SOFTWARE TESTING LABORATORY

Sub Code : 18ISL66
Hrs / Week : 03
Total Hrs : 36

CIE Marks : 40
Exam Hours : 03
SEE Marks: 60

Syllabus

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.
.

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.

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.

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

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.

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.

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 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. Derive test cases for your program based on decision-table approach, execute the test cases and discuss the results.

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.

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.

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.

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.

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.

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.

Requirements Specification:

Req1: The program should accept three numbers for a,b,c that is the three sides of the triangle. By giving the input it should be determine whether the triangle can be formed or not.

Req2: The numbers given as input to the triangle a,b,c shouldn't exceed to 10.

Req3: If the requirement req1 is satisfied then the program should determine the types of the triangle. The type of triangle can be,

Equilateral Triangle → Three sides are equal.

Isosceles Triangle → Any two sides are equal.

Scalene Triangle → Three sides are not equal.

Req4: If the triangle is not formed print the message saying triangle is not formed.

Design:

C1: a<b+c
C2: b<a+c
C3: c<a+b
C4: a=b
C5: a=c
C6: b=c

According to the property of triangle if any one of the three condition C1, C2, C3 are not satisfied, then triangle cannot be constructed. If C1, C2, C3 are true then triangle can be formed, with C4, C5, C6 we decide what type of triangle can be formed.

Coding:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
inta,b,c;
printf("Enter the sides of the triangle:\n");
scanf("%d%d%d",&a,&b,&c);
if((a<1 || a>10)&&(b<1 || b>10)&&(c<1 || c>10))
printf("Sides a,b and c are of out of range");
else if((a<1 || a>10)&&(b<1 || b>10))
printf("Sides a and b are out of range");
else if((b<1 || b>10)&&(c<1 || c>10))
printf("Sides b and c are out of range");
else if((a<1 || a>10)&&(c<1 || c>10))
printf("Sides a and c are out of range");
else if(a<1 || a>10)
printf("side a is out of range.");
else if(b<1 || b>10)
printf("side b is out of range.");
else if(c<1 || c>10)
printf("side c is out of range.");
else
  {
if((a<b+c)&&(b<a+c)&&(c<a+b))
    {
printf("Triangle is formed..\n");
          if(a==b && b==c && c==a )
          printf("Equilateral triangle..\n");
          else if(a==b || b==c || c==a)
          printf("Isoceles triangle..\n");
          else
          printf("Scalene triangle...\n");
    }
else
    {
printf("not a triangle..\n");
    }
  }
return 0;}
```

Testing:

Technique used Boundary Value Analysis Approach:

Test cases:

TC ID	Purpose	Actual input			Expected output	Actual output	Status
		A	B	C			
1.	Testing for R3	5	5	1	Isosceles triangle		
2.	Testing for R3	5	5	2	Isosceles triangle		
3.	Testing for R3	5	5	9	Isosceles triangle		
4.	Testing for R3	5	5	10	Not a Triangle		
5.	Testing for R4	5	5	5	Equilateral triangle		
6.	Testing for R3	5	1	5	Isosceles triangle		
7.	Testing for R3	5	2	5	Isosceles triangle		
8.	Testing for R3	5	9	5	Isosceles triangle		
9.	Testing for R4	5	10	5	Not a Triangle		
10.	Testing for R3	1	5	5	Isosceles triangle		
11.	Testing for R3	2	5	5	Isosceles triangle		
12.	Testing for R3	9	5	5	Isosceles triangle		
13.	Testing for R4	10	5	5	Not a Triangle		

Inputs	Min	Min+	Nom	Max-	Max
A	1	2	5	9	10
B	1	2	5	9	10
C	1	2	5	9	10

Report Generation:

- Number of test cases executed:
- Number of test cases passed:
- Number of test cases failed:

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

Requirements Specification:

R1. The system should accept 3 values for locks, stocks and barrels.

R2. Limit ranges 1-70 for locks, 1-80 for stocks and 1-90 for barrels.

R3. Cost of each lock= $45, stock= $30, barrel= $25.

R4. Commission for total sales includes:

Sales up to & including $1000= 10%

Sales up to & including $1800= 15%

Sales in excess of $1800= 20%

DESIGN

C1: 1<=locks<=70
C2: 1<=stocks<=80
C3: 1<=barrels<=90
C4: sales>1800
C5: sales>1000
C6: sales<=1000

Coding:

```
#include<stdio.h>
int main()
{
intlocks,stocks,barrels,sales,flag=0;
float commission;
printf("Enter the total number of locks\n");
scanf("%d",&locks);
printf("Enter the total number of stocks\n");
scanf("%d",&stocks);
printf("Enter the total number of barrels\n");
scanf("%d",&barrels);
if((locks<=0) || (locks>70))
{
```

```
flag=1;
printf("Locks Out of Range\n");
}
if((stocks<=0) || (stocks>80))
{
flag=1;
printf("Stocks Out of Range\n");
}
if((barrels<=0) || (barrels>90))
{
flag=1;
printf("Barrels Out of Range\n");
}
if(flag == 1)
{
printf("Invalid input\n");
return 0;
}
sales= locks*45 + stocks*30 + barrels*25;
if(sales <= 1000)
{
commission = 0.10 * sales;
}
else if(sales <= 1800)
{
commission = 0.10 * 1000;
commission = commission + (0.15 * (sales - 1000));
}
else
{
commission = 0.10 * 1000;
commission = commission + (0.15 * 800);
commission = commission + (0.20 * (sales - 1800));
}
printf("Total sales: %d and Commission: %f\n",sales,commission);
return 0;
}
```

TESTING:

Technique used: Boundary value analysis:

BVA is a black box technique. For BVA problem the test case generation depends on the output and constraints on the output. This method is based on single fault assumption. This

method does not calculate the values for out of range. Since BVA consists of (4n+1) test cases according to Single Fault Theory.

Inputs	Min	Min+	Nom	Max-	Max
Locks	1	2	35	69	70
Stocks	1	2	40	79	80
Barrels	1	2	45	89	90

Test Cases:

TCID	Purpose	Expected input			Expectedoutput		Actual output		Status
		Lock	Stock	Barrel	T_sale	Comm	T_sale	Comm	
1	Testing for R1	35	40	1	2800	420	2800	420	pass
2	Testing for R1	35	40	2	2825	425	2825	425	pass
3	Testing for R1	35	40	45	3900	640	3900	640	pass
4	Testing for R1	35	40	89	5000	860	5000	860	pass
5	Testing for R1	35	40	90	5025	865	5025	865	pass
6	Testing for R1	35	1	45	2730	406	2730	406	pass
7	Testing for R1	35	2	45	2760	412	2760	412	pass
8	Testing for R1	35	79	45	5070	874	5070	874	pass
9	Testing for R1	35	80	45	5100	880	5100	880	pass
10	Testing for R1	1	40	45	2370	334	2370	334	pass
11	Testing for R1	2	40	45	2415	343	2415	343	pass
12	Testing for R1	69	40	45	5430	946	5430	946	pass
13	Testing for R1	70	40	45	5475	955	5475	955	pass

REPORT GENERATION

Number of test cases executed : 13
Number of test cases passed : 13
Number of test cases failed : 0

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

Requirements Specification:

R1: Accepting the input date

1<=month<=12
1<=day<=31
1812<=year<=2016

R2: if R1 is true, return the next date.
R3: if R1 is false, display the appropriate error message

Coding:

```
#include<stdio.h>
int main()
{
        intdd,mm,yy,flag=0;
        Printf("Enter the date,month and year\n");
        scanf("%d%d%d",&dd,&mm,&yy);
        if(dd<=0||dd>31)
        {
                printf("Day out of range\n");
                flag=1;
        }
        if(mm<=0||mm>12)
        {
                printf("Month out of range\n");
                flag=1;
        }
        if(yy<=1812||yy>2012)
        {
                printf("Year out of range\n");
                flag=1;
        }
        if(flag==0)
        {
                if(mm==1||mm==3||mm==5||mm==7||mm==8||mm==10)
                {
                        if(dd<31)
```

```
                dd=dd+1;
        else
        {
                dd=1;
                mm=mm+1;
        }
}
else if(mm==4||mm==6||mm==9||mm==11)
{
        if(dd<30)
                dd=dd+1;
        else if(dd==31)
        {
                printf("Date 31 does not exist in this month\n");
                return 0;
        }
        else
        {
                dd=1;
                mm=mm+1     ;
        }
}
else if(mm==12)
{
        if(dd<31)
                dd=dd+1;
        else
        {
                dd=1;
                mm=1;
                yy=yy+1;
        }
}
else
{
        if((yy%4==0 && yy%100!=0) || (yy%400==0))
        {
                if(dd<29)
                        dd=dd+1;
                else if(dd>29)
                {
```

```
                        printf("Date %d does not exist in this month\n",dd);
                                return 0;
                        }
                        else
                        {
                                dd=1;
                                mm=3;
                        }
                }
                else
                {

                        if(dd<28)
                                dd=dd+1;
                        else if(dd>28)
                        {
printf("Date %d does not exist in this month\n",dd);
                                return 0;
                        }
                        else
                        {
                                dd=1;
                                mm=3;
                        }
                }
        }
        printf("Next date:%d-%d-%d\n",dd,mm,yy);

        return 0;
    }
    else
        return 0;
}
```

TESTING:

Technique used is Boundary value analysis. It is a functional testing method. This technique is used to identify errors at boundaries rather than finding those exist in center of input domain. It consists of 4n+1 test cases, where n is no of input.

Here n=3, hence 13 test cases.

	Min	Min+	Nom	Max-	Max
Date	1	2	15	30	31
Month	1	2	6	11	12
Year	1812	1813	1914	2015	2016

Test cases:

TC id	Test case description	Input data dd mm yyyy	Expected output dd mm yyyy	Actual output	Status
1	Testing for R2	15 6 1812	16 6 1812	16 6 1812	Pass
2	Testing for R2	15 6 1813	16 6 1813	16 6 1813	Pass
3	Testing for R2	15 6 1914	16 6 1914	16 6 1914	Pass
4	Testing for R2	15 6 2015	16 6 2015	16 6 2015	Pass
5	Testing for R2	15 6 2016	16 6 2016	16 6 2016	Pass
6	Testing for R2	15 1 1914	16 1 1914	16 1 1914	Pass
7	Testing for R2	15 2 1914	16 2 1914	16 2 1914	Pass
8	Testing for R2	15 6 1914	16 6 1914	16 6 1914	Pass
9	Testing for R2	15 11 1914	16 11 1914	16 11 1914	Pass
10	Testing for R2	1 6 1914	2 6 1914	2 6 1914	Pass
11	Testing for R2	2 6 1914	3 6 1914	3 6 1914	Pass
12	Testing for R2	30 6 1914	1 7 1914	1 7 1914	Pass
13	Testing for R3	31 6 1914	Date 31 does not exist in this month	Date 31 does not exist in this month	Pass

TEST REPORT:

No of TC's executed: 13
No of TC's passed: 13
No of TC's failed: 0

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.

Requirements Specification:

Req1: The program should accept three numbers for a,b,c that is the three sides of the triangle. By giving the input it should be determine wheather the triangle can be formed or not.

Req2: The numbers given as input to the triangle a,b,c shouldn't exceed to 10.

Req3: If the requirement req1 is satisfied then the program should determine the types of the triangle. The type of triangle can be,

Equilateral Triangle → Three sides are equal.

Isosceles Triangle → Any two sides are equal.

Scalene Triangle → Three sides are not equal.

Req4: If the triangle is not formed print the message saying triangle is not formed.

Design:

C1: a<b+c
C2: b<a+c
C3: c<a+b
C4: a=b
C5: a=c
C6: b=c

According to the property of triangle if any one of the three condition C1, C2, C3 are not satisfied, then triangle cannot be constructed. If C1, C2, C3 are true then triangle can be formed, with C4, C5, C6 we decide what type of triangle can be formed.

Coding:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
inta,b,c;
printf("Enter the sides of the triangle:\n");
scanf("%d%d%d",&a,&b,&c);
if((a<1 || a>10)&&(b<1 || b>10)&&(c<1 || c>10))
printf("Sides a, b and c are of out of range");
else if((a<1 || a>10)&&(b<1 || b>10))
printf("Sides a and b are out of range");
else if((b<1 || b>10)&&(c<1 || c>10))
printf("Sides b and c are out of range");
else if((a<1 || a>10)&&(c<1 || c>10))
printf("Sides a and c are out of range");
else if(a<1 || a>10)
printf("side a is out of range.");
else if(b<1 || b>10)
printf("side b is out of range.");
else if(c<1 || c>10)
printf("side c is out of range.");
else
  {
if((a<b+c)&&(b<a+c)&&(c<a+b))
```

```
    {
printf("Triangle is formed..\n");
        if(a==b && b==c && c==a )
        printf("Equilateral triangle..\n");
        else if(a==b || b==c || c==a)
        printf("Isoceles triangle..\n");
        else
        printf("Scalene triangle...\n");
    }
else
    {
printf("not a triangle..\n");
    }
  }
return 0;
}
```

Testing

Technique used equivalence class partitioning Approach:

Test cases 1: Weak normal equivalence class

TC ID	Purpose	Actual input			Expected output	Actual output	status
		a	b	c			
WN1	Testing for R3	2	2	2	Equilateral triangle		
WN2	Testing for R3	2	1	2	Isosceles triangle		
WN3	Testing for R3	3	4	5	Scalene triangle		
WN4	Testing for R4	1	2	3	Not a triangle		

Test cases 2

Strong normal equivalence class: The data given cannot be divided into subsets. We dont get extra test cases. Hence they get same test cases as weak normal equivalence class.

Test cases 3: Week Robust

TC ID	Purpose	Actual input			Expected output	Actual output	status
		a	b	c			
WR1	Testing for R2	-1	2	2	Side a is out of range		
WR2	Testing for R2	11	2	2	Side a is out of range		
WR3	Testing for R2	7	-5	6	Side b is out of range		
WR4	Testing for R2	7	12	6	Side b is out of range		
WR5	Testing for R2	3	4	-2	Side c is out of range		
WR6	Testing for R2	4	5	16	Side c is out of range		

Test cases 4: Strong robust

TC ID	Purpose	Actual input			Expected output	Actual output	status
		a	b	c			
SR1	Testing for R3	-2	-2	1	Sides a and b are out of range		
SR2	Testing for R3	11	12	3	Sides a and b are out of range		
SR3	Testing for R3	2	-3	-1	Sides b and c are out of range		
SR4	Testing for R3	2	11	16	Sides b and c are out of range		
SR5	Testing for R3	-3	6	-9	Sides a and c are out of range		

SR6	Testing for R3	11	8	22	Sides a and c are out of range		
SR7	Testing for R3	-3	-4	-1	Sides a, b and c are of out of range		
SR8	Testing for R3	12	19	26	Sides a, b and c are of out of range		

Report Generation:

- Number of test cases executed:
- Number of test cases passed:
- Number of test cases failed:

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

Requirements Specification:

R1. The system should accept 3 values for locks, stocks and barrels.

R2. Limit ranges 1-70 for locks, 1-80 for stocks and 1-90 for barrels.

R3. Cost of each lock= $45, stock= $30, barrel= $25.

R4. Commission for total sales includes:

Sales up to & including $1000= 10%

Sales up to & including $1800= 15%

Sales in excess of $1800= 20%

DESIGN

C1: 1<=locks<=70
C2: 1<=stocks<=80
C3: 1<=barrels<=90
C4: sales>1800
C5: sales>1000
C6: sales<=1000

Coding:

```
#include<stdio.h>
int main()
{
intlocks,stocks,barrels,sales,flag=0;
float commission;
printf("Enter the total number of locks\n");
scanf("%d",&locks);
printf("Enter the total number of stocks\n");
scanf("%d",&stocks);
printf("Enter the total number of barrels\n");
scanf("%d",&barrels);
if((locks<=0) || (locks>70))
{
flag=1;
```

```
printf("Locks out of range\n");
}
if((stocks<=0) || (stocks>80))
{
flag=1;
printf("Stocks out of range\n");
}
if((barrels<=0) || (barrels>90))
{
flag=1;
printf("Barrels out of range\n");
}
if(flag == 1)
{
printf("Invalid input\n");
return 0;
}
sales= locks*45 + stocks*30 + barrels*25;
if(sales <= 1000)
{
commission = 0.10 * sales;
}
else if(sales <= 1800)
{
commission = 0.10 * 1000;
commission = commission + (0.15 * (sales - 1000));
}
else
{
commission = 0.10 * 1000;
commission = commission + (0.15 * 800);
commission = commission + (0.20 * (sales - 1800));
}
printf("Total sales: %d and Commission: %f\n",sales,commission);
return 0;
}
```

TESTING:

Technique used: Equivalence class testing
This technique focuses on input domain. We can obtain a set of test cases. It is divided into 'n' subsets which would be mutually disjoint.
The first class is the valid input , the other two are invalid .

The valid classes of input variables are:
L1= {locks: 1<=locks<=70}
S1= {stocks: 1<=stocks<=80}
B1= {barrels: 1<=barrels<=90}
The invalid classes of input variables are:
L3= {locks: locks<=0}
L4= {locks: locks>70}
S2= {stocks: stocks<=0}
S3= {stocks: stocks>80}
B2= {barrels: barrels<=0}
B3= {barrels: barrels>90}

TEST CASES:

Weak Normal Equivalence Class:

TCID	Purpose	Actual input			Expected output		Actual output		Status
		locks	Stocks	barrels	t_sales	comm	t_sales	comm	
WN1	Testing for R4	20	30	50	3050	470	3050	470	pass

Strong Normal Equivalence Class:

The data given cannot be divided into subsets. Hence we get same test case as weak normal.

Weak Robust Equivalent Class:

TCID	Purpose	Actual input			Expected output	Actual output	Status
		locks	stocks	barrels			
WR1	Testing for R2	0	65	70	Locks out of range	Locks out of range	Pass
WR2	Testing for R2	75	60	75	Locks out of range	Locks out of range	Pass
WR3	Testing for R2	50	-2	85	Stocks out of range	Stocks out of range	Pass
WR4	Testing	60	86	88	Stocks	Stocks	Pass

	for R2				out of range	out of range	
WR5	**Testing for R2**	55	70	-50	Barrels out of range	Barrels out of range	**pass**
WR6	**Testing for R2**	62	75	95	Barrels out of range	Barrels out of range	**pass**

Strong Robust Equivalence Class:

TCID	**Purpose**	**Actual input**			**Expected output**	**Actual output**	**Status**
		locks	**stocks**	**barrels**			
SR1	**Testing for R2**	0	-1	85	Locks out of range Stocks out of range Invalid input	Locks out of range Stocks out of range Invalid input	**Pass**
SR2	**Testing for R2**	72	86	88	Locks out of range Stocks out of range Invalid input	Locks out of range Stocks out of range Invalid input	**Pass**
SR3	**Testing for R2**	-2	75	96	Locks out of range Barrels out of range Invalid input	Locks out of range Barrels out of range Invalid input	**Pass**
SR4	**Testing for R2**	-10	76	99	Locks out of range Barrels out of range Invalid input	Locks out of range Barrels out of range Invalid input	**Pass**
SR5	**Testing for R2**	10	0	-20	Stocks out of range Barrels out of range Invalid input	Stocks out of range Barrels out of range Invalid input	**Pass**
SR6	**Testing for R2**	20	86	95	Stocks out of range	Stocks out of range	**Pass**

					Barrels out of range Invalid input	Barrels out of range Invalid input	
SR7	**Testing for R2**	-2	0	-15	Locks out of range Stocks out of range Barrels out of range Invalid input	Locks out of range Stocks out of range Barrels out of range Invalid input	**Pass**
SR8	**Testing for R2**	75	85	95	Locks out of range Stocks out of range Barrels out of range Invalid input	Locks out of range Stocks out of range Barrels out of range Invalid input	**Pass**

REPORT GENERATION

- Number of test cases executed : 16
- Number of test cases passed : 16
- Number of test cases failed : 0

6. Design, develop, code and run the program in any suitable language to implement the NextDate function. Analyze it from the perspective of equivalence class analysis. Derive different test cases, execute these test cases and discuss the test results.

Requirement Specification:

R1: Accepting the input date

1<=month<=12
1<=day<=31
1812<=year<=2016

R2: if R1 is true, return the next date.
R3: if R1 is false, display the appropriate error message

coding:

```
#include<stdio.h>
int main()
{
        intdd,mm,yy,flag=0;
        Printf("Enter the date,month and year\n");
        scanf("%d%d%d",&dd,&mm,&yy);
        if(dd<=0||dd>31)
        {
                printf("Day out of range\n");
                flag=1;
        }
        if(mm<=0||mm>12)
        {
                printf("Month out of range\n");
                flag=1;
        }
        if(yy<=1812||yy>2012)
        {
                printf("Year out of range\n");
                flag=1;
        }
        if(flag==0)
        {
                if(mm==1||mm==3||mm==5||mm==7||mm==8||mm==10)
                {
```

```
        if(dd<31)
                dd=dd+1;
        else
        {
                dd=1;
                mm=mm+1;
        }
}
else if(mm==4||mm==6||mm==9||mm==11)
{
        if(dd<30)
                dd=dd+1;
        else if(dd==31)
        {
                printf("Date 31 does not exist in this month\n");
                return 0;
        }
        else
        {
                dd=1;
                mm=mm+1     ;
        }
}
else if(mm==12)
{
        if(dd<31)
                dd=dd+1;
        else
        {
                dd=1;
                mm=1;
                yy=yy+1;
        }
}
else
{
        if((yy%4==0 && yy%100!=0) || (yy%400==0))
        {
                if(dd<29)
                        dd=dd+1;
                else if(dd>29)
```

```
                        {
                        printf("Day %d does not exist in this month\n",dd);
                                return 0;
                        }
                        else
                        {
                                dd=1;
                                mm=3;
                        }
                }
                else
                {

                        if(dd<28)
                                dd=dd+1;
                        else if(dd>28)
                        {
printf("Day %d does not exist in this month\n",dd);
                                return 0;
                        }
                else
                        {
                                dd=1;
                                mm=3;
                        }
                }
        }
        printf("Next date:%d-%d-%d\n",dd,mm,yy);

        return 0;
    }
    else
        return 0;
}
```

TESTING:

Testing technique: Equivalence class testing.

In this technique, input data is subdivided into four equivalence classes based on which test cases are written. Four equivalence classes are weak normal, strong normal, weak robust and strong robust.

FIRST ATTEMPT

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

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

1. **WEAK ROBUST**: The word 'weak' means single fault assumption theory and the word 'Robust' refers to invalid values.

Tc id	Test case description	Actual input DD MM YYYY	Expected output DD MM YYYY	Actual output DD MM YYYY	Status
1	WR1	18 -12 2000	Month out of range	Month out of range	Pass
2	WR2	20 15 2000	Month out of range	Month out of range	Pass
3	WR3	-30 3 1990	Day out of range	Day out of range	Pass
4	WR4	64 4 1995	Day out of range	Day out of range	Pass
5	WR5	4 7 1810	Year out of range	Year out of range	Pass
6	WR6	18 8 2020	Year out of range	Year out of range	Pass

2. STRONG ROBUST: Robust means consideration of invalid values and the strong means multiple fault assumption. We obtain the test cases from each element of the Cartesian product of all the equivalence classes

Tc id	Test case description	Actual input DD MM YYYY	Expected output DD MM YYYY	Actual output DD MM YYYY	Status
1	SR1	-3 -2 2001	Day out of range Month out of range	Day out of range Month out of range	Pass
2	SR2	-64 20 2002	Day out of range Month out of range	Day out of range Month out of range	Pass

3	SR3	-16 12 1766	Day out of range Year out of range	Day out of range Year out of range	Pass
4	SR4	56 5 2050	Day out of range Year out of range	Day out of range Year out of range	Pass
5	SR5	25 -12 1010	Month out of range Year out of range	Month out of range Year out of range	Pass
6	SR6	17 25 2070	Month out of range Year out of range	Month out of range Year out of range	Pass
7	SR7	-16 -30 1030	Day out of range Month out of range Year out of range	Day out of range Month out of range Year out of range	Pass
8	SR8	90 80 2019	Day out of range Month out of range Year out of range	Day out of range Month out of range Year out of range	Pass

3.WEAK NORMAL AND STRONG NORMAL:

The word 'weak' means 'single fault assumption'. This type of testing is accomplished by using one variable from each valid equivalence class.

Strong normal is same as weak normal in this case, as Cartesian product of equivalence classes cannot be obtained.

Tc id	Test case description	Actual input MM DD YYYY	Expected output MM DD YYYY	Actual output MM DD YYYY	Status
1	Testing for valid input	15 6 1900	16 6 1900	16 6 1900	Pass

SECOND ATTEMPT

Valid intervals:

M1: {month: month has 30 days}

M2: {month: month has 31 days}

M3: {month: month is February}

D1: {day: 1<=day<=28}

D2: {day: day=29}

D3: {day: day=30}

D4: {day: day=31}

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

WEAK NORMAL:

Tc id	Test case description	Actual input DD MM YYYY	Expected output DD MM YYYY	Actual output DD MM YYYY	Status
1	WN1	20 11 2000	21 11 2000	21 11 2000	Pass
2	WN2	29 01 2012	30 1 2012	30 1 2012	Pass
3	WN3	30 02 2009	Day 30 does not exist in this month	Day 30 does not exist in this month	Pass
4	WN4	31 06 2011	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass

STRONG NORMAL:

Tc id	Test case description	Actual input DD MM YYYY	Expected output DD MM YYYY	Actual output DD MM YYYY	Status
1	SN1	20 11 2000	21 11 2000	21 11 2000	Pass
2	SN2	20 11 2012	21 11 2012	21 11 2012	Pass
3	SN3	20 11 2009	21 11 2009	21 11 2009	Pass
4	SN4	20 11 2011	21 11 2011	21 11 2011	Pass
5	SN5	20 01 2000	21 01 2000	21 01 2000	Pass
6	SN6	20 01 2012	21 01 2012	21 01 2012	Pass
7	SN7	20 01 2009	21 01 2009	21 01 2009	Pass
8	SN8	20 01 2011	21 01 2011	21 01 2011	Pass
9	SN9	20 02 2000	21 02 2000	21 02 2000	Pass
10	SN10	20 02 2012	21 02 2012	21 02 2012	Pass
11	SN11	20 02 2009	21 02 2009	21 02 2009	Pass
12	SN12	20 02 2011	21 02 2011	21 02 2011	Pass
13	SN13	20 06 2000	21 06 2000	21 06 2000	Pass
14	SN14	20 06 2012	21 06 2012	21 06 2012	Pass
15	SN15	20 06 2009	21 06 2009	21 06 2009	Pass
16	SN16	20 06 2011	21 06 2011	21 06 2011	Pass
17	SN17	29 11 2000	30 11 2000	30 11 2000	Pass
18	SN18	29 11 2012	30 11 2012	30 11 2012	Pass
19	SN19	29 11 2009	30 11 2009	30 11 2009	Pass

20	SN20	29 11 2011	30 11 2011	30 11 2011	Pass
21	SN21	29 01 2000	30 01 2000	30 01 2000	Pass
22	SN22	29 01 2012	30 01 2012	30 01 2012	Pass
23	SN23	29 01 2009	30 01 2009	30 01 2009	Pass
24	SN24	29 01 2011	30 01 2011	30 01 2011	Pass
25	SN25	29 02 2000	01 03 2000	01 03 2000	Pass
26	SN26	29 02 2012	01 03 2012	01 03 2012	Pass
27	SN27	29 02 2009	01 03 2009	01 03 2009	Pass
28	SN28	29 02 2011	01 03 2011	01 03 2011	Pass
29	SN29	29 06 2000	30 06 2000	30 06 2000	Pass
30	SN30	29 06 2012	30 06 2012	30 06 2012	Pass
31	SN31	29 06 2009	30 06 2009	30 06 2009	Pass
32	SN32	29 06 2011	30 06 2011	30 06 2011	Pass
33	SN33	30 11 2000	01 12 2000	01 12 2000	Pass
34	SN34	30 11 2012	01 12 2012	01 12 2012	Pass
35	SN35	30 11 2009	01 12 2009	01 12 2009	Pass
36	SN36	30 11 2011	01 12 2011	01 12 2011	Pass
37	SN37	30 01 2000	31 01 2000	31 01 2000	Pass
38	SN38	30 01 2012	31 01 2012	31 01 2012	Pass
39	SN39	30 01 2009	31 01 2009	31 01 2009	Pass
40	SN40	30 01 2011	31 01 2011	31 01 2011	Pass
41	SN41	30 02 2000	Day 30 does not exist in this month	Day 30 does not exist in this month	Pass
42	SN42	30 02 2012	Day 30 does not exist in this month	Day 30 does not exist in this month	Pass
43	SN43	30 02 2009	Day 30 does not exist in this month	Day 30 does not exist in this month	Pass
44	SN44	30 02 2011	Day 30 does not exist in this month	Day 30 does not exist in this month	Pass
45	SN45	30 06 2000	01 07 2000	01 07 2000	Pass
46	SN46	30 06 2012	01 07 2012	01 07 2012	Pass
47	SN47	30 06 2009	01 07 2009	01 07 2009	Pass
48	SN48	30 06 2011	01 07 2011	01 07 2011	Pass
49	SN49	31 11 2000	01 12 2000	01 12 2000	Pass
50	SN50	31 11 2012	01 12 2012	01 12 2012	Pass
51	SN51	31 11 2009	01 12 2009	01 12 2009	Pass
52	SN52	31 11 2011	01 12 2011	01 12 2011	Pass
53	SN53	31 01 2000	01 02 2000	01 02 2000	Pass
54	SN54	31 01 2012	01 02 2012	01 02 2012	Pass
55	SN55	31 01 2009	01 02 2009	01 02 2009	Pass
56	SN56	31 01 2011	01 02 2011	01 02 2011	Pass

57	SN57	31 02 2000	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass
58	SN58	31 02 2012	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass
59	SN59	31 02 2009	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass
60	SN60	31 02 2011	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass
61	SN61	31 06 2000	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass
62	SN62	31 06 2012	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass
63	SN63	31 06 2009	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass
64	SN64	31 06 2011	Day 31 does not exist in this month	Day 31 does not exist in this month	Pass

TEST REPORT:

No of TC'S generated: 68

No of TC'S passed: 68

No of TC'S failed: 00

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 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. Derive test cases for your program based on decision-table approach, execute the test cases and discuss the results.

Requirements Specification:

Req1: The program should accept three numbers for a,b,c that is the three sides of the triangle. By giving the input it should be determine whether the triangle can be formed or not.

Req2: The numbers given as input to the triangle a,b,c shouldn't exceed to 10.

Req3: If the requirement req1 is satisfied then the program should determine the types of the triangle. The type of triangle can be,
Equilateral Triangle → Three sides are equal.
Isosceles Triangle → Any two sides are equal.
Scalene Triangle → Three sides are not equal.

Req4: If the triangle is not formed print the message saying triangle is not formed.

Design:

C1: a<b+c
C2: b<a+c
C3: c<a+b
C4: a=b
C5: a=c
C6: b=c

According to the property of triangle if any one of the three condition c1,c2,c3 are not satisfied, then triangle cannot be constructed. If c1, c2, c3 are true then triangle can be formed, with c4, c5, c6 we decide what type of triangle can be formed.

Coding:

```
#include<stdio.h>

#include<stdlib.h>

int main()
```

```
{
inta,b,c;
printf("Enter the sides of the triangle:\n");
scanf("%d%d%d",&a,&b,&c);
if((a<1 || a>10)&&(b<1 || b>10)&&(c<1 || c>10))
printf("Sides a, b and c are of out of range");
else if((a<1 || a>10)&&(b<1 || b>10))
printf("Sides a and b are out of range");
else if((b<1 || b>10)&&(c<1 || c>10))
printf("Sides b and c are out of range");
else if((a<1 || a>10)&&(c<1 || c>10))
printf("Sides a and c are out of range");
else if(a<1 || a>10)
printf("side a is out of range.");
else if(b<1 || b>10)
printf("side b is out of range.");
else if(c<1 || c>10)
printf("side c is out of range.");
else
  {
if((a<b+c)&&(b<a+c)&&(c<a+b))
    {
printf("Triangle is formed..\n");
          if(a==b && b==c && c==a )
          printf("Equilateral triangle..\n");
```

```
        else if(a==b || b==c || c==a)
        printf("Isoceles triangle..\n");
        else
        printf("Scalene triangle...\n");
    }
else
    {
printf("not a triangle..\n");
    }
  }
return 0;
}
```

Testing:

Technique used Decision Table Approach:

Conditions	**Condition Entries**										
	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
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<a+b?	--	--	F	T	T	T	T	T	T	T	T
C4:a=b?	--	--	--	F	T	T	T	F	F	F	T
C5:a=c?	--	--	--	T	F	T	F	T	F	F	T
C6:b=c?	--	--	--	T	T	F	F	F	T	F	T
Actions	**Action entries**										
a1:not a triangle	X	X	X								

a2:Scalene										X	
a3:Isosceles							X	X	X		
a4:Equilateral											X
a5:Impossible				X	X	X					

Test Cases for the Triangle problem

TC ID	Purpose	Actual input			Expected output	Actual output	Status
		a	b	c			
1.	Testing for R4	6	3	2	Not a triangle		
2.	Testing for R4	3	6	2	Not a triangle		
3.	Testing for R4	3	2	6	Not a triangle		
4.	Testing for R3	3	3	3	Equilateral triangle		
5.	Testing for R3	2	2	1	Isosceles triangle		
6.	Testing for R3	2	1	2	Isosceles triangle		
7.	Testing for R3	1	2	2	Isosceles triangle		
8.	Testing for R3	3	4	50	Scalene triangle		

Report Generation:

- Number of test cases executed:
- Number of test cases passed:
- Number of test cases failed:

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

Requirement Specification:

R1. The system should accept 3 values for locks, stocks and barrels.

R2. Limit ranges 1-70 for locks, 1-80 for stocks and 1-90 for barrels.

R3. Cost of each lock= $45, stock= $30, barrel= $25.

R4. Commission for total sales includes:

Sales up to & including $1000= 10%

Sales up to & including $1800= 15%

Sales in excess of $1800= 20%

DESIGN

C1: 1<=locks<=70
C2: 1<=stocks<=80
C3: 1<=barrels<=90
C4: sales>1800
C5: sales>1000
C6: sales<=1000

Coding:

```
#include<stdio.h>
int main()
{
intlocks,stocks,barrels,sales,flag=0;
float commission;
printf("Enter the total number of locks\n");
scanf("%d",&locks);
printf("Enter the total number of stocks\n");
scanf("%d",&stocks);
printf("Enter the total number of barrels\n");
scanf("%d",&barrels);
if((locks<=0) || (locks>70))
{
```

```
flag=1;
printf("Locks Out of Range\n");
}
if((stocks<=0) || (stocks>80))
{
flag=1;
printf("Stocks Out of Range\n");
}
if((barrels<=0) || (barrels>90))
{
flag=1;
printf("Barrels Out of Range\n");
}
if(flag == 1)
{
printf("Invalid input\n");
return 0;
}
sales= locks*45 + stocks*30 + barrels*25;
if(sales <= 1000)
{
commission = 0.10 * sales;
}
else if(sales <= 1800)
{
commission = 0.10 * 1000;
commission = commission + (0.15 * (sales - 1000));
}
else
{
commission = 0.10 * 1000;
commission = commission + (0.15 * 800);
commission = commission + (0.20 * (sales - 1800));
}
printf("Total sales: %d and Commission: %f\n",sales,commission);
return 0;
}
```

TESTING:

Technique used: Decision Table Approach

It is used to depict complex logical relationships between input data. A decision table is the method to build a complete set of test cases without using the internal structure of the program in the question. In order to create test cases we use a table to contain the input and output values of the program.

Conditions	**Condition Entries**					
	R1	**R2**	**R3**	**R4**	**R5**	**R6**
C1:1<=locks<=70	**F**	**T**	**T**	**T**	**T**	**T**
C2:1<=stock<=80	**-**	**F**	**T**	**T**	**T**	**T**
C3:1<=barrel<=90	**-**	**-**	**F**	**T**	**T**	**T**
C4:sales>1800	**-**	**-**	**-**	**T**	**F**	**F**
C5:sales>1000	**-**	**-**	**-**	**F**	**T**	**F**
C6:sales<=1000	**-**	**-**	**-**	**F**	**F**	**T**
Actions	**Action Entries**					
A1:com1=0.1*sales						**X**
A2:com2=com1+ 0.15*(sales-1000)					**X**	
A3:com3=com2+ 0.2*(sales-1800)				**X**		
A4:Out of range	**X**	**X**	**X**			

Test Cases:

TCID	**Purpose**	**Expected input**			**Expected output**	**Actual output**	**Status**
		locks	**stocks**	**barrels**			
1	**Testing for R2**	-2	40	45	Locks Out of range	Locks Out of range	pass
2	**Testing for R2**	90	40	45	Locks Out of range	Locks Out of range	pass
3	**Testing for R2**	35	-3	45	Stocks Out of range	Stocks Out of range	pass
4	**Testing for R2**	35	100	45	Stocks Out of range	Stocks Out of range	**pass**
5	**Testing for R2**	35	40	-10	Barrels Out of range	Barrels Out of range	**pass**
6	**Testing for R2**	35	40	150	Barrels Out of range	Barrels Out of range	**pass**
7	**Testing for R4**	9	9	9	Sales=900 Com=90	Sales=900 Com=90	**pass**

8	**Testing for R4**	15	15	15	Sales=1500 Com=175	Sales=1500 Com=175	**pass**
9	**Testing for R4**	25	25	25	Sales=2500 Com=360	Sales=2500 Com=360	**pass**

Report Generation:

Number of test cases executed : 9
Number of test cases passed : 9
Number of test cases failed : 0

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

Requirement Specification:

R1. The system should accept 3 values for locks, stocks and barrels.

R2. Limit ranges 1-70 for locks, 1-80 for stocks and 1-90 for barrels.

R3. Cost of each lock= $45, stock= $30, barrel= $25.

R4. Commission for total sales includes:

Sales up to & including $1000= 10%

Sales up to & including $1800= 15%

Sales in excess of $1800= 20%

DESIGN

C1: 1<=locks<=70
C2: 1<=stocks<=80
C3: 1<=barrels<=90
C4: sales>1800
C5: sales>1000
C6: sales<=1000

Coding

```
1. #include<stdio.h>
2. int main()
3. {
4. int locks, stocks, barrels, t_sales, flag= 0;
5. float commission;
6. printf("enter the total number of locks");
7. scanf("%d", &locks);
8. if((locks<=0)||(locks>70))
9. {
10. flag= 1;
11. }
12. printf("enter the total number of stocks");
13. scanf("%d", &stocks);
14. if((stocks<=0)||(stocks>80))
```

```
15. {
16. flag= 1;
17. }
18. printf("enter the total number of barrels");
19. scanf("%d", &barrels);
20. if((barrels<=0)||(barrels>90))
21. {
22. flag= 1;
23. }
24. if(flag==1)
25. {
26. printf("invalid input");
27. return 0;
28. }
29. t_sales=(locks*45)+(stocks*30)+(barrels*25);
30. if(t_sales<=1000)
31. {
32. commission=0.10*t_sales;
33. }
34. else if(t_sales<1800)
35. {
36. commission=0.10*1000;
37. commission=commission+(0.15*(t_sales-1000));
38. }
39. else
40. {
41. commission=0.10*1000;
42. commission=commission+(0.15*800);
43. commission=commission+(0.20*(t_sales-1800));
44. }
45. printf("the total sales is %d\n the commission is %f",t_sales,commission);
46. return;
47. }
```

TESTING

1. Number the lines.
2. List the variables.
3. List occurrences & assign a category to each variable.
4. Identify du-pairs and their use (p-use or c-use).
5. Define test cases, depending on the required coverage.

Line number	Definition	c-use	p-use
1.			
2.			
3.			
4.	flag		
5.			
6.			
7.	locks		
8.			locks
9.			
10.	flag		
11.			
12.			
13.	stocks		
14.			stocks
15.			
16.	flag		
17.			
18.			
19.	barrels		
20.			barrels
21.			
22.	flag		
23.			
24.			flag
25.			
26.		locks, stocks, barrels	
27.			
28.			
29.	t_sales	locks, stocks, barrels	
30.			t_sales
31.			
32.	commission	t_sales	
33.			
34.			t_sales
35.			
36.	commission		
37.	commission	commission, t_sales	
38.			

39.			
40.			
41.	commission		
42.	commission	commission	
43.	commission	commission, t_sales	
44.			
45.		commission, t_sales	

Test Cases:

Variables	du-pairs	Sub paths	lock	Stock	barrel	Sales	commission	Expected output	Actual output	Status
Locks	7-26	7,8,10,24,26	72	-	-	-	-	Invalid	invalid	**pass**
Stocks	13-26	13,14,16,24,26	-	82	-	-	-	Invalid	invalid	**pass**
barrels	19-26	19,20,22,24,26	-	-	92	-	-	Invalid	Invalid	**pass**
Commission	32-35	32,35	5	5	5	500	50	50050	50050	**pass**
Commission	36-45	36,37,45	15	15	15	1500	175	1500175	1500175	**pass**
Commission	41-45	41,42,43,45	25	25	25	2500	360	2500360	2500360	**Pass**

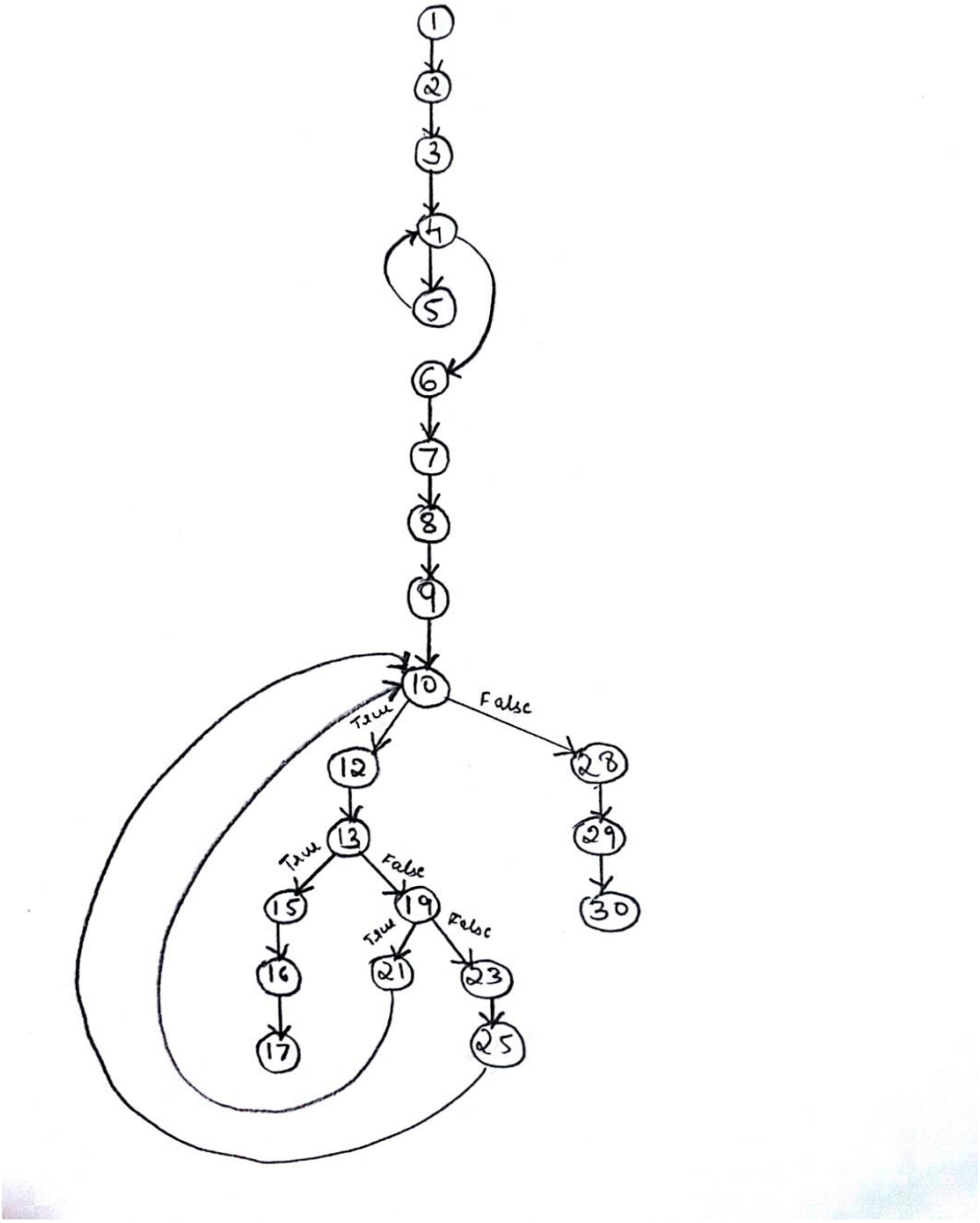
Report Generation:

- Number of test cases executed : 6
- Number of test cases passed : 6
- Number of test cases failed : 0

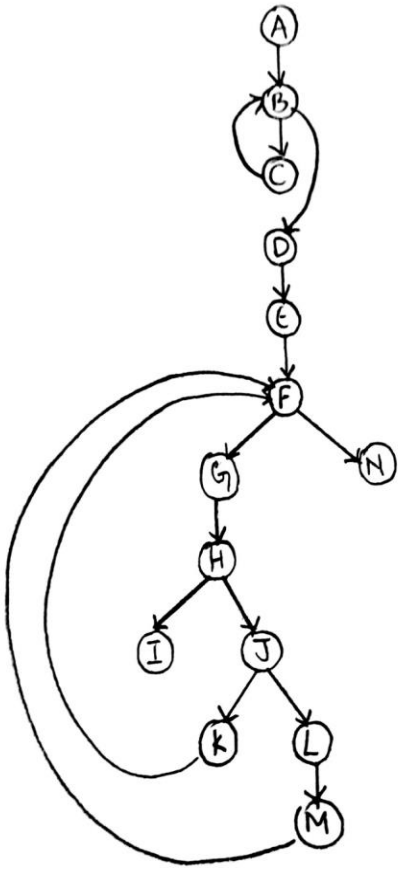
10. Design, develop, code and run the program in any suitable language to implement the binary search algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
int main()
{
int a[20],n,low,high,mid,key,i,flag=0;
1) printf("Enter the size of array \n");
2) scanf("%d",&n);
3) printf("Enter the elements in ascending order\n");
4) for(i=0;i<n;i++)
5) scanf("%d",&a[i]);
6) printf("Enter the key element\n");
7) scanf("%d",&key);
8) low=0;
9) high=n-1;
10) while(low<=high)
11) {
12) mid=(low+high)/2;
13) if(a[mid]==key)
14) {
15) printf("Element found at position %d",mid+1);
16) flag=1;
17) return 0;
18) }
19) else if(key<a[mid])
20) {
21) high=mid-1;
22) }
23) else
24) {
25) low=mid+1;
26) }
27) }
28) if(flag==0)
29) printf("Key element not found");
30) return 0;
31) }
```

Program Graph



DD Graph



Program graph node	DD Path Name
1,2,3	A
4	B
5	C
6	D
7,8,9	E
10	F
12	G
13	H
15,16,17	I
19	J
21	K
23	L
25	M
28,29,30	N

Cyclomatic Complexity

V(G)= e-n+2p

V(G)=16-14+2(1)

V(G)=4

e = Number of edges

n = Number of nodes

p = Connected regions

Independent Paths

P1: A-B-C-B-D-E-F-N (Key element not found)

P2: A-B-C-B-D-E-F-G-H-I (Element found at middle position)

P3: A-B-C-B-D-E-F-G-H-J-K-F-G-H-I (Key element less than middle)

P4: A-B-C-B-D-E-F-G-H-J-L-M-F-G-H-I (Key element greater than middle)

TEST CASES

Test case ID	Purpose	Input	Expected Result	Actual Result	Status
1	Testing for P1	n=5 a[5]={5,6,7,8,9} key=10	Key element not found		
2	Testing for P2	n=5 a[5]={5,6,7,8,9} key=7	Element found at position 3		
3	Testing for P3	n=5 a[5]={5,6,7,8,9} key=6	Element found at position 2		
4	Testing for P4	n=5 a[5]={5,6,7,8,9} key=9	Element found at position 5		

TEST REPORT:

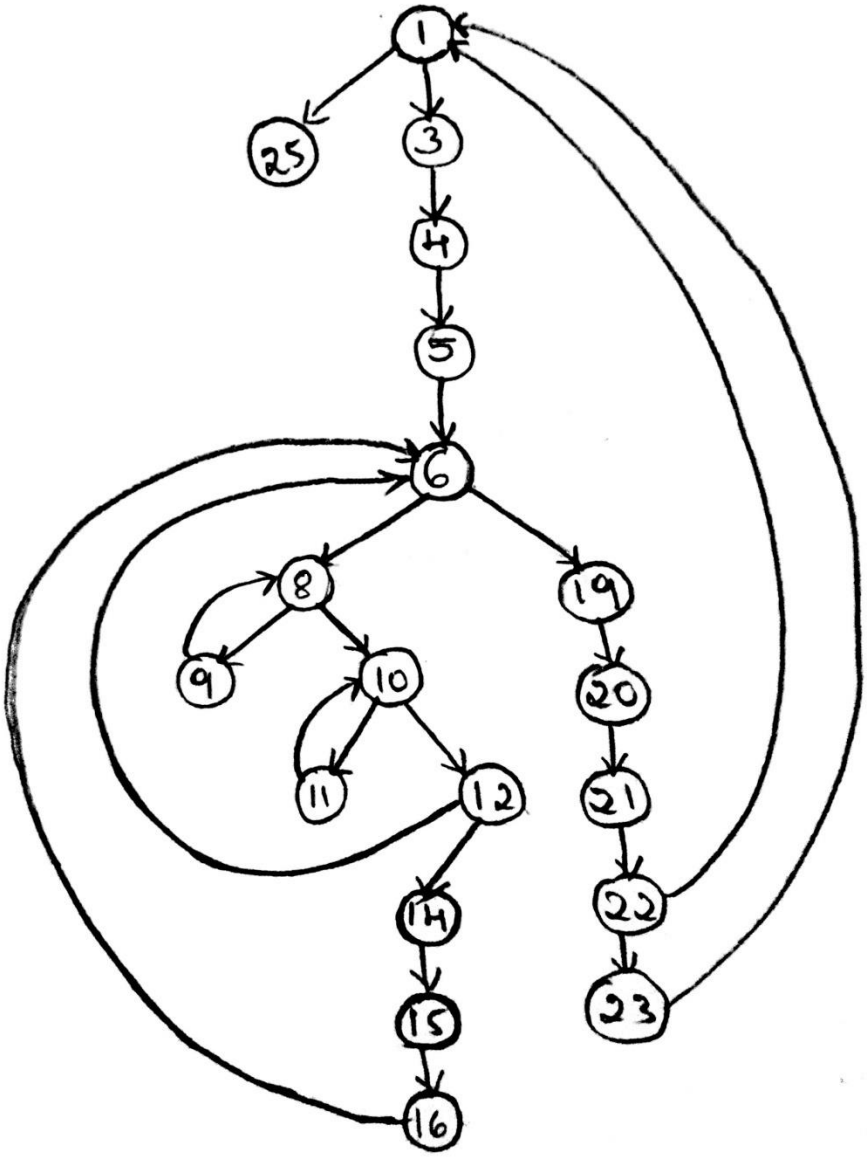
No of TC'S generated: 04
No of TC'S passed: 04
No of TC'S failed: 00

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.

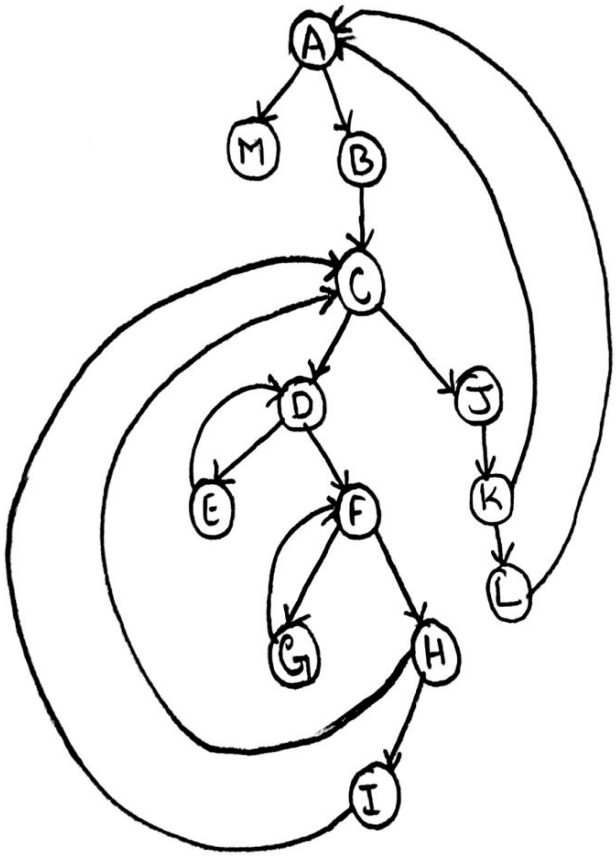
```
#include<stdio.h>
void quicksort(int x[10],intfirst,int last)
{
inttemp,pivot,i,j;
1. if(first<last)
2. {
3. pivot=first;
4. i=first;
5. j=last;
6. while(i<j)
7. {
8. while(x[i]<=x[pivot]&&i<last)
9. i++;
10. while(x[j]>x[pivot])
11. j--;
12. if(i<j)
13. {
14. temp=x[i];
15. x[i]=x[j];
16. x[j]=temp;
17. }
18. }
19. temp=x[pivot];
20. x[pivot]=x[j];
21. x[j]=temp;
22. quicksort(x,first,j-1);
23. quicksort(x,j+1,last);
24. }
25. }
```

```
void main()
 {
 int a[20],i,key,n;
 printf("Enter the size of the array\n");
 scanf("%d",&n);
 if(n>0)
 {
 printf("Enter the elements of the array\n");
 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");
 }
 }
```

Program Graph



DD Graph



Program graph node	DD Path Name
1,	A
3,4,5	B
6	C
8	D
9	E
10	F
11	G
12	H
14,15,16	I
19,20,21	J
22	K
23	L
25	M

Paths

P1: A-M

P2: A-B-C-J-K-A

P3: A-B-C-J-K-L-A

P4: A-B-C-D-E-D-F-H-C

P5: A-B-C-D-E-D-F-H-I-C

P6: A-B-C-D-E-D-F-H

P7: A-B-C-D-E-D-F-G-F-H

Cyclomatic Complexity

V(G)= e-n+2p
V(G)=18-13+2(1)
V(G)=07
e = Number of edges
n = Number of nodes
p = Connected regions

TEST CASES

Test case ID	Purpose	Input	Expected Result	Actual Result	Status
1	Testing for P1	n= 1 5	5		
2	Testing for P2	n=2 5,4	4,5		
3	Testing for P3	n=3 3,1,2	1,2,3		
4	Testing for P4	n=5 1,2,3,4,5	1,2,3,4,5		
5	Testing for P5	n=5 5,4,3,2,1,	1,2,3,4,5		
6	Testing for P6	n=5 2,2,2,2,2	2,2,2,2,2		
7	Testing for P7	n=5 5,2,3,1,4	1,2,3,4,5		

Test Report:

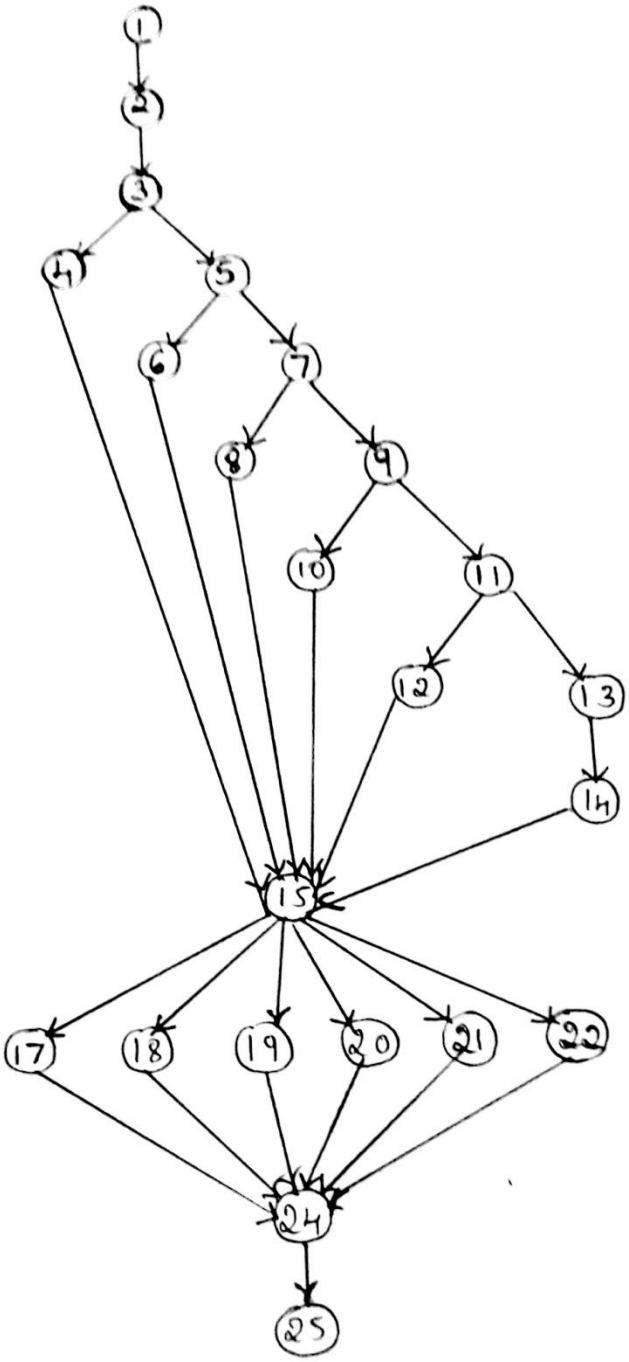
No of TC'S generated: 07
No of TC'S passed: 07
No of TC'S failed: 00

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.

Coding

```
#include<stdio.h>
int main()
{
float per;
char grade;
printf("Enter the percentage from 1 to 100\n");
scanf("%f",&per);
if(per>=90 && per<=100)
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 if(per>=35 && per<60)
grade='E';
else
grade='F';
switch(grade)
{
case 'A':printf("EXCELLENT\n");break;
case 'B':printf("VERY GOOD\n");break;
case 'C':printf("GOOD\n");break;
case 'D':printf("ABOVE AVERAGE\n");break;
case 'E':printf("SATISFACTORY\n");break;
case 'F':printf("FAIL\n");break;
}
printf("The percentage=%f and grade is %c",per,grade);
return 0;
}
```

Program Graph



Cyclomatic Complexity

V(G)= e-n+2p

V(G)=32-23+2(1)

V(G)=11

e = Number of edges

n = Number of nodes

p = Connected regions

Independent Paths

P1: 1-2-3-4-15-17-24-25 (A Grade)

P2: 1-2-3-5-6-15-18-24-25 (B Grade)

P3: 1-2-3-5-7-8-15-19-24-25 (C Grade)

P4: 1-2-3-5-7-9-10-15-20-24-25 (D Grade)

P5: 1-2-3-5-7-9-11-12-15-21-24-25 (E Grade)

P6: 1-2-3-5-7-9-11-13-14-15-22-24-25 (F Grade)

P7: 1-2-3-5-7-9-11-13-14-15-17-24-25 (Infeasible)

P8: 1-2-3-5-7-9-11-13-14-15-18-24-25 (Infeasible)

P9: 1-2-3-5-7-9-11-13-14-15-19-24-25 (Infeasible)

P10: 1-2-3-5-7-9-11-13-14-15-20-24-25 (Infeasible)

P11: 1-2-3-5-7-9-11-13-14-15-21-24-25 (Infeasible)

Test Cases

TC ID	PURPOSE	INPUTS	EXPECTED RESULT	ACTUAL RESULT	STATUS
1	Testing for P1	95	EXCELLENT The percentage=95.000000 and grade is A	EXCELLENT The percentage=95.000000 and grade is A	PASS
2	Testing for	85	VERY GOOD	VERY GOOD	PASS

	P2		The percentage=85.000000 and grade is B	The percentage=85.000000 and grade is B	
3	Testing for P3	75	GOOD The percentage=75.000000 and grade is C	GOOD The percentage=75.000000 and grade is C	PASS
4	Testing for P4	65	ABOVE AVERAGE The percentage=65.000000 and grade is D	ABOVE AVERAGE The percentage=65.000000 and grade is D	PASS
5	Testing for P5	45	SATISFACTORY The percentage=45.000000 and grade is E	SATISFACTORY The percentage=45.000000 and grade is E	PASS
6	Testing for P6	30	FAIL The percentage=30.000000 and grade is F	FAIL The percentage=30.000000 and grade is F	PASS

Test Report:

No of TC'S generated: 06
No of TC'S passed: 06
No of TC'S failed: 00