

}
```

BOUNDARY VALUE ANALYSIS

Test Case Name: Boundary Value Analysis for triangle problem

Test Data: Enter the 3 Integer Value (a , b and c)

Pre-condition: $1 \leq a \leq 10$, $1 \leq b \leq 10$ and $1 \leq c \leq 10$ and $a < b + c$, $b < a + c$ and $c < a + b$

Conditions evaluated for: whether given value for a Equilateral, Isosceles, Scalene triangle or can't form a triangle.

Case id(BVA)	Description	Input data			Expected output	Actual output	status
		a	b	c			
1.	Enter min value for a b and c	1	1	1	Equilateral triangle		Pass/Fail
2.	Enter min value for two items and min+1 for any one item	1	1	2	Isosceles triangle		
3.	Enter valid values for a b and c	5	5	5	Equilateral triangle		
4.	Enter valid values for two items and max values for one item	5	5	10	Isosceles triangle		
5.	Enter valid values for two items and max-1 values for one item	10	9	10	Isosceles triangle		
6.	Enter max values for a b and c	10	10	10	Equilateral triangle		
7.	Enter different values for a b and c	1	2	3	Scalene triangle		
8.	Enter different values for a b and c	2	3	1	Scalene triangle		

Note : ***Also Perform robustness testing , worst case testing and special value testing .***

2. Design, develop code and run the program in any suitable language to solve the commission problem. Analyze it from the perspective of boundary value testing, derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
int main()
{
int locks, stocks, barrels, t_sales;
float commission;
printf("Enter the total number of locks");
scanf("%d",&locks);
printf("Enter the total number of stocks");
scanf("%d",&stocks);
printf("Enter the total number of barrelss");
scanf("%d",&barrels);
if ((locks <= 0) || (locks> 70)||(stocks <= 0) || (stocks > 80)||
(barrels <= 0) || (barrels > 90))
{
if(lock==-1)
Printf(“program terminates”);
else
printf("invalid input");
}
else
{
t_sales = (locks * 45) + (stocks * 30) + (barrels * 25);
if (t_sales <= 1000)
{
commission = 0.10 * t_sales;
}
else if (t_sales < 1800)
{
commission = 0.10 * 1000;
commission = commission + (0.15 * (t_sales - 1000));
}
else
{
commission = 0.10 * 1000;
commission = commission + (0.15 * 800);
commission = commission + (0.20 * (t_sales - 1800));
}
printf("The total sales is %d \n the commission is %f",t_sales,
commission);
}
return 0;
}
```

BOUNDARY VALUE ANALYSIS

Test Case Name: Boundary Value Analysis for Commission problem

Test Data: Enter the 3 Integer Value (locks, Stocks and barrels)

Pre-condition: 1<=Locks<=70 {1,2,35,70,69}
1<=Stocks<=80 {1,2,40,80,79}
1<=Barrels<=90 {1,2,45,90,89}

No of test cases=(4n+1, n is the number of inputs) {min, min+, norm, max, max-}

Case id(BVA)	Description	Input			Expected output		Actual output		status
		Locks	Stocks	Barrels	Sales	Commission	Sales	Commission	
1	Barrels is min	35	40	1	2800	420.00			
2	Barrels is min+	35	40	2	2825	425.00			
3	Barrels is norm	35	40	45	3900	640.00			
4	Barrels is max	35	40	90	5025	865.00			
5	Barrels is max-	35	40	89	5000	860.00			
6	Stocks is min	35	1	45	2730	406.00			
7	Stocks is min+	35	2	45	2760	412.00			
8	Stocks is norm	35	40	45	3900	640.00			
9	Stocks is max	35	80	45	5100	880.00			
10	Stocks is max-	35	79	45	5070	874.00			

Robust testing: Test Cases (min-, max+)

Case id	Description	Input data			Expected output		Actual output		Status
		Locks	Stocks	Barrels	Sales	commission	Sales	Commission	
1	Locks invalid	-2	40	45	Invalid Input				
2	Locks invalid	71	40	45	Invalid Input				
3	Stocks invalid	35	-2	45	Invalid Input				
4	Stocks invalid	35	81	45	Invalid Input				
5	Barrels invalid	35	40	-2	Invalid Input				
6	Barrels invalid	35	40	91	Invalid Input				

Special Value Testing: Test Cases(Validated test cases)

Case id	Description	Input			Expected output		Actual output		Status
		Locks	Stocks	Barrels	Sales	Commission	Sales	Commission	
1	Less than 1800	1	40	1	1270	275.00			
2	Less than 1000	1	30	1	970	97.00			
3	Greater than 1800	1	1	90	2325	325.00			

Worst Case Testing: Test Cases(min,max, nom,min-,max+)

Test Case id	Description	Input			Expected output		Actual output		status
		Locks	Stocks	Barrels	Sales	Commission	Sales	Commission	
1	Cartesian product of 5 element set.	1	2	1	130	13			
2		1	2	2	155	15.5			
3		1	2	45	1230	134.5			
4		1	2	90	2355	331			
5		1	2	89	2330	326			
6	Cartesian product of 5 element set.	1	1	1	100	10			
7		1	1	2	125	12.5			
8		1	1	45	1200	130			
9		1	1	90	2325	325			
10		1	1	89	2300	320			
11	Cartesian product of 5 element set.	1	40	1	1270	140.5			
12		1	40	20	1745	211.75			
13		1	40	45	2370	334			
14		1	40	90	3495	559			
15		1	40	89	3470	554			
16	Cartesian product of 5 element set.	1	80	1	2470	354			
17		1	80	2	2495	359			
18		1	80	45	3570	574			
19		1	80	90	4695	799			
20		1	80	89	4670	794			
21	Cartesian product of 5 element set.	1	79	1	2440	348			
22		1	79	2	2465	353			
23		1	79	45	3540	568			
24		1	79	90	4665	793			
25		1	79	89	4640	788			

3. Design, develop, code and run the program in any suitable language to implement the Next Date function. Analyze it from the perspective of boundary value testing, derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
main( )
{
int month[12]={31,28,31,30,31,30,31,31,30,31,30,31};
int d,m,y,nd,nm,ny,ndays;
printf("enter the date,month,year");
scanf("%d%d%d",&d,&m,&y);
if((y%100!=0)&&(y%4==0)||(y%400==0))
{
Month[i]=29;
}
ndays=month [m-1];
(y<=1812) || ( y>2012) || (d<=0) || (d>ndays)|| (m<1) || ( m>12)
{
Printf("invalid input");
}
else
{
if(m==2)
{
if(y%100==0)
{
if(y%400==0)
ndays=29;
}
else
if(y%4==0)
ndays=29;
}
nd=d+1;
nm=m;
ny=y;
if(nd>ndays)
{
nd=1;
nm++;
}
if(nm>12)
{
nm=1;
ny++;
}
```

```
printf("\n Given date is %d:%d:%d",d,m,y);
printf("\n Next day‟s date is %d:%d:%d",nd,nm,ny);
}
Return 0;
}
```

BOUNDARY VALUE ANALYSIS

(No. of test cases = 4n+1)

Pre Conditions :
Day:1<=day<=31{1,2,15,31,30}(min,min-,norm,max,max-)
Month:1<=month<=12 {1,2,6,12,11}
Year: 1812<=year<=2012 {1812,1813,1912,2012,2011}

Boundary value analysis test cases

Case id	Description	Input data			Expected output	Actual output	status
		Day	Month	year			
1	Year is min	15	6	1812	16-6-1812		
2	Year is min+	15	6	1813	16-6-1813		
3	Year is norm	15	6	1912	16-6-1912		
4	Year is max	15	6	2012	16-6-2012		
5	Year is max-	15	6	2011	16-6-2011		
6	Month is min	15	1	1912	16-1-1912		
7	Month is min+	15	2	1912	16-2-1912		
8	Month is norm	15	6	1912	16-6-1912		
9	Month is max	15	12	1912	16-12-1912		
10	Month is max-	15	11	1912	16-11-1912		
11	Day is min	1	6	1912	2-6-1912		
12	Day is min+	2	6	1912	3-6-1912		
13	Day is norm	15	6	1912	16-6-1912		
14	Day is max	31	6	1912	Invalid input		
15	Day is max-	30	6	1912	1-7-1912		

Robust Testing Test Cases

Case id	Description	Input data			Expected output	Actual output	Status
		Day	Month	Year			
1	Day invalid	-1	6	1912	Invalid input		
2	Day invalid	32	6	1912	Invalid input		
3	Month invalid	15	-1	1912	Invalid input		

4	Month invalid	15	13	1912	Invalid input		
5	Year invalid	15	6	1811	Invalid input		
6	Year invalid	15	6	2013	Invalid input		

Special value testing Test Cases

Case id	Description	Input data			Expected output	Actual output	Status
		Day	Month	Year			
1	For leap year	28	2	2000	29-2-2000		
2	For non-leap year	28	2	2001	1-3-2001		
3	For non-leap year	29	2	2001	Invalid input		

Worst case testing: Test Cases

(No. of test cases =5^n)

Day: 1<=day<=31 {1,2,15,31,30}(min,min-,norm,max,max-)

Month:1<=month<=12 {1,2,6,12,11}

Year: 1812<=year<=2012 {1812, 1813, 1912, 2012, 2011}

case id	Description	Input data			Expected output	Actual output	status
		Day	Month	Year			
1	Cartesian Product of 5 element set	1	1	1812	2-1-1812		
2		1	1	1813	2-1-1813		
3		1	1	1912	2-1-1912		
4		1	1	2012	2-1-2012		
5		1	1	2011	2-1-2011		
6	Cartesian Product of 5 element set	1	2	1812	2-2-1812		
7		1	2	1813	2-2-1913		
8		1	2	1912	2-2-1912		
9		1	2	2012	2-2-2012		
10		1	2	2011	2-2-2011		
11	Cartesian Product of 5 element set	1	6	1812	2-6-1812		
12		1	6	1813	2-6-1813		
13		1	6	1912	2-6-1912		
14		1	6	2012	2-6-2012		
15		1	6	2011	2-6-2011		

16	Cartesian Product of 5 element set	1	12	1812	2-12-1812		
17		1	12	1813	2-12-1813		
18		1	12	1912	2-12-1912		
19		1	12	2012	2-12-2012		
20		1	12	2011	2-12-2011		
21	Cartesian Product of 5 element set	1	11	1812	2-11-1812		
22		1	11	1813	2-11-1813		
23		1	11	1912	2-11-1912		
24		1	11	2012	2-11-2012		
25		1	11	2011	2-11-2011		

4. **Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume that the upper limit for the size of any side is 10. Derive test cases for your program based on equivalence class partitioning, execute the test cases and discuss the results**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c,c1,c2,c3;
char istriangle;
do
{
printf("\nenter 3 integers which are sides of triangle\n");
scanf("%d%d%d",&a,&b,&c);
printf("\na=%d\tb=%d\tc=%d",a,b,c);

c1 = a>=1 && a<=10;
c2= b>=1 && b<=10;
c3= c>=1 && c<=10;
if (!c1)

printf("\nthe value of a=%d is not the range of permitted value",a);

if (!c2)
printf("\nthe value of b=%d is not the range of permitted value",b);

if (!c3)

printf("\nthe value of c=%d is not the range of permitted value",c);
} while(!(c1 && c2 && c3));

// to check is it a triangle or not

if( a<b+c && b<a+c && c<a+b )
istriangle='y';

else

istriangle ='n';

if (istriangle=='y')
if ((a==b) && (b==c))
printf("equilateral triangle\n");
```

```
else if ((a!=b) && (a!=c) && (b!=c))
printf("scalene triangle\n");

else
printf("isosceles triangle\n");
else

printf("Not a triangle\n");
getch();

}
```

EQUIVALENCE CLASS TESTING

Test Case Name : Equivalence class Analysis for triangle

Test Data : Enter the 3 Integer Value(a , b And c)

Pre-condition : $1 \leq a \leq 10$, $1 \leq b \leq 10$ and $1 \leq c \leq 10$ and $a < b + c$, $b < a + c$ and $c < a + b$

Brief Description : Check whether given value for a Equilateral, Isosceles , Scalene triangle or can't form a triangle

Weak equivalence class testing

Case id	Description	Input data			Expected output	Actual output	status
		a	b	c			
1	Enter min value for a b and c	5	5	5	Equilateral triangle		
2	Enter min value for a b and c	2	2	3	Isosceles triangle		
3	Enter min value for a b and c	3	4	5	Scalene triangle		
4	Enter min value for a b and c	4	1	2	Cannot form a triangle		

Weak robust equivalence class testing

Case id	Description	Input data			Expected output	Actual output	status
		A	b	C			
1	Enter one invalid input and two valid inputs	-1	5	5	Value of a is not in range of permitted values		
2	Enter one invalid input and two valid inputs	5	-1	5	Value of b is not in range of permitted values		
3	Enter one invalid input and two valid inputs	5	5	-1	Value of c is not in range of permitted values		
4	Enter one in valid input and two valid inputs	11	5	5	Value of a is not in range of permitted values		
5	Enter one in valid input and two valid inputs	5	11	5	Value of b is not in range of permitted values		

6	Enter one in valid input and two valid inputs	5	5	11	Value of c is not in range of permitted values		

Strong robust equivalence class testing

Case id	Description	Input data			Expected output	Actual output	status
		a	b	c			
1	Enter one invalid input and two valid inputs	-1	5	5	Value of a is not in range of permitted values		
2	Enter one invalid input and two valid inputs	5	-1	5	Value of b is not in range of permitted values		
3	Enter one invalid input and two valid inputs	5	5	-1	Value of c is not in range of permitted values		
4	Enter two in valid input and one valid inputs	-1	-1	5	Value of a and b I are not in range of permitted values		
5	Enter two in valid input and one valid inputs	-1	5	-1	Value of b and a are not in range of permitted values		
6	Enter two in valid input and one valid inputs	5	-1	-1	Value of c and b are not in range of permitted values		
7	Enter all three invalid inputs	-1	-1	-1	Input values are not in range of permitted values		

5. Design, develop, code and run the program in any suitable language to solve the commission problem. Analyze it from the perspective of equivalence class testing, derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
int main()
{
int locks, stocks, barrels, t_sales;
float commission;
printf("Enter the total number of locks");
scanf("%d",&locks);
printf("Enter the total number of stocks");
scanf("%d",&stocks);
printf("Enter the total number of barrelss");
scanf("%d",&barrels);
if ((locks <= 0) || (locks> 70)||(stocks <= 0) || (stocks > 80)||
(barrels <= 0) || (barrels > 90))
{
if(lock==-1)
Printf(“program terminates”);
else
printf("invalid input");
}
else
{
t_sales = (locks * 45) + (stocks * 30) + (barrels * 25);
if (t_sales <= 1000)
{
commission = 0.10 * t_sales;
}
else if (t_sales < 1800)
{
commission = 0.10 * 1000;
commission = commission + (0.15 * (t_sales - 1000));
}
else
{
commission = 0.10 * 1000;
commission = commission + (0.15 * 800);
commission = commission + (0.20 * (t_sales - 1800));
}
printf("The total sales is %d \n the commission is %f",t_sales,
commission);
}
return 0;
}
```

EQUIVALENCE CLASS TESTING

Test Case Name: Equivalence Class for Commission Problem

Test data: price Rs for lock - 45.0 , stock - 30.0 and barrel - 25.0

Sales = total lock * lock price + total stock * stock price + total barrel * barrel price

Commission: 10% up to sales Rs 1000 , 15 % of the next Rs 800 and 20 % on any sales in excess of 1800 .

Pre-condition: lock = -1 to exit and 1< =lock < = 70, 1<=stock <=80 and 1<=barrel<=90.

Brief Description: The salesperson had to sell at least one complete rifle per month. Checking boundary value for locks, stocks and barrels and commission

Valid Equivalence Classes:

L1 ={LOCKS :1 <=LOCKS<=70}

L2 ={Locks=-1}(occurs if locks=-1 is used to control input iteration)

L3={stocks : 1<=stocks<=80}

L4= {barrels :1<=barrels<=90}

Invalid Equivalence Classes

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}

Commission Problem Output Equivalence Class Testing

Weak & Strong Normal Equivalence Class

Case id	Description	Input data			Expected output		Actual output		status
		Locks	stocks	barrels	Sales	commission	Sales	commission	
1	Valid input	35	40	45	3900	640.00			Pass
2	Locks =-1	-1	40	45	Program terminates				pass

Weak Robust Equivalence Class Test Cases

Case	Description	Input data	Expected output	Actual output	status

id									
		Locks	stocks	barrels	Sales	commission	Sales	commission	
WR1	Valid input	35	40	45	3900	640.00			
WR2	Locks is invalid	-2	40	45	Program terminates				
WR3	Locks is invalid	71	40	45	Invalid input				
WR4	Stocks is invalid	35	-2	45	Invalid input				
WR5	Stocks is invalid	35	81	45	Invalid input				
WR6	Barrels is invalid	35	40	-2	Invalid input				
WR7	Barrels is invalid	35	40	91	Invalid input				

Strong Robust Equivalence Classes Test Cases

Case Id	Description	Input data			Expected output		Actual output		status
		Locks	stocks	Barrels	Sales	commission	Sales	commission	
SR1	Invalid input	-2	40	45	Invalid input				
SR2	Invalid input	35	-2	45	Invalid input				
SR3	Invalid input	35	40	-2	Invalid input				
SR4	Invalid input	-2	-2	45	Invalid input				
SR5	Invalid input	-2	40	-2	Invalid input				
SR6	Invalid input	35	-2	-2	Invalid input				
SR7	Invalid input	-2	-2	-2	Invalid input				

Second try(depending on o/p)

S1= {<locks , stocks ,barrels>: sales<=1000}
S2= {<locks , stocks ,barrels>: 1000<=sales<=1800}
S3= {<locks , stocks ,barrels>: sales>1800}

Weak Normal Equivalence Classes Test Cases

Case Id	Description	Input			Expected output		Actual output		status
		Locks	stocks	barrels	Sales	commission	Sales	commission	
WN1	Sales<=1000	5	5	5	500	50.00			

WN2	1000<=sales<=1800	15	15	15	1500	310.00			
WN3	Sales>1800	25	25	25	2500	360.00			

Strong Normal Equivalence Classes Test Cases

Case id	**Description**	**Input**			**Expected output**		**Actual output**		**status**
		Locks	**Stocks**	**barrels**	**Sales**	**commission**	**Sales**	**commission**	
SN1	Sales<=1000	5	5	5	500	50.00			
SN2	1000<=sales<=1800	15	15	15	1500	310.00			
SN3	Sales>1800	25	25	25	2500	360.00			

6. Design, develop, code and run the program in any suitable language to implement the Next Date function. Analyze it from the perspective of equivalence class value testing, derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
main( )
{
int month[12]={31,28,31,30,31,30,31,31,30,31,30,31};
int d,m,y,nd,nm,ny,ndays;
printf("enter the date,month,year");
scanf("%d%d%d",&d,&m,&y);
if((y%100!=0)&&(y%4==0)||(y%400==0))
{
Month[i]=29;
}
Ndays =month [m-1];
(y<1812) || ( y>2012) || (d<1) || (d>ndays)|| (m<1) || ( m>12)
{
Printf (“invalid input”);
}
else
{
if(m==2)
{
if(y%100==0)
{
if(y%400==0)
ndays=29;
}
else
if(y%4==0)
ndays=29;
}
nd=d+1;
nm=m;
ny=y;
if(nd>ndays)
{
nd=1;
nm++;
}
if(nm>12)
```

```
{
nm=1;
ny++;
}
printf("\n Given date is %d:%d:%d",d,m,y);
printf("\n Next day‟s date is %d:%d:%d",nd,nm,ny);
}
return 0;
}
```

EQUIVALENCE CLASS TESTING

First Attempt
A first attempt at creating an equivalence relation might produce intervals Such as these:

Valid Equivalence Classes

M1 = {month: $1 \leq$ month ≤ 12}
D1 = {day: $1 \leq$day ≤ 31}
Y1 = {year: $1812 \leq$ 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}

At a first glance it seems that everything has been taken into account and our day, month and year intervals have been defined well. Using these intervals we produce test cases using the four different types of Equivalence Class testing.

Weak Normal and Strong Normal Equivalence Classes Test Cases

Case id	Description	Input			Expected output	Actual output	status
		Day	Month	Year			
1	Enter the M1, D1 and Y1 valid cases	15	6	1912	16-6-1912		

Weak Robust Equivalence Classes Test Cases

Case	Description	Input	Expected	Actual output	status

id		Day	Month	Year	output		
1	Valid input	15	6	1912	Invalid input		
2	Day is invalid	-1	6	1912	Invalid input		
3	Day is invalid	32	6	1912	Invalid input		
4	Month is invalid	15	-1	1912	Invalid input		
5	Month is invalid	15	13	1912	Invalid input		
6	Year is invalid	15	6	1811	Invalid input		
7	Year is invalid	15	6	2013	Invalid input		

Strong Robust Equivalence Classes Test Cases

TC ID	TC Description	Input			Expected output	Actual output	status
		Day	Month	Year			
SR1	Invalid input	-1	6	1912	Invalid input		
SR2	Invalid input	15	-1	1912	Invalid input		
SR3	Invalid input	15	6	-1	Invalid input		
SR4	Invalid input	-1	-1	1912	Invalid input		
SR5	Invalid input	-1	6	1912	Invalid input		
SR6	Invalid input	15	-1	-1	Invalid input		
SR7	Invalid input	-1	-1	-1	Invalid input		

Second Attempt

As said before the equivalence relation is vital in producing useful test cases and more time must be spent on designing it. If we focus more on the equivalence relation and consider more greatly what must happen to an input date we might produce the following equivalence classes:

M1 = {month: month has 30 days}
M2 = {month: month has 31 days}
M3 = {month: month is February}

Here month has been split up into 30 days (April, June, September and November), 31 days (January, March, April, May, July, August, October and December) and February.

D1 = {day: $1 \leq day \leq 28$}
D2 = {day: day = 29}
D3 = {day: day = 30}
D4 = {day: day = 31}

Day has been split up into intervals to allow months to have a different number of days; we also have the special case of a leap year (February 29days).

Y1 = {year: year = 2000}
Y2 = {year: year is a leap year}
Y3 = {year: year is a common year}

Year has been split up into common years, leap years and the special case the year 2000 so we can determine the date in the month of February. Here are the test cases for the new equivalence relation using the four types of Equivalence Class testing.

Weak Normal Equivalence Classes Test Cases

Test case id	Description	Input			Expected output	Actual output	status
		Day	Month	Year			
WN1	Valid input	14	6	2000	14-6-2000		
WN2	Valid input	29	7	1996	29-7-1996		
WN3	Valid input	30	2	2000	30-2-2000		
WN4	Valid input	31	6	2000	31-6-2000		

Strong Normal Equivalence Classes Test Cases

(No. of test cases =3*4*3=36 Where 3= no of month class; 4= no of day class; 3=no of year class respectively)

Test case id	Description	Input			Expected output	Actual output	status
		Day	Month	Year			
SN1	Valid input	14	6	2000	Valid input		
SN2	Valid input	14	6	1996	Valid input		
SN3	Valid input	14	6	2002	Valid input		
SN4	Valid input	29	6	2000	Valid input		
SN5	Valid input	29	6	1996	Valid input		
SN6	Valid input	29	6	2002	Valid input		
SN7	Valid input	30	6	2000	Valid input		

SN8	Valid input	30	6	1996	Valid input		
SN9	Valid input	30	6	2002	Valid input		
SN10	Valid input	31	6	2000	Valid input		
SN11	Valid input	31	6	1996	Valid input		
SN12	Valid input	31	6	2002	Valid input		
SN13	Valid input	14	7	2000	Valid input		
SN14	Valid input	14	7	1996	Valid input		
SN15	Valid input	14	7	2002	Valid input		
SN16	Valid input	29	7	2000	Valid input		
SN17	Valid input	29	7	1996	Valid input		
SN18	Valid input	29	7	2002	Valid input		
SN19	Valid input	30	7	2000	Valid input		
SN20	Valid input	30	7	1996	Valid input		
SN21	Valid input	30	7	2002	Valid input		
SN22	Valid input	31	7	2000	Valid input		
SN23	Valid input	31	7	1996	Valid input		
SN24	Valid input	31	7	2002	Valid input		
SN25	Valid input	14	2	2000	Valid input		
SN26	Valid input	14	2	1996	Valid input		
SN27	Valid input	14	2	2002	Valid input		
SN28	Valid input	29	2	2000	Valid input		
SN29	Valid input	29	2	1996	Valid input		
SN30	Valid input	29	2	2002	Valid input		
SN31	Valid input	30	2	2000	Valid input		

SN32	Valid input	30	2	1996	Valid input		
SN33	Valid input	30	2	2002	Valid input		
SN34	Valid input	31	2	2000	Valid input		
SN35	Valid input	31	2	1996	Valid input		
SN36	Valid input	31	2	2002	Valid input		

Invalid Equivalence Classes

M4 = {month: month < 1}
M5 = {month: month > 12}
D5= {day: day < 1}
D6 = {day: day > 31}
Y4 = {year: year < 1812}
Y5 = {year: year > 2012}

Weak Robust Equivalence Classes Test Cases

Test case id	Description	Input data			Expected output	Actual output	status
		Day	Month	Year			
WR1	Valid input	15	6	1912	Invalid input		
WR2	Day is invalid	-1	6	1912	Invalid input		
WR3	Day is invalid	32	6	1912	Invalid input		
WR4	Month is invalid	15	-1	1912	Invalid input		
WR5	Month is invalid	15	13	1912	Invalid input		
WR6	Year is invalid	15	6	1811	Invalid input		
WR7	Year is invalid	15	6	2013	Invalid input		

Strong Robust Equivalence Classes Test Cases

Test case id	Description	Input data			Expected output	Actual output	status
		Day	Month	Year			
SR1	Invalid input	-1	6	1912	Invalid input		
SR2	Invalid input	15	-1	1912	Invalid input		
SR3	Invalid input	15	6	-1	Invalid input		
SR4	Invalid input	-1	-1	1912	Invalid input		

SR5	Invalid input	-1	6	1912	Invalid input		
SR6	Invalid input	15	-1	-1	Invalid input		
SR7	Invalid input	-1	-1	-1	Invalid input		

7. Design and develop a program in a language of your choice to solve the triangle problem defined as follows : Accept three integers which are supposed to be the three sides of triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Derive test cases for your program based on decision-table approach, execute the test cases and discuss the results

```
#include<stdio.h>
#include <conio.h>

void main()
{
     int a,b,c;
    char istriangle;

printf("enter 3 integers which are sides of triangle\n");
scanf("%d%d%d",&a,&b,&c);
printf("a=%d\t,b=%d\t,c=%d",a,b,c);

// to check is it a triangle or not

if(a<b+c && b<a+c && c<a+b)

      istriangle='y';
else
     istriangle ='n';

if (istriangle=='y')

     if ((a==b) && (b==c))

         printf("equilateral triangle\n");

else if ((a!=b) && (a!=c) && (b!=c))

        printf("scalene triangle\n");
else
    printf("isosceles triangle\n");

else
  printf("Not a triangle\n");
```

```
getch();
}
```

DESCISION TABLE TESTING

Test Case Name: Decision table for triangle problem

Test Data: Enter the 3 Integer Value (a, b And c)

Pre-condition: a < b + c , b < a + c and c < a + b

Conditions evaluated for: whether given value for a equilateral, isosceles , Scalene triangle or can't form a triangle

Rules	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**
Conditions:											
C1: a<(b+c)?	F	T	T	T	T	T	T	T	T	T	T
C2: b<(a+c)?	-	F	T	T	T	T	T	T	T	T	T
C3: c<(b+a)?	-	-	F	T	T	T	T	T	T	T	T
C4: a=b?	-	-	-	F	T	T	T	F	F	F	T
C5: b=c?	-	-	-	T	F	T	F	T	F	F	T
C6: c=a?	-	-	-	T	T	F	F	F	T	F	T
Actions:											
A1: not a triangle	X	X	X								
A2: equilateral											X
A3: isosceles							X	X	X		
A4: scalene										X	
A5: impossible				X	X	X					

Triangle Problem -Decision Table Test cases for input data

Case id	**Description**	**Input data**			**Expected output**	**Actual output**	**Status**
		A	**b**	**c**			
1	Enter the value of a,b and c such that a is not less than sum of two sides	20	5	5	Cannot form a triangle		
2	Enter the value of a,b and c such that b is not less than sum of two sides and a is less than sum of two sides	3	15	11	Cannot form a triangle		
3	Enter the value of a , b and c such that c is not	4	5	20	Cannot form a triangle		

	less than sum of two sides and a and b is less than sum of other two sides						

8. Design, develop, code and run the program in any suitable language to solve the commission problem. Analyze it from the perspective of decision table-based testing, derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
int main()
{
int locks, stocks, barrels, t_sales;
float commission;
printf("Enter the total number of locks");
scanf("%d",&locks);
printf("Enter the total number of stocks");
scanf("%d",&stocks);
printf("Enter the total number of barrelss");
scanf("%d",&barrels);
if ((locks <= 0) || (locks> 70)||(stocks <= 0) || (stocks > 80)||
(barrels <= 0) || (barrels > 90))
{
if(lock==-1)
Printf(“program terminates”);
else
printf("invalid input");
}
else
{
t_sales = (locks * 45) + (stocks * 30) + (barrels * 25);
if (t_sales <= 1000)
{
commission = 0.10 * t_sales;
}
else if (t_sales < 1800)
{
commission = 0.10 * 1000;
commission = commission + (0.15 * (t_sales - 1000));
}
else
{
commission = 0.10 * 1000;
commission = commission + (0.15 * 800);
commission = commission + (0.20 * (t_sales - 1800));
}
```

```
printf("The total sales is %d \n the commission is %f",t_sales,
commission);
}
return 0;
}
```

DECISION TABLE

Test data: price Rs for lock - 45.0, stock - 30.0 and barrel - 25.0

Sales = total lock * lock price + total stock * stock price + total barrel * barrel price

Commission: 10% up to sales Rs 1000, 15 % of the next Rs 800 and 20 % on any sales in excess of 1800.

Pre-condition: lock = -1 to exit and 1< =lock < = 70 , 1<=stock <=80 and 1<=barrel<=90

Brief Description: The salesperson had to sell at least one complete rifle per month.

RULES		R1	R2	R3	R4	R5	R6	R7	R8	R9
Conditions	C1: Locks = -1	T	F	F	F	F	F	F	F	F
	C2 : 1 ≤ Locks ≤ 70	-	T	T	F	T	F	F	F	T
	C3 : 1 ≤ Stocks ≤ 80	-	T	F	T	F	T	F	F	T
	C4 : 1 ≤ Barrels ≤ 90	-	F	T	T	F	F	T	F	T
Actions	a1 : Terminate the input loop	X								
	a2 : Invalid locks input				X		X	X	X	
	a3 : Invalid stocks input			X		X		X	X	
	a4 : Invalid barrels input		X			X	X		X	
	a5 : Calculate total locks, stocks and barrels		X	X	X	X	X	X		X
	a5 : Calculate Sales	X								
	a6: proceed to commission decision table	X								

Decision table for commission problem (Precondition: lock = -1)

Rules		R1	R2	R3	R4
Conditions	C1:sales=0	T	F	F	F
	C2: sales>0 and sales< =1000	-	T	F	F
	C3: sales>1001 and sales< =1800	-	-	T	F
	C4: sales >= 1801	-	-	-	T
Actions	A1:Terminate the program	X			
	A2:comm =10% * sales		X		
	A3:comm= 10%*1000+(sales-1000)*15%			X	
	A4:comm = 10%*1000+15%*800+(sales-1800)*20%				X

Precondition : Initial Value Total Locks= 0 , Total Stocks=0 and Total Barrels=0

Precondition Limit : Total locks, stocks and barrels should not exceed the limit 70,80 and 90 respectively.

Commission problem decision table for input data test cases

Case Id	Description	Input data			Expected output	Actual output	status
		Locks	stocks	barrels			
1	Enter the value for locks =-1	-1			Terminate the input loop check for sales if(sales=0) exit from program else calculate commission		
2	Enter valid inputs for locks and stocks and invalid input for barrels	20	30	-5	Total of locks stocks is updated if it is with in a precondition limit and should be display value of barrels is not in range1..90		
3	Enter valid inputs for locks and barrels and invalid input for stocks	15	-2	45	Total of locks barrels is updated if it is with in a precondition limit and should be display value of stocks is not in range1..80		
4	Enter valid inputs for barrels and stocks and invalid input for locks	-4	15	16	Total of barrels stocks is updated if it is with in a precondition limit and should be display value of locks is not in range1..70		
5	Enter valid inputs for locks and invalid input for barrels and stocks	15	80	100	Total of locks is updated if it is with in a precondition limit and (i) should be display value of barrels is not in range1..90 (ii) should be display value of stocks is not in range1..80		
6	Enter valid inputs for stocks and invalid input	88	20	99	Total of stocks is updated if it is with in a precondition limit and (i)should be display		

	for barrels and locks				value of barrels is not in range1..90(ii) should be display value of locks is not in range1..70		
7	Enter valid inputs for barrels and invalid input for locks and stocks	100	200	25	Total of barrels is updated if it is with in a precondition limit and (i) should be display value of stocks is not in range1..80(ii) should be display value of locks is not in range1..70		
8	Enter invalid input for barrels and stocks and locks	-5	400	-9	(i) should be display value of stocks is not in range1..80(ii) should be display value of locks is not in range1..70(iii) should be display value of barrels is not in range1..90		
9	Enter valid inputs for locks and barrels and stocks	15	20	25	Total of locks,stocks and barrels is updated if it is with in a precondition limit and calculate sales and commission		

Commission problem - decision table for pre condition locks=-1

Case id	**Description**	**Input**	**Expected output**		**Actual output**	**Status**
		Sales	**commission**	**values**		
1	Check values for sales	0	Terminates the program and commission =0	0		
2	Sales>0 and sales< =1000	900	Comm =10% * sales	900		
3	Sales>1000 and sales< =1800	1400	Comm= 10%*1000+(sales-1000)*15%	1600		
4	Sales >= 1800	2500	Comm=10%*1000+15%*800+(sales-1800)*20%	3400		

9. Design, develop, code and run the program in any suitable language to solve the commission problem. Analyze it from the perspective of dataflow testing, derive different test cases, execute these test cases and discuss the test results.

```
1.  #include<stdio.h>
2.   int main()
{
3.   int locks, stocks, barrels, tlocks, tstocks, tbarrels;
4.   float lprice,sprice,bprice,lsales,ssales,bsales,sales,comm;
5.   lprice=45.0;
6.   sprice=30.0;
7.   bprice=25.0;
8.   tlocks=0;
9.   tstocks=0;
10. tbarrels=0;
11. printf("\nenter the number of locks and to exit the loop enter -1 for locks\n");
12. scanf("%d", &locks);
13.while(locks!=-1)
     {
14.printf("enter the number of stocks and barrels\n");
15.scanf("%d%d",&stocks,&barrels);
16.tlocks=tlocks+locks;
17. tstocks=tstocks+stocks;
18. tbarrels=btarrels+barrels;
19. printf("\nenter the number of locks and to exit the loop enter -1 for
20.locks\n");
21.scanf("%d",&locks);
22.}
23. printf("\ntotal locks = %d\",tlocks);
24. printf("total stocks =%d\n",tstocks);
25. printf("total barrels =%d\n",tbarrels);

26. lsales = lprice*tlocks;
27. ssales=sprice*tstocks;
28. bsales=bprice*tbarrels;
29. sales=lsales+ssales+bsales;
30. printf("\nthe total sales=%f\n",sales);
31. if(sales > 1800.0)
    {
32. comm=0.10*1000.0;
33. comm=comm+0.15*800;
```

```
34. comm=comm+0.20*(sales-1800.0);
    }
35. else if(sales > 1000)
    {
36.comm =0.10*1000;
37.comm=comm+0.15*(sales-1000);
38.}
39. else
40. comm=0.10*sales;
41. printf("the commission is=%f\n",comm);
42.  return ( ) ;
43. }
```

DATA FLOW TESTING

Test Case Name: Data Flow Testing for Commission Program

Precondition: Enter -1 for locks to exit from input loop Brief

Description: Enter the locks, stocks and barrels > 0

Define /Use nodes for variables in the commission problem

Variable name	Defined at node	Used at node
Lprice	5	26
Sprice	6	27
Bprice	7	28
Tlocks	8,16	16
Tstocks	9,17	17
Tbarrels	10,18	18
Locks	12	21

Selected Define/use paths for commission problem

Test case id	Description	Variable path (beginning and end nodes)	Du paths	Definition clear?
1	Check for lock price variable DEF(lprice,5) and USE(lprice,26)	(5,26)	<5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26>	Yes
2	Check for lock variable DEF(locks ,12) and USE(locks,)	(12,22)	<12-13-14-15-16-17-18-19-20-**21**-22>	No

Note: Find path for all variables used and identify Definition Clear and non-Definition Clear paths

10. Design, develop, code and run the program in any suitable language to implement the binary search algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results

```
#include<stdio.h>
int binsrc(int x[],int low,int high,int key)
{
        int             mid;
        while(low<=high)
        {
                mid=(low+high)/2;
                if(x[mid]==key)
                        return mid;
                if(x[mid]<key)
                        low=mid+1;
                else
                        high=mid-1;
        }
        return -1;
}

int main()
{
        int         a[20],key,i,n,succ;
        printf("Enter the n value");
        scanf("%d",&n);
        if(n>0)
        {
        printf("enter the elements in ascending order\n");
                for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        printf("enter the key element to be searched\n");
              scanf("%d",&key);
                succ=binsrc(a,0,n-1,key);
                if(succ>=0)
                        printf("Element found in position = %d\n",succ+1);
                else
```

```
                printf("Element not found \n");
        }
        else
                printf("Number of element should be greater than zero\n");
        return 0;
}
```

PATH TESTING

Binary Search function with line number

```
int binsrc(int x[],int low, int high, int key)
{
        int mid;                                1
        while(low<=high)                        2
                {
                mid=(low+high)/2;
                if(x[mid]==key)                 3
                        return mid;             8
                if(x[mid]<key)                  4
                        low=mid+1;              5
                        else
                        high=mid-1;             6
        }                                       7
        return -1;                              8
}                                               9
```

PATH TESTING

Independent paths :

#Edges =11 ,#nodes= 9, #p =1

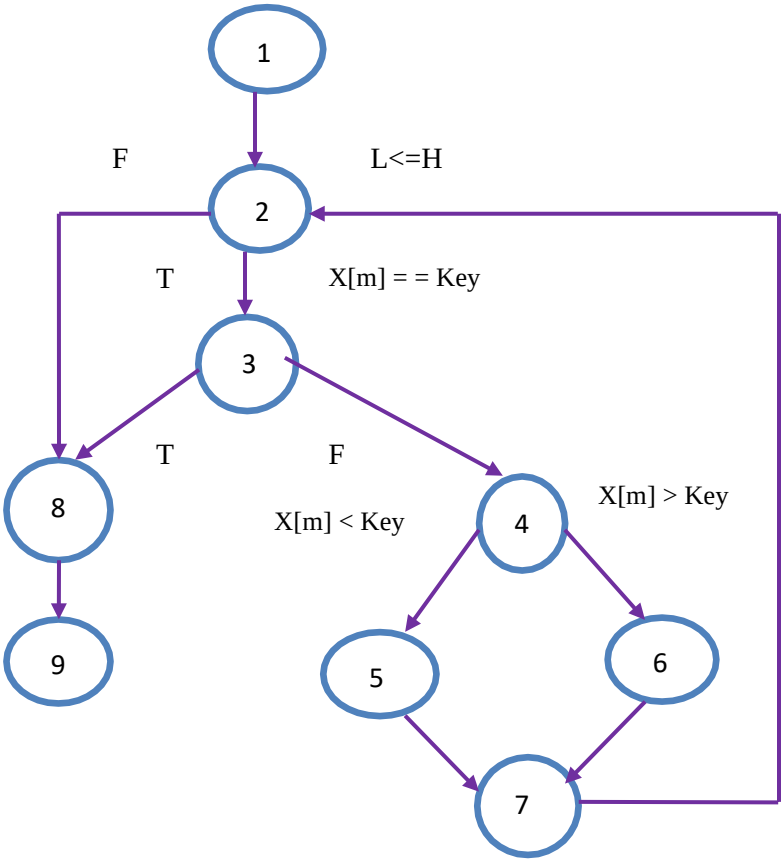
V(G)=E-N+2P=11-9+2=4

P1: 1-2-3-8-9

P2: 1-2-3-4-5-7-2

P3: 1-2-3-4-6-7-2

P4: 1-2-8-9



PRE CONDITIONS:

Array element is in ascending order True /False
Key element is in the array True /False
Array has ODD number of elements True /False

Paths	Inputs		Expected output	Remarks
	X[ ]	Key		
P1: 1-2-3-8-9	{10,20,30,40,50}	30	Success	Key ∈X[ ] & Key==X[mid]
P2: 1-2-3-4-5-7-2	{10,20,30,40,50}	20	Repeat and success	Key < X[mid] search 1st half
P3: 1-2-3-4-6-7-2	{10,20,30,40,50}	40	Repeat and success	Key >X[mid] search 2nd half
P4: 1-2-8-9	{10,20,30,40,50}	60 or 50	Repeat and Failure	Key∉X[mid]

P5: 1-2-8-9	Empty	Any key	Failure	Empty list

”

11. Design, develop, code and run the program in any suitable language to implement the quick sort algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
void quicksort(int x[10],int first,int last)
{
        int temp,pivot,i,j;
        if(first<last)
        {
                pivot=first;
                i=first;
                j=last;
                while(i<j)
                  {
                        while(x[i]<=x[pivot] && i<last)
                                i++;
                       while(x[j]>x[pivot]) j--
                                ;
                         if(i<j)
                        {
                                  temp=x[i];
```

```
                                x[i]=x[j];
                                x[j]=temp;
                        }
                    }
                 temp=x[pivot];
                x[pivot]=x[j];
                x[j]=temp;
               quicksort(x,first,j-1);
               quicksort(x,j+1,last);
        }
}
```

// main program

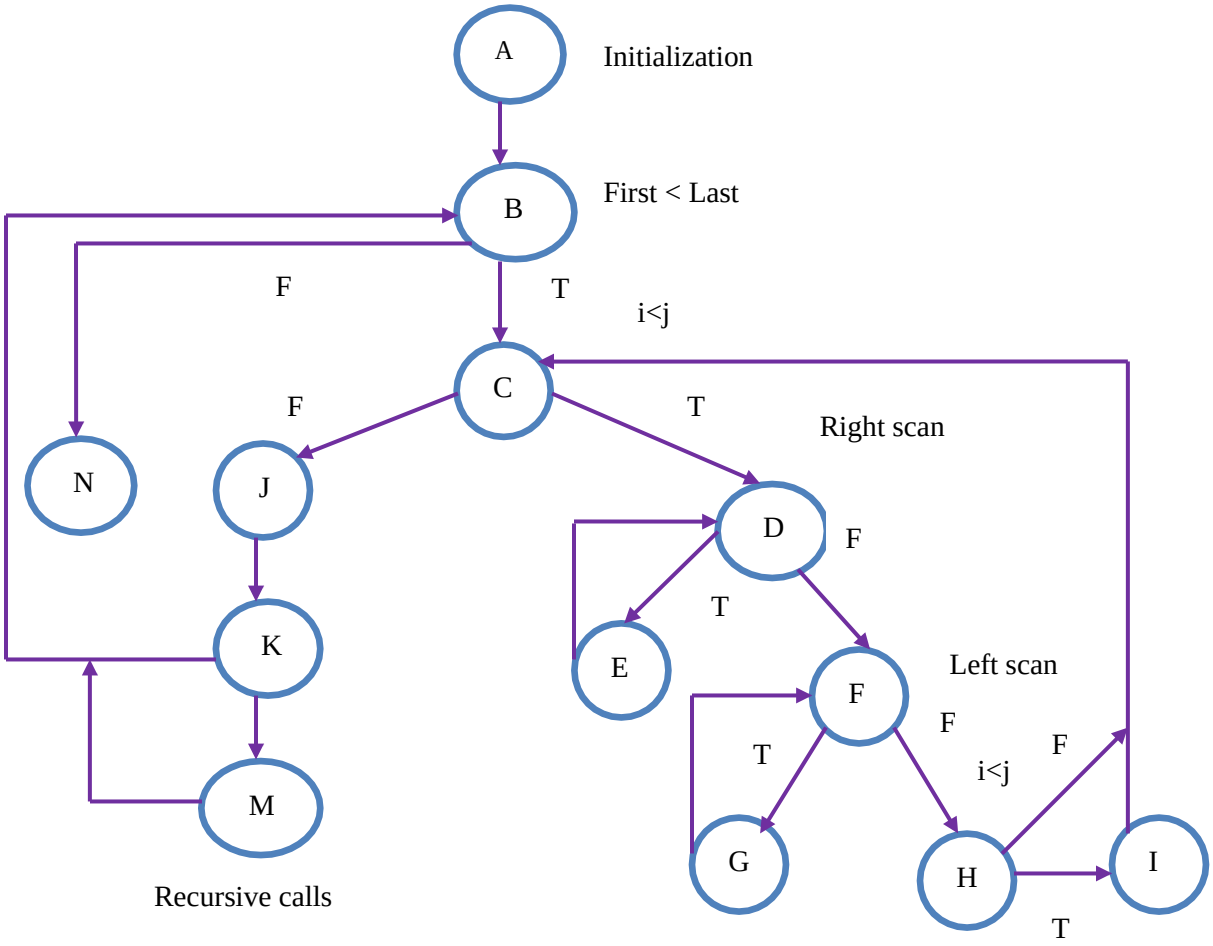
```
int main()
{
        int a[20],i,key,n;
        printf("enter the size of the array");
        scanf("%d",&n);
        if(n>0)
        {
                printf("enter the elements of the array");
                for(i=0;i<n;i++)
                        scanf("%d",&a[i]);
                quicksort(a,0,n-1);
                printf("the elements in the sorted array is:\n");


              for(i=0;i<n;i++)
                  printf("%d\t",a[i]);
        }
        else
        {
                printf("size of array is invalid\n");
}
```

PATH TESTING

Quick sort function with line number

```
void quicksort(int x[10],int first,int last)
{
        int temp,pivot,i,j;                                    1
        if(first<last)                                         2
```

```
        {
                pivot=first;                                    3
         i=first;                                               4
                j=last;                                         5
                while(i<j)                                      6
                  {
                while(x[i]<=x[pivot] && i<last)                 7
                                i++;                            8
                        while(x[j]>x[pivot])                    9
                                j--;                            10
                                 if(i<j)                        11
                        {
                                 temp=x[i];                     12
                                 x[i]=x[j];                     13
                                 x[j]=temp;                     14
                        }
          }
                        temp=x[pivot];                          15
                        x[pivot]=x[j];                          16
                x[j]=temp;                                      17
                quicksort(x,first,j-1);                         18
                quicksort(x,j+1,last);                          19
        }
}                                                               20
```



Independent paths – Quick sort

P1: A-B-N
P2: A-B-C-J-K-B
P3: A-B-C-J-K-M-B
P4: A-B-C-D-F-H-C
P5: A-B-C-D-F-H-I-C
P6: A-B-C-D-E-D-F-H
P7: A-B-C-D-F-G-F-H

Independent Paths: #Edges =18, #Nodes =13, #P =1

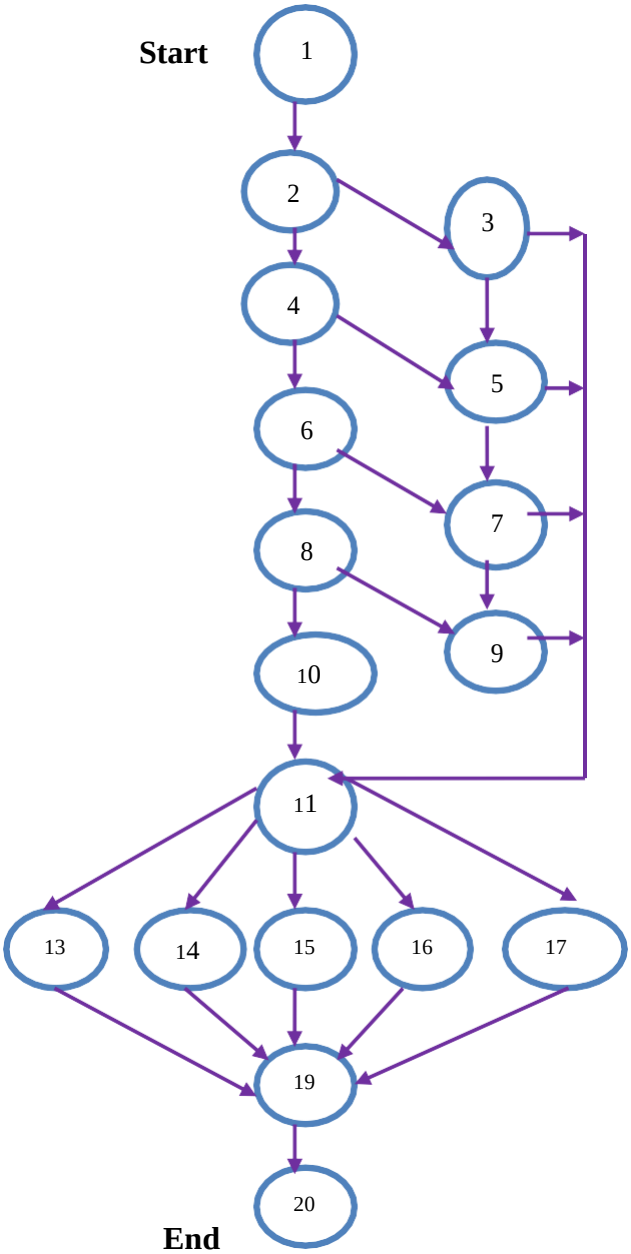
V (G) =E-N+2P=18-13+2=7

12. Design, develop, code and run the program in any suitable language to implement an absolute letter grading procedure, making suitable assumptions. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
 int main()
{

float per;
char grade;

scanf("%f",&per);

if(per>=90)
 grade= 'A';
else if(per>=80 && per<90)
grade ='B';
else if(per>=70 && per<80)
grade ='C';
else if(per>=60 && per<70)
grade='D';
else grade='E';

switch(grade)
{
case 'A': printf("\nEXCELLENT");
break;
case 'B':printf("\nVery Good");
break;
case 'C' : printf("\nGood");
 break;
case 'D': printf("\nAbove Average");
break;
case 'E': printf("\n Satisfactory");
break;
}
printf("\t The percentage = %f and grade is %c ",per,grade);
return 0;
}
```

Structural Testing: Path Testing

```
    int main()
    {
    float per;
    char grade;
1.  scanf("%f",&per);

2.  if(per>=90)
3.   grade= 'A';

4.  else if(per>=80 && per<90)
5.  grade ='B';

6.  else if(per>=70 && per<80)
7.  grade ='C';

8.  else if(per>=60 && per<70)
9.  grade='D';

10. else grade='E';

11. switch(grade)
12. {
13. case 'A': printf("\nEXCELLENT");
    break;
14. case 'B':printf("\nVery Good");
    break;
15. case 'C' : printf("\nGood");
    break;
16. case 'D': printf("\nAbove Average");
    break;
17. case 'E': printf("\n Satisfactory");
    break;
18. }
19. printf("\t The percentage = %f and grade is %c ",per,grade);
20. return ( ) ;
    }
```



Independent Paths:

#Edges=25, #Nodes=18, #P=1
V(G)= E-N+2P = 25-18+2 = **09**

P1: 1-2-4-6-8-10-11-17-19-20	E Grade
P2: 1-2-4-6-8-9-11-16-19-20	D Grade
P3: 1-2-4-6-7-11-15-19-20	C Grade
P4: 1-2-4-5-11-14-19-20	B Grade
P5: 1-2-3-11-13-19-20	A Grade
P6: 1-2-4-6-8-10-11-13-19-20	
P7: 1-2-4-6-8-10-11-14-19-20	
P8: 1-2-4-6-8-10-11-15-19-20	
P9: 1-2-4-6-8-10-11-16-19-20	

Pre-Conditions/Issues:

Percentage Per is a positive Float Number

Test Cases – Absolute Grading

Paths	Input per	Expected output	Remarks
P1:1-2-4-6-8-10-11-17-19-20	< 60	E grade , satisfactory	Pass
P2:1-2-4-6-8-9-11-16-19-20	60-69	D grade , Above average	Pass
P3:1-2-4-6-7-11-15-19-20	70-79	C grade ,good	Pass
P4:1-2-4-5-11-14-19-20	80-89	B grade , very good	Pass
P5:1-2-3-11-13-19-20	>=90	A grade , excellent	Pass
P6:1-2-4-6-8-10-11-13-19-20	< 60	Excellent	Pass
P7: 1-2-4-6-8-10-11-14-19-20	< 60	Very good	Pass
P8: 1-2-4-6-8-10-11-15-19-20	< 60	Good	Pass
P9: 1-2-4-6-8-10-11-16-19-20	< 60	Above average	Pass

ADDITIONAL LAB EXPERIMENTS:

1. Write a program to implement dataflow testing for Triangle problem.
2. Write a program to implement Decision-table approach for Next Date problem.
3. Study of Any Testing Tool.
4. Create a test plan document for any application.
5. Write the test cases for any known application also given the justification about the testing technique suitable.
6. Take any system (e.g. ATM system) and study its system specifications and report the various bugs.
7. Explain the data flow testing metrics evaluation for any program/application.
8. Perform path testing form any program with while loop.
9. Perform path testing form any program with switch case statement.
10. Perform path testing to demonstrate cyclomatic complexity greater than 10. Also Discuss the essential complexity for the same.

VIVA QUESTIONS

1. Define the following
   i. error
   ii. Fault
   iii. Failure
   iv. Incident
   v. Test
   vi. Test cases

2. What are pre conditions?

3. What is post Conditions?

4. What is Functional (Black box) testing?

5. What is structural testing (Clear box, White box testing)

6. Different types of faults?

7. List different types of functional testing?

8. List different types of structural testing?

9. What is boundary value analysis?

10. Number of test cases for boundary value analysis for n numbers?

11. What type of variables we are using boundary value analysis?

12. What are the 5 boundary value analysis values of a variable?

13. What is Robust testing?

14. What is worst case casting?

15. Number of test cases for worst case testing?

16. What is robust worst case testing?

17. Number of test cases for robust worst case testing?

18. What is special value testing (skirt testing)?

19. What is the limitation of boundary value analysis?

20. What is equivalence class?

21. What is equivalence class testing?

22. What is weak normal equivalence class testing?

23. What is strong normal equivalence class testing?

24. What is weak robust equivalence class testing?

25. What is strong robust equivalence class testing?

26. When equivalence class is appropriate?

27. What is decision table based testing?

28. What are the different positions of decision table?

29. What is a rule?

30. What are the different types of decision tables?

31. What are limited entry decision tables?

32. What is extended entry decision tables?

33. When decision table based testing is appropriate?

34. What is a program graph?

35. What is DD (decision to decision) path?

36. What is test coverage metrics?

37. What is basis path testing?

38. What is data flow testing?

39. What is Def (v.n)?

40. What is Use (v,n)?

41. What is definition use path?

42. What is a slice?

43. Different types of use nodes?

44. Different types of definition nodes?

45. What is a system thread?

46. What is automic system function?

47. What is unit testing?

48. What is system testing?

49. What is Integration testing?

50. What is interaction testing?