CANARA ENGINEERING COLLEGE

BENJANAPADAVU-574219

DEPT. OF INFORMATION SCIENCE AND ENGINEERING

LABORATORY MANUAL

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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 NextDate 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 NextDate 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 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. 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 quicksort 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

EXPERIMENTS

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.

1.1 REQUIREMENTS SPECIFICATION:

R1: The program should accept 3 integer values a, b & c which represent the 3 sides of a triangle

R2: If the R1 satisfies, based on the input, program should determine whether triangle can be formed or not

R3: If R1 & R2 holds good, then the program should determine the type of triangle

i.e.,

Equilateral (Three sides are equal)

Isosceles (Two sides are equal)

Scalene (Three sides are unequal)

Else report suitable error message

R4. Upper Limit for the size of any side is 10

1.2 DESIGN

Based on the requirements, the following conditions are drawn:

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 a triangle, if any one of the conditions C1, C2 & C3 fails then triangle cannot be constructed.
- If all the conditions C1, C2 & C3 are true then a triangle is constructed
- Based on the condition C4, C5 & C6 the type of triangle can be determined

ALGORITHM

Step 1: Get input a, b and c i.e., 3 integers which are sides of a triangle

Step 2: Is a triangle?

```
If (a < (b + c)) AND (b < (a + c)) AND (c < (a + b))
    Then IsATriangle = True
    Else IsATriangle = False
EndIf
```

Step 3: Determine Triangle Type

```
If IsATriangle
    Then If (a = b) AND (b = c)
        Then triangle is an equilateral
        Else If (a ≠ b) AND (a ≠ c) AND (b ≠ c)
            Then triangle is a Scalene
            Else triangle is an Isosceles
        EndIf
    EndIf
    Else "Not a Triangle"
  EndIf
End triangle
```

1.3 PROGRAM

```
#include<stdio.h>
#include<ctype.h>
void main()
{
int a,b,c;
printf("\n Enter 3 integer which are side of triangle \n");
scanf("%d %d %d",&a,&b,&c);
if((a>=1&&a<=10)&&(b>=1&&b<=10)&&(c>=1&&c<=10))
{
if((a<(b+c))&&(b<(a+c))&&(c<(a+b)))
{
if((a==b)&&(b==c))
printf("Triangle is equilateral");
else if((a!=b)&&(a!=c)&&(b!=c))
printf("Triangle is scalene\n");
else
printf("Triangle is isoceles\n");
}
else
printf("\n Triangle cannot not be formed");
}
else
printf("\n The values are out of range\n");
}
```

1.4 TESTING

Technique: Boundary value analysis

- Boundary value analysis focuses on the boundary of the input space to identify test cases. The rationale behind boundary value testing is that errors tend to occur near the extreme values of an input variable.
- The basic idea of boundary value analysis is to use input variable values at their minimum, just above the minimum, a nominal value, just below their maximum, and their maximum.
- The boundary value analysis is based on a critical assumption; it is known as the single fault assumption in reliability theory. This says that failures are only rarely the result of the simultaneous occurrence of two (or more) faults.
- If we have a function of n variables, we hold all but one at the nominal values and let the remaining variable assume the min, min+, nom, max-, and max values, repeating this for each variable. Thus, for a function of n variables, boundary value analysis yields (4n + 1) unique test cases.

Test Cases:

We can derive 13 test cases based on (4n+1) of BVA

Table 2.1 Triangle Problem Boundary Value Analysis Test Cases

TC ID	Test Case Descriptio	Input			Expected Output	Actual Output	Comments
		a	b	c			
01	Testing for 1, 5, 5	1	5	5	Isosceles	Isosceles	Pass
02	Testing for 2, 5, 5	2	5	5	Isosceles	Isosceles	Pass
03	Testing for 5, 5, 5	5	5	5	Equilateral	Equilateral	Pass
04	Testing for 9, 5, 5	9	5	5	Isosceles	Isosceles	Pass
05	Testing for 10, 5, 5	10	5	5	Not a triangle	Not a triangle	Pass
06	Testing for 5, 1, 5	5	1	5	Isosceles	Isosceles	Pass
07	Testing for 5, 2, 5	5	2	5	Isosceles	Isosceles	Pass
08	Testing for 5, 9, 5	5	9	5	Isosceles	Isosceles	Pass
09	Testing for 5, 10, 5	5	10	5	Not a triangle	Not a triangle	Pass
10	Testing for 5, 5, 1	5	5	1	Isosceles	Isosceles	Pass
11	Testing for 5, 5, 2	5	5	2	Isosceles	Isosceles	Pass
12	Testing for 5, 5, 9	5	5	9	Isosceles	Isosceles	Pass
13	Testing for 5, 5, 10	5	5	10	Not a triangle	Not a triangle	Pass

Table 2.2 Triangle Problem Robustness Test Cases

TC ID	Test Case Description	Input			Expected Output	Actual Output	Comments
		a	b	c			
01	Testing for 5, 5, -1	5	5	-1	Out of Range	Out of Range	Pass
02	Testing for 5, 5, 1	5	5	1	Isosceles	Isosceles	Pass
03	Testing for 5, 5, 1	5	5	2	Isosceles	Isosceles	Pass
04	Testing for 5, 5, 5	5	5	5	Equilateral	Equilateral	Pass
05	Testing for 5, 5, 9	5	5	9	Isosceles	Isosceles	Pass
06	Testing for 5, 5, 10	5	5	10	Not a triangle	Not a triangle	Pass
07	Testing for 5, 5, 11	5	5	11	Out of Range	Out of Range	Pass
08	Testing for 5, -1, 5	5	-1	5	Out of Range	Out of Range	Pass
09	Testing for 5, 1, 5	5	1	5	Isosceles	Isosceles	Pass
10	Testing for 5, 2, 5	5	2	5	Isosceles	Isosceles	Pass
11	Testing for 5, 9, 5	5	9	5	Isosceles	Isosceles	Pass
12	Testing for 5, 10, 5	5	10	5	Not a triangle	Not a triangle	Pass
13	Testing for 5, 11, 5	5	11	5	Out of Range	Out of Range	Pass
14	Testing for -1, 5, 5	-1	5	5	Out of Range	Out of Range	Pass
15	Testing for 1, 5, 5	1	5	5	Isosceles	Isosceles	Pass
16	Testing for 2, 5, 5	2	5	5	Isosceles	Isosceles	Pass
17	Testing for 9, 5, 5	9	5	5	Isosceles	Isosceles	Pass
18	Testing for 10, 5, 5	10	5	5	Not a triangle	Not a triangle	Pass
19	Testing for 11, 5, 5	11	5	5	Out of Range	Out of Range	Pass

Table 2.3 Triangle Problem Worst-Case Test Cases

1.5

TC ID	TestCase Description	Input			Expected Output	Actual Output	Comments
		a	b	c			
01	Testing for 1, 1, 1	1	1	1	Equilateral	Equilateral	Pass
02	Testing for 1, 1, 2	1	1	2	Not a Triangle	Not a Triangle	Pass
03	Testing for 1, 1, 5	1	1	5	Not a Triangle	Not a Triangle	Pass
04	Testing for 1, 1, 9	1	1	9	Not a Triangle	Not a Triangle	Pass
05	Testing for 1, 1, 10	1	1	10	Not a Triangle	Not a Triangle	Pass
06	Testing for 1, 2, 1	1	2	1	Not a Triangle	Not a Triangle	Pass
07	Testing for 1, 2, 2	1	2	2	Isosceles	Isosceles	Pass
08	Testing for 1, 2, 5	1	2	5	Not a Triangle	Not a Triangle	Pass
09	Testing for 1, 2, 9	1	2	9	Not a Triangle	Not a Triangle	Pass
10	Testing for 1, 2, 10	1	2	10	Not a Triangle	Not a Triangle	Pass
11	Testing for 1, 5, 1	1	5	1	Not a Triangle	Not a Triangle	Pass
12	Testing for 1, 5, 2	1	5	2	Not a Triangle	Not a Triangle	Pass
13	Testing for 1, 5, 5	1	5	5	Isosceles	Isosceles	Pass
14	Testing for 1, 5, 9	1	5	9	Not a Triangle	Not a Triangle	Pass
15	Testing for 1, 5, 10	1	5	10	Not a Triangle	Not a Triangle	Pass
16	Testing for 1, 9, 1	1	9	1	Not a Triangle	Not a Triangle	Pass
17	Testing for 1, 9, 2	1	9	2	Not a Triangle	Not a Triangle	Pass
18	Testing for 1, 9, 5	1	9	5	Not a Triangle	Not a Triangle	Pass
19	Testing for 1, 9, 9	1	9	9	Isosceles	Isosceles	Pass
20	Testing for 1, 9, 10	1	9	10	Not a Triangle	Not a Triangle	Pass
21	Testing for 1, 10, 1	1	10	1	Not a Triangle	Not a Triangle	Pass
22	Testing for 1, 10, 2	1	10	2	Not a Triangle	Not a Triangle	Pass
23	Testing for 1, 10, 5	1	10	5	Not a Triangle	Not a Triangle	Pass
24	Testing for 1, 10, 9	1	10	9	Not a Triangle	Not a Triangle	Pass
25	Testing for 1, 10, 10	1	10	10	Isosceles	Isosceles	Pass

EXECUTIONS

Execute the program and test the test cases in above tables against program and complete the table with for Actual output column and Status column

TEST REPORT:

1. No of TC's Executed: 57
2. No of Defects Raised: 0
3. No of TC's Pass: 57
4. No of TC's Failed: 0

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.

2.1 REQUIREMENTS SPECIFICATIONS:

- A rifle salesperson sold rifle locks, stocks, and barrels made by a gunsmith and the cost are as follows:

  Locks - $45
  Stocks - $30
  Barrels - $25

- The salesperson had to sell at least one complete rifle per month, and production limits were such that the most the salesperson could sell in a month was 70 locks, 80 stocks, and 90 barrels.
- After each town visit, the salesperson sent a telegram to the gunsmith with the number of locks, stocks, and barrels sold in that town. At the end of a month, the salesperson sent a very short telegram showing -1 lock sold.
- The gunsmith then knew the sales for the month were complete and computed the salesperson's commission as follows:

  Sales from $1 to $1000 = 10%
  Sales from $1001 to $1800 = 15%
  Sales in excess of $1800 = 20%

  The commission program produced a monthly sales report that gave the total number of locks, stocks, and barrels sold, the salesperson's total dollar sales, and the commission.

2.2 DESIGN

ALGORITHM:

Step 1: Define locks_price = 45$, stock_price= 30$, barrelprice = 25 $

Step 2: Input (locks)

Step 3: while (locks !=-1) 'Input device uses -1 to indicate end of data'
Input (stocks, barrels)
calculate total_locks, total_stocks, total_barrels
Input (locks)
End while

Step 4: calculate total sales
total_sales = (total_locks *45) + (total_stocks*30) + (total_barrels * 25)

Step 5: if (total_sales <= 1000)
commission = 0.10 * total_sales;

```
        else if (total_sales < 1800)
                {
                        commission = 0.10 * 1000;
                        commission = commission + (0.15 * (total_sales - 1000));
                }
        else
                {
                        commission = 0.10 * 1000;
                        commission = commission + (0.15 * 800);
                        commission = commission + (0.20 * (total_sales - 1800));
                }
```

Step 6: Output (“Commission is $”, Commission)

Step 7: Exit

2.3 PROGRAM

```
#include<stdio.h>
void main()
{
int locks,stocks,barrels,totalsales;
int totallocks=0,totalstocks=0,totalbarrels=0;
float commission=0;
printf("Enter the number of locks\n");
scanf("%d",&locks);
while(locks!=-1)
{
printf("Enter the no. of stocks\n");
scanf("%d",&stocks);
printf("Enter the no. of barrels");
scanf("%d",&barrels);
totallocks=totallocks+locks;
totalstocks=totalstocks+stocks;
totalbarrels=totalbarrels+barrels;
printf("\nEnter -1 to end of the sales\n Else Enter the number of locks\n");
scanf("%d",&locks);


}
if((totallocks>=0&&totallocks<=70)&&(totalstocks>=0&&totalstocks<=80)&&(totalbarrels
>=0&&totalbarrels<=90))
{
totalsales=(totallocks*45)+(totalstocks*30)+(totalbarrels*25);
if(totalsales<=1000)
{
```

```
commission=0.10*totalsales;
}
else if(totalsales<1800)
{
commission=0.10*1000;
commission=commission+(0.15*(totalsales-1000));
}
else
{
commission=0.10*1000;
commission=commission+(0.15*800);
commission=commission+(0.20*(totalsales-1800));
}
printf("The total sales is %d\n The commission is %f",totalsales,commission);
}
else
{
printf("\n invalid input");
}
}
```

2.4 TESTING

Technique: Boundary value analysis

- Boundary value analysis focuses on the boundary of the input space to identify test cases. The rationale behind boundary value testing is that errors tend to occur near the extreme values of an input variable.
- The basic idea of boundary value analysis is to use input variable values at their minimum, just above the minimum, a nominal value, just below their maximum, and their maximum.
- The boundary value analysis is based on a critical assumption; it is known as the single fault assumption in reliability theory. This says that failures are only rarely the result of the simultaneous occurrence of two (or more) faults.
- If we have a function of n variables, we hold all but one at the nominal values and let the remaining variable assume the min, min+, nom, max-, and max values, repeating this for each variable. Thus, for a function of n variables, boundary value analysis yields (4n + 1) unique test cases.

Test Cases:

We can derive 13 test cases based on (4n+1) of BVA

Table 2.1 Commission Problem Boundary Value Analysis Test Cases

TC ID	Test Case Description	Input			Expected Output		Actual o/p		Comment
		Locks	Stocks	Barrels	Commission	Sales	Commision	sales	
01	Testing for 35, 40, 1	35	40	1	420	2800	420	2800	Pass
02	Testing for 35, 40, 2	35	40	2	425	2825	425	2825	Pass
03	Testing for 35, 40, 45	35	40	45	640	3900	640	3900	Pass
04	Testing for 35, 40, 89	35	40	89	860	5000	860	5000	Pass
05	Testing for 35, 40, 90	35	40	90	865	5025	865	5025	Pass
06	Testing for 35, 1, 45	35	1	45	406	2730	406	2730	Pass
07	Testing for 35, 2, 45	35	2	45	412	2760	412	2760	Pass
08	Testing for 35, 79, 45	35	79	45	874	5070	874	5070	Pass
09	Testing for 35, 80, 45	35	80	45	880	5100	880	5100	Pass
10	Testing for 1, 40, 45	1	40	45	334	2370	334	2370	Pass
11	Testing for 2, 40, 45	2	40	45	343	2415	343	2415	Pass
12	Testing for 69, 40, 45	69	40	45	940	5430	940	5430	Pass
13	Testing for 70, 40, 45	70	40	45	955	5475	955	5475	Pass

Table 2.2 Commission Problem Robustness Test Cases

TC ID	Test Case Description	Input			Expected Output		Actual Output (Commission)	Comment
		Locks	Stocks	Barrels	Sales	Commission		
01	Testing for 35, 40, -1	35	40	-1	-	-	-	Pass
02	Testing for 35, 40, 1	35	40	1	420	2800	2800	Pass
03	Testing for 35, 40, 2	35	40	2	425	2825	2825	Pass
04	Testing for 35, 40, 45	35	40	45	640	3900	3900	Pass

05	Testing for 35, 40, 89	35	40	89	860	5000	5000	Pass
06	Testing for 35, 40, 90	35	40	90	865	5025	5025	Pass
07	Testing for 35, 40, 91	35	40	91	-	-	-	Pass
08	Testing for 35, -1, 45	35	-1	45	-	-	-	Pass
09	Testing for 35, 1, 45	35	1	45	406	2730	2730	Pass
10	Testing for 35, 2, 45	35	2	45	412	2760	2760	Pass
11	Testing for 35, 79, 45	35	79	45	874	5070	5070	Pass
12	Testing for 35, 80, 45	35	80	45	880	5100	5100	Pass
13	Testing for 35, 81, 45	35	81	45	-	-	-	Pass
14	Testing for -1, 40, 45	-1	40	45	-	-	-	Pass
15	Testing for 1, 40, 45	1	40	45	334	2370	2370	Pass
16	Testing for 2, 40, 45	2	40	45	343	2415	2415	Pass
17	Testing for 69, 40, 45	69	40	45	940	5430	5430	Pass
18	Testing for 70, 40, 45	70	40	45	955	5475	5475	Pass
19	Testing for 71, 40, 45	71	40	45	-	-	-	Pass

Table 2.3 Commision Problem Worst-Case Test Cases

TC Id	Test Case Description	Input Data			Sales	Expected Output (Commission)	Sales	Commision	Comment
		Locks	Stocks	Barrels					
1	Testing for 1,1,1	1	1	1	100	10	100	10	Pass
2	Testing for 1,1,2	1	1	2	125	12.5	125	12.5	Pass
3	Testing for 1,1,45	1	1	45	1200	130	1200	130	Pass
4	Testing for 1,1,89	1	1	89	2300	320	2300	320	Pass
5	Testing for 1,1,90	1	1	90	2325	325	2325	325	Pass
6	Testing for 1,2,1	1	2	1	130	13	130	13	Pass
7	Testing for 1,2,2	1	2	2	155	15.5	155	15.5	Pass

8	Testing for 1,2,45	1	2	45	1230	134.5	1230	134.5	Pass
9	Testing for 1,2,89	1	2	89	2330	326	2330	326	Pass
10	Testing for 1,2,90	1	2	90	2355	331	2355	331	Pass
11	Testing for 1,40,1	1	40	1	1270	140.5	1270	140.5	Pass
12	Testing for 1,40,2	1	40	2	1295	144.25	1295	144.25	Pass
13	Testing for 1,40,45	1	40	45	2370	334	2370	334	Pass
14	Testing for 1,40,89	1	40	89	3470	554	3470	554	Pass
15	Testing for 1,40,90	1	40	90	3495	559	3495	559	Pass
16	Testing for 1,79,1	1	79	1	2440	348	2440	348	Pass
17	Testing for 1,79,2	1	79	2	2465	353	2465	353	Pass
18	Testing for 1,79,45	1	79	45	3540	568	3540	568	Pass
19	Testing for 1,79,89	1	79	89	4640	788	4640	788	Pass
20	Testing for 1,79,90	1	79	90	4665	793	4665	793	Pass
21	Testing for 1,80,1	1	80	1	2470	354	2470	354	Pass
22	Testing for 1,80,2	1	80	2	2495	359	2495	359	Pass
23	Testing for 1,80,45	1	80	45	3570	574	3570	574	Pass
24	Testing for 1,80,89	1	80	89	4670	794	4670	794	Pass
25	Testing for 1,80,90	1	80	90	4695	799	4695	799	Pass

Execute the program and test the test cases in above tables against program and complete the table with for Actual output column and Status column

TEST REPORT:

1. No of TC's Executed: 57
2. No of Defects Raised: 00
3. No of TC's Pass: 57
4. No of TC's Failed: 00

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.

3.1 REQUIREMENTS SPECIFICATIONS:

"Next Date" is a function of three variables: day, month and year. It returns the date of the day after the input date. The day, month, year variable have integer values subject to these conditions.

C1:1 ≤ month ≤ 12
C2:1 ≤ day ≤ 31
C3: 1812 ≤ year ≤ 2012.

If any one condition out of C1, C2 or C3 fails, NextDate produces an output indicating the corresponding variable has an out-of-range value - for example, "Value of month not in the range 1..12." Because numerous invalid day- month- year combinations exist, NextDate collapses these into one message: "Invalid Input Date."

The complexity of the problem is the rule that determines when a year is leap year. A year is called as a leap year if it is divisible by 4, unless it is a century year. Century years are leap years only if they are multiples of 400. So, 1992, 1996 and 2000 are leap years while 1900 is not a leap year.

3.2 DESIGN

ALGORITHM

```
Step 1: Input (month, day, year)
Step 2: if ((1 <= day) AND  (day <= 31)) AND ((1 <= month) AND  (month <= 12))
        AND ((1812 <= year)  AND  (year <= 2012))
                goto step 3
        else goto step 7

step 3: if month Is 1, 3, 5, 7, 8, Or 10:   '31 day months   (except Dec.)
            if (day < 31)
                Then tomorrowDay = day + 1
                Else
                        tomorrowDay = 1 tomorrow
                        Month = month + 1
          else goto step 4

Step 4: if month Is 4, 6, 9 Or 11  '30 day months
            if (day < 30)
                then tomorrowDay = day + 1
                else
```

```
                    if day =30
                        then   tomorrowDay = 1
                               tomorrowMonth = month + 1
                    else    Output  ("Invalid Input Date")
          else goto step 5
Step 5: if month Is 12: December
           if day < 31
              then tomorrowDay = day + 1
              else
                    tomorrowDay = 1
                    tomorrowMonth = 1
              if year = 2012
                    then Output   ("Invalid Input Date")
                    else tomorrow.year = year + 1
        else goto step 6
Step 4: if month is 2: 'February
          if day < 28
              then tomorrowDay = day + 1
          else
              if day = 28
                 then
                    if  (year is a leap year)
                        then tomorrowDay = 2 9  'leap day
                    else    'not a leap year
                               tomorrowDay = 1
                               tomorrowMonth = 3
              else
                if day =29
                    if (Year is a leap year)
                        then    tomorrowDay = 1
                               tomorrowMonth = 3
                    else
                        if day > 29
                            Then Output   ("Invalid Input Date")
Step 6: Output   ("Tomorrow's date is",   tomorrowMonth,
              tomorrowDay, tomorrowYear)
Step 7: Exit
```

3.3 PROGRAM CODE

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

```
void main()
{
int day,month,year;
int nextday,nextmonth,nextyear;
printf("\n Enter the date format DD MM YYYY:");
scanf("%d%d%d",&day,&month,&year);
if(((day>=1)&&(day<=31))&&((month>=1)&&(month<=12))&&((year>=1812)&&(year<=
2012)))
{
nextmonth=month;
nextyear=year;

if((month==1)||(month==3)||(month==5)||(month==7)||(month==8)||(month==10))
{
if(day<31)
{
nextday=day+1;
}
else
{
nextday=1;
nextmonth=month+1;
}}
else if((month==4)||(month==6)||(month==9)||(month==11))
{
if(day<30)
nextday=day+1;
else if(day==30)
{
nextday=1;
nextmonth=month+1;
}
else
{
printf("invalid");
exit(0);}
}
else if(month==12)
{
if(day<31)
nextday=day+1;
else
{
nextday=1;
nextmonth=1;
nextyear=year+1;
if(nextyear>2012)
```

```
{
printf("Invalid");
exit(0);
}}}
else
{
if((year%4==0&&year%100!=0)||(year%400==0))
{
if(day<29)
nextday=day+1;

else if(day==29)
{
nextday=1;
nextmonth=month+1;
}
else
{
printf("Invalid date");
exit(0);
}}
else
{
if(day<28)
nextday=day+1;
else if(day==28)
{
nextday=1;
nextmonth=month+1;
}
else
{printf("Invalid date");
exit(0);
}}}
printf("\nThe next date is = %d %d %d",nextday,nextmonth,nextyear);
}
else
printf("\n the date is invalid\n");
}
```

3.4 TESTING

Technique: Boundary value analysis

Boundary value analysis focuses on the boundary of the input space to identify test cases. The rationale behind boundary value testing is that errors tend to occur near the extreme values of an input variable.

The basic idea of boundary value analysis is to use input variable values at their minimum, just above the minimum, a nominal value, just below their maximum, and their maximum.

The boundary value analysis is based on a critical assumption; it is known as the single fault assumption in reliability theory. This says that failures are only rarely the result of the simultaneous occurrence of two (or more) faults.

If we have a function of n variables, we hold all but one at the nominal values and let the remaining variable assume the min, min+, nom, max-, and max values, repeating this for each variable. Thus, for a function of n variables, boundary value analysis yields (4n + 1) unique test cases.

Test Cases:

We can derive 13 test cases based on (4n+1) of BVA

Table 3.1 NextDate Problem Boundary Value Analysis Test Cases

TC ID	Test Case Description	Input			Expected Output	Actual Output	Comment
		Day	Month	Year			
01	Testing for d=1, m=6, y= 1912	1	6	1912	2-6-1912	2-6-1912	Pass
02	Testing for d=2, m=6, y= 1912	2	6	1912	3-6-1912	3-6-1912	Pass
03	Testing for d=15, m=6, y= 1912	15	6	1912	16-6-1912	16-6-1912	Pass
04	Testing for d=30, m=6, y= 1912	30	6	1912	1-7-1912	1-7-1912	Pass
05	Testing for d=31, m=6, y= 1912	31	6	1912	Invalid	Invalid	Pass
06	Testing for d=15, m=1, y= 1912	15	1	1912	16-1-1912	16-1-1912	Pass
07	Testing for d=15, m=6, y= 1912	15	2	1912	16-2-1912	16-2-1912	Pass
08	Testing for d=15, m=11, y= 1912	15	11	1912	16-11-1912	16-11-1912	Pass
09	Testing for d=15, m=12, y= 1912	15	12	1912	16-12-1912	16-12-1912	Pass
10	Testing for d=15, m=6, y= 1812	15	6	1812	16-6-1812	16-6-1812	Pass

11	Testing for d=15, m=6, y= 1813	15	6	1813	16-6-1813	16-6-1813	Pass
12	Testing for d=15, m=6, y= 2011	15	6	2011	16-6-2011	16-6-2011	Pass
13	Testing for d=15, m=6, y= 2012	15	6	2012	16-6-2012	16-6-2012	Pass

Table 3.2 NextDate Problem Robustness Test Cases

TC ID	Test Case Description	Input			Expected Output	Actual Output	Comment
		Day	Month	Year			
01	Testing for d=-1, m=6, y= 1912	-1	6	1912	Invalid	Invalid	Pass
02	Testing for d=1, m=6, y= 1912	1	6	1912	2-6-1912	2-6-1912	Pass
03	Testing for d=2, m=6, y= 1912	2	6	1912	3-6-1912	3-6-1912	Pass
04	Testing for d=15, m=6, y= 1912	15	6	1912	16-6-1912	16-6-1912	Pass
05	Testing for d=30, m=6, y= 1912	30	6	1912	1-7-1912	1-7-1912	Pass
06	Testing for d=31, m=6, y= 1912	31	6	1912	Invalid	Invalid	Pass
07	Testing for d=32, m=6, y= 1912	32	6	1912	Invalid	Invalid	Pass
08	Testing for d=15, m=-1, y= 1912	15	-1	1912	Invalid	Invalid	Pass
09	Testing for d=15, m=1, y= 1912	15	1	1912	16-1-1912	16-1-1912	Pass
10	Testing for d=15, m=6, y= 1912	15	2	1912	16-2-1912	16-2-1912	Pass
11	Testing for d=15, m=11, y= 1912	15	11	1912	16-11-1912	16-11-1912	Pass
12	Testing for d=15, m=12, y= 1912	15	12	1912	16-12-1912	16-12-1912	Pass

13	Testing for d=15, m=13, y= 1912	15	13	1912	Invalid	Invalid	Pass
14	Testing for d=15, m=6, y= 1811	15	6	1811	Invalid	Invalid	Pass
15	Testing for d=15, m=6, y= 1812	15	6	1812	16-6-1812	16-6-1812	Pass
16	Testing for d=15, m=6, y= 1813	15	6	1813	16-6-1813	16-6-1813	Pass
17	Testing for d=15, m=6, y= 2011	15	6	2011	16-6-2011	16-6-2011	Pass
18	Testing for d=15, m=6, y= 2012	15	6	2012	16-6-2012	16-6-2012	Pass
19	Testing for d=15, m=6, y= 2013	15	6	2013	Invalid	Invalid	Pass

Table 3.3 Triangle Problem Worst-Case Test Cases

TC ID	Test Case Description	Input			Expected Output	Actual Output	Comments
		Day	Month	Year			
01	Testing for d=1, m=1, y= 1812	1	1	1812	2-1-1812	2-1-1812	Pass
02	Testing for d=1, m=1, y= 1813	1	1	1813	2-1-1813	2-1-1813	Pass
03	Testing for d=1, m=1, y= 1912	1	1	1912	2-1-1912	2-1-1912	Pass
04	Testing for d=1, m=1, y= 2011	1	1	2011	2-1-2011	2-1-2011	Pass
05	Testing for d=1, m=1, y= 2012	1	1	2012	2-1-2012	2-1-2012	Pass
06	Testing for d=2, m=1, y= 1812	2	1	1812	3-1-1812	3-1-1812	Pass
07	Testing for d=2, m=1, y= 1813	2	1	1813	3-1-1813	3-1-1813	Pass
08	Testing for d=2, m=1, y= 1912	2	1	1912	3-1-1912	3-1-1912	Pass

EXECUTIONS

09	Testing for d=2, m=1, y= 2011	2	1	2011	3-1-2011	3-1-2011	Pass
10	Testing for d=2, m=1, y= 2012	2	1	2012	3-1-2012	3-1-2012	Pass
11	Testing for d=15, m=1, y= 1812	15	1	1812	16-1-1812	16-1-1812	Pass
12	Testing for d=15, m=1, y= 1813	15	1	1813	16-1-1813	16-1-1813	Pass
13	Testing for d=15, m=1, y= 1912	15	1	1912	16-1-1912	16-1-1912	Pass
14	Testing for d=15, m=1, y= 2011	15	1	2011	16-1-2011	16-1-2011	Pass
15	Testing for d=15, m=1, y= 2012	15	1	2012	16-1-2012	16-1-2012	Pass
16	Testing for d=30, m=1, y= 1812	30	1	1812	31-1-1812	31-1-1812	Pass
17	Testing for d=30, m=1, y= 1813	30	1	1813	31-1-1813	31-1-1813	Pass
18	Testing for d=30, m=1, y= 1912	30	1	1912	31-1-1912	31-1-1912	Pass
19	Testing for d=30, m=1, y= 2011	30	1	2011	31-1-2011	31-1-2011	Pass
20	Testing for d=30, m=1, y= 2012	30	1	2012	31-1-2012	31-1-2012	Pass
21	Testing for d=31, m=1, y= 1812	31	1	1812	1-2-1812	1-2-1812	Pass
22	Testing for d=31, m=1, y= 1813	31	1	1813	1-2-1813	1-2-1813	Pass
23	Testing for d=31, m=1, y= 1912	31	1	1912	1-2-1912	1-2-1912	Pass
24	Testing for d=31, m=1, y= 2011	31	1	2011	1-2-2011	1-2-2011	Pass
25	Testing for d=31, m=1, y= 2012	31	1	2012	1-2-2012	1-2-2012	Pass

Execute the program and test the test cases in above tables against program and complete the table with for Actual output column and Status column

TEST REPORT:

1. No of TC's Executed: 57
2. No of Defects Raised: 00
3. No of TC's Pass: 57
4. No of TC's Failed: 00

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.

4.1 REQUIREMENTS

R1: The program should accept 3 integer values a, b & c which represent the 3 sides of a triangle

R2: If the R1 satisfies, based on the input program should determine whether triangle can be formed or not

R3: If R1 & R2 holds good, then the program should determine the type of triangle i.e.,

Equilateral (Three sides are equal)

Isosceles (Two sides are equal)

Scalene (Three sides are unequal)

Else report suitable error message

R4: Upper Limit for the size of any side is 10

4.2 DESIGN

Based on the requirements, the following conditions are drawn:

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 a triangle, if any one of the conditions C1, C2 & C3 fails then triangle cannot be constructed.
- If all the conditions C1, C2 & C3 are true then a triangle is constructed
- Based on the condition C4, C5 & C6 the type of triangle can be determined

ALGORITHM

Step 1: Get input a, b and c i.e., 3 integers which are sides of a triangle

Step 2: Is a triangle?

If (a < (b + c)) AND (b < (a + c)) AND (c < (a + b))

Then IsATriangle = True

Else IsATriangle = False

EndIf

Step 3: Determine Triangle Type

If IsATriangle

Then If (a = b) AND (b = c)

Then triangle is an equilateral

Else If (a ≠ b) AND (a ≠ c) AND (b ≠ c)

Then triangle is a Scalene

Else triangle is an Isosceles

EndIf

EndIf

Else "Not a Triangle"

EndIf

End triangle

4.3 PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int a,b,c;
printf("\n Enter 3 integer which are side of triangle \n");
scanf("%d %d %d",&a,&b,&c);
if(a<1||a>10)

printf("a is out of range\n");


if(b<1 ||b>10)

printf("b is out of range\n");


if(c<1||c>10)

printf("c is out of range");

if((a>=1&&a<=10)&&(b>=1&&b<=10)&&(c>=1&&c<=10))
{
```

```
if((a<(b+c))&&(b<(a+c))&&(c<(a+b)))
{
if((a==b)&&(b==c))
printf("Triangle is equilateral");
else if((a!=b)&&(a!=c)&&(b!=c))
printf("Triangle is scalene\n");
else
printf("Triangle is isoceles\n");
}
else
printf("\n Triangle does not formed");
}}
```

4.4 TESTING

Technique: Equivalence Class Testing

The use of equivalence classes for functional testing has two motivations:

To have a sense of complete testing

Hope to avoid redundancy.

The important aspect of equivalence classes is that they form a partition of a set, where partition refers to a collection of mutually disjoint subsets, the union of which is the entire set. Here entire set is represented provides a form of completeness, and the disjointedness ensures a form of non-redundancy.

The idea of equivalence class testing is to identify test cases by using one element from each equivalence class. If the equivalence classes are chosen wisely, this greatly reduces the potential redundancy among test cases.

Equivalence Class Test Cases for the Triangle Problem

- The maximum limit of each side a, b & c of the triangle is 10 units according to requirement R4.
  $0 \leq a \leq 10$
  $0 \leq b \leq 10$
  $0 \leq c \leq 10$

In the problem statement, note that four possible outputs can occur: Not a Triangle, Scalene, Isosceles and Equilateral.

We can use these to identify output (range) equivalence classes as follows:

R1 = {<a, b, c>: the triangle with sides a, b, and c is equilateral}
R2 = {<a, b, c>: the triangle with sides a, b, and c is isosceles}
R3 = {<a, b, c>: the triangle with sides a, b, and c is scalene}
R4 = {<a, b, c>: sides a, b, and c do not form a triangle}

Weak Normal Equivalence Class Test Cases

TC ID	Test Case Description	Input			Expected Output	Actual Output	Status
		a	b	c			
1	WN1	5	5	5	Equilateral	Equilateral	Pass
2	WN2	2	2	3	Isosceles	Isosceles	Pass
3	WN3	3	4	5	Scalene	Scalene	Pass
4	WN4	4	1	2	Not a Triangle	Not a Triangle	Pass

Because no valid subintervals of variables a, b and c exist, *the strong normal equivalence class test cases* are identical to the weak normal equivalence class test cases.

Weak Robust Equivalence Class Test Cases

Considering the invalid values for a, b, and c yields the following additional weak robust equivalence class test cases

TC ID	Test Case Description	Input			Expected Output	Actual Output	Status
		a	b	c			
1	WR1	-1	5	5	Value of a is not in the range of permitted values	Value of a is not in the range of permitted values	Pass
2	WR2	5	-1	5	Value of b is not in the range of permitted values	Value of b is not in the range of permitted values	Pass
3	WR3	5	5	-1	Value of c is not in the range of permitted values	Value of c is not in the range of permitted values	Pass
4	WR4	11	5	5	Value of a is not in the range of permitted values	Value of a is not in the range of permitted values	Pass
5	WR5	5	11	5	Value of b is not in the range of permitted values	Value of b is not in the range of permitted values	Pass
6	WR6	5	5	11	Value of c is not in the range of permitted values	Value of c is not in the range of permitted values	Pass
7	WR7	1	5	5	Isosceles	Isosceles	Pass

Strong Robust Equivalence Class Test Cases

TC ID	Test Case Description	Input			Expected Output	Actual Output	Status
		a	b	c			
1	SR1	-1	5	5	Value of a is not in the range of permitted values	Value of a is not in the range of permitted values	Pass
2	SR2	5	-1	5	Value of b is not in the range of permitted values	Value of b is not in the range of permitted values	Pass
3	SR3	5	5	-1	Value of c is not in the range of permitted values	Value of c is not in the range of permitted values	Pass
4	SR4	-1	-1	5	Value of a & b is not in the range of permitted values	Value of a & b is not in the range of permitted values	Pass
5	SR5	-1	5	-1	Value of a & c is not in the range of permitted values	Value of a & c is not in the range of permitted values	Pass
6	SR6	5	-1	-1	Value of b & c is not in the range of permitted values	Value of b & c is not in the range of permitted values	Pass
7	SR7	-1	-1	-1	Value of a, b & c is not in the range of permitted values	Value of a, b & c is not in the range of permitted values	Pass

4.5 EXECUTION:

Execute the program and test the test cases in above tables against program and complete the table with for Actual output column and Status column

Test Report:
1. No of TC's Executed: 17
2. No of Defects Raised: 0
3. No of TC's Pass: 17
4. No of TC's Failed: 0

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.

5.1 REQUIREMENTS

R1: The system should read the number of Locks, Stocks and Barrels sold in a month.

(1≤ Locks≤ 70)

(1 ≤ Stocks ≤ 80)

(1 ≤ Barrels ≤ 90)

R2: If R1 is satisfied the system should compute the salesperson's commission depending on the total number of Locks, Stocks & Barrels sold else it should display suitable error message.

Following is the percentage of commission for the sales done:

10% on sales up to (and including) $1000

15% on next $800

20% on any sales in excess of $1800

The system should compute the total dollar sales and output salespersons total dollar sales, and his commission.

5.2 DESIGN

ALGORITHM

Step 1: Define locks_price = 45 $, stock_price= 30$, barrelprice = 25 $

Step 2: Input (locks)

Step 3: while (locks !=-1) 'Input device uses -1 to indicate end of data'
Input (stocks, barrels)
calculate total_locks, total_stocks, total_barrels
Input (locks)
End while

Step 4: calculate total sales
total_sales = (total_locks *45) + (total_stocks*30) + (total_barrels * 25)

Step 5: if (total_sales <= 1000)
commission = 0.10 * total_sales;

else if (total_sales < 1800)
{

```
                commission = 0.10 * 1000;
                commission = commission + (0.15 * (total_sales - 1000));
        }
    else
        {
                commission = 0.10 * 1000;
                commission = commission + (0.15 * 800);
                commission = commission + (0.20 * (total_sales - 1800));
        }
```

Step 6: Output (“Commission is $”, Commission)

Step 7: Exit

5.3 PROGRAM

```
#include<stdio.h>
void main()
{
int locks,stocks,barrels,totalsales;
int totallocks=0,totalstocks=0,totalbarrels=0;
float commission=0;
printf("Enter the number of locks\n");
scanf("%d",&locks);
while(locks!=-1)
{
printf("Enter the no. of stocks\n");
scanf("%d",&stocks);
printf("Enter the no. of barrels");
scanf("%d",&barrels);
totallocks=totallocks+locks;
totalstocks=totalstocks+stocks;
totalbarrels=totalbarrels+barrels;
printf("\nenter number of locks or -1 for end of sale");
scanf("%d",&locks);
}

if(totallocks<-1 ||totallocks>70)
printf("\nLock is out of range\n");
if(totalstocks<0 || totalstocks>80)
printf("\nStock is out of range\n");
if(totalbarrels<0 ||totalbarrels>90)
printf("\nBarrel is out of range\n");
if((totallocks>=0&&totallocks<=70)&&(totalstocks>=0&&totalstocks<=80)&&(totalbarrels
>=0&&totalbarrels<=90))
{
totalsales=(totallocks*45)+(totalstocks*30)+(totalbarrels*25);
```

```
if(totalsales<=1000)
{
commission=0.10*totalsales;
}
else if(totalsales<1800)
{
commission=0.10*1000;
commission=commission+(0.15*(totalsales-1000));
}
else
{
commission=0.10*1000;
commission=commission+(0.15*800);
commission=commission+(0.20*(totalsales-1800));
}
printf("\nThe total sales is %d\n The commission is %f",totalsales,commission);
}
}
```

5.4 TESTING Technique: Equivalence Class Testing

Equivalence partitioning is a black box testing method that divides the input and/or output domain of a program into classes of data from which test cases can be derived.

An equivalence classes form a partition of a set, where partition refers to a collection of mutually disjoint subsets, the union of which is the entire set.

The key goals for equivalence class testing are:

Completeness of test coverage

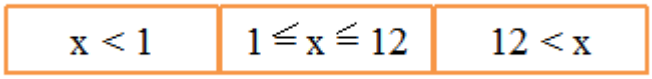
Non-redundancy of test coverage

The idea of equivalence class testing is to identify test cases by using one element from each equivalence class. If the equivalence classes are chosen wisely, this reduces the potential redundancy among test cases.

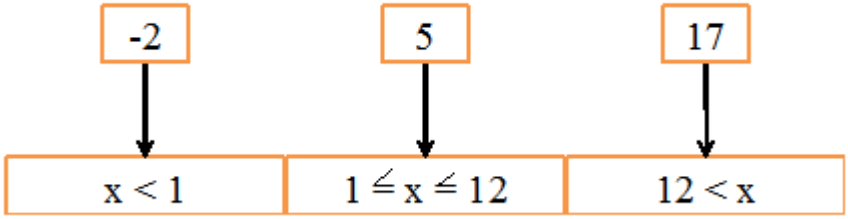
Example: A function which takes a parameter “month”.

The valid range for the month is 1 to 12, representing January to December. This valid range is called a partition.

In this example there are two further partitions of invalid ranges.



Test cases are chosen so that each partition would be tested.



TEST CASE DESIGN

The input domain of the commission problem is partitioned by the limits on locks, stocks and barrels. The first class is the valid input; the other two are invalid. The input domain equivalence classes lead to very unsatisfactory sets of test cases. Equivalence classes defined on the output range of the commission function will be an improvement.

The valid classes of the input variables are:

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

The corresponding invalid classes of the input variables are:

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

One problem occurs, that is the variables locks are also used as a sentinel to indicate no more telegrams. When a value of -1 is given for locks, the while loop terminates, and the values of totallocks, totalstocks and totalbarrels are used to compute sales, and then commission

Weak & Strong Normal Test Cases:

TC Id	Test Case Description	Input Data			Sales	Expected Output (Commission)	Actual Output	Status
		Locks	Stocks	Barrels				
WN1, SN1	Testing for valid values of locks, stocks & barrels	35	40	45	640	3900	3900	Pass

Weak Robust Test Cases:

TC Id	Test Case Description	Input Data			Sales	Expected Output (Commission)	Actual Output	Status
		Locks	Stocks	Barrels				
WR1	Testing for valid value of locks, stocks & barrels	10	10	10	1000	100	100	Pass
WR2	Testing for termination of program with locks=-1 value	-1	40	45	0	0	0	Pass
WR3	Testing for invalid value of locks, i.e., negative value	-2	40	45	----------	Values of locks not in the range 1...70	Values of locks not in the range 1...70	Pass
WR4	Testing for invalid value of locks, i.e., for positive value	71	40	45	----------	Values of locks not in the range 1...70	Values of locks not in the range 1...70	Pass
WR5	Testing for invalid value of stocks, i.e., negative value	35	-1	45	----------	Values of stocks not in the range 1...80	Values of stocks not in the range 1...80	Pass
WR6	Testing for invalid value of stocks, i.e., for positive value	35	81	45	----------	Values of stocks not in the range 1...80	Values of stocks not in the range 1...80	Pass
WR7	Testing for invalid value of barrels, i.e., negative value	35	40	-1	----------	Values of stocks not in the range 1...90	Values of stocks not in the range 1...90	Pass
WR8	Testing for invalid value of barrels, i.e., for positive value	35	40	91	----------	Values of stocks not in the range 1...90	Values of stocks not in the range 1...90	Pass

Strong Robust Test Cases:

A corner of the cube will be in 3 space of the additional strong robust equivalence class test cases:

TC Id	Test Case Description	Input Data			Sales	Expected Output (Commission)	Actual Output	Status
		Locks	Stocks	Barrels				
SR1	Testing for Invalid value of locks	-2	40	45	----------	Values of locks not in the range 1...70	Values of locks not in the range 1...70	Pass
SR2	Testing for Invalid value of stocks	35	-1	45	----------	Values of stocks not in the range 1...80	Values of stocks not in the range 1...80	Pass
SR3	Testing for Invalid value of barrels	35	40	-2	----------	Values of barrels not in the range 1...90	Values of barrels not in the range 1...90	Pass
SR4	Testing for Invalid values of locks & stocks	-2	-1	45	----------	Values of locks not in the range 1...70 Values of stocks not in the range 1...80	Values of locks not in the range 1...70 Values of stocks not in the range 1...80	Pass
SR5	Testing for Invalid values of locks & barrels	-2	40	-1	----------	Values of locks not in the range 1...70 Values of barrels not in the range 1...90	Values of locks not in the range 1...70 Values of barrels not in the range 1...90	Pass
SR6	Testing for Invalid values of stocks & barrels	35	-1	-1	----------	Values of stocks not in the range 1...80 Values of barrels not in the range 1...90	Values of stocks not in the range 1...80 Values of barrels not in the range 1...90	Pass
SR7	Testing for Invalid values of locks, stocks & barrels	-2	-1	-1	----------	Values of locks not in the range 1...70 Values of stocks not in the range 1...80 Values of barrels not in the range 1...90	Values of locks not in the range 1...70 Values of stocks not in the range 1...80 Values of barrels not in the range 1...90	Pass

5.5 EXECUTION & RESULT DISCUSION:

Execute the program against the designed test cases and complete the table for Actual output column and status column.

Test Report: 1. No of TC's Executed: 16

2. No of Defects Raised: 00
3. No of TC's Pass: 16
4. No of TC's Failed: 00

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 value testing,derive different test cases, execute these test cases and discuss the test results.

6.1 REQUIREMENT SPECIFICATION

Problem Definition: "Next Date" is a function consisting of three variables like: month, date and year. It returns the date of next day as output. It reads current date as input date.

The constraints are

C1: $1 \leq \text{month} \leq 12$
C2: $1 \leq \text{day} \leq 31$
C3: $1812 \leq \text{year} \leq 2012$.

If any one condition out of C1, C2 or C3 fails, then this function produces an output "value of month not in the range 1...12".

Since many combinations of dates can exist, hence we can simply displays one message for this function: "Invalid Input Date".

A very common and popular problem occurs if the year is a leap year. We have taken into consideration that there are 31 days in a month. But what happens if a month has 30 days or even 29 or 28 days ?

A year is called as a leap year if it is divisible by 4, unless it is a century year. Century years are leap years only if they are multiples of 400. So, 1992, 1996 and 2000 are leap years while 1900 is not a leap year.

Furthermore, in this Next Date problem we find examples of Zipf's law also, which states that "80% of the activity occurs in 20% of the space". Thus in this case also, much of the source-code of Next Date function is devoted to the leap year considerations.

6.2 DESIGN

Algorithm

STEP 1: Input date in format DD.MM.YYYY

STEP2: if MM is 01, 03, 05,07,08,10 do STEP3 else STEP6

STEP3:if DD < 31 then do STEP4 else if DD=31 do STEP5 else output(Invalid Date);

STEP4: tomorrowday=DD+1 goto STEP18

STEP5: tomorrowday=1; tomorrowmonth=month + 1 goto STEP18

STEP6: if MM is 04, 06, 09, 11 do STEP7

STEP7: if DD<30 then do STEP4 else if DD=30 do STEP5 else output(Invalid Date);

STEP8: if MM is 12

STEP9: if DD<31 then STEP4 else STEP10

STEP10: tomorrowday=1, tommorowmonth=1, tommorowyear=YYYY+1; goto STEP18

STEP11: if MM is 2

STEP12: if DD<28 do STEP4 else do STEP13

STEP13: if DD=28 & YYYY is a leap do STEP14 else STEP15

STEP14: tommorowday=29 goto STEP18

STEP15: tommorowday=1, tomorrowmonth=3, goto STEP18;

STEP16: if DD=29 then do STEP15 else STEP17

STEP17: output("Cannot have feb", DD); STEP19

STEP18: output(tomorrowday, tomorrowmonth, tomorrowyear);

STEP19: exit

6.3 PROGRAM CODE:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int day,month,year;
int nextday,nextmonth,nextyear;

printf("\n Enter the date format DD MM YYYY:");
scanf("%d%d%d",&day,&month,&year);
if(day<1||day>31)
printf("\nday is out of range\n");
if(month<1||month>12)
printf("\nMonth is out of range\n");
```

```
if(year<1812||year>2012)
printf("\nYear is out of range\n");
if(((day>=1)&&(day<=31))&&((month>=1)&&(month<=12))&&((year>=1812)&&(year<=2012)))
{
nextmonth=month;
nextyear=year;

if((month==1)||(month==3)||(month==5)||(month==7)||(month==8)||(month==10))
{
if(day<31)
{
nextday=day+1;
}
else
{
nextday=1;
nextmonth=month+1;
}}
else if((month==4)||(month==6)||(month==9)||(month==11))
{
if(day<30)
nextday=day+1;
else if(day==30)
{
nextday=1;
nextmonth=month+1;
}
else
{
printf("invalid");
```

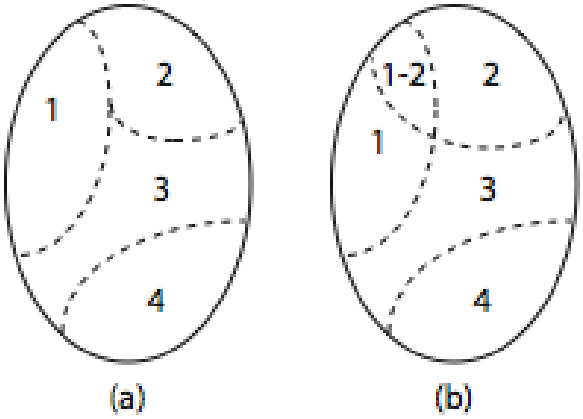
```
exit(0);}
}
else if(month==12)
{
if(day<31)
nextday=day+1;
else
{
nextday=1;
nextmonth=1;
nextyear=year+1;
if(nextyear>2012)
{
printf("Invalid");
exit(0);
}}}
else
{
if((year%4==0&&year%100!=0)||(year%400==0))
{
if(day<29)
nextday=day+1;

else if(day==29)
{
nextday=1;
nextmonth=month+1;
}
else
{
```

```
printf("Invalid date");
exit(0);
}}
else
{
if(day<28)
nextday=day+1;
else if(day==28)
{
nextday=1;
nextmonth=month+1;
}
else
{printf("Invalid date");
exit(0);
}}}
printf("\nThe next date is = %d %d %d",nextday,nextmonth,nextyear);
}}
```

6.4 TESTING

Technique used: Equivalence Class testing

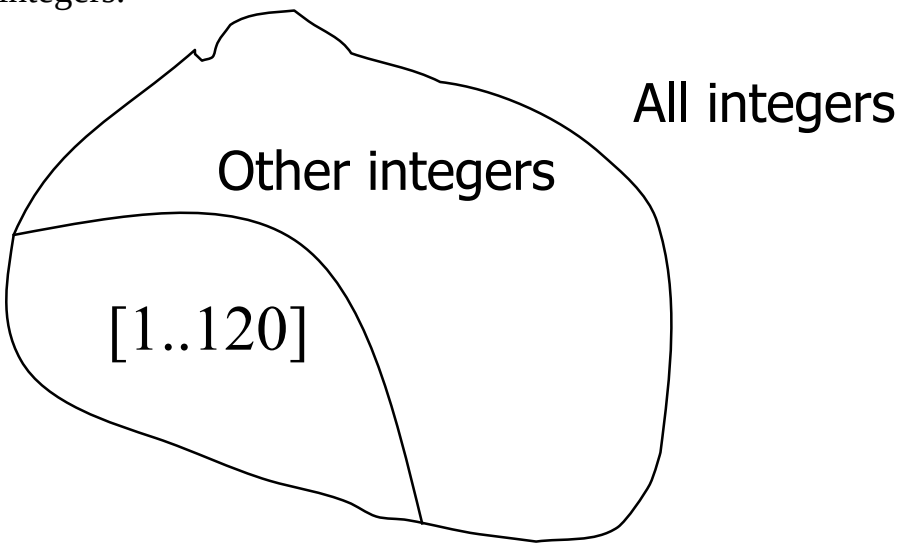
Test selection using equivalence partitioning allows a tester to subdivide the input domain into a relatively small number of sub-domains, say N>1, as shown.



In strict mathematical terms, the sub-domains by definition are disjoint. The four subsets shown in (a) constitute a partition of the input domain while the subsets in (b) are not. Each subset is known as an equivalence class.

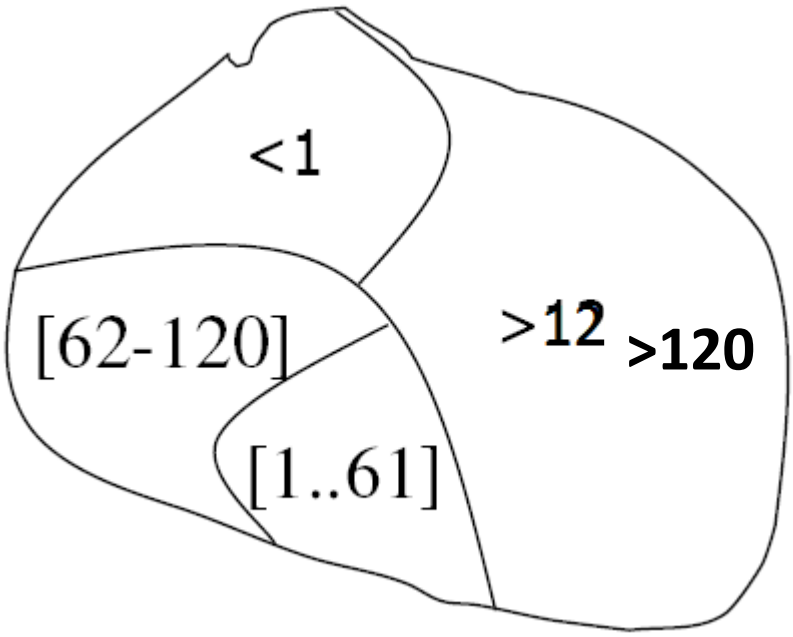
Example:

Consider an application A that takes an integer denoted by age as input. Let us suppose that the only legal values of age are in the range [1..120]. The set of input values is now divided into a set E containing all integers in the range [1..120] and a set U containing the remaining integers.



Further, assume that the application is required to process all values in the range [1..61] in accordance with requirement R1 and those in the range [62..120] according to requirement R2. Thus E is further subdivided into two regions depending on the expected behavior.

Similarly, it is expected that all invalid inputs less than or equal to 1 are to be treated in one way while all greater than 120 are to be treated differently. This leads to a subdivision of U into two categories.



Tests selected using the equivalence partitioning technique aim at targeting faults in the application under test with respect to inputs in any of the four regions, i.e. two regions containing expected inputs and two regions containing the unexpected inputs.

It is expected that any single test selected from the range [1...61] will reveal any fault with respect to R1. Similarly, any test selected from the region [62...120] will reveal any fault with respect to R2. A similar expectation applies to the two regions containing the unexpected inputs.

Test Case design

The NextDate function is a function which will take in a date as input and produces as output the next date in the Georgian calendar. It uses three variables (month, day and year) which each have valid and invalid intervals.

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

Valid Intervals

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

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 and Strong Normal

TC Id	Test Case Description	Input Data			Expected Output	Actual Output	Status
		MM	DD	YYYY			
1	Testing for Valid input changing the day within the month.	6	15	1900	6/16/1900	6/16/1900	pass

Table 1: Weak and Strong Normal

Since the number of variables is equal to the number of valid classes, only one weak normal equivalence class test case occurs, which is the same as the strong normal equivalence class test case (Table 1).

Weak Robust:

TC Id	Test Case Description	Input Data			Expected Output	Actual Output	Status
		MM	DD	YYYY			
1	Testing for Valid input changing the day within the month.	6	15	1900	6/16/1900	6/16/1900	Pass
2	Testing for Invalid Day, day with negative number it is not possible	6	-1	1900	Day not in range	Day not in range	Pass
3	Testing for Invalid Day, day with Out of range i.e., DD=32	6	32	1900	Day not in range	Day not in range	Pass
4	Testing for Invalid Month, month with negative number it is not possible	-1	15	1900	Month not in range	Month not in range	Pass
5	Testing for Invalid month, month with out of range i.e., MM=13 it should MM<=12	13	15	1900	Month not in range	Month not in range	Pass
6	Testing for Year, year is out of range YYYY=1899, it should <=1812	6	15	1811	Year not in range	Year not in range	Pass
7	Testing for Year, year is out of range YYYY=2013, it should <=2012	6	15	2013	Year not in range	Year not in range	Pass

Table 2:Weak Robust

(Table 2) we can see that weak robust equivalence class testing will just test the ranges of the input domain once on each class. Since we are testing weak and not normal, there will only be at most one fault per test case (single fault assumption) unlike Strong Robust Equivalence class testing

Strong Robust

TC Id	Test Case Description	Input Data			Expected Output	Actual Output	Status
		MM	DD	YYYY			
1	Testing for Month is not in range MM=-1 i.e., in negative number there is not possible have to be month in negative number	-1	15	1900	Month not in range	Month not in range	Pass
2	Testing for Day is not in range DD=-1 i.e., in negative number there is not possible have to be Day in negative number	6	-1	1900	Day not in range	Day not in range	Pass
3	Testing for Year is not in range YYYY=1899 i.e., Year should <=1812	6	15	1811	Year not in range	Year not in range	Pass
4	Testing for Day and month is not in range MM=-1, DD=-1 i.e., in negative number there is not possible have to be Day and Month in negative number	-1	-1	1900	i) Day not in range ii) Month not in range	i) Day not in range ii) Month not in range	Pass
5	i) Testing for Day is not in range and Year is not in range DD=-1 i.e., in negative number there is not possible have to be Day in negative number, and ii) YYYY=1899, so the range of year is <=1812	6	-1	1811	i) Day not in range ii) Year not in range	i) Day not in range ii) Year not in range	Pass

6	i) Testing for Month is not in range MM=-1 and i.e., in negative number there is not possible have to be Day in negative number, and ii) Year is not in range YYYY=1899, year should <=1812	-1	15	1811	i) Month not in range ii) Year not in range	i) Month not in range ii) Year not in range	Pass
7	i) Testing for Day is not in range DD=-1 i.e., in negative number there is not possible have to be Day in negative number ii) Testing for Month is not in range MM=-1 and i.e., in negative number there is not possible have to be Day in negative number, and iii) Year is not in range YYYY=1899, year should <=1812	-1	-1	1811	i) Day not in range ii) Month not in range iii) Year not in range	i) Day not in range ii) Month not in range iii) Year not in range	Pass

6.5 EXECUTIONS

Execute the program and test the test cases in Table-1 against program and complete the table with for Actual output column and Status column

Test Report:
1. No of TC's Executed: 15
2. No of Defects Raised: 00
3. No of TC's Pass: 15
4. 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 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. Derive test cases for your program based on decision-table approach, execute the test cases and discuss the results.

7.1 REQUIREMENTS

R1: The program should accept 3 integer values a, b & c which represent the 3 sides of a triangle

R2: If the R1 satisfies, based on the input program should determine whether triangle can be formed or not

R3: If R1 & R2 holds good, then the program should determine the type of triangle i.e.,

Equilateral (Three sides are equal)

Isosceles (Two sides are equal)

Scalene (Three sides are unequal)

Else report suitable error message

7.2 DESIGN

Based on the requirements, the following conditions are drawn:

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 a triangle, if any one of the conditions C1, C2 & C3 fails then triangle cannot be constructed.
- If all the conditions C1, C2 & C3 are true then a triangle is constructed
- Based on the condition C4, C5 & C6 the type of triangle can be determined

ALGORITHM

Step 1: Get input a, b and c i.e., 3 integers which are sides of a triangle

Step 2: Is a triangle?

If (a < (b + c)) AND (b < (a + c)) AND (c < (a + b))

Then IsATriangle = True

Else IsATriangle = False

EndIf

Step 3: Determine Triangle Type

If IsATriangle

Then If (a = b) AND (b = c)

Then triangle is an equilateral

Else If (a ≠ b) AND (a ≠ c) AND (b ≠ c)

Then triangle is a Scalene

Else triangle is an Isosceles

EndIf

EndIf

Else "Not a Triangle"

EndIf

End triangle

7.3 PROGRAM

```
#include<stdio.h>
#include<ctype.h>
void main()
{
int a,b,c;
printf("\n Enter 3 integer which are side of triangle \n");
scanf("%d %d %d",&a,&b,&c);
if((a>=1&&a<=10)&&(b>=1&&b<=10)&&(c>=1&&c<=10))
{
if((a<(b+c))&&(b<(a+c))&&(c<(a+b)))
{
```

```
if((a==b)&&(b==c))
printf("Triangle is equilateral");
else if((a!=b)&&(a!=c)&&(b!=c))
printf("Triangle is scalene\n");
else
printf("Triangle is isoceles\n");
}
else
printf("\n Triangle does not formed");
}
else
printf("\n Impossible\n");
}
```

7.4 TESTING

Technique: Decision Table-Based Testing

Decision tables have been used to represent and analyze complex logical relationship since the early 1960s.

A decision table has four portions:

1. Leftmost column is the Stub portion
2. Right columns of stub is the Entry portion
3. The Condition portion is noted by c's
4. The Action portion is noted by a's

Table 1.1 Portion of a Decision Table

Stub	Rule1	Rule2	Rule3,4	Rule5	Rule6	Rule7,8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	--	T	F	--
a1	x	x		x		
a2	x				x	
a3		x		x		
a4			x			x

- Rules indicates which action are taken for the circumstance indicated in the condition portion of the rule

- The don't care entries are denoted by " -- " or " n/a " and it means " must be false" or " the condition is irrelevant "
- If conditions are allowed to have binary values (true/false, yes/no, 0/1) in the decision table are called limited entry decision tables.
- If conditions are allowed to have several values, the resulting tables are called extended entry decision tables.

Table 1.2 Decision Table for the Triangle Problem

Condition & Actions	Rules										
	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?	-	-	-	T	T	T	T	F	F	F	F
c5: a = c?	-	-	-	T	T	F	F	T	T	F	F
c6: b =c?	-	-	-	T	F	T	F	T	F	T	F
Rule Count	32	16	8	1	1	1	1	1	1	1	1
a1: Not a Triangl	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

The decision table in Table 1.2 illustrates the Triangle Problem

Action: Not a Triangle
When c1 fails, the three integers a, b & c do not constitute sides of a triangle
If any one of the condition c1, c2 & c3 fails, the three integers a, b & c do not constitute sides of a triangle hence it's Not a Triangle
Action: Scalene
When conditions c1, c2 & c3 are true and c4, c5 & c6 are false, then action is Scalene Triangle means all sides are not equal
Action: Isosceles
When conditions c1, c2 & c3 are true and if any two condition among c4, c5 & c6 are false then action is Isosceles Triangle

Action: Equilateral

If all the conditions c1, c2, c3, c4, c5 & c6 are true then action is Equilateral Triangle means all sides are equal.

Action: Impossible

If conditions c1, c2 & c3 are true and two conditions among c4, c5 & c6 is true , then no 1 condition to be false hence it is Impossible

Test Cases:

Based on Decision Table-Based Testing, we can derive 11 test cases: 3 test cases for failing triangle property, 1 case for Scalene, 3 cases for Isosceles, 1 case for Equilateral and 3 cases for Impossible.

TC ID	Test Case Description	Input			Expected Output	Actual Output	Observations
		a	b	c			
01	Testing for Not a Triangle	7	2	3	Not a Triangle	Not a Triangle	Pass
02	Testing for Not a Triangle	2	7	3	Not a Triangle	Not a Triangle	Pass
03	Testing for Not aTriangle	3	2	7	Not a Triangle	Not a Triangle	Pass
04	Testing for Equilateral	8	8	8	Equilateral	Equilateral	Pass
05	Testing for impossible	?	?	?	Impossible	Impossible	Pass
06	Testing for impossible	?	?	?	Impossible	Impossible	Pass
07	Testing for Isosceles	6	6	7	Isosceles	Isosceles	Pass
08	Testing for impossible	?	?	?	Impossible	Impossible	Pass
09	Testing for Isosceles	6	7	6	Isosceles	Isosceles	Pass
10	Testing for Isosceles	7	6	6	Isosceles	Isosceles	Pass
11	Testing for Scalene	3	4	5	Scalene	Scalene	Pass

7.5 EXECUTION

Execute the program against the designed test cases and complete the table for Actual Output and Observation column

Test Report:

1. No. of Test cases Executed: 11
2. No. of Defect detected:0
3. No. of Test cases Passed:11
4. No. of Test cases Failed: 0

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.

8.1 REQUIREMENTS SPECIFICATIONS:

R1: The system should read the number of Locks, Stocks and Barrels sold in a month.

(1≤ Locks≤ 70)

(1 ≤ Stocks ≤ 80)

(1 ≤ Barrels ≤ 90)

R2: If R1 is satisfied the system should compute the salesperson's commission depending on the total number of Locks, Stocks & Barrels sold else it should display suitable error message.

Following is the percentage of commission for the sales done:

10% on sales up to (and including) $1000

15% on next $800

20% on any sales in excess of $1800

The system should compute the total dollar sales and output salespersons total dollar sales, and his commission.

8.2 DESIGN

ALGORITHM

Step 1: Define locks_price = 45 $, stock_price= 30$, barrelprice = 25 $

Step 2: Input (locks)

Step 3: while (locks !=-1) 'Input device uses -1 to indicate end of data'
Input (stocks, barrels)
calculate total_locks, total_stocks, total_barrels
Input (locks)
End while

Step 4: calculate total sales
total_sales = (total_locks *45) + (total_stocks*30) + (total_barrels * 25)

Step 5: if (total_sales <= 1000)

```
                commission = 0.10 * total_sales;

        else if (total_sales < 1800)
                {
                        commission = 0.10 * 1000;
                        commission = commission + (0.15 * (total_sales - 1000));
                }
        else
                {
                        commission = 0.10 * 1000;
                        commission = commission + (0.15 * 800);
                        commission = commission + (0.20 * (total_sales - 1800));
                }
```

Step 6: Output ("Commission is $", Commission)

Step 7: Exit

8.3 PROGRAM CODE

```
#include<stdio.h>
void main()
{
int locks,stocks,barrels,totalsales;
int totallocks=0,totalstocks=0,totalbarrels=0;
float commission=0;
printf("Enter the number of locks\n");
scanf("%d",&locks);
while(locks!=-1)
{
printf("Enter the no. of stocks\n");
scanf("%d",&stocks);
printf("Enter the no. of barrels\n");
scanf("%d",&barrels);
totallocks=totallocks+locks;
totalstocks=totalstocks+stocks;
totalbarrels=totalbarrels+barrels;
printf("\nEnter -1 to end of the sales\n Else Enter the number of locks\n");
scanf("%d",&locks);



}
if((totallocks>=0&&totallocks<=70)&&(totalstocks>=0&&totalstocks<=80)&&(totalbarrels
>=0&&totalbarrels<=90))
{
totalsales=(totallocks*45)+(totalstocks*30)+(totalbarrels*25);
```

```
if(totalsales<=1000)
{
commission=0.10*totalsales;
}
else if(totalsales<1800)
{
commission=0.10*1000;
commission=commission+(0.15*(totalsales-1000));
}
else
{
commission=0.10*1000;
commission=commission+(0.15*800);
commission=commission+(0.20*(totalsales-1800));
}
printf("The total sales is %d\n The commission is %f",totalsales,commission);
}
else
{
printf("\n Out of Range");
}}
```

8.4 TESTING

Technique: Decision Table-Based

Using the decision table we get 6 functional test cases: 3 cases out of range, 1 case each for sales greater than $1800, sales greater than $1000, sales less than or equal to $1000.

Conditions	R1	R2	R3	R4	R5	R6
C1: 1≤locks≤70?	F	T	T	T	T	T
C2: 1≤stocks≤80?	--	F	T	T	T	T
C3: 1≤barrels≤90?	--	--	F	T	T	T
C4: sales>1800?	--	--	--	T	F	F
C5: sales>1000?	--	--	--	--	T	F
C6: sales≤1000?	--	--	--	--	--	T
Rule Count	32	16	08	04	02	01
a1: com1 = 0.10*Sales						X
a2: com2 = com1+0.15*(sales-1000)					X	
a3: com3 = com2+0.20*(sales-1800)				X		
a4: Out of Range.	X	X	X			

Testing Table 7.1 Decision Table for the Triangle Problem

Deriving test cases using Decision table:

TC ID	Test Case Description	Locks	Stocks	Barrels	Expected Output		Actual Output	Status
1	Testing for Requirement 1 Condition 1 (C1)	-2	40	45	Out of Range		Out of Range	Pass
2	Testing for Requirement 1 Condition 1 (C1)	90	40	45	Out of Range		Out of Range	Pass
3	Testing for Requirement 1 Condition 2 (C2)	35	-3	45	Out of Range		Out of Range	Pass
4	Testing for Requirement 1 Condition 2 (C2)	35	100	45	Out of Range		Out of Range	Pass
5	Testing for Requirement 1 Condition 3 (C3)	35	40	-10	Out of Range		Out of Range	Pass
6	Testing for Requirement 1 Condition 3 (C3)	35	40	150	Out of Range		Out of Range	Pass
7	Testing for Requirement 2	5	5	5	500	a1: 50	500	Pass
8	Testing for Requirement 2	15	15	15	1500	a2: 175	1500	Pass
9	Testing for Requirement 2	25	25	25	2500	a3: 360	2500	Pass

8.5 EXECUTION & RESULT DISCUSION:

Execute the program against the designed test cases and complete the table for Actual output column and status column.

TEST REPORT:

1. No of TC's Executed: 09
2. No of Defects Raised: 00
3. No of TC's Pass: 09
4. No of TC's Failed: 00

The commission problem is not well served by a decision table analysis because it has very little decisional. Because the variables in the equivalence classes are truly independent, no impossible rules will occur in a decision table in which condition correspond to the equivalence classes.

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.

9.1 REQUIREMENTS SPECIFICATIONS:

A rifle salesperson sold rifle locks, stocks, and barrels made by a gunsmith and the cost are as follows:
Locks - $45
Stocks - $30
Barrels - $25

The salesperson had to sell at least one complete rifle per month, and production limits were such that the most the salesperson could sell in a month was 70 locks, 80 stocks, and 90 barrels.

After each town visit, the salesperson sent a telegram to the gunsmith with the number of locks, stocks, and barrels sold in that town. At the end of a month, the salesperson sent a very short telegram showing -1 lock sold.

The gunsmith then knew the sales for the month were complete and computed the salesperson's commission as follows:
Sales from $1 to $1000 = 10%
Sales from $1001 to $1800 = 15%
Sales in excess of $1800 = 20%

The commission program produced a monthly sales report that gave the total number of locks, stocks, and barrels sold, the salesperson's total dollar sales, and the commission.

9.2 DESIGN ALGORITHM

Step 1: Define locks_price = 45 $, stock_price= 30$, barrelprice = 25 $
Step 2: Input (locks)
Step 3: while (locks !=-1) ‘Input device uses -1 to indicate end of data’
Input (stocks, barrels)
calculate total_locks, total_stocks, total_barrels
Input (locks)
End while
Step 4: calculate total sales
total_sales = (total_locks *45) + (total_stocks*30) + (total_barrels * 25)

Step 5: if (total_sales <= 1000)
commission = 0.10 * total_sales;

```
        else if (total_sales < 1800)
                {
                        commission = 0.10 * 1000;
                        commission = commission + (0.15 * (total_sales - 1000));
                }
        else
                {
                        commission = 0.10 * 1000;
                        commission = commission + (0.15 * 800);
                        commission = commission + (0.20 * (total_sales - 1800));
                }
```

Step 6: Output ("Commission is $", Commission)

Step 7: Exit

9.3 PROGRAM

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

2.      int main()
3.      {
4.              int locks, stocks, barrels, total_sales;
5.              int total_locks=0, total_stocks=0,total_barrels=0;
6.              float commission=0;

7.              printf("Enter the number of locks \n");
8.              scanf("%d",&locks);

9.              while( locks !=-1)
10.             {
11.                     printf("Enter the number of stocks \n");
12.                     scanf("%d",&stocks);
13.                     printf("Enter the number of barrelss \n");
14.                     scanf("%d",&barrels);

15.                     total_locks = total_locks + locks;
16.                     total_stocks = total_stocks + stocks;
17.                     total_barrels = total_barrels + barrels;

18.                     printf(" \n Enter -1 to end of sale \n Else enter the number of locks
                                \n");
19.                     scanf("%d", &locks);
20.               }
```

```
21.   if ((total_locks >=0 && total_locks <= 70) && (total_stocks >= 0 && total_stocks <=
      80) && (total_barrels >= 0 && total_barrels <= 90))
            {

22.   total_sales = (total_locks * 45) + (total_stocks * 30) + (total_barrels * 25);

23.               if (total_sales <= 1000)
                  {
24.                     commission = 0.10 * total_sales;
25.               }

26.               else if (total_sales < 1800)
                  {
27.                     commission = 0.10 * 1000;
28.                     commission = commission + (0.15 * (total_sales - 1000));
29.               }

30.               else
                  {
31.                     commission = 0.10 * 1000;
32.                     commission = commission + (0.15 * 800);
33.                     commission = commission + (0.20 * (total_sales - 1800));
34.               }

35.               printf("The total sales is %d \n The commission is %f",total_sales,
                  commission);
36.         }

37.         else
              {
38.               printf("\n Invalid Data \n");
39.           }
40.   }
```

9.4 TESTING

Dataflow testing refers to forms of structural testing that focus on the points at which variables receive values and the points at which these values are used (or referenced).

DEF(v, n): Node n ϵ G(P) is a *defining node of the variable* v ϵ V, written as DEF(v, n), iff the value of the variable v is defined at the statement fragment corresponding to node n.

USE(v, n): Node n ϵ G(P) is a *usage node of the variable* v ϵ V, written as USE(v, n), iff the value of the variable v is used at the statement fragment corresponding to node n.

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

The nodes corresponding to predicate uses always have an outdegree ≥ 2, and nodes corresponding to computation uses always have an outdegree ≥1.

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

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

DATA FLOW TESTING: KEY STEPS

For a given program:

1. Assign the line numbers to the program
2. List define/ use nodes for variables.
3. List occurrences & assign a category for each variable.
4. Identify du-pairs and their p-use and c-use
5. Define test cases, depending on the required coverage.

Table 9.1: Define/Use Nodes for Variables in the Commission Problem

Variable	Defined at Node	Used at Node
total_locks	5, 15	15, 21,22
total_stocks	5, 16	16, 21,22
total_barrels	5, 17	17, 21,22
Locks	8, 19	9, 15
stocks	12	16
barrels	14	17
total_sales	22	23, 24, 26, 28, 33, 35
Commission	6, 24, 27, 28, 31, 32, 33	28. 32, 33, 35

Table 9.2: Occurrences and assignment of category for each variable

line no.s	Category		
	Definition	c-use	p-use
5	total_locks total_stocks total_barrels		
6	commission		
7			
8	locks		
9			locks
10			
11			
12	stocks		
13			
14	barrels		
15	total_locks	total_locks, locks	

16	total_stocks	total_stocks, stocks	
17	total_barrels	total_barrels, barrels	
18			
19	locks		
20			
21			total_locks total_stocks total_barrels
22	total_sales	total_locks total_stocks total_barrels	
23			total_sales
24	commission	total_sales	
25			
26			total_sales
27	commission		
28	commission	total_sales commission	
29			
30			
31	commission		
32	commission	commission	
33	commission	total_sales commission	
34			
35		total_sales commission	

Table 9.3: Define / Use Paths for variables

Variables	definition - use pair
	Beginning node → end Node
total_locks	5 →15 5 →21 15 →21
total_stocks	5 →16 5 →21 16 →21
total_barrels	5 →17 5 →21 17 →21
locks	8 →9 8 →15 19 →9 19 →15
stocks	12 →16

barrels	14 →17
total_sales	22 →23 22 →24 22 →26 22 →28 22 →33 22 →35
commission	Selected Pairs 24 →35 28 →35 33 →35

TEST CASES BASED ON ALL DEFINITION: To achieve 100% All-definitions data flow coverage at least one sub-path from each variable definition to some use of that definition (either c- or p- use) must be executed.

Test Cases:

				Inputs			Expected output	
Tc ID	Variable (s)	du-pair	sub-path	locks	stocks	barrels	total_sales	Commi -ssion
1	locks	8 →9	8 →9	35	-	-	-	-
2		8 →15	8, 9, 15	35	40	45	3900	640
3		19 →9	19 →9	-1	40	45	Invalid	
4		19 →15	19, 9,15	35	40	45	3900	640
5	stocks	12 →16	12 →16	35	40	-	-	-
6	barrels	14 →17	14 →17	35	40	45	3900	640
7	total_sales	22 →23	22 →23	5	5	5	500	50
8		22 →24	22, 23, 24	5	5	5	500	50
9		22 →26	22 →26	14	14	14	1400	160
10		22 →28	22, 26, 28	14	14	14	1400	160
11		22 →33	22 →33	48	48	48	4800	820
12		22 →35	22, 23, 26, 33	35	40	45	3900	640
13	commission	24 →35	24 →35	5	5	5	500	50
14		28 →35	28 →35	14	14	14	1400	160
15		33 →35	33 →35	48	48	48	4800	820

9.5 EXECUTION

Execute the program and test the test cases in above Tables against program and complete the table with for Actual output column and Status column

Test Report:

1. No of TC's Executed: 15
2. No of Defects Raised: 00
3. No of TC's Pass: 15
4. No of TC's Failed: 00

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.

10.1 REQUIREMENTS

R1: The system should accept ‘n’ number of elements and key element that is to be searched among ‘n’ elements.

R2: Check if the key element is present in the array and display the position if present otherwise print unsuccessful search.

10.2 DESIGN

Use integer array to store ‘n’ number of elements. Iterative programming technique is used.

ALGORITHM

Step 1: Input value of ‘n’. Enter ‘n’ integer numbers in array int mid;
Step 2: Initialize low = 0, high = n -1
Step 3: until (low <= high) do
mid = (low + high) / 2
if (a[mid] == key)
then do Step 5
else if (a[mid] > key)
then do
high = mid - 1
else
low = mid + 1
Step 4: Print unsuccessful search do step 6.
Step 5: Print Successful search. Element found at position mid+1.
Step 6: Stop.

10.3 PROGRAM

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

2.  int main()
3.  {
4.    int i, low, high, mid, n, key, array[100];

5.    printf("Enter number of elements\n");
```

```
6.      scanf("%d",&n);

7.      printf("Enter %d integers\n", n);
8.      for (i=0; i<n; i++ )
9.        scanf("%d",&array[i]);

10.     printf("Enter value to find\n");
11.     scanf("%d",&key);

12.     low = 0;
13.     high = n-1;

14.     while( low <= high )
        {
15.         mid = (low + high)/2;

16.         if ( array[mid] == key )
            {
17.             printf("%d found at location %d.\n", key, mid+1);
18.             return;
19.         }

20.         else if ( array[mid] < key )
21.             low = mid + 1;

22.         else
23.             high = mid- 1;
24.       }

25.     if ( low > high )
26.         printf("Not found! %d is not present in the list.\n", key);

27.     return;
28.   }
```

10.4 TESTING

Technique Used: Basis Path Testing

Basis path testing is a form of Structural testing (White Box testing). The method devised by McCabe to carry out basis path testing has four steps. These are:

1. Compute the program graph.
2. Calculate the cyclomatic complexity.
3. Select a basis set of paths.
4. Generate test cases for each of these paths.

Step 1: The program graph and DD-path graph for binary search program is shown below

Table 8.1: DD-Path in Fig 8.1

Program Graph Nodes	DD-Path
5	First
6-7	A
8	B
9	C
10-13	D
14	E
15	F
16	G
17-19	H
20	I
21	J
23	K
24	L
25	M
26	N
27	O
28	Last

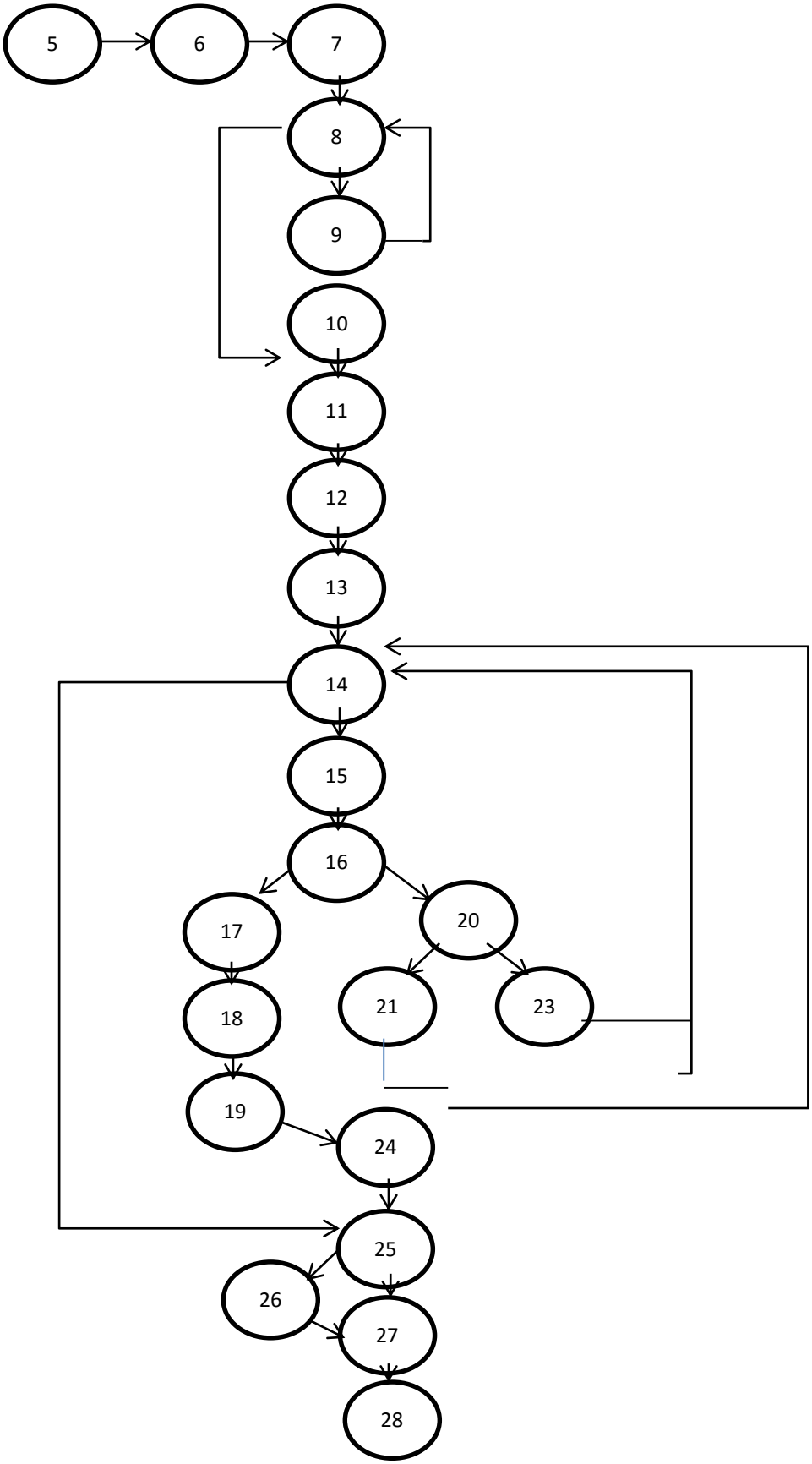


Fig 10.1: Program Graph for Binary Search Program

Using the program graph, DD (Decision-to-Decision) path graph is derived for binary search program

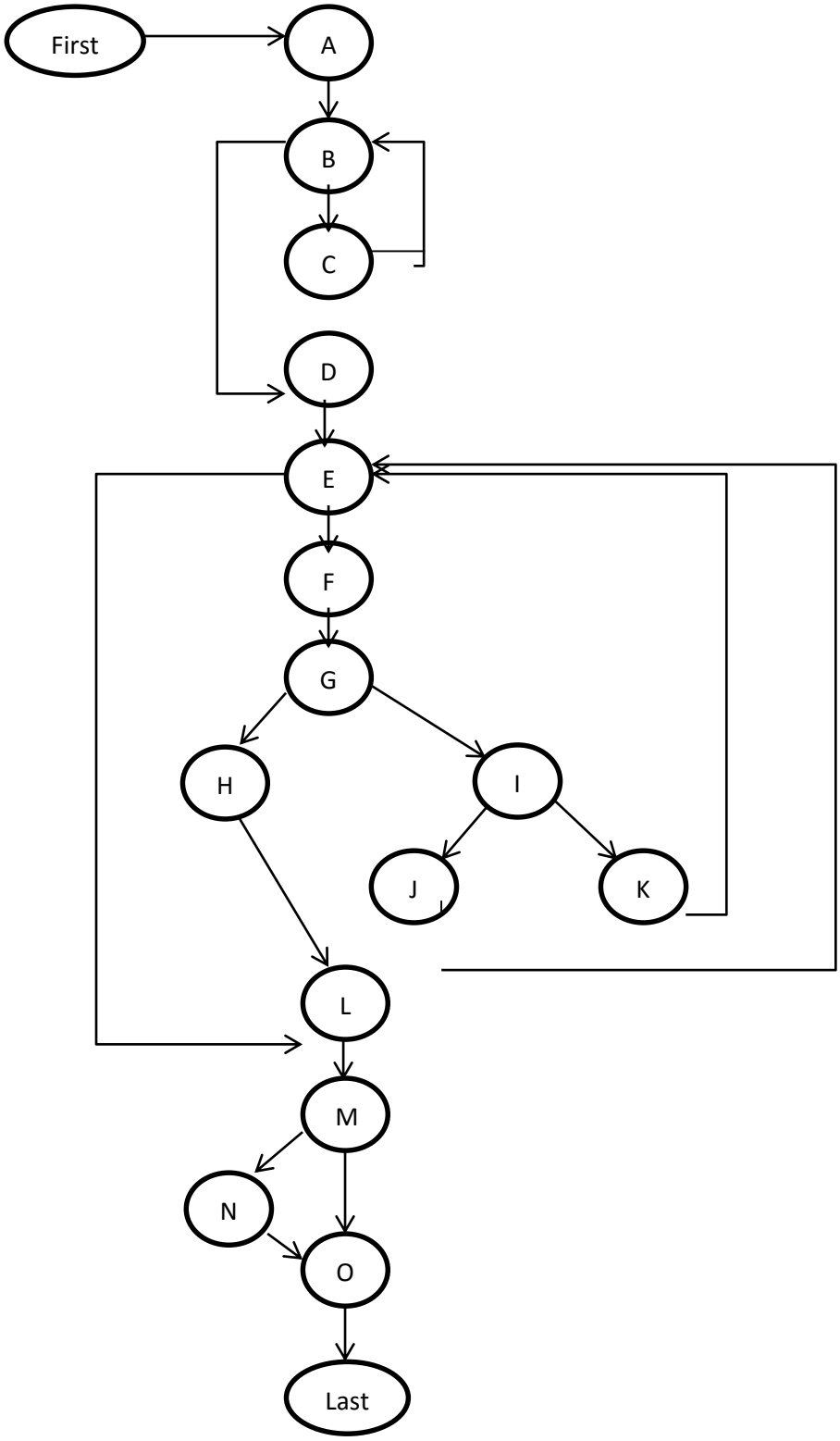


Fig 10.2: DD- Path Graph for Binary Search Program

Step 2: Calculate the cyclomatic complexity

The formula for cyclomatic complexity is given by

V(G) = e - n + p or V(G) = e - n + 2p

Where, e: is the number of edges
n: is the number of nodes
p: is the number of connected regions

The number of linearly independent paths from the source node to the sink node in Fig. 8.2 is

$$V(G) = e - n + 2p = 21 - 17 + 2(1) = 6$$

Step 3: The six linearly independent paths of our graph are as follows:

P1: First, A, B, C, B, D, E, F, G, H, L, M, O, Last
P2: First, A, B, C, B, D, E, F, G, I, J, E, F, G, H, L, M, O, Last
P3: First, A, B, C, B, D, E, F, G, I, K, E, F, G, H, L, M, O, Last
P4: First, A, B, D, E, L, M, N, Last
P5: First, A, B, C, B, D, E, F, G, I, J, E, L, M, N, O, Last
P6: First, A, B, C, B, D, E, F, G, I, K, E, L, M, N, O, Last

Step 4: Deriving test cases using Basis Path Testing

TC ID	Test Case Description	Value of 'n'	array elements	key	Expected Output	Actual Output	Status
1	Testing for Path P1	5	2,4,6,8,10	6	Key Found at position 3	Key Found at position 3	Pass
2	Testing for Path P2	5	2,4,6,8,10	4	Key Found at position 2	Key Found at position 2	Pass
3	Testing for Path P3	5	2,4,6,8,10	10	Key Found at position 5	Key Found at position 5	Pass
4	Testing for Path P4	0	----------	5	Key Not Found	Key Not Found	Pass
5	Testing for Path P5	5	2,4,6,8,10	1	Key Not Found	Key Not Found	Pass
6	Testing for Path P6	5	2,4,6,8,10	11	Key Not Found	Key Not Found	Pass

10.5 EXECUTION:

Execute the program against the designed test cases and complete the table for Actual output column and status column.

Test Report:1. No of TC's Executed: 06

2. No of Defects Raised: 00

3. No of TC's Pass: 06

4. No of TC's Failed: 00

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

11.1 REQUIREMENTS

- Quicksort is the sorting algorithm that is based on the divide-and-conquer approach. Quicksort divides its input elements according to their value.
- A partition is an arrangement of the array's elements so that all the elements to the left of some element A[s] are less than or equal to A[s], and all the elements to the right of A[s] are greater than or equal to it
- After a partition is achieved, $A[s]$ will be in its final position in the sorted array and continue sorting the two sub-arrays to the left and to the right of $A[s]$ independently

R1: The system should accept Subarray of array $A[0..n-1]$ defined by its left and right indices l and r $(l<r)$
R2: Partitions a subarray using the first element as a pivot
R3: Partition of $A[l..r]$, with the split position returned as this function's value

11.2 DESIGN

Integer array is used as a data structure to store 'n' number of elements. Iterative programming technique is used.

11.3 PROGRAM CODE

```
1.  #include<stdio.h>
2.  void main()
3.  {
4.      int a[20],i,j,n;

5.      printf("\n QUICKSORT \n");
6.      printf(" \n Enter the value of n: ");
7.      scanf("%d", &n);

8.      printf("\n The Unsorted elements are: \n");
9.      for(i=0;i<n;i++)
10.     scanf("%d",&a[i]);


11.     for(i=0;i<n;i++)
12.     printf("%d\n",a[i]);

13.     qsort(a,0,n-1);

14.     printf(" \n SORTED array is: \t");
15.     for(i=0;i<n;i++)
```

```
16.             printf(" %d\t", a[i]);

17.     }

18.     int qsort(int a[], int l, int r)
19.     {
20.             int s;
21.             if(l<r)
22.             {
23.                     s=partition(a,l,r);
24.                     qsort(a,l,s-1);
25.                     qsort(a,s+1,r);
26.             }
27.             return;
28.     }

29.     int partition(int a[], int l, int r)
30.     {
31.             int i, j, p;
32.             p=a[l];
33.             i= l+1;
34.             j= r;
35.             while(i<=j)
                {
36.                     while(a[i]<=p)
37.                             i++;
38.                     while(a[j]>p)
39.                             j--;
40.                     swap(&a[i], &a[j]);
41.             }
42.             swap(&a[i], &a[j]);
43.             swap(&a[l], &a[j]);
44.             return j;
45.      }
46.     int swap(int *x, int *y)
47.     {
48.             int temp;
49.             temp=*x;
50.             *x=*y;
51.             *y=temp;
52.             return;
53.     }
```

11.4 TESTING

Technique Used: Basis Path Testing

Basis path testing is a form of Structural testing (White Box testing). The method devised by McCabe to carry out basis path testing has four steps. These are:

1. Compute the program graph.
2. Calculate the cyclomatic complexity.
3. Select a basis set of paths.
4. Generate test cases for each of these paths.

Testing qsort:

Step 1: The program graph for qsort function is shown below.

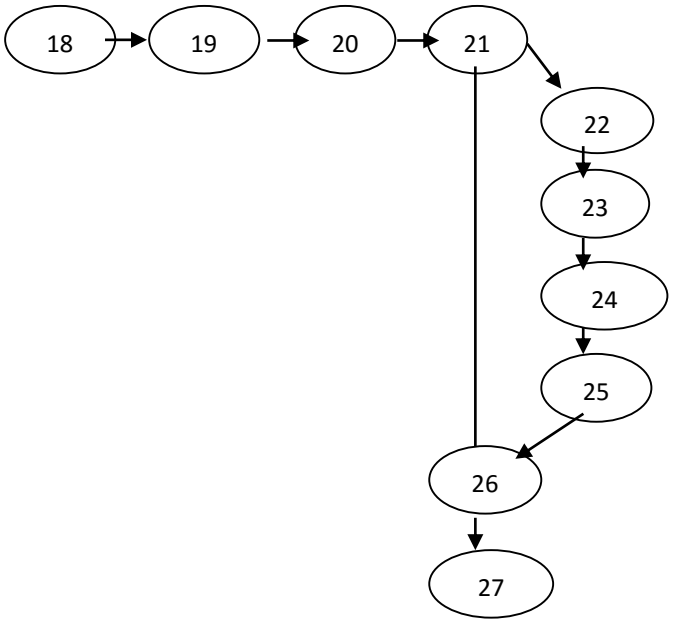


Fig 11.1: Program Graph for qsort function

Table 11.1: DD-Path in Fig 9.1

Program Graph Nodes	DD-Path
18	First
19,20	A
21	B
22-25	C
26	D
27	Last

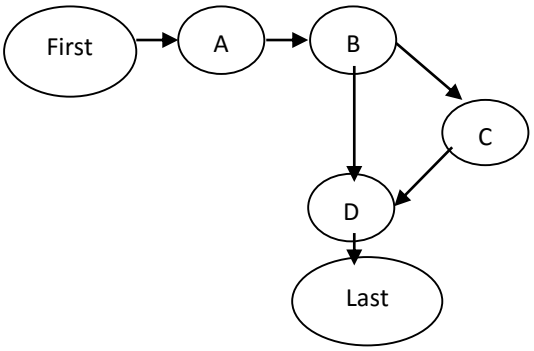


Fig 11.2: DD-Path Graph for qsort function

Step 2: Calculate the cyclomatic complexity

The formula for cyclomatic complexity is given by

$$V(G) = e - n + p \quad \text{or} \quad V(G) = e - n + 2p$$

The number of linearly independent paths from the source node to the sink node in Fig. 9.2 is

$$V(G) = e - n + 2p = 6\text{-}6 + 2(1) = 2$$

Step 3: Two linearly independent paths of DD-Path graph are as follows:

P1: First, A, B, C, D, Last
P2: First, A, B, D, Last

Test Cases:

TC ID	Test Description	Input	Expected Output	Actual Output	Status
1	Testing for path P1	2 9 1 7 5	1 2 5 7 9	1 2 5 7 9	Pass
2	Testing for path P2	3	3	3	Pass

Testing partition

Step 1: The program graph for partition function is shown below.

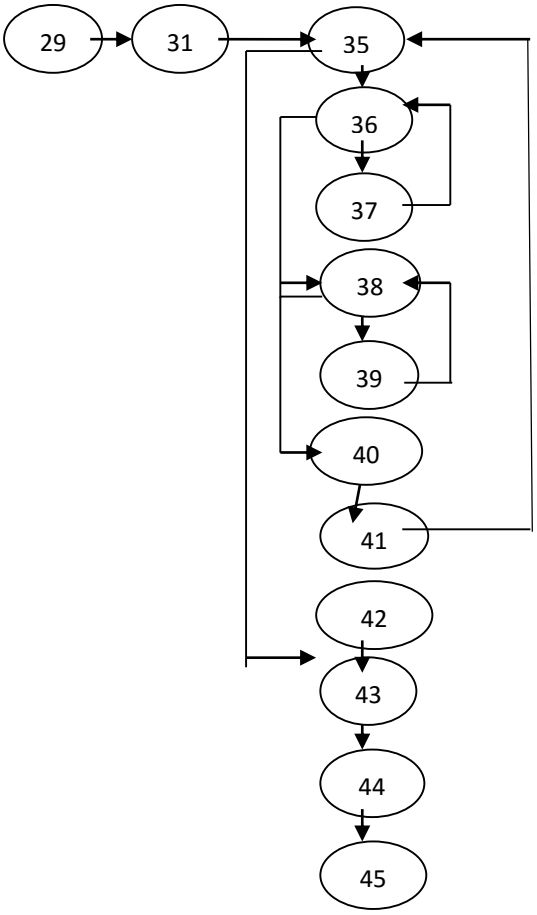


Fig 9.3: Program Graph for partition function

Table 9.1: DD-Path in Fig 9.1

Program Graph Nodes	DD-Path
29	First
31	A
35	B
36	C
37	D
38	E
39	F
40	G
41	H
42-44	I
45	Last

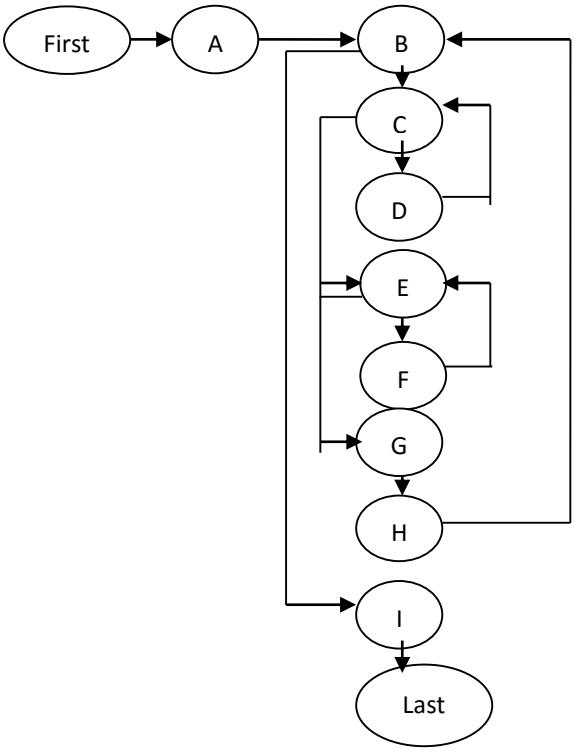


Fig 9.4: DD-Path Graph for partition function

Step 2: Calculate the cyclomatic complexity

The number of linearly independent paths from the source node to the sink node in Fig. 9.4 is

$$V(G) = e - n + 2p = 13-11 + 2(1) = 4$$

Step 3: Four linearly independent paths of DD-Path graph are as follows:

P1: First, A, B, C, D, C, E, F, E, G, H, B, I, Last
P2: First, A, B, C, D, C, E, G, H, B, I, Last
P3: First, A, B, C, E, F, E, G, H, B, I, Last
P4: First, A, B, I, Last

Test Cases:

TC ID	Test Description	Input (Array Elements)	Expected Output	Actual Output	Status
1	Testing for path P1	2 9 5 5 2 6	1 2 5 5 6 9	1 2 5 5 6 9	Pass
2	Testing for path P2	4 3 6 1 2	1 2 3 4 6	1 2 3 4 6	Pass
3	Testing for path P3	15 10 2 9 3	2 3 9 10 15	2 3 9 10 15	Pass
4	Testing for path P4	15	15	15	Pass

11.5 EXECUTION

Compile the program and enter array values as input and test cases are executed based on basis paths

Test Report:

1. No of TC’s Executed: 04
2. No of Defects Raised: 00
3. No of TC’s Pass: 04
4. 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

12.1 REQUIREMENTS:

R1: The system should accept marks of 6 subjects & each mark should be in the range of 1 to 100.

For example, 1<=sub1<=100
1<=sub2<=100
1<=sub3<=100

R2: If R1 is satisfied, compute average of marks scored and based on average marks display the grade.

10.2 DESIGN:

Use the total average marks scored to grade the student.

If average marks lies b/w <35 && >0 then make it as FAIL
avmar>35 && avmar<=40 make it as Grade C
avmar>40 && avmar<=50make it as Grade C+
avmar>50 && avmar<=60make it as Grade B
avmar>60 && avmar<=70make it as Grade B+
avmar>70 && avmar<=80 make it as Grade A
avmar>80 && avmar<=100 make it as Grade A+

12.3 PROGRAM CODE:

```
1.  #include<stdio.h>
2.  main()
3.  {
4.  float me,usp,fs,cn2,st,cd,avmar;
5.  printf("Letter Grading Program");
6.  printf("Enter the marks for ME");
7.  scanf("%f",&me);
8.  printf("Enter the marks for USP");
9.  scanf("%f",&usp);
10. printf("Enter the marks for FS");
```

```
11.     scanf("%f",&fs);
12.     printf("Enter the marks for CN2");
13.     scanf("%f",&cn2);
14.     printf("Enter the marks for ST");
15.     scanf("%f",&st);
16.     printf("Enter the marks for CD");
17.     scanf("%f",&cd);
18.
if((me>=0&&me<=100)&&(usp>=0&&usp<=100)&&(fs>=0&&fs<=100)&&(c
n2>=0&&cn2<=100)&&(st>=0&&st<=100)&&(cd>=0&&cd<=100))


19.     {
20.     if(me<35||usp<35||fs<35||cn2<35||st<35||cd<35)
21.     {
22.     printf("\nFail");
23.     goto end;
24.     }
25.     avmar=(me+usp+fs+cn2+st+cd)/6;
26.     printf("The average marks= %2f\n",avmar);
27.     if((avmar>=35)&&(avmar<=40))
28.     printf("Grade C");
29.     else if((avmar>40)&&(avmar<=50))
30.     printf("grade C+");
31.     else if((avmar>50)&&(avmar<=60))
32.     printf("Grade B");
33.     else if((avmar>60)&&(avmar<=70))
34.     printf("Grade B+");
35.     else if((avmar>70)&&(avmar<=80))
36.     printf("Grade A");
37.     else printf("Grade A+");
38.     }
39.     else{
40.     printf("\n Invalid Data");
41.     }
42.     end:return;
43.     }
```

12.4 TESTING

Technique Used: Basis Path Testing

Basis path testing is a form of Structural testing (White Box testing). The method devised by McCabe to carry out basis path testing has four steps. These are:

1. Compute the program graph.
2. Calculate the cyclomatic complexity.
3. Select a basis set of paths.
4. Generate test cases for each of these paths.

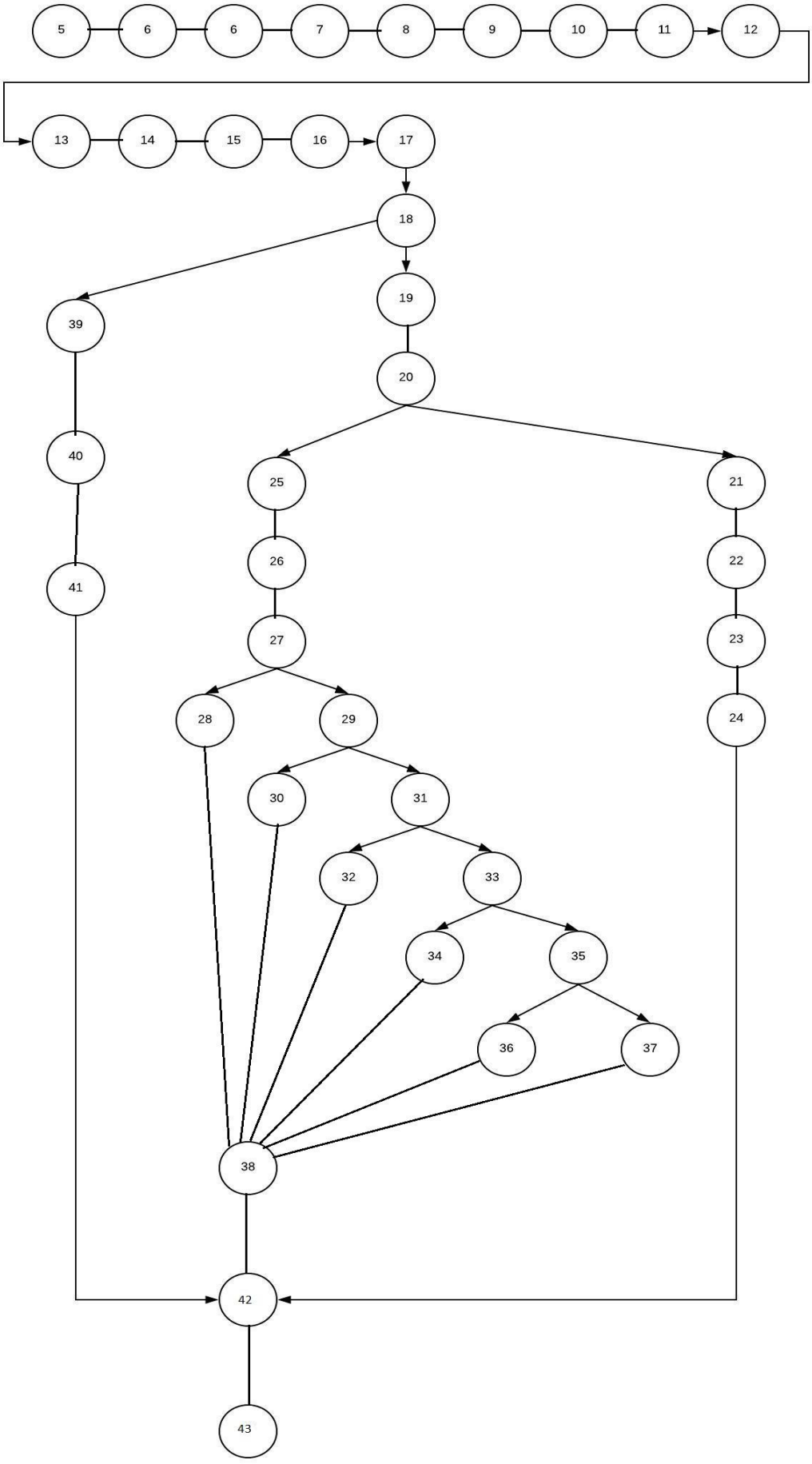


Fig 12.1: Program Graph for absolute letter grading procedure

Table 12.1: DD-Path in Fig 10.1

Program Graph Nodes	DD-Path
5	First
6-17	A
18	B
19	C
20	D
21-24	E
25-26	F
27	G
28	H
29	I
30	J
31	K
32	L
33	M
34	N
35	O
36	P
37	Q
38	R
39-41	S
42	T
43	Last

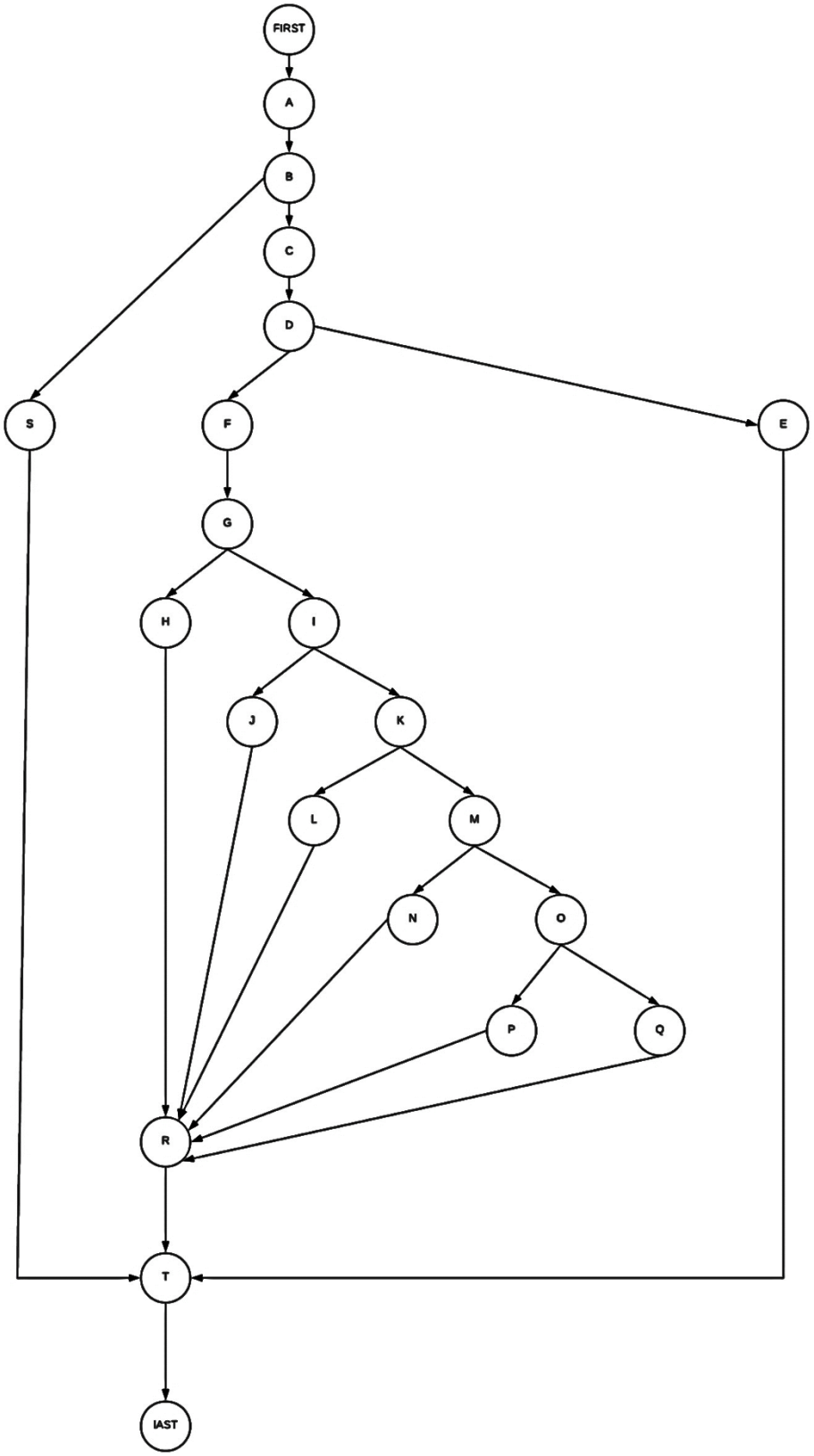


Fig 12.2: DD-Path Graph for absolute letter grading procedure

Step 2: Calculate the cyclomatic complexity

The formula for cyclomatic complexity is given by
$V(G) = e - n + p$ or $V(G) = e - n + 2p$
Where, e: is the number of edges
n: is the number of nodes
p: is the number of connected regions

The number of linearly independent paths from the source node to the sink node in Fig. 10.2 is
$V(G) = e - n + 2p = 28-22 + 2(1) = 8$

Step 3: The eight linearly independent paths of DD-Path graph are as follows:

P1: First, A, B, C, D, E, T, Last
P2: First, A, B, C, D, F, G, H, R, T, Last
P3: First, A, B, C, D, F, G, I, J, R, T, Last
P4: First, A, B, C, D, F, G, I, K, L, R, T, Last
P5: First, A, B, C, D, F, G, I, K, M, N, R, T, Last
P6: First, A, B, C, D, F, G, I, K, M, O, P, R, T, Last
P7: First, A, B, C, D, F, G, I, K, M, O, Q, R, T, Last
P8: First, A, B, S, T, Last

Test Cases:

TC ID	Test Description	Input	Expected Output	Actual Output	Status
1	Testing for path P1	ME = 22 USP = 12 FS = 20 CN2 = 21 ST = 22 CD = 25	Fail	Fail	Pass
2	Testing for path P2	ME = 36 USP = 38 FS = 38 CN2 = 39 ST = 37 CD = 35	Grade C	Grade C	Pass
3	Testing for path P3	ME = 40 USP = 41 FS = 42 CN2 = 45 ST = 48 CD = 49	Grade C+	Grade C+	Pass
4	Testing for path P4	ME = 55 USP = 58 FS = 59 CN2 = 57 ST = 52 CD = 53	Grade B	Grade B	Pass

5	Testing for path P5	ME = 65 USP = 62 FS = 68 CN2 = 67 ST = 63 CD = 61	Grade B+	Grade B+	Pass
6	Testing for path P6	ME = 72 USP = 74 FS = 78 CN2 = 75 ST = 74 CD = 78	Grade A	Grade A	Pass
7	Testing for path P7	ME = 85 USP = 90 FS = 87 CN2 = 95 ST = 98 CD = 99	Grade A+	Grade A+	Pass
8	Testing for path P8	ME = -1 USP = 25 FS = 62 CN2 = 9 ST = 120 CD = 20	Invalid Input	Invalid Input	Pass

12.5 EXECUTION

Compile the program and enter inputs for subject marks, then depending on the average marks, the Grades are displayed and test cases are executed based on basis paths.

Test Report:

1. No of TC's Executed: 08
2. No of Defects Raised: 00
3. No of TC's Pass: 08
4. No of TC's Failed: 00