SOFTWARE TESTING LABORATORY

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, and 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. Discuss the test results.

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

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

1.1 REQUIREMENT SPECIFICATIONS

R1. The system should accept 3 positive integer numbers (a, b, c) which represents 3 sides of the triangle.

R2. Based on the input should determine if a triangle can be formed or not.

R3. If the requirement R2 is satisfied then the system should determine the type of the triangle, which can be

- Equilateral (i.e. all the three sides are equal)
- Isosceles (i.e Two sides are equal)
- Scalene (i.e All the three sides are unequal)

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

1.2 DESIGN

Algorithm:

```
Step 1: Input a, b & c i.e three integer values which represent three sides of
the triangle.
Step 2: if (a < (b + c)) and (b < (a + c)) and (c < (a + b) then
do Step 3
            else
            print not a triangle. do Step 6.
Step 3: if (a=b) and (b=c) then
      Print triangle formed is equilateral. do Step 6.
Step 4: if (a ≠ b) and (a ≠ c) and (b ≠ c) then
      Print triangle formed is scalene. do Step 6.
```

```
Step 5: Print triangle formed is Isosceles.
Step 6: stop
```

1.3 PROGRAM CODE:

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#include<process.h>
 int main()
 {
        int a, b, c;
        clrscr();
        printf("Enter three sides of the triangle");
        scanf("%d%d%d", &a, &b, &c);
        if((a > 10) || (b > 10) || (c > 10))
        {
                printf("Out of range");
                getch();
                exit(0);
        }
        if((a<b+c)&&(b<a+c)&&(c<a+b))
        {
                if((a==b)&&(b==c))
                {
                        printf("Equilateral triangle");
                }
                else if((a!=b)&&(a!=c)&&(b!=c))
                {
                         printf("Scalene triangle");
               }
               else
```

```
            {
                    printf("Isosceles triangle");
            }
    }
    else
        {
                  printf("triangle cannot be formed");
        }
       getch();
       return 0;
      }
```

1.4 TESTING

1. **Technique used: Boundary value analysis**
2. **Test Case design**

For BVA problem the test cases can be generation depends on the output and the constraints on the output. Here we least worried on the constraints on Input domain.

The Triangle problem takes 3 sides as input and checks it for validity, hence n = 3. Since BVA yields (4n + 1) test cases according to single fault assumption theory, hence we can say that the total number of test cases will be (4*3+1) =12+1=13.

The maximum limit of each side a, b, and c of the triangle is 10 units according to requirement R4. So a, b and c lies between

$1 \leq a \leq 10$

$1 \leq b \leq 10$

$1 \leq c \leq 10$

Equivalence classes for a:

E1: Values less than 1.
E2: Values in the range.
E3: Values greater than 10.

Equivalence classes for b:

E4: Values less than 1
E5: Values in the range.
E6: Values greater than 10.

Equivalence classes for c:

E7: Values less than 1.
E8: Values in the range.
E9: Values greater than 10.

From the above equivalence classes we can derive the following test cases using boundary value analysis approach.

TC Id	Test Case Description	Input Data			Expected Output	Actual Output	Status
		A	b	C			
1	For A input is not given	X	3	6	Not a Triangle		
2	For B input is not given	5	X	4	Not a Triangle		
3	For C input is not given	4	7	X	Not a Triangle		
4	Input of C is in negative(-)	5	5	-1	Not a Triangle		
5	Two sides are same one side is given different input	5	5	1	Isosceles		

6	All Sides of inputs are equal	5	5	5	Equilateral		
7	Two sides are same one side is given different input	5	5	9	Isosceles		
8	The input of C is out of range (i.e., range is <10)	5	5	10	Not a Triangle		
9	Two sides are same one side is given different input (i.e., A & C are 5, B=1)	5	1	5	Isosceles		
10	Two sides are same one side is given different input (i.e., A & C are 5, B=2)	5	2	5	Isosceles		
11	Two sides are same one side is given different input (i.e., A & C are 5, B=9)	5	9	5	Isosceles		
12	Two sides are same one side is given different input (i.e., A & C are 5, B=10 so, it isout of given range)	5	10	5	Not a Triangle		

13	Two sides are same one side is given different input (i.e., B & C are 5, A=1)	1	5	5	Isosceles		
14	Two sides are same one side is given different input (i.e., B & C are 5, A=2)	2	5	5	Isosceles		
15	Two sides are same one side is	9	5	5	Isosceles		
	given different input (i.e., B &						
16	Two sides are same one side is	10	5	5	Not a Triangle		

Table-1: Test case for Triangle Problem

1.5 EXECUTION:

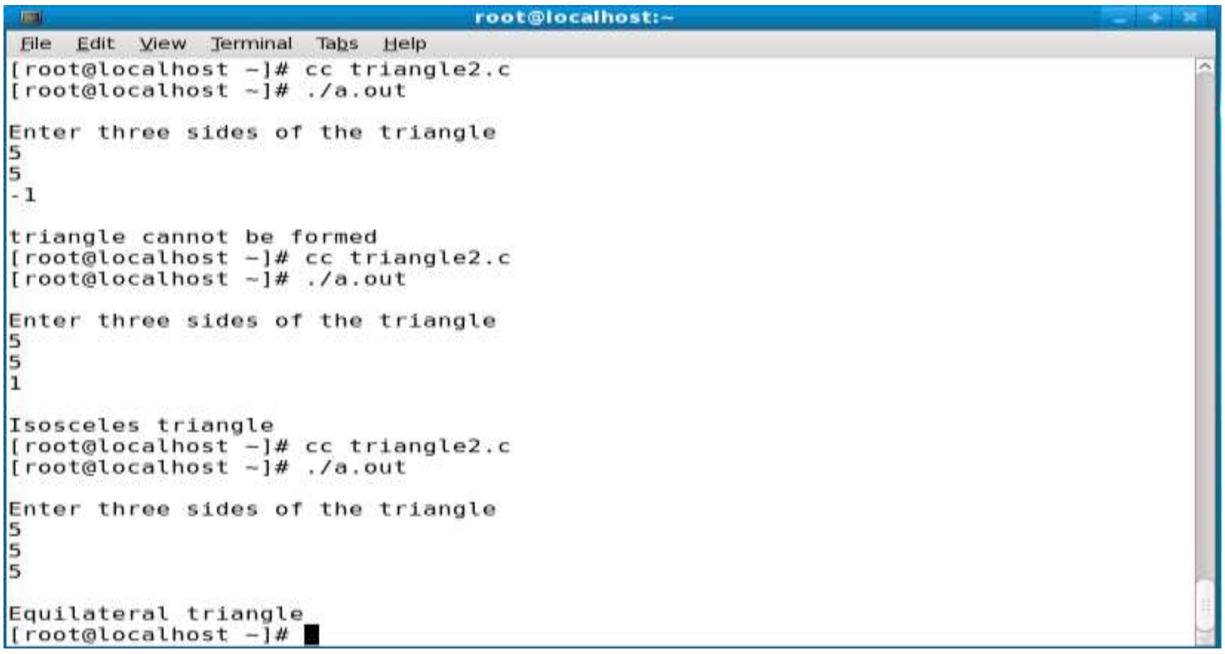
Execute the program and test the test cases in Table-1 against program and complete the table with for Actual output column and Status column.

Test Report:

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

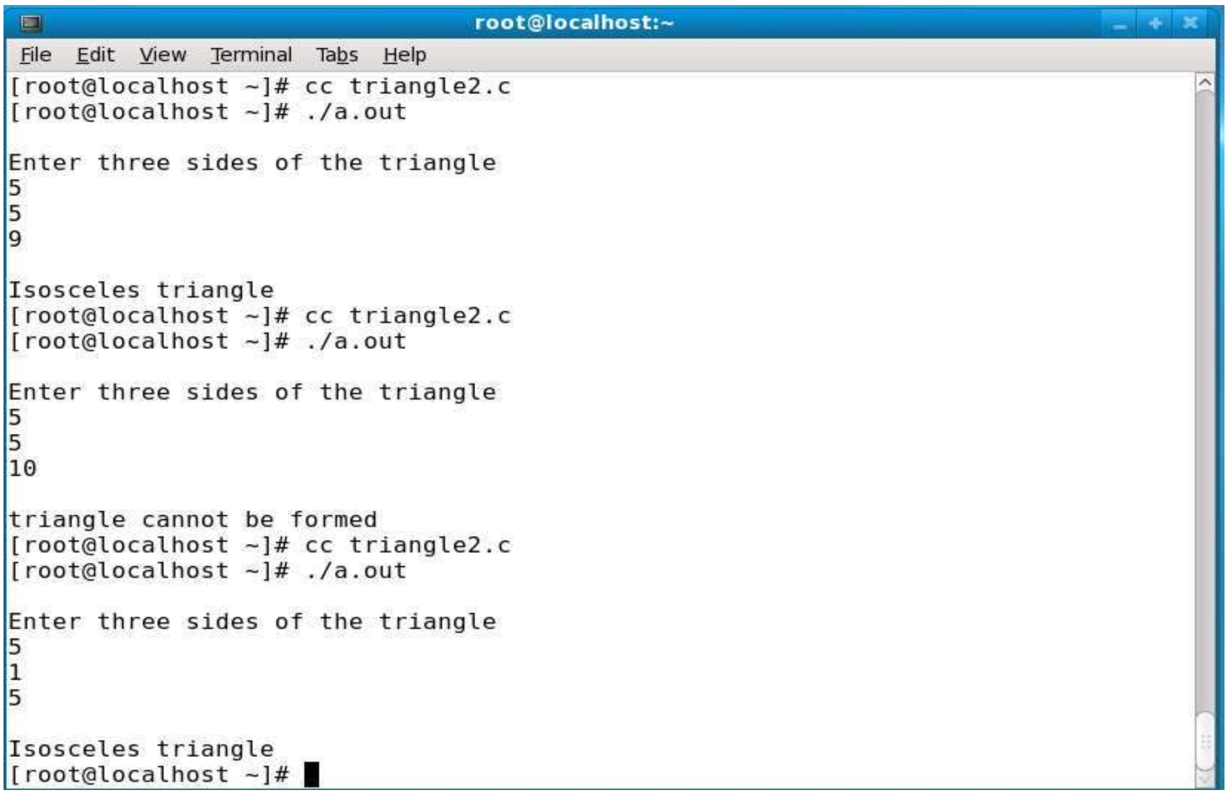
1.6 SNAPSHOTS:

1. Snapshot of Isosceles and Equilateral triangle and triangle can not be formed.



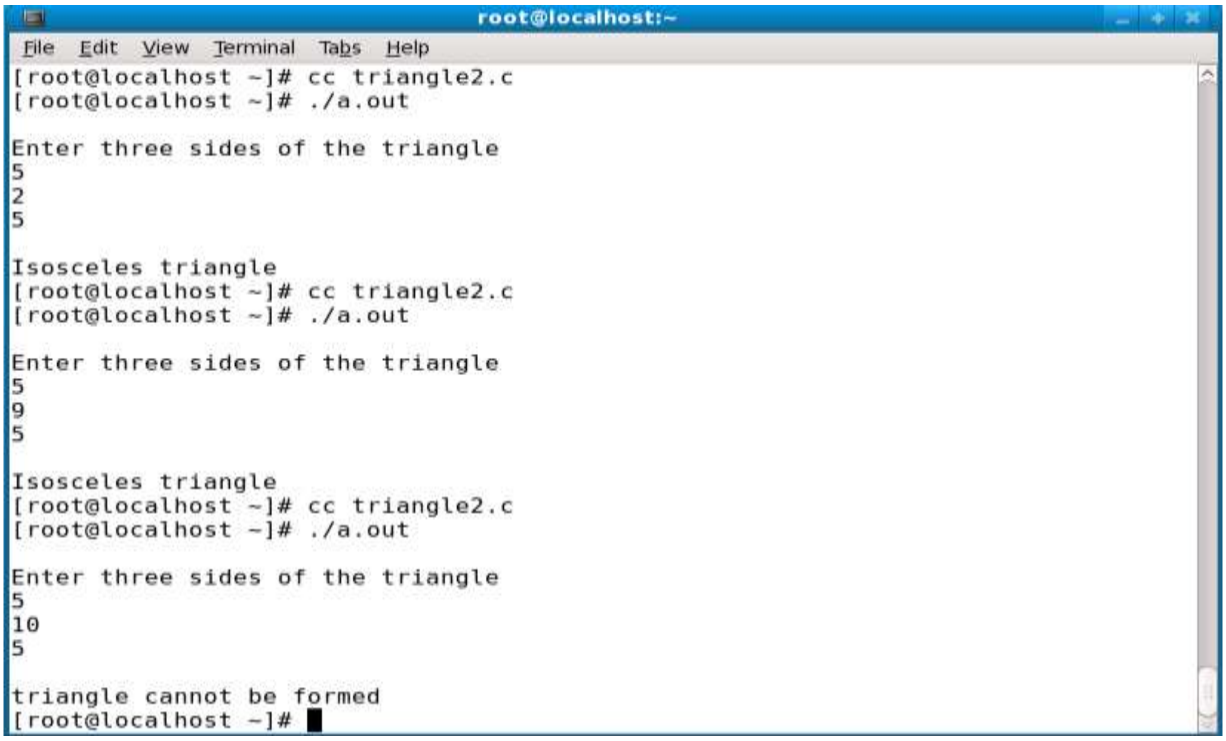
```
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
-1

triangle cannot be formed
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
1

Isosceles triangle
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
5

Equilateral triangle
[root@localhost ~]#
```

2. Snapshot for Isosceles and triangle cannot be formed



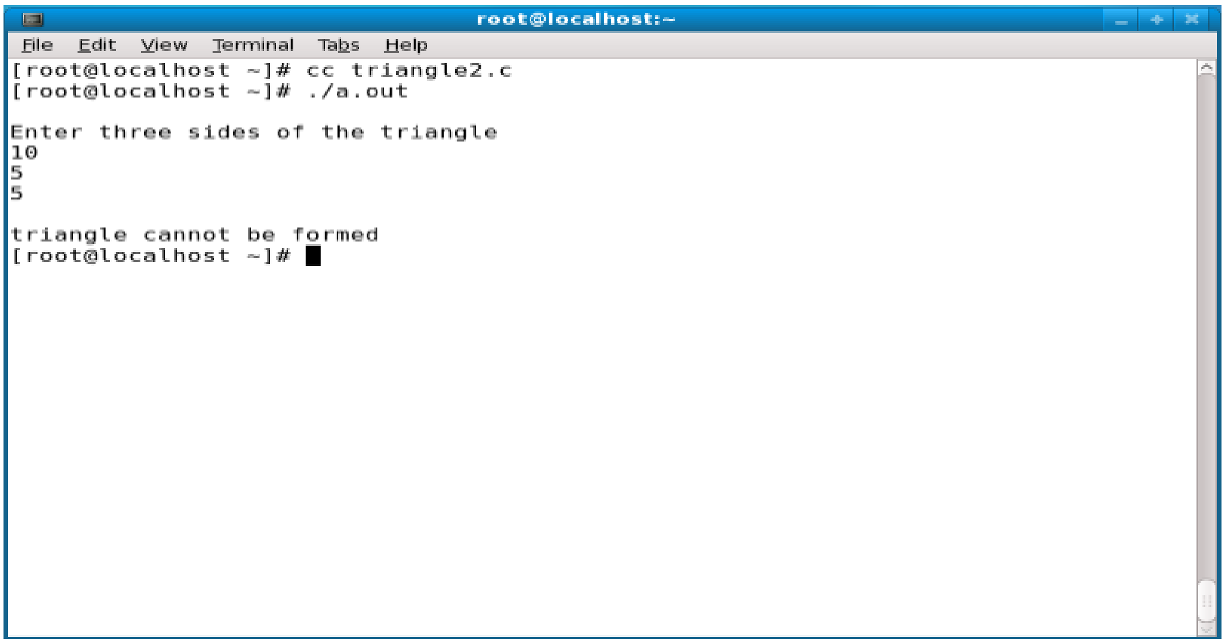
```
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
9

Isosceles triangle
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
10

triangle cannot be formed
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
1
5

Isosceles triangle
[root@localhost ~]#
```

3. Snapshot for Isosceles and triangle cannot be formed



```
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
2
5

Isosceles triangle
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
9
5

Isosceles triangle
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
10
5

triangle cannot be formed
[root@localhost ~]# █
```

4. Output screen for Triangle cannot be formed



```
[root@localhost ~]# cc triangle2.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
10
5
5

triangle cannot be formed
[root@localhost ~]# █
```

1.7 REFERENCES

1. Requirement Specification
2. Assumptions

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 REQUIREMENT SPECIFICATION

Problem Definition: The Commission Problem includes a salesperson in the former Arizona Territory sold rifle locks, stocks and barrels made by a gunsmith in Missouri. Cost includes,
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 sales person sent a telegram to the Missouri gunsmith with the number of locks, stocks and barrels sold in the town. At the end of the 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:

On sales up to(and including) $1000= 10% On the
sales up to(and includes) $1800= 15% On the sales
in excess of $1800= 20%
The commission program produces a monthly sales report that gave the total number of locks, stocks and barrels sold, the salesperson's total dollar sales and finally the commission

2.2 DESIGN

Algorithm

```
Step 1: Define lockPrice=45.0, stockPrice=30.0, barrelPrice=25.0
Step 2: Input locks
Step 3: while(locks!=-1) 'input device uses -1 to indicate end of data goto
            Step 12
Step 4:input (stocks, barrels)
Step 5: compute lockSales, stockSales, barrelSales and sales
Step 6: output("Total sales:" sales)
Step 7: if (sales > 1800.0)  goto Step 8 else goto  Step 9
Step 8: commission=0.10*1000.0; commission=commission+0.15*800.0;
        commission = commission + 0.20 * (sales-1800.0)
Step 9: if (sales > 1000.0) goto Step 10 else goto Step 11
Step10: commission=0.10* 1000.0; commission=commission + 0.15 * (sales-
1000.0)
Step 11: Output("Commission is $", commission) Step
12: exit
```

2.3 PROGRAM CODE:

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

        }
        printf("Enter the total number of stocks");
        scanf("%d",&stocks);
        if ((stocks <= 0) || (stocks > 80))
        {
                flag = 1;
        }
        printf("Enter the total number of barrelss");
```

```
    scanf("%d",&barrels);
    if ((barrels <= 0) || (barrels > 90))
    {
            flag = 1;
    }
    if (flag == 1)
    {
            printf("invalid input");
            getch();
            exit(0);
    }
    t_sales = (locks * 45) + (stocks * 30) + (barrels * 25);
    if (t_sales <= 1000)
    {
            commission = 0.10 * t_sales;
    }
    else if (t_sales < 1800)
    {
             commission = 0.10 * 1000;
             commission = commission + (0.15 * (t_sales - 1000));
     }
   else
   {
          commission = 0.10 * 1000;
          commission = commission + (0.15 * 800);
          commission = commission + (0.20 * (t_sales - 1800));
   }
  printf("The total sales is %d \n The commission is %f",t_sales,
  commission);
        getch();  return;
 }
```

2.4 TESTING

Technique used: Boundary value analysis

‘Boundary value analysis’ testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

BVA: Procedure

1. Partition the input domain using unidimensional partitioning. This leads to as many partitions as there are input variables. Alternately, a single partition of an input domain can be created using multidimensional partitioning. We will generate several sub-domains in this Step.
2. Identify the boundaries for each partition. Boundaries may also be identified using special relationships amongst the inputs.
3. Select test data such that each boundary value occurs in at least one test input.
4. BVA: Example: Create equivalence classes

Assuming that an item code must be in the range 99...999 and quantity in the range 1...100,

Equivalence classes for code:

E1: Values less than 99.

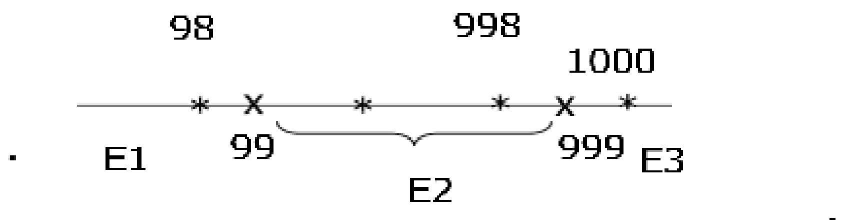
E2: Values in the range.

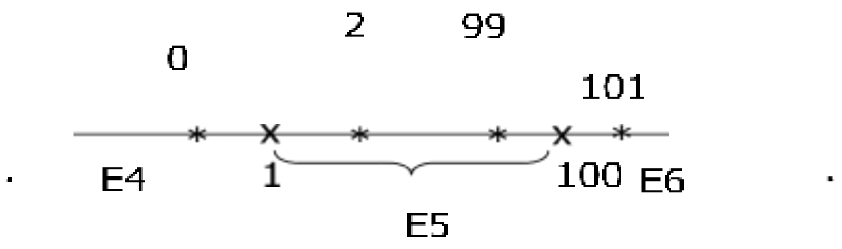
E3: Values greater than 999.

Equivalence classes for qty:

E4:Value less than 1.
E5:Value in the range.
E6: Value greater than 100.

BVA: Example: Identify boundaries





Equivalence classes and boundaries for find Price. Boundaries are indicated with an x. Points near the boundary are marked *.

Test Case design

The Commission Problem takes locks, stocks and barrels as input and checks it for validity. If it is valid, it returns the commission as its output. Here we have three inputs for the program, hence n = 3.

Since BVA yields (4n + 1) test cases according to single fault assumption theory, hence we can say that the total number of test cases will be (4*3+1) =12+1=13.

The boundary value test case can be generated over an output by using fallowing constraints and these constraints are generated over commission:

C1: Sales up to(and including) $1000= 10% commission
C2: Sales up to(and includes) $1800= 15% commission
C3: Sales in excess of $1800= 20% commission

Here from these constraints we can extract the test cases using the values of Locks, Stocks, and Barrels sold in month. The boundary values for commission are 10%, 15% and 20%.

Equivalence classes for 10% Commission:

E1: Sales less than
1000. E2: Sales equals
to 1000.

Equivalence classes for 15% Commission:

E3: Sales greater than 1000 and less than 1800.
E4: Sales equals to 1800

Equivalence classes for 20% Commission:

E5: Sales greater then 1800

From the above equivalence classes we can derive the following test cases using boundary value analysis approach.

TC Id	Test Case Description	Input Data			Sales	Expected Output(Commission)	Actual Output	Status
		Locks	Stocks	Barrels				
1	Input test cases for Locks=1, Stocks=1, Barrels=1	1	1	1	100	10		
2	Input test cases for Locks=1, Stocks=1, Barrels=2	1	1	2	125	12.5		
3	Input test cases for Locks=1, Stocks=2, Barrels=1	1	2	1	130	13		
4	Input test cases for Locks=2, Stocks=1, Barrels=1	2	1	1	145	14.5		
5	Input test cases for Locks=5, Stocks=5, Barrels=5	5	5	5	500	50		
6	Input test cases for Locks=10, Stocks=10,	10	10	9	975	97.5		

	Barrels=9							
7	Input test cases for Locks=10, Stocks=9, Barrels=10	10	9	10	970	97		
8	Input test cases for Locks=9, Stocks=10, Barrels=10	9	10	10	955	95.5		
9	Input test cases for Locks=10, Stocks=10, Barrels=10	10	10	10	1000	100		
10	Input test cases for Locks=10, Stocks=10, Barrels=11	10	10	11	1025	103.75		
11	Input test cases for Locks=10, Stocks=11, Barrels=10	10	11	10	1030	104.5		
12	Input test cases for Locks=11, Stocks=10, Barrels=10	11	10	10	1045	106.75		
13	Input test cases for Locks=14, Stocks=14, Barrels=13	14	14	13	1400	160		
14	Input test cases for Locks=18, Stocks=18, Barrels=17	18	18	17	1775	216.25		
15	Input test cases for Locks=18, Stocks=17, Barrels=18	18	17	18	1770	215.5		
16	Input test cases for Locks=17, Stocks=18,	17	18	18	1755	213.25		

	Barrels=18							
17	Input test cases for Locks=18, Stocks=18, Barrels=18	18	18	18	1800	220		
18	Input test cases for Locks=18, Stocks=18, Barrels=19	18	18	19	1825	225		
19	Input test cases for Locks=18, Stocks=19, Barrels=18	18	19	18	1830	226		
20	Input test cases for Locks=19, Stocks=18, Barrels=18	19	18	18	1845	229		
21	Input test cases for Locks=48, Stocks=48, Barrels=48	48	48	48	4800	820		

Table-1 BVA Test case for commission problem.

This is how we can apply BVA technique to create test cases for our Commission Problem.

2.5 EXECUTIONS

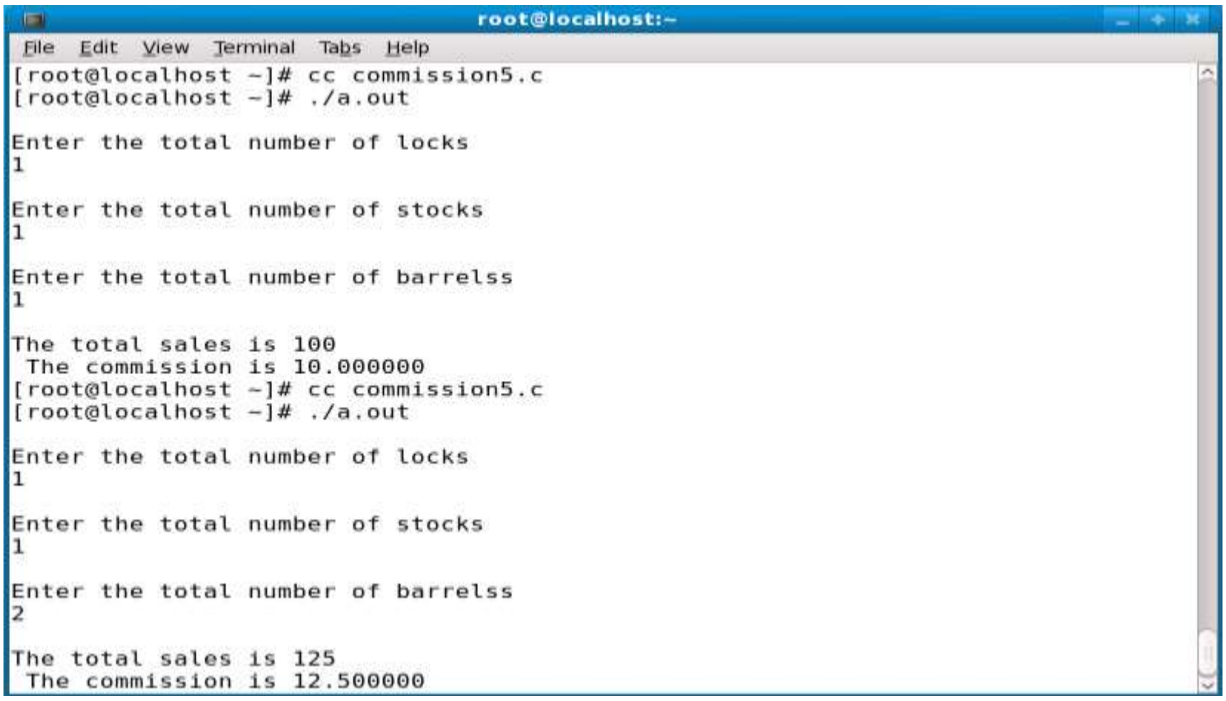
Execute the program and test the test cases in Table-1 against program and complete the table with for Actual output column and Status column

TEST REPORT:

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

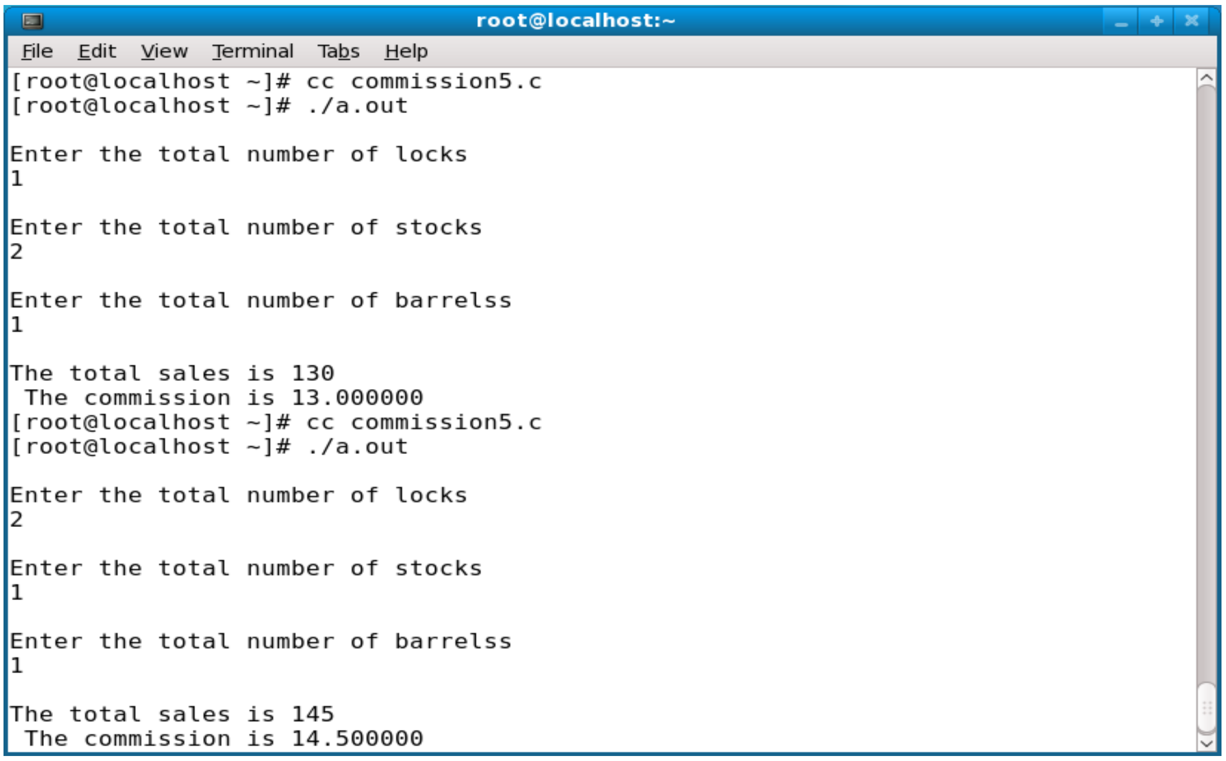
2.6 SNAPSHOTS:

1. Snapshot for valid inputs



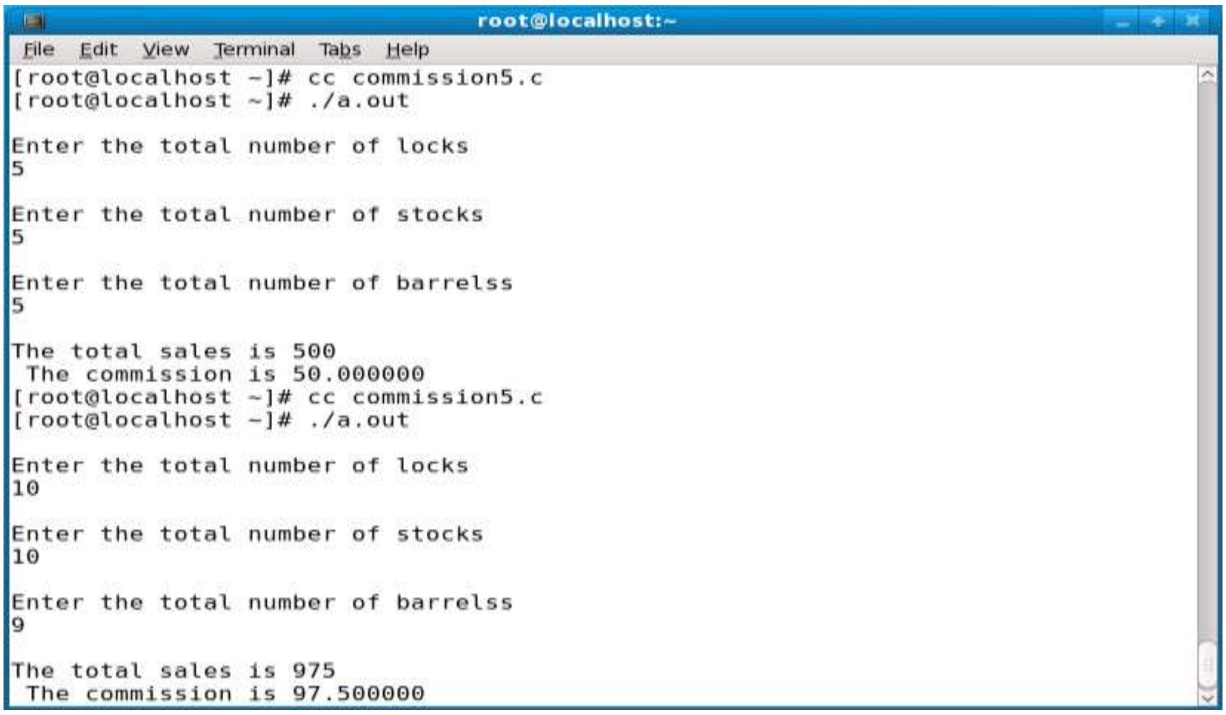
```
[root@localhost ~]# cc commission5.c
[root@localhost ~]# ./a.out

Enter the total number of locks
1

Enter the total number of stocks
1

Enter the total number of barrelss
1

The total sales is 100
 The commission is 10.000000
[root@localhost ~]# cc commission5.c
[root@localhost ~]# ./a.out

Enter the total number of locks
1

Enter the total number of stocks
1

Enter the total number of barrelss
2

The total sales is 125
 The commission is 12.500000
```

2. Snapshots when the two inputs are same

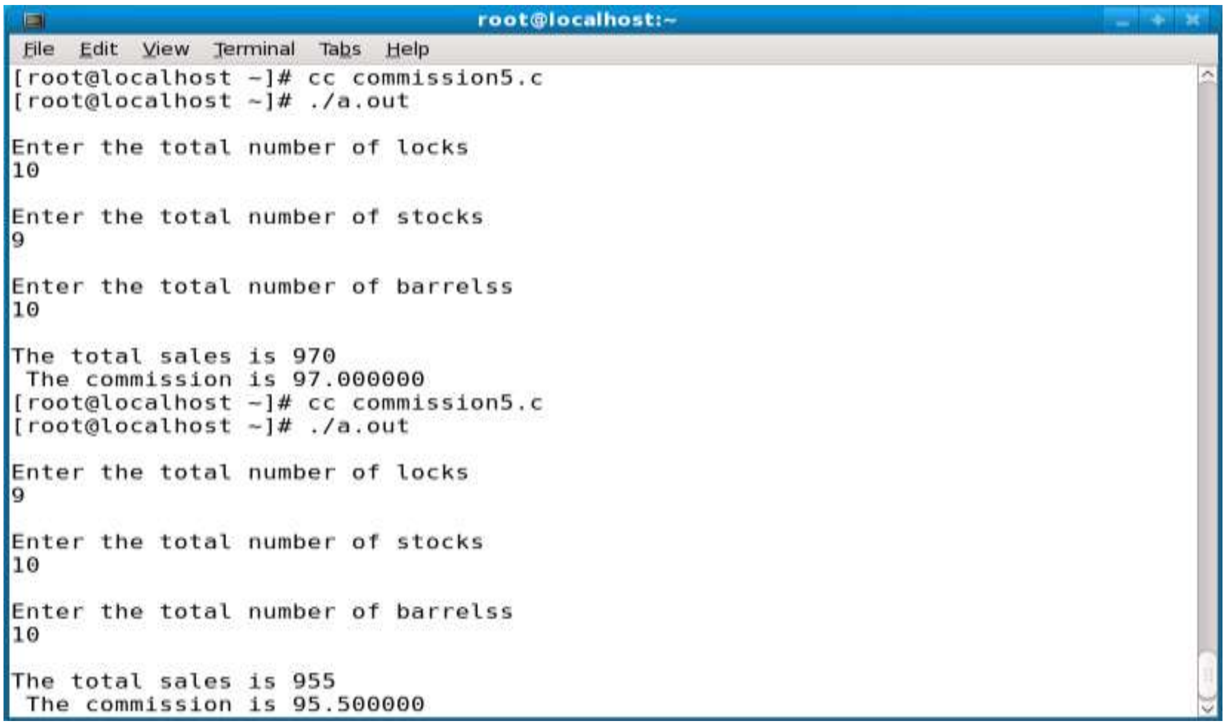


```
[root@localhost ~]# cc commission5.c
[root@localhost ~]# ./a.out

Enter the total number of locks
1

Enter the total number of stocks
2

Enter the total number of barrelss
1

The total sales is 130
 The commission is 13.000000
[root@localhost ~]# cc commission5.c
[root@localhost ~]# ./a.out

Enter the total number of locks
2

Enter the total number of stocks
1

Enter the total number of barrelss
1

The total sales is 145
 The commission is 14.500000
```

3. Snapshots when the two inputs and all the inputs are same



```
[root@localhost ~]# cc commission5.c
[root@localhost ~]# ./a.out

Enter the total number of locks
5

Enter the total number of stocks
5

Enter the total number of barrelss
5

The total sales is 500
 The commission is 50.000000
[root@localhost ~]# cc commission5.c
[root@localhost ~]# ./a.out

Enter the total number of locks
10

Enter the total number of stocks
10

Enter the total number of barrelss
9

The total sales is 975
 The commission is 97.500000
```



```
[root@localhost ~]# cc commission5.c
[root@localhost ~]# ./a.out

Enter the total number of locks
10

Enter the total number of stocks
9

Enter the total number of barrelss
10

The total sales is 970
 The commission is 97.000000
[root@localhost ~]# cc commission5.c
[root@localhost ~]# ./a.out

Enter the total number of locks
9

Enter the total number of stocks
10

Enter the total number of barrelss
10

The total sales is 955
 The commission is 95.500000
```

2.7 REFERENCES

1. Requirement Specification
2. Assumptions

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 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.

3.2 DESIGN

Algorithm

```
Step 1: Input date in format DD.MM.YYYY
Step 2: if MM is 01, 03, 05,07,08,10 do Step 3 else Step 6
Step 3:if DD < 31 then do Step 4 else if DD=31 do Step 5 else
output(Invalid Date);
Step 4: tomorrowday=DD+1 goto Step 18
Step 5: tomorrowday=1; tomorrowmonth=month + 1 goto Step 18
Step 6: if MM is 04, 06, 09, 11 do Step 7
Step 7: if DD<30 then do Step 4 else if DD=30 do Step 5 else
output(Invalid Date);
Step 8: if MM is 12
Step 9: if DD<31 then Step 4 else Step 10
Step 10: tomorrowday=1, tommorowmonth=1, tommorowyear=YYYY+1;
goto Step 18
Step 11: if MM is 2
Step 12: if DD<28 do Step 4 else do Step 13
Step 13: if DD=28 & YYYY is a leap do Step14 else Step 15
Step 14: tommorowday=29 goto Step 18
Step 15: tommorowday=1, tomorrowmonth=3, goto Step18;
Step 16: if DD=29 then do Step 15 else Step 17
Step 17: output("Cannot have feb", DD); Step19
Step 18: output(tomorrowday, tomorrowmonth, tomorrowyear);
Step 19: exit
```

3.3 PROGRAM CODE:

```
#include<stdio.h>
#include<conio.h>
main( )
{
 int month[12]={31,28,31,30,31,30,31,31,30,31,30,31};
 int d,m,y,nd,nm,ny,ndays;
 clrscr( );
printf("enter the date,month,year");
scanf("%d%d%d",&d,&m,&y);
ndays=month[m-1];
if(y<=1812 && y>2012)
{
        printf("Invalid Input Year");
        exit(0);
}
if(d<=0 || d>ndays)
{

        printf("Invalid Input Day");
        exit(0);
}
if(m<1 && m>12)
{
        printf("Invalid Input Month");
        exit(0);
}
if(m==2)
{
        if(y%100==0)
        {
          if(y%400==0)
          ndays=29;
        }
      else
      if(y%4==0)
      ndays=29;
}
nd=d+1;
nm=m;
ny=y;
if(nd>ndays)
{
        nd=1;
        nm++;
}
```

```
if(nm>12)
{
        nm=1;
        ny++;
}
printf("\n Given date is %d:%d:%d",d,m,y); printf("\n
Next day’s date is %d:%d:%d",nd,nm,ny); getch( );
}
```

3.4 TESTING

Technique used: Boundary value analysis

‘Boundary value analysis’ testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

BVA: Procedure

1. Partition the input domain using unidimensional partitioning. This leads to as many partitions as there are input variables. Alternately, a single partition of an input domain can be created using multidimensional partition. We will generate several sub-domains in this Step.
2. Identify the boundaries for each partition. Boundaries may also be identified using special relationships amongst the inputs.
3. Select test data such that each boundary value occurs in at least one test input.

BVA: Example: Create equivalence classes

Assuming that an item code must be in the range 99...999 and quantity in the range 1...100,

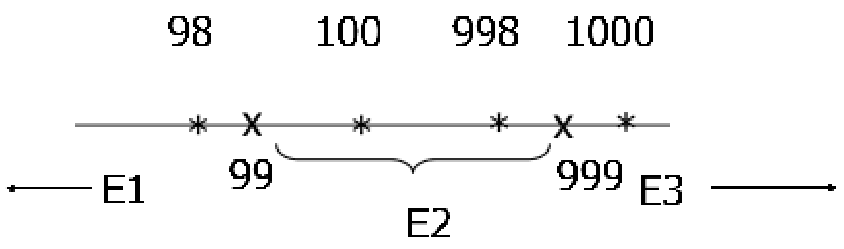
Equivalence classes for code:

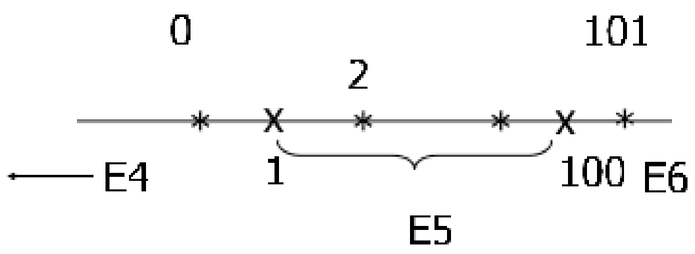
E1: Values less than 99.
E2: Values in the range.
E3: Values greater than 999.

Equivalence classes for qty:

E4: Values less than 1.
E5: Values in the range.
E6: Values greater than 100.

BVA: Example: Identify boundaries





Equivalence classes and boundaries for find Price. Boundaries are indicated with an x. Points near the boundary are marked *.

Test Case design

The Next Date program takes date as input and checks it for validity. If it is valid, it returns the next date as its output. Here we have three inputs for the program, hence n = 3.

Since BVA yields (4n + 1) test cases according to single fault assumption theory, hence we can say that the total number of test cases will be (4*3+1) =12+1=13.

The boundary value test cases can be generated by using following constraints

C1: $1 \leq MM \leq 12$
C2: $1 \leq DD \leq 31$
C3: $1812 \leq YYYY \leq 2012$.

Here from these constraints we can extract the test cases using the values of MM, DD, and YYYY. The following equivalence classes can be generated for each variable.

Equivalence classes for MM:
E1: Values less than 1.
E2: Values in the range.
E3: Values greater than 12.
Equivalence classes for DD:
E4: Values less than 1.
E5: Values in the range.
E6: Values greater than 31.
Equivalence classes for YYYY:
E7: Values less than 1812.
E8: Values in the range.
E9: Values greater than 2012.

From the above equivalence classes we can derive the following test cases using boundary value analysis approach.

TC Id	**Test Case Description**	**Input Data**			**Expected Output**	**Actual Output**	**Status**
		MM	**DD**	**YYYY**			
1	Testing for Invalid months with character is typed	Aa	15	1900	Invalid Input Month		
2	Testing for Invalid Day with character is typed	06	Dd	1901	Invalid Input Day		
3	Testing for Invalid Year with character is typed	06	15	196y	Invalid Input Year		
4	Testing for Invalid Day, day with 00	03	00	2000	Invalid Input Day		
5	Testing for Valid input changing the day within the month.	03	30	2000	03/31/2000		
6	Testing for Valid input changing the day within the month.	03	02	2000	03/03/2000		
7	Testing for Invalid Day, day with 32	03	32	2000	Invalid Input Day		
8	Testing for Invalid Day, month with 00	00	15	2000	Invalid Input Month		
9	Testing for Valid input changing the day within the month. MM=11 DD=15	11	15	2000	11/16/2000		
10	Testing for Valid input changing the day within the month. MM=02 DD=15	02	15	2000	02/16/2000		

11	Testing for Invalid Month, month with 13	13	15	2000	Invalid Input Month		
12	Testing for Invalid year, year should >=1812	03	15	1811	Invalid Input Year		
13	Testing for Valid input changing the day within the month. MM=03 DD=15 YYYY=2011	03	15	2011	03/16/2011		
14	Testing for Valid input changing the day within the month.	03	15	1813	03/16/1813		
15	Testing for Invalid year, year should <=2012	03	15	2013	Invalid Input Year		

Table-1: Test case for Next Date Problem

This is how we can apply BA technique to create test cases for our Next Date Problem.

3.5 EXECUTIONS

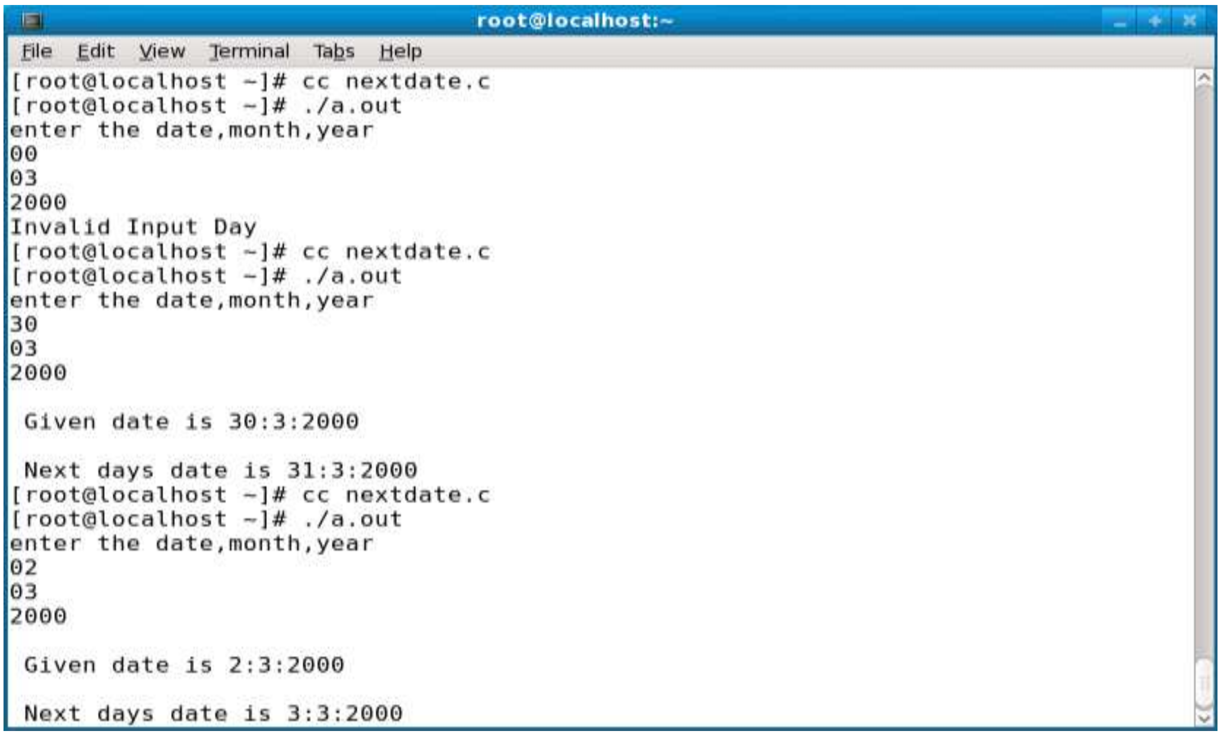
Execute the program and test the test cases in Table-1 against program and complete the table with for Actual output column and Status column

Test Report:

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

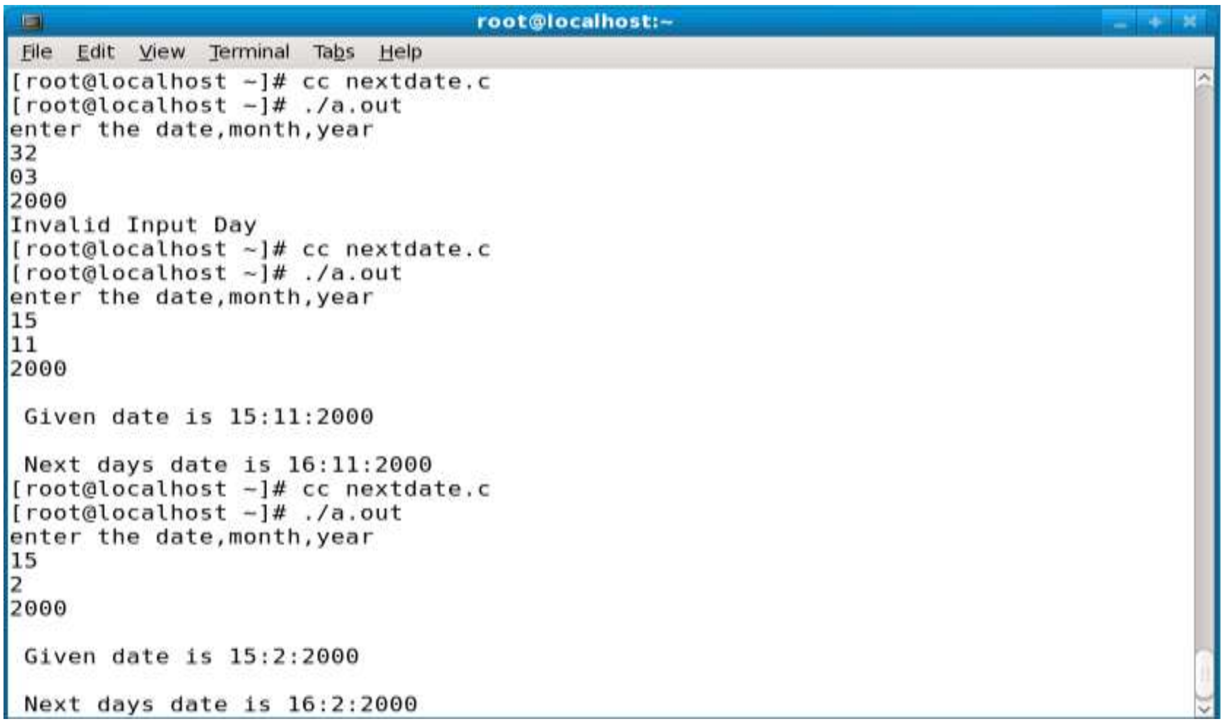
3.6 SNAPSHOTS:

1. Snapshot for Invalid Input day and next date



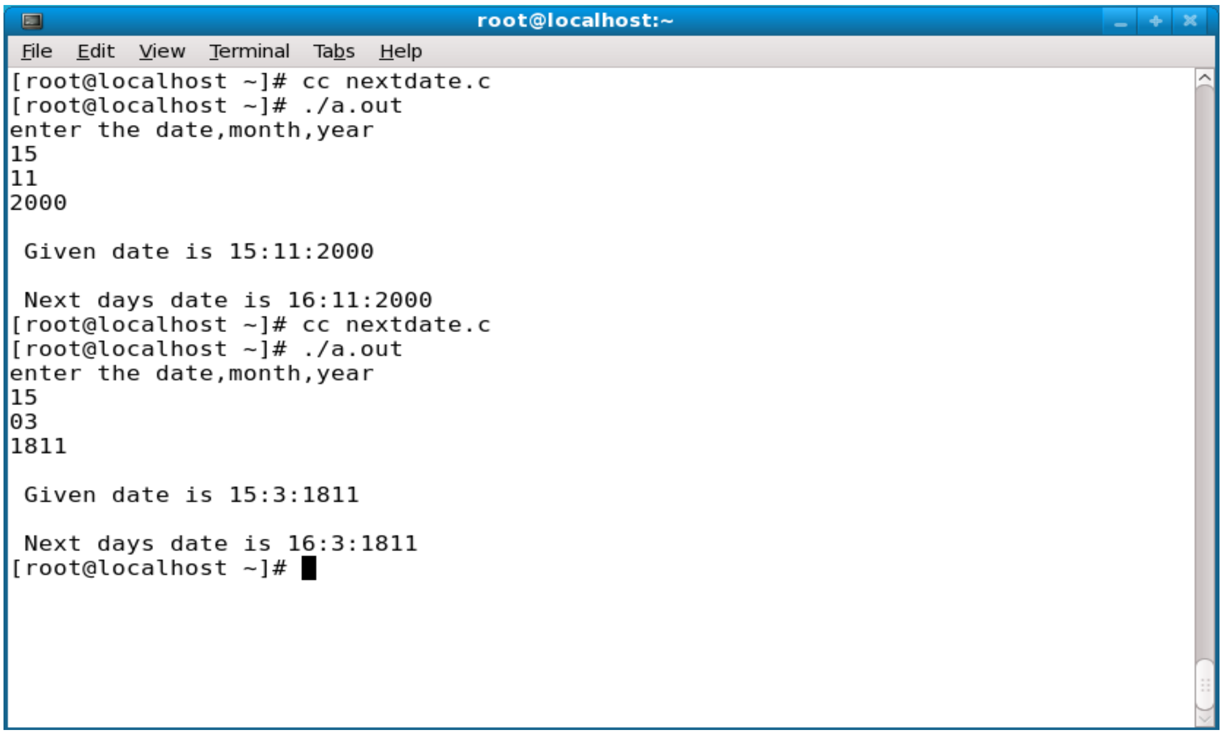
```
[root@localhost ~]# cc nextdate.c
[root@localhost ~]# ./a.out
enter the date,month,year
00
03
2000
Invalid Input Day
[root@localhost ~]# cc nextdate.c
[root@localhost ~]# ./a.out
enter the date,month,year
30
03
2000

 Given date is 30:3:2000

 Next days date is 31:3:2000
[root@localhost ~]# cc nextdate.c
[root@localhost ~]# ./a.out
enter the date,month,year
02
03
2000

 Given date is 2:3:2000

 Next days date is 3:3:2000
```

2. Snapshot to show the invalid day when the DD=32



```
[root@localhost ~]# cc nextdate.c
[root@localhost ~]# ./a.out
enter the date,month,year
32
03
2000
Invalid Input Day
[root@localhost ~]# cc nextdate.c
[root@localhost ~]# ./a.out
enter the date,month,year
15
11
2000

 Given date is 15:11:2000

 Next days date is 16:11:2000
[root@localhost ~]# cc nextdate.c
[root@localhost ~]# ./a.out
enter the date,month,year
15
2
2000

 Given date is 15:2:2000

 Next days date is 16:2:2000
```

3. Valid Output:



```
[root@localhost ~]# cc nextdate.c
[root@localhost ~]# ./a.out
enter the date,month,year
15
11
2000

 Given date is 15:11:2000

 Next days date is 16:11:2000
[root@localhost ~]# cc nextdate.c
[root@localhost ~]# ./a.out
enter the date,month,year
15
03
1811

 Given date is 15:3:1811

 Next days date is 16:3:1811
[root@localhost ~]# 
```

3.7 REFERENCES:

1. Requirement Specification
2. Assumptions

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 REQUIREMENT SPECIFICATION

R1. The system should accept 3 positive integer numbers (a, b, c) which represents 3 sides of the triangle.
R2. Based on the input should determine if a triangle can be formed or not. R3. If the requirement R2 is satisfied then the system should determine the type of the triangle, which can be

- Equilateral (i.e. all the three sides are equal)
- Isosceles (i.e. two sides are equal)
- Scalene (i.e. All the three sides are unequal)

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

4.2 DESIGN

Form the given requirements we can draw the following conditions:

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 the triangle, if any one of the three conditions C1, C2 and C3 are not satisfied then triangle cannot be constructed. So only when C1, C2 and C3 are true the triangle can be formed, then depending on conditions C4, C5 and C6 we can decide what type of triangle will be formed(i.e requirements R3)

Algorithm:

```
Step 1: Input a, b & c i.e three integer values which represent three sides of the
triangle.
Step 2: if (a < (b + c)) and (b < (a + c)) and (c < (a + b) then do
            Step 3
            else
            print not a triangle. do Step 6.
Step 3: if (a=b) and (b=c) then
        Print triangle formed is equilateral. do Step 6.
Step 4: if (a ≠ b) and (a ≠ c) and (b ≠ c) then
        Print triangle formed is scalene. do Step 6.
Step 5: Print triangle formed is Isosceles.
Step 6: stop
```

4.3 PROGRAM CODE

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#include<process.h>
 int main()
 {
        int a, b, c;
        clrscr();
        printf("Enter three sides of the triangle");
        scanf("%d%d%d", &a, &b, &c);
        if((a > 10) || (b > 10) || (c > 10))
        {
                printf("Out of range");
                getch();
                exit(0);
        }
                if((a<b+c)&&(b<a+c)&&(c<a+b))
```

```
            {
                    if((a==b)&&(b==c))
            {
                printf("Equilateral triangle");
            }
            else if((a!=b)&&(a!=c)&&(b!=c))
            {

            printf("Scalene triangle");

            }
          else
              printf("Isosceles triangle");
        }
     else
     {
     printf("triangle cannot be formed");
     } getch(); return 0;
}
```

4.4 TESTING

1. **Technique used: Equivalence class partitioning**
2. **Test Case design**

Equivalence class partitioning technique focus on the Input domain, we can obtain a richer set of test cases. What are some of the possibilities for the three integers, a, b, and c? They can all be equal, exactly one pair can be equal.

The maximum limit of each side a, b, and c of the triangle is 10 units according to requirement R4. So a, b and c lies between

$1 \leq a \leq 10$

$1 \leq b \leq 10$

$1 \leq c \leq 10$

First Attempt

Weak normal equivalence class: In the problem statement, we 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:

R_1= {<a,b,c>: the triangle with sides a, b, and c is equilateral}
R_2= {<a,b,c>: the triangle with sides a, b, and c is isosceles}
R_3= {<a,b,c>: the triangle with sides a, b, and c is scalene}
R_4= {<a,b,c>: sides a, b, and c do not form a triangle}

Four weak normal equivalence class test cases, chosen arbitrarily from each class, and invalid values for weak robust equivalence class test cases are as follows.

TC Id	**Test Case Description**	**Input Data**			**Expected Output**	**Actual Output**	**Status**
		a	**b**	**c**			
1	WN1	5	5	5	Equilateral		
2	WN2	2	2	3	Isosceles		
3	WN3	3	4	5	Scalene		
4	WN4	4	1	2	Not a Triangle		
5	WR1	-1	5	5	Value of a is not in the range of permitted values		
6	WR2	5	-1	5	Value of b is not in the range of permitted values		
7	WR3	5	5	-1	Value of c is not in the range of permitted values		

8	WR4	11	5	5	Value of a is not in the range of permitted values		
9	WR5	5	11	5	Value of b is not in the range of permitted values		
10	WR6	5	5	11	Value of c is not in the range of permitted values		

Table-1: Weak Normal and Weak Robust Test case for Triangle Problem

Second attempt

The strong normal equivalence class test cases can be generated by using following possibilities:

D1 = {<a, b, c>: a=b=c}

D2 = {<a, b, c>: a=b, a≠ c}

D3= {<a, b, c>: a=c, a≠ b}

D4 = {<a, b, c>: b=c, a≠ b}

D5 = {<a, b, c>: a≠ b, a≠ c, b≠ c}

D6 = {<a, b, c>: a≥ b+ c}

D7 = {<a, b, c>: b≥ a+ c}

D8 = {<a, b, c>: c≥ a+ b}

TC Id	**Test Case**	**Input Data**			**Expected Output**	**Actual Output**	**Status**
		a	**b**	**c**			
1	SR1	-1	5	5	Value of a is not in the range of permitted values		
2	SR 2	5	-1	5	Value of b is not in the range of permitted values		

3	SR3	5	5	-1	Value of c is not in the range of permitted values		
5	SR5	5	-1	-1	Value of b, c is not in the range of permitted values		
6	SR6	-1	5	-1	Value of a, c is not in the range of permitted values		
7	SR7	-1	-1	-1	Value of a, b, c is not in the range of permitted values		

Table-2: Strong Robust Test case for Triangle Problem

4.5 EXECUTION:

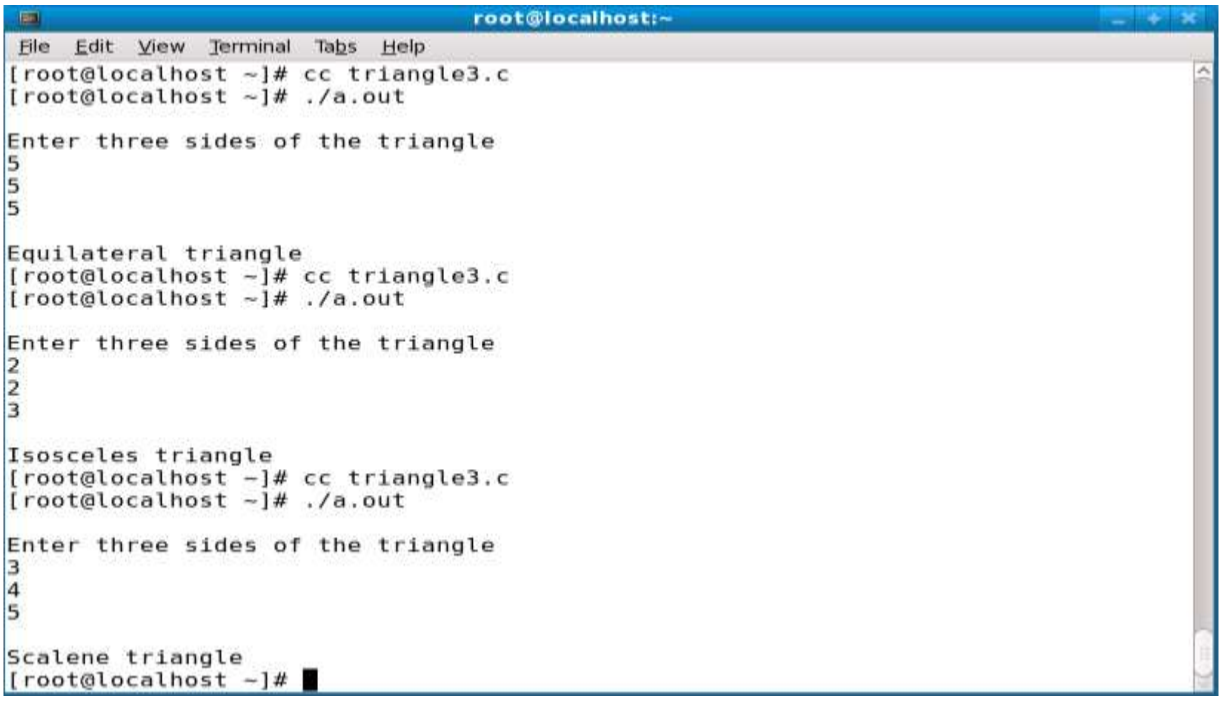
Execute the program and test the test cases in Table-1 and Table-2 against program and complete the table with for Actual output column and Status column

Test Report:

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

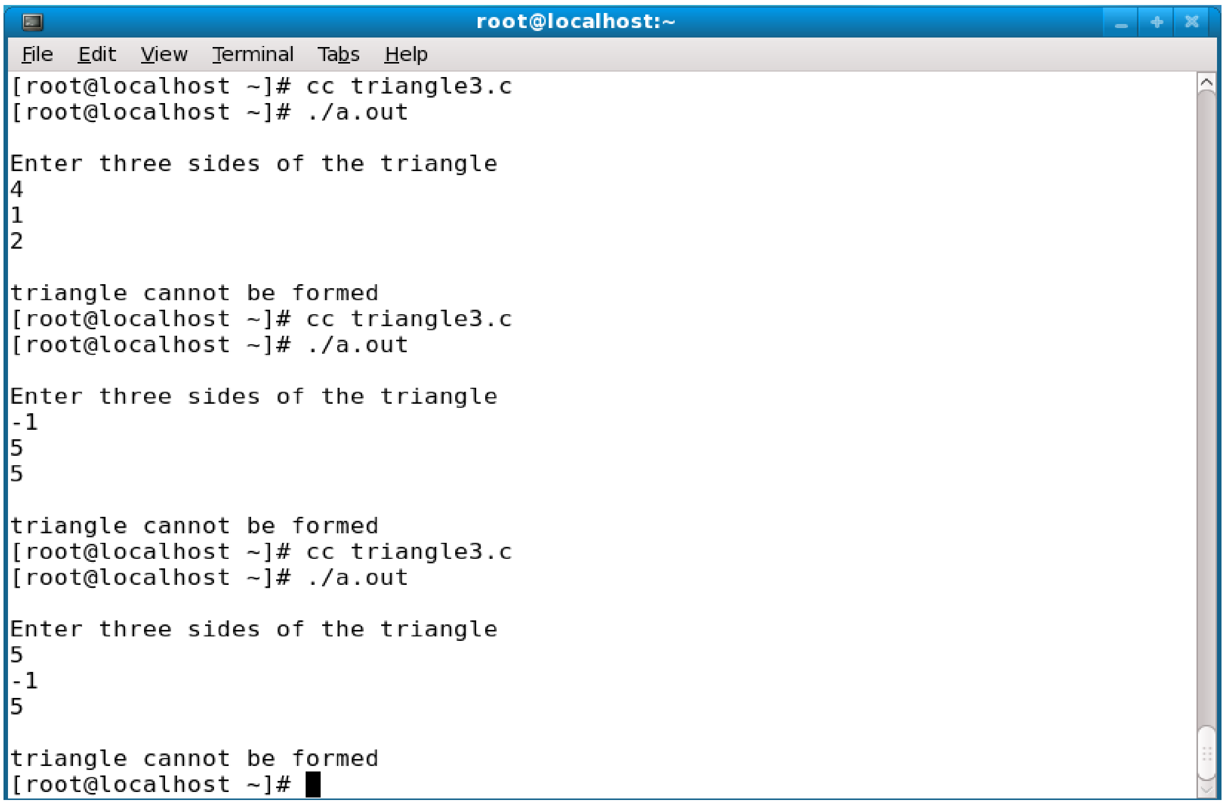
4.6 SNAPSHOTS:

1. Snapshot of Equilateral. Isosceles and scalene triangle.



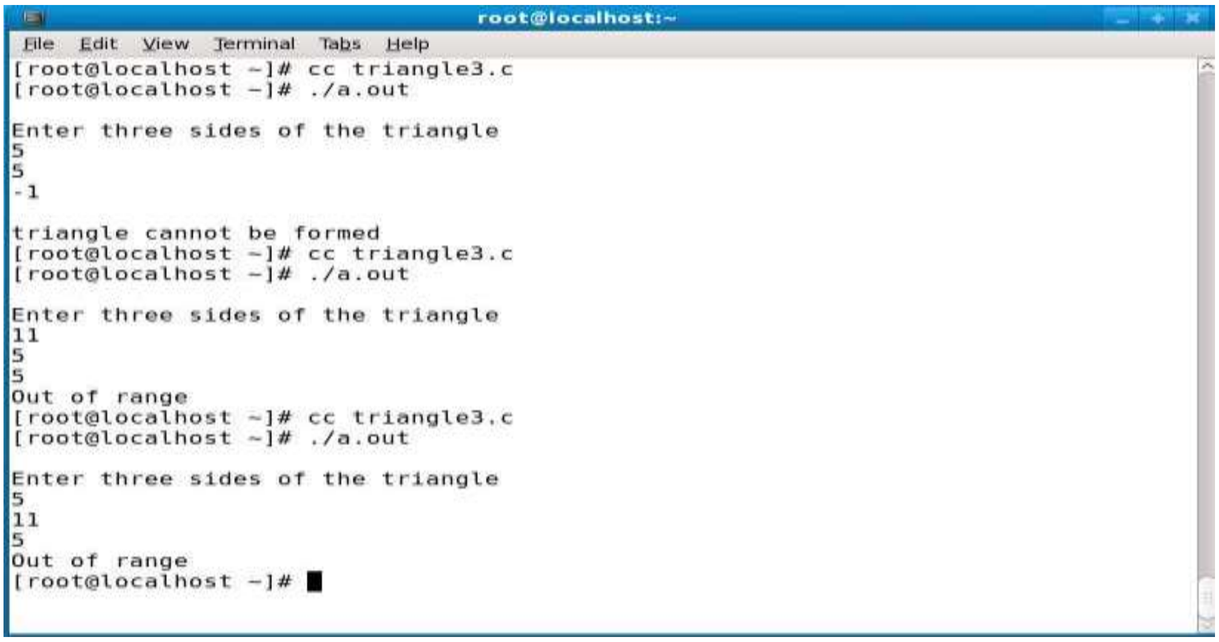
```
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
5

Equilateral triangle
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
2
2
3

Isosceles triangle
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
3
4
5

Scalene triangle
[root@localhost ~]#
```

2. Snapshot for Triangle cannot be formed

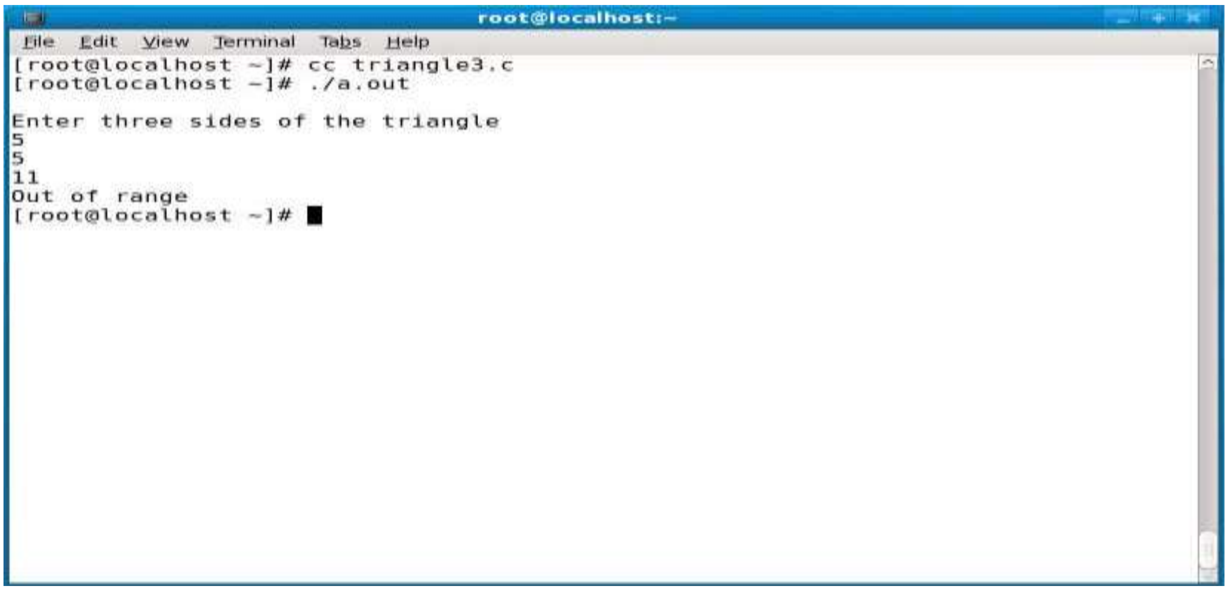


```
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
4
1
2

triangle cannot be formed
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
-1
5
5

triangle cannot be formed
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
-1
5

triangle cannot be formed
[root@localhost ~]#
```

3. Snapshot for the given range is Out of range and Triangle cannot be formed.



```
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
-1

triangle cannot be formed
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
11
5
5
Out of range
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
11
5
Out of range
[root@localhost ~]# █
```



```
[root@localhost ~]# cc triangle3.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
11
Out of range
[root@localhost ~]# █
```

4.7 REFERENCES

1. Requirement Specification
2. Assumptions

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 REQUIREMENT SPECIFICATION

Problem Definition: The Commission Problem includes a salesperson in the former Arizona Territory sold rifle locks, stocks and barrels made by a gunsmith in Missouri. Cost includes

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 sales person sent a telegram to the Missouri gunsmith with the number of locks, stocks and barrels sold in the town. At the end of the 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:

On sales up to(and including) $1000= 10% On
the sales up to(and includes) $1800= 15% On the
sales in excess of $1800= 20%
The commission program produces a monthly sales report that gave the total number of locks, stocks and barrels sold, the salesperson's total dollar sales and finally the commission.

5.2 DESIGN

Algorithm:

```
Step 1: Define lockPrice=45.0, stockPrice=30.0, barrelPrice=25.0
Step2: Input locks
```

```
Step3: while(locks!=-1) ‘input device uses -1 to indicate end of data goto
Step 12
Step 4:input (stocks, barrels)
 Step 5: compute lockSales, stockSales, barrelSales and sales
Step 6: output(“Total sales:” sales)
Step 7: if (sales > 1800.0)  goto Step 8 else goto Step 9
Step 8: commission=0.10*1000.0; commission=commission+0.15 * 800.0;
    commission = commission + 0.20 * (sales-1800.0)
Step 9: if (sales > 1000.0) goto Step 10 else goto Step 11
Step 10: commission=0.10* 1000.0; commission=commission + 0.15 *
(sales-1000.0)
Step 11: Output(“Commission is $”, commission)
Step 12: exit
```

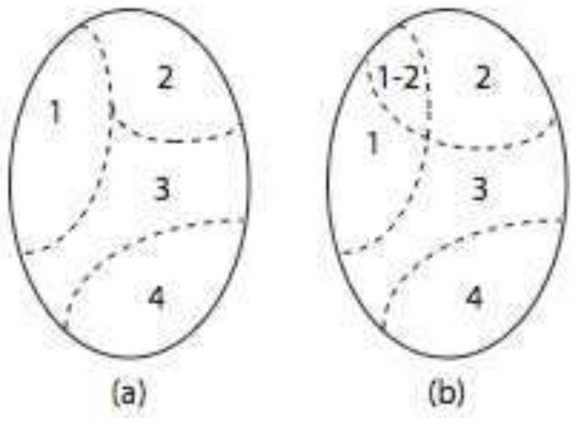
5.3 PROGRAM CODE:

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

5.4 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.

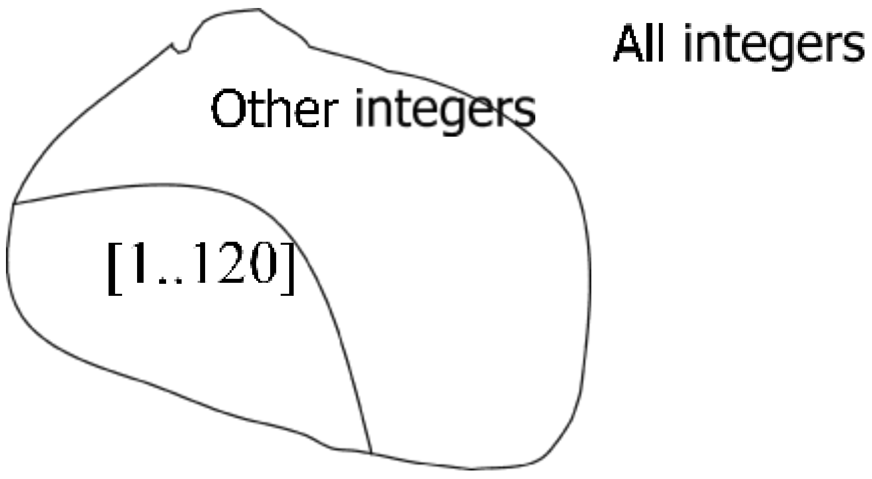
Example:

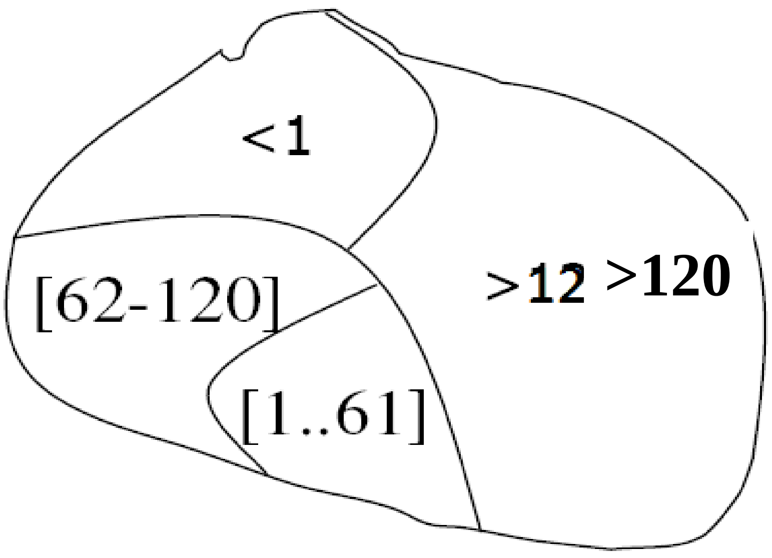
Consider an application A that takes an integer denoted by age as input. Let us suppose that the only legal values of age are in the range [1..120]. The set of input values is now divided into a set E containing all integers in the range [1..120] and a set U containing the remaining integers.

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 input domain of the commission problem is naturally partitioned by the limits on locks, stocks and barrels. These equivalence classes are exactly those that would also be identified by traditional equivalence class testing. 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, however. The variables lock 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.
Expect for the names of the variables and the interval endpoint values, this isidentical to our first version of the NextDate function. therefore we will have exactly one week normal equivalence class test case – and again, it is identical to the strong normal equivalence class test case. Note that the case for locks =-1 just terminates the iteration.

First attempt

We will have eight weak robust test cases.

TC Id	Test Case Description	Input Data			Sales	Expected Output(Commission)	Actual Output	Status
		Locks	Stocks	Barrels				
1	WR1	10	10	10	$100	10		
2	WR2	-1	40	45	Program terminates	Program terminates		
3	WR3	-2	40	45	Values of locks not inthe range 1...70	Values of locks not in the range 1...70		
4	WR4	71	40	45	Values of locks not inthe range 1...70	Values of locks not in the range 1...70		
5	WR5	35	-1	45	Values of stocks not inthe range 1...80	Values of stocks not in the range 1...80		
6	WR6	35	81	45	Values of stocks not inthe range 1...80	Values of stocks not in the range 1...80		
7	WR7	10	9	10	970	97		
8	WR8	9	10	10	955	95.5		

Second attempt:

Finally, 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**				
1	SR1	-2	40	45	Values of locks not in the range 1...70	Values of locks not in the range 1...70		
2	SR2	35	-1	45	Values of stocks not in the range 1...80	Values of stocks not in the range 1...80		
3	SR3	35	40	-2	Values of barrels not in the range 1...90	Values of barrels not in the range 1...90		
4	SR4	-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		
5	SR5	-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		
6	SR6	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		
7	SR7	-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	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		

5.5 EXECUTIONS

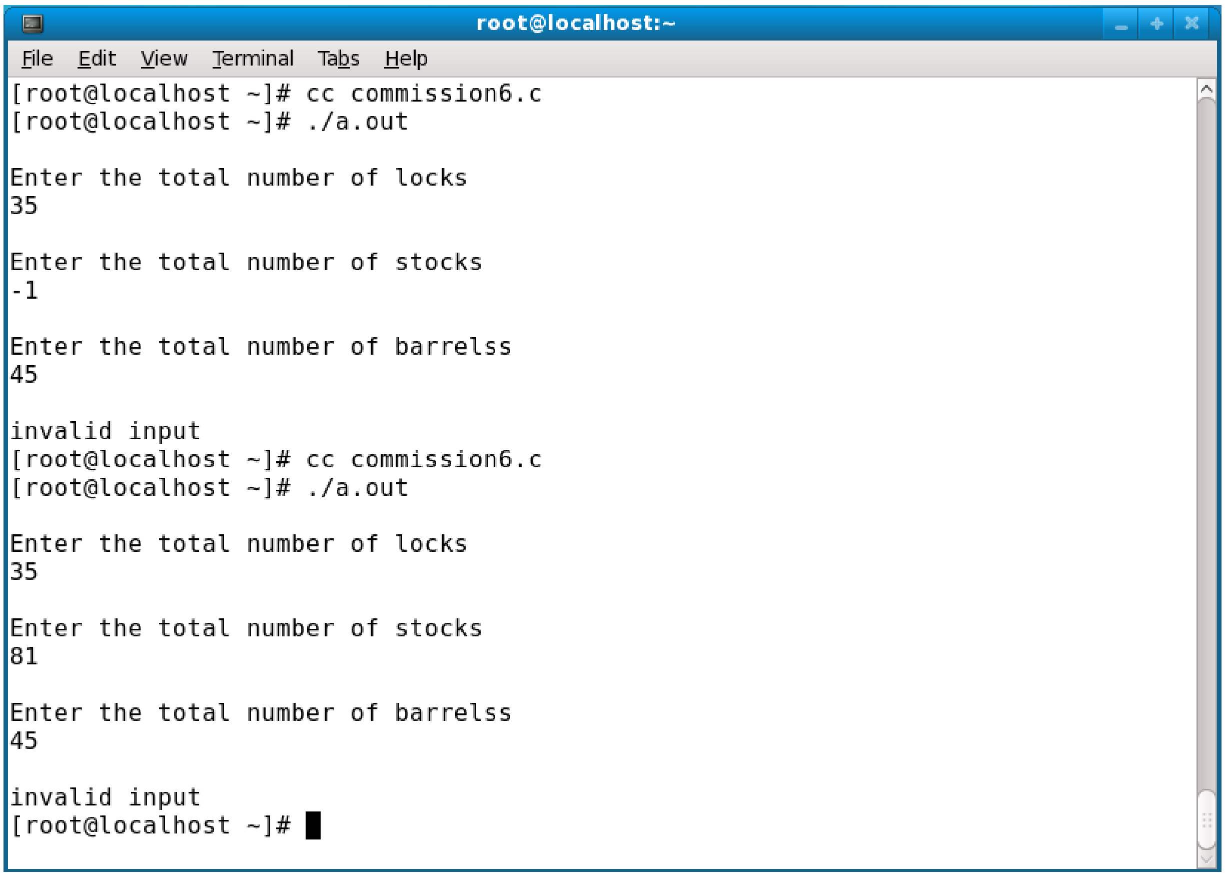
Execute the program and test the test cases in Table-1 against program and complete the table with for Actual output column   and Status column

Test Report:

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

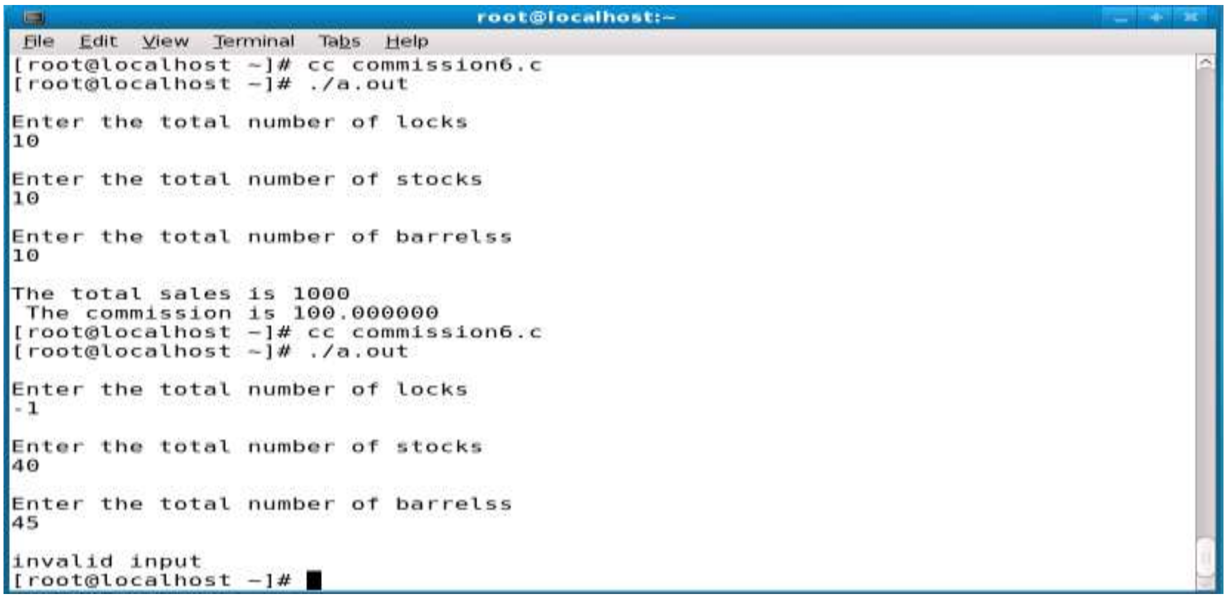
5.6 SNASHOTS

1. Snapshot for invalid inputs

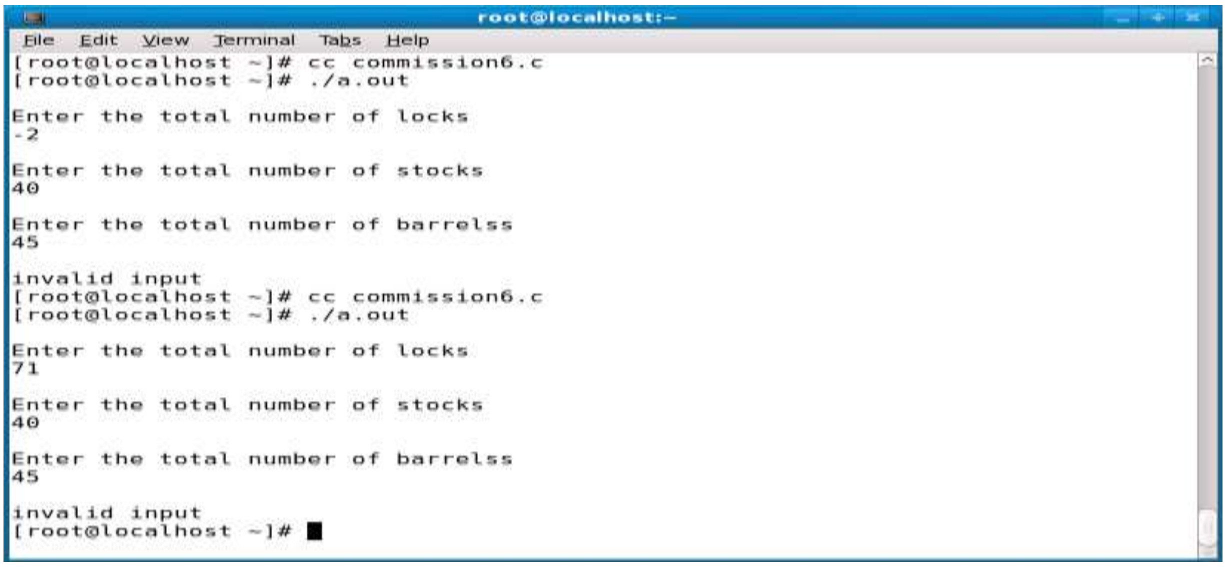


```
[root@localhost ~]# cc commission6.c
[root@localhost ~]# ./a.out

Enter the total number of locks
35

Enter the total number of stocks
-1

Enter the total number of barrelss
45

invalid input
[root@localhost ~]# cc commission6.c
[root@localhost ~]# ./a.out

Enter the total number of locks
35

Enter the total number of stocks
81

Enter the total number of barrelss
45

invalid input
[root@localhost ~]# 
```

2. Invalid Input and commission for when the all inputs are 10



```
[root@localhost ~]# cc commission6.c
[root@localhost ~]# ./a.out

Enter the total number of locks
10

Enter the total number of stocks
10

Enter the total number of barrelss
10

The total sales is 1000
 The commission is 100.000000
[root@localhost ~]# cc commission6.c
[root@localhost ~]# ./a.out

Enter the total number of locks
-1

Enter the total number of stocks
40

Enter the total number of barrelss
45

invalid input
[root@localhost ~]# 
```



```
[root@localhost ~]# cc commission6.c
[root@localhost ~]# ./a.out

Enter the total number of locks
-2

Enter the total number of stocks
40

Enter the total number of barrelss
45

invalid input
[root@localhost ~]# cc commission6.c
[root@localhost ~]# ./a.out

Enter the total number of locks
71

Enter the total number of stocks
40

Enter the total number of barrelss
45

invalid input
[root@localhost ~]# 
```

5.7 REFERENCES

1. Requirement Specification
2. Assumptions

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
Step 2: if MM is 01, 03, 05,07,08,10 do Step 3 else Step 6
Step 3:if DD < 31 then do Step 4 else if DD=31 do Step 5 else output(Invalid
Date);
Step 4: tomorrowday=DD+1 goto Step 18
Step 5: tomorrowday=1; tomorrowmonth=month + 1 goto Step 18
Step 6: if MM is 04, 06, 09, 11 do Step 7
Step 7: if DD<30 then do Step 4 else if DD=30 do Step 5 else output(Invalid
Date);
Step 8: if MM is 12
Step 9: if DD<31 then Step 4 else Step 10
Step 10: tomorrowday=1, tommorowmonth=1, tommorowyear=YYYY+1; goto
Step 18
Step 11: if MM is 2
Step12: if DD<28 do Step 4 else do Step 13
Step 13: if DD=28 & YYYY is a leap do Step 14 else Step 15
Step 14: tommorowday=29 goto Step18
Step 15: tommorowday=1, tomorrowmonth=3, goto Step 18;
Step 16: if DD=29 then do Step15 else Step 17
Step 17: output("Cannot have feb", DD); Step 19
Step 18: output(tomorrowday, tomorrowmonth, tomorrowyear);
Step 19: exit
```

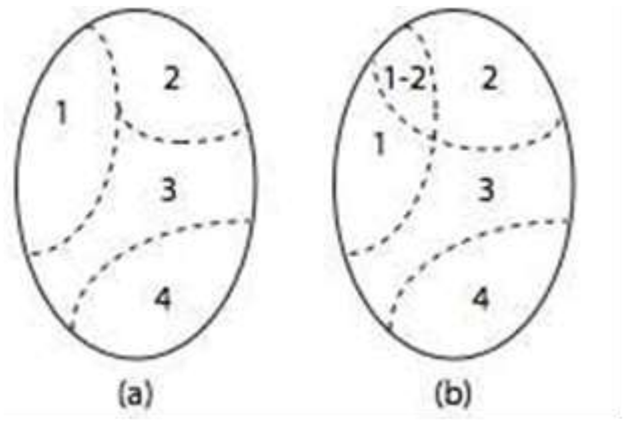
6.3 PROGRAM CODE:

```
#include<stdio.h>
#include<conio.h>
main( )
{
int month[12]={31,28,31,30,31,30,31,31,30,31,30,31};
int d,m,y,nd,nm,ny,ndays;
clrscr( );
printf("enter the date,month,year");
scanf("%d%d%d",&d,&m,&y);
ndays=month[m-1];
if(y<=1812 && y>2012)
{
        printf("Invalid Input Year");
        exit(0);
}

if(d<=0 || d>ndays)
{


        printf("Invalid Input Day");
        exit(0);
}
if(m<1 && m>12)
{
        printf("Invalid Input Month");
        exit(0);
}


if(m==2)
{
        if(y%100==0)
        {
           if(y%400==0)
           ndays=29;
        }
        else
        if(y%4==0)
                ndays=29;
nd=d+1;
nm=m;
ny=y;
}
if(nd>ndays)
{
        nd=1;
        nm++;
}
if(nm>12)
{
        nm=1;
        ny++;
}
if(nm>12)
{
        nm=1;
        ny++;
}
printf("\n Given date is %d:%d:%d",d,m,y); printf("\n
Next day's date is %d:%d:%d",nd,nm,ny); getch( );
}
```

6.4 TESTING

Technique used: Equivalence Class testing

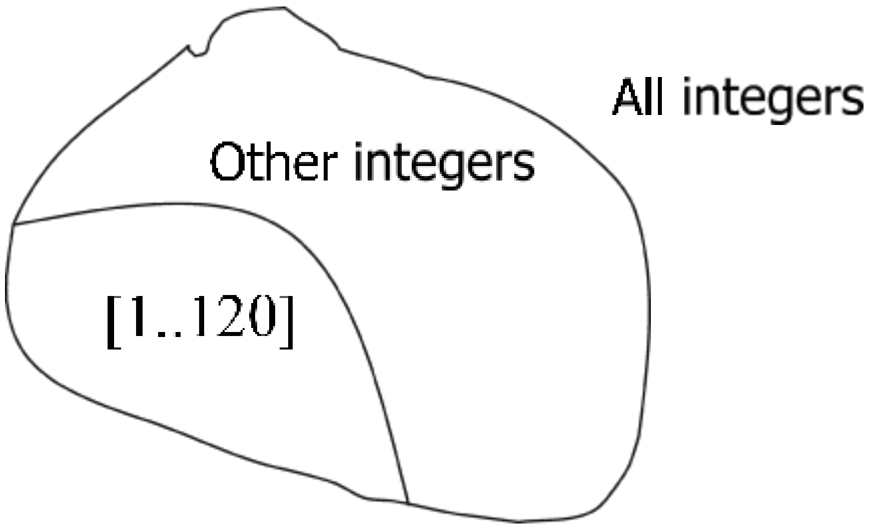
Test selection using equivalence partitioning allows a tester to subdivide the input domain into a relatively small number of sub-domains, say N>1, as shown.



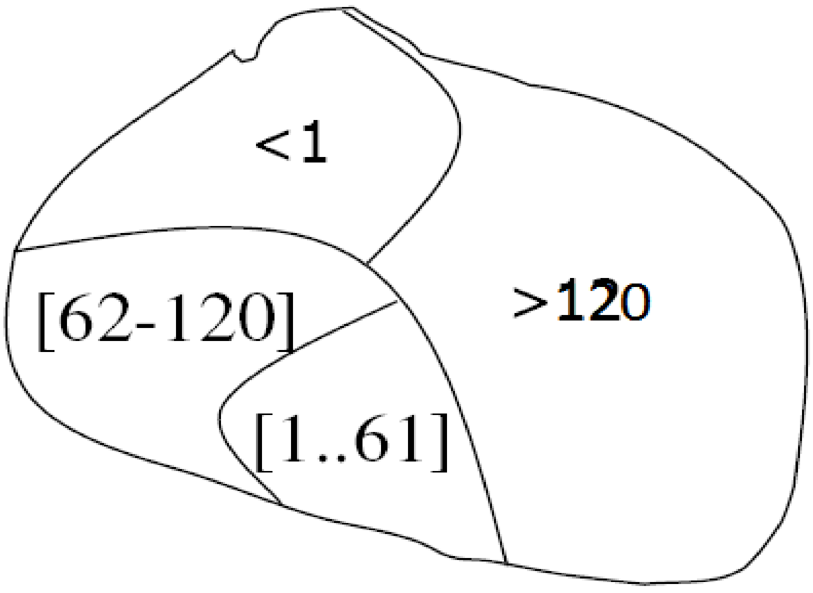
In strict mathematical terms, the sub-domains by definition are disjoint. The four subsets shown in (a) constitute a partition of the input domain while the subsets in (b) are not. Each subset is known as an equivalence class.

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:

<u>Valid Intervals</u>

M1 = {month: $1 \le$ month ≤ 12}
D1 = {day: $1 \le$day $\le$31}
Y1 = {year: $1812 \le$ year $\le$2012}

<u>Valid Intervals</u>

M1 = {month: $1 \le$ month ≤ 12}
D1 = {day: $1 \le$day $\le$31}
Y1 = {year: $1812 \le$ year $\le$2012}

<u>Invalid Intervals</u>

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		

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		
2	Testing for Invalid Day, day with negative number it is not possible	6	-1	1900	Day not in range		
3	Testing for Invalid Day, day with Out of range i.e., DD=32	6	32	1900	Day not in range		
4	Testing for Invalid Month, month with negative number it is not possible	-1	15	1900	Month not in range		

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		
6	Testing for Year, year is out of range YYYY=1899, it should <=1812	6	15	1899	Year not in range		
7	Testing for Year, year is out of range YYYY=2013, it should <=2012	6	15	2013	Year not in range		

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:

This is a table showing one corner of the cube in 3d-space (the three other corners would include a different combination of variables) since the complete table would be too large to show.

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		
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		
3	Testing for Year is not in range YYYY=1899 i.e., Year should <=1812	6	15	1899	Year not in range		
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		
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	1899	i) Day not in range ii) Year not in range		

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	-1	15	1899	i) Month not in range ii) Year not in range		
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	1899	i) Day not in range ii) Month not in range iii) Year not in range		

Second Attempt

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

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

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

D1 = {day: 1 ≤ day ≤ 28}
D2 = {day: day = 29}
D3 = {day: day = 30}
D4 = {day: day = 31}

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

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

Year has been split up into common years, leap years and the special case the year 2000 so we can determine the date in the month of February.

Here are the test cases for the new equivalence relation using the four types of Equivalence Class testing.

Weak Normal

TC Id	Test Case Description	Input Data			Expected Output	Actual Output	Status
		MM	DD	YYYY			
1	Testing for all Valid input changing the day within the month.	6	14	2000	6/15/2000		
2	Testing for Valid input changing the day within the month.	7	29	1996	7/30/1996		
3	Testing for Leaf year, i.e., MM=2 (Feb) the input DD=30, there is not possible date 30, in leaf year only 28 and 29 will occur.	2	30	2002	Impossible date		
4	Testing for Impossible Date, i.e., MM=6 (June) the input DD=31, there is only 30 days in the month of June, So, DD=31 is Impossible Date.	6	31	2000	Impossible input date		

Table 3: Weak normal

Strong Normal

TC ID	Test Case Description	Input Data			Expected Output	Actual Output	Status
		MM	DD	YYYY			
1	SN1	6	14	2000	6/15/2000		
2	SN2	6	14	1996	6/15/1996		
3	SN3	6	14	2002	6/15/2002		
4	SN4	6	29	2000	6/30/2000		
5	SN5	6	29	1996	6/30/1996		
6	SN6	6	29	2002	6/30/2002		
7	SN7	6	30	2000	Invalid Input Date		
8	SN8	6	30	1996	Invalid Input Date		
9	SN9	6	30	2002	Invalid Input Date		
10	SN10	6	31	2000	Invalid Input Date		
11	SN11	6	31	1996	Invalid Input Date		
12	SN12	6	31	2002	Invalid Input Date		
13	SN13	7	14	2000	7/15/2000		
14	SN14	7	14	1996	7/15/1996		
15	SN15	7	14	2002	7/15/2002		
16	SN16	7	29	2000	7/30/2000		
17	SN17	7	29	1996	7/30/1996		
18	SN18	7	29	2002	7/30/2002		
19	SN19	7	30	2000	7/31/2000		

20	SN20	7	30	1996	7/31/1996		
21	SN21	7	30	2002	7/31/2002		
22	SN22	7	31	2000	8/1/2000		
23	SN23	7	31	1996	8/1/1996		
24	SN25	7	31	2002	8/1/2002		
25	SN24	2	14	2000	2/15/2000		
26	SN26	2	14	1996	2/15/1996		
27	SN27	2	14	2002	2/15/2002		
28	SN28	2	29	2000	Invalid Input Date		
29	SN29	2	29	1996	3/1/1996		
30	SN30	2	29	2002	Invalid Input Date		
31	SN31	2	30	2000	Invalid Input Date		
32	SN32	2	30	1996	Invalid Input Date		
33	SN33	2	30	2002	Invalid Input Date		
34	SN34	2	31	2000	Invalid Input Date		
35	SN35	2	31	1996	Invalid Input Date		
36	SN36	2	31	2002	Invalid Input Date		

Table 4: Strong Normal

6.5 EXECUTIONS

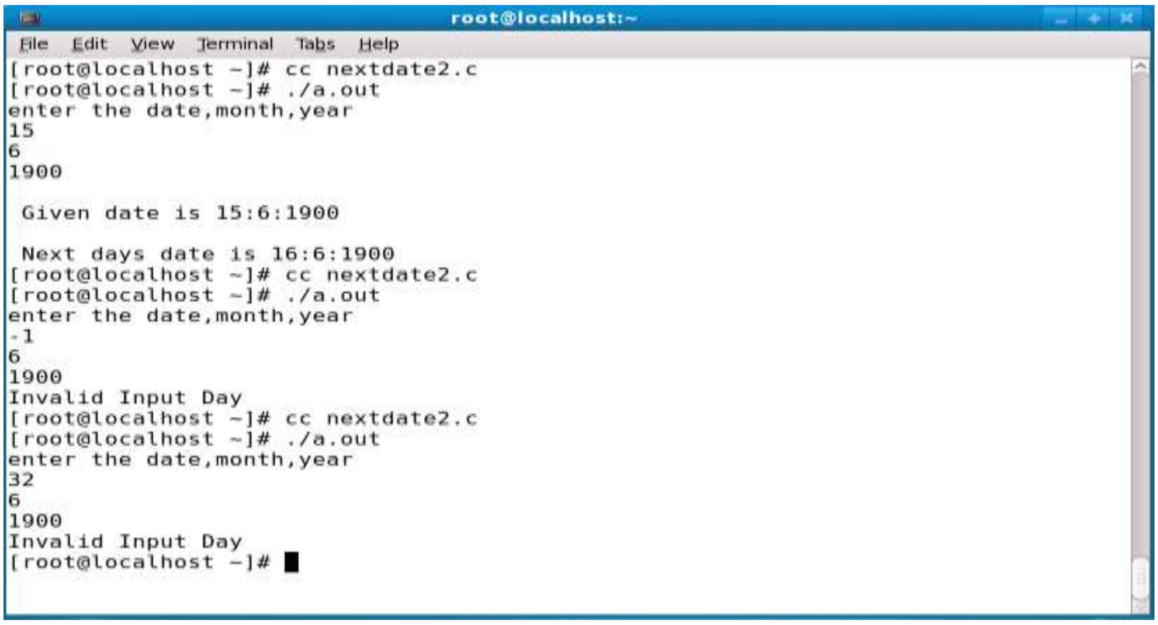
Execute the program and test the test cases in Table-1 against program and complete the table with for Actual output column and Status column

Test Report:

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

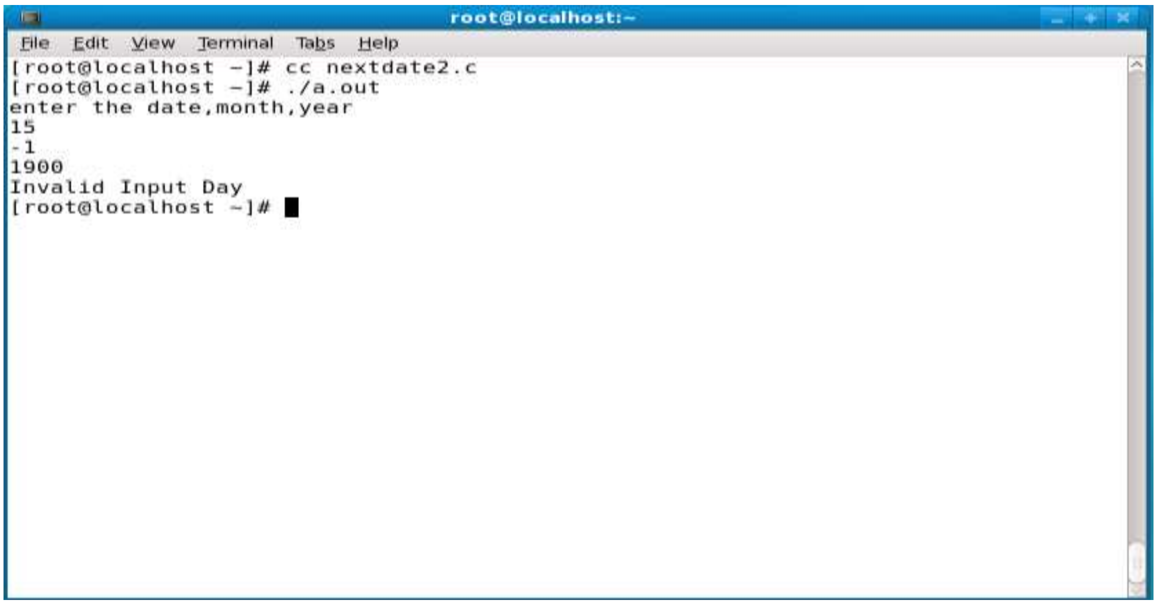
6.6 SNAPSHOTS:

1. Snapshot to show the nextdate for current date and invalid day is entered



```
[root@localhost ~]# cc nextdate2.c
[root@localhost ~]# ./a.out
enter the date,month,year
15
6
1900

 Given date is 15:6:1900

 Next days date is 16:6:1900
[root@localhost ~]# cc nextdate2.c
[root@localhost ~]# ./a.out
enter the date,month,year
-1
6
1900
Invalid Input Day
[root@localhost ~]# cc nextdate2.c
[root@localhost ~]# ./a.out
enter the date,month,year
32
6
1900
Invalid Input Day
[root@localhost ~]#
```

2. Invalid Input



```
[root@localhost ~]# cc nextdate2.c
[root@localhost ~]# ./a.out
enter the date,month,year
15
-1
1900
Invalid Input Day
[root@localhost ~]#
```

6.7 REFERENCES:

1. Requirement Specification
2. Assumptions

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 REQUIREMENT SPECIFICATION:

R1. The system should accept 3 positive integer numbers (a, b, c) which represents 3 sides of the triangle. Based on the input it should determine if a triangle can be formed or not.

R2. If the requirement R1 is satisfied then the system should determine the type of the triangle, which can be

- Equilateral (i.e. all the three sides are equal)
- Isosceles (i.e Two sides are equal)
- Scalene (i.e All the three sides are unequal)

else suitable error message should be displayed. Here we assume that user gives three positive integer numbers as input.

7.2 DESIGN:

Form the given requirements we can draw the following conditions: 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 the triangle, if any one of the three conditions C1,
C2 and C3 are not satisfied then triangle cannot be constructed. So only when C1, C2 and C3 are true the triangle can be formed, then depending on conditions C4, C5 and C6 we can decide what type of triangle will be formed. (i.e requirement R2).

Algorithm:

```
Step 1: Input a, b & c i.e three integer values which represent three sides of the triangle.
Step 2: if (a < (b + c)) and (b < (a + c)) and (c < (a + b) then
        do Step 3
        else
        print not a triangle. do Step 6.
Step 3: if (a=b) and (b=c) then
        Print triangle formed is equilateral. do Step 6.
Step 4: if (a ≠ b) and (a ≠ c) and (b ≠ c) then
        Print triangle formed is scalene. do Step 6.
Step 5: Print triangle formed is Isosceles.
Step 6: stop
```

7.3 PROGRAM CODE:

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#include<process.h>
 int main()
 {
        int a, b, c;
        clrscr();
        printf("Enter three sides of the triangle");
        scanf("%d%d%d", &a, &b, &c);
        if((a<b+c)&&(b<a+c)&&(c<a+b))
        {

                if((a==b)&&(b==c))
                {
                        printf("Equilateral triangle");
                }
                else if((a!=b)&&(a!=c)&&(b!=c))
                {
                        printf("Scalene triangle");
                        }
                        else
                        printf("Isosceles triangle");

                  }
                else
                {
                        printf("triangle cannot be formed");
                } getch(); return 0;
}
```

7.4
TESTING:

Technique Used: Decision Table Approach

Decision Table-Based Testing has been around since the early 1960's; it is used to depict complex logical relationships between input data. A Decision Table is the method used to build a complete set of test cases without using the internal structure of the program in question. In order to create test cases we use a table to contain the input and output values of a program.The decision table is as given below:

Conditions	**Condition Entries (Rules)**										
	R 1	R 2	R 3	R 4	R 5	R 6	R 7	R 8	R 9	R 10	R 11
C1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
C2: b<a+c?	--	F	T	T	T	T	T	T	T	T	T
C3: c<a+b?	--	--	F	T	T	T	T	T	T	T	T
C4: a=b?	--	--	--	F	T	T	T	F	F	F	T
C5: a=c?	--	--	--	T	F	T	F	T	F	F	T
C6: b=c?	--	--	--	T	T	F	F	F	T	F	T
Actions	**Action Entries**										
a1: Not a Triangle	X	X	X								
a2: Scalene										X	
a3: Isosceles							X	X	X		
a4: Equilateral											X
a5: Impossible				X	X	X					

The “--“ symbol in the table indicates don’t care values. The table shows the six conditions and 5 actions. All the conditions in the decision table are binary; hence, it is called as “Limited Entry decision table”.

Each column of the decision table represents a test case. That is,
The table is read as follows:
Action: Not a Triangle

1. When condition C1 is false we can say that with the given ‘a’ ‘b’ and ‘c’

values, it’s Not a
triangle.

2. Similarly condition C2 and C3, if any one of them are false, we can say that with the given ‘a’ ‘b’ and ‘c’ values it’s Not a triangle.
Action: Impossible

3. When conditions C1, C2, C3 are true and two conditions among C4, C5, C6 is true, there is no chance of one conditions among C4, C5, C6 failing. So we can neglect these rules.
   Example: if condition C4: a=b is true and C5: a=c is true

   Then it is impossible, that condition C6: b=c will fail, so the action is Impossible.

Action: Isosceles

4. When conditions C1, C2, C3 are true and any one condition among C4, C5 and C6 is true with remaining two conditions false then action is Isosceles triangle.
   Example: If condition C4: a=b is true and C5: a=c and C6: b=c are false, it means two sides are equal. So the action will be Isosceles triangle.
Action: Equilateral

5. When conditions C1, C2, C3 are true and also conditions C4, C5 and C6 are true then, the action is Equilateral triangle.
Action: Scalene

6. When conditions C1, C2, C3 are true and conditions C4, C5 and C6 are false

i.e sides a, b and c are different, then action is Scalene triangle.
Number of Test Cases = Number of Rules.
Using the decision table we obtain 11 functional test cases: 3 impossible cases,

3 ways of failing the triangle property, 1 way to get an equilateral triangle, 1 way to get a scalene triangle, and 3 ways to get an isosceles triangle.
Deriving test cases using
Decision Table Approach:
Test Cases:

TC ID	**Test Case Description**	**a**	**B**	**c**	**Expected Output**	**Actual Output**	**Status**
1	Testing for Requirement 1	4	1	2	Not a Triangle		
2	Testing for Requirement 1	1	4	2	Not a Triangle		
3	Testing for Requirement 1	1	2	4	Not a Triangle		
4	Testing for Requirement 2	5	5	5	Equilateral		
5	Testing for Requirement 2	2	2	3	Isosceles		
6	Testing for Requirement 2	2	3	2	Isosceles		
7	Testing for Requirement 2	3	2	2	Isosceles		
8	Testing for Requirement 2	3	4	5	Scalene		

7.5 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: 08
2. No of Defects Raised:
3. No of TC’s Pass:
4. No of TC’s Failed:

The decision table technique is indicated for applications characterised by any of the following:
Prominent if-then-else logic

Logical relationships among input variables

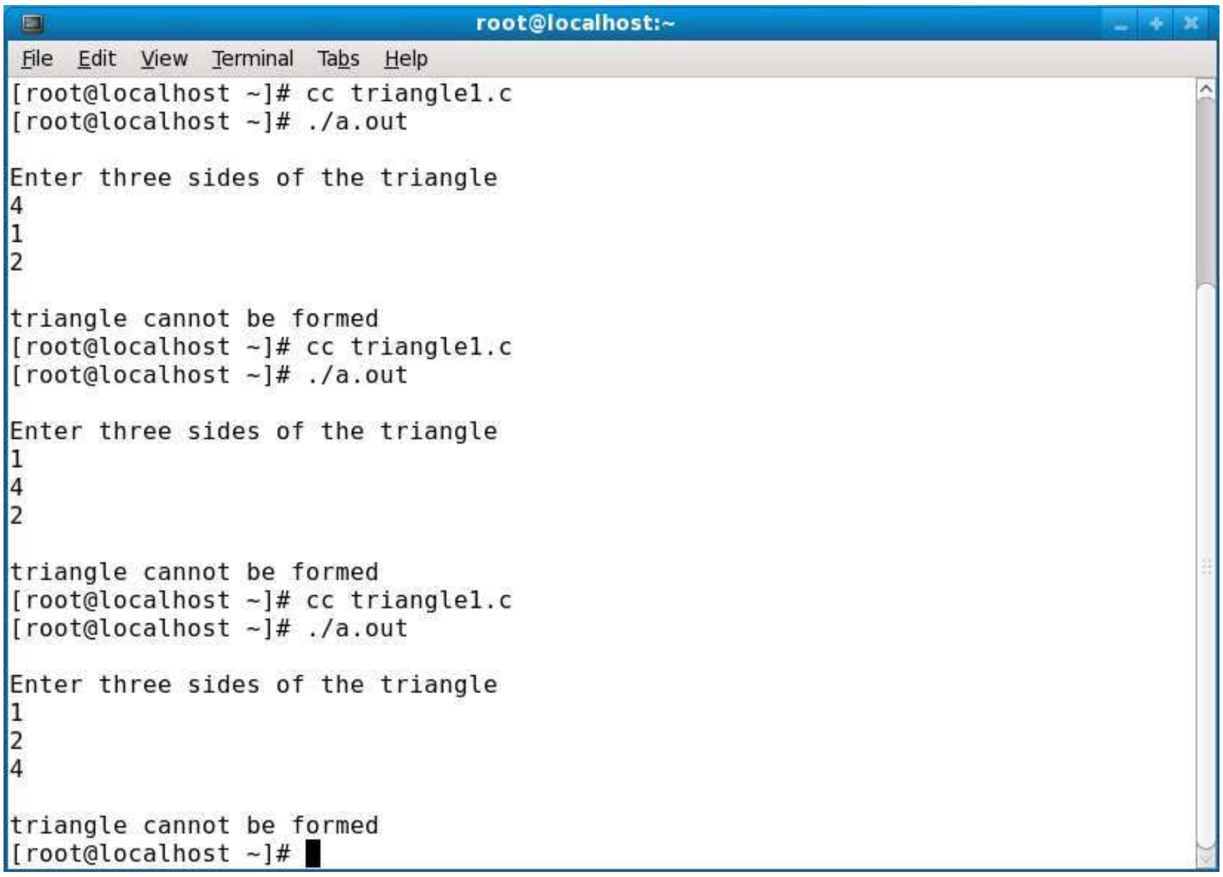
Calculations involving subsets of the input variables

Cause-and-effect relationship between inputs and outputs

The decision table-based testing works well for triangle problem because a lot of decision making i.e if-then-else logic takes place.

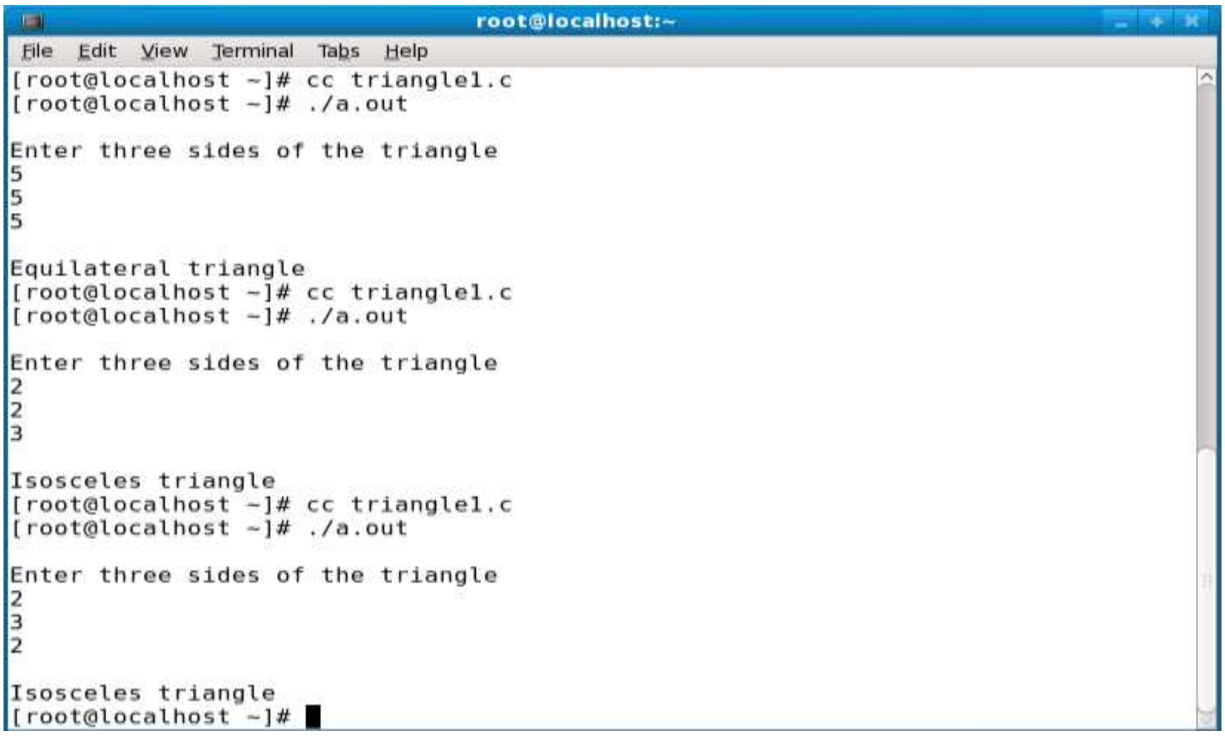
7.6 SNAPSHOTS:

1. Output screen of Triangle cannot be formed



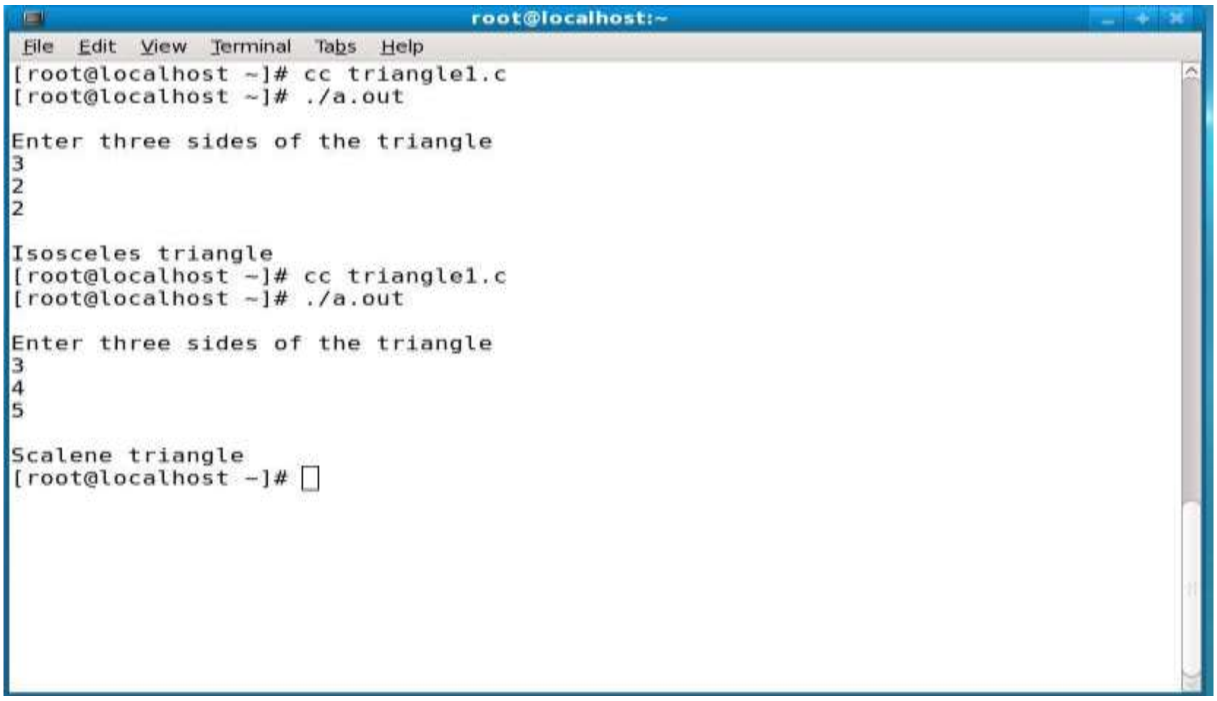
```
[root@localhost ~]# cc triangle1.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
4
1
2

triangle cannot be formed
[root@localhost ~]# cc triangle1.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
1
4
2

triangle cannot be formed
[root@localhost ~]# cc triangle1.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
1
2
4

triangle cannot be formed
[root@localhost ~]# 
```

2. Output screen of Equilateral and Isosceles Triangle.



```
[root@localhost ~]# cc triangle1.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
5
5
5

Equilateral triangle
[root@localhost ~]# cc triangle1.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
2
2
3

Isosceles triangle
[root@localhost ~]# cc triangle1.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
2
3
2

Isosceles triangle
[root@localhost ~]#
```

3. Output screen for Scalene Triangle



```
[root@localhost ~]# cc triangle1.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
3
2
2

Isosceles triangle
[root@localhost ~]# cc triangle1.c
[root@localhost ~]# ./a.out

Enter three sides of the triangle
3
4
5

Scalene triangle
[root@localhost ~]#
```

7.7. REFERENCES:

1. Requirement Specification
2. Assumption.

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 REQUIREMENT SPECIFICATION:

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

(i.e 1≤ Locks≤ 70) (i.e
1 ≤ Stocks ≤ 80) (i.e 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

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

8.2 DESIGN:

Form the given requirements we can draw the following conditions:

C1: 1≤locks≤70? Locks = -1? (occurs if locks = -1 is used to control input iteration).

C2: 1≤stocks≤80?

Here C1 can be expanded as:

C1a: 1≤locks

C1b: locks≤70

C3: 1≤barrels≤90?

C4: sales>1800?

C5: sales>1000?

C6: sales≤1000?

Algorithm:

```
Step 1: Input 3 integer numbers which represents number of Locks, Stocks and Barrels sold.
Step 2: compute the total sales =
        (Number of Locks sold *45) + (Number of Stocks sold *30) + (Number of Barrels sold *25)
Step 3: if a totals sale in dollars is less than or equal to $1000
            then commission = 0.10* total Sales do Step 6
Step 4: else if total sale is less than $1800
            then commission1 = 0.10* 1000
                commission = commission1 + (0.15 * (total sales – 1000))
                do Step 6
Step 5: else commission1 = 0.10* 1000
            commission2 = commission1 + (0.15 * 800))
            commission = commission2 + (0.20 * (total sales – 1800)) do Step 6
Step 6: Print commission.
Step 7: Stop.
```

8.3 PROGRAM CODE:

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

```
 printf("Enter the total number of stocks");
scanf("%d",&stocks);
if ((stocks <= 0) || (stocks > 80))
{
        flag = 1;
}
 printf("Enter the total number of stocks");
scanf("%d",&stocks);
if ((stocks <= 0) || (stocks > 80))
{
        flag = 1;
}
printf("Enter the total number of barrelss");
scanf("%d",&barrels);
if ((barrels <= 0) || (barrels > 90))
{
        flag = 1;
}
if (flag == 1)
{
        printf("invalid input");
        getch();
        exit(0);
}
t_sales = (locks * 45) + (stocks * 30) + (barrels * 25);
if (t_sales <= 1000)
{
        commission = 0.10 * t_sales;
}
else if (t_sales < 1800)
{
```

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

8.4 TESTING

Technique Used: Decision Table Approach

The decision table is given below

Conditions	**Condition Entries (Rules)**					
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
Actions	**Action Entries**					
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			

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.

DERIVING TEST CASES USING Decision Table Approach:

Test Cases

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			
2	Testing for Requirement 1 Condition 1 (C1)	90	40	45	Out of Range			
3	Testing for Requirement 1 Condition 2 (C2)	35	-3	45	Out of Range			
4	Testing for Requirement 1 Condition 2 (C2)	35	100	45	Out of Range			
5	Testing for Requirement 1 Condition 3 (C3)	35	40	-10	Out of Range			
6	Testing for Requirement 1 Condition 3 (C3)	35	40	150	Out of Range			
7	Testing for Requirement 2	5	5	5	500	a1:50		
8	Testing for Requirement 2	15	15	15	1500	a2: 175		
9	Testing for Requirement 2	25	25	25	2500	a3: 360		

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:
2. No of Defects Raised:

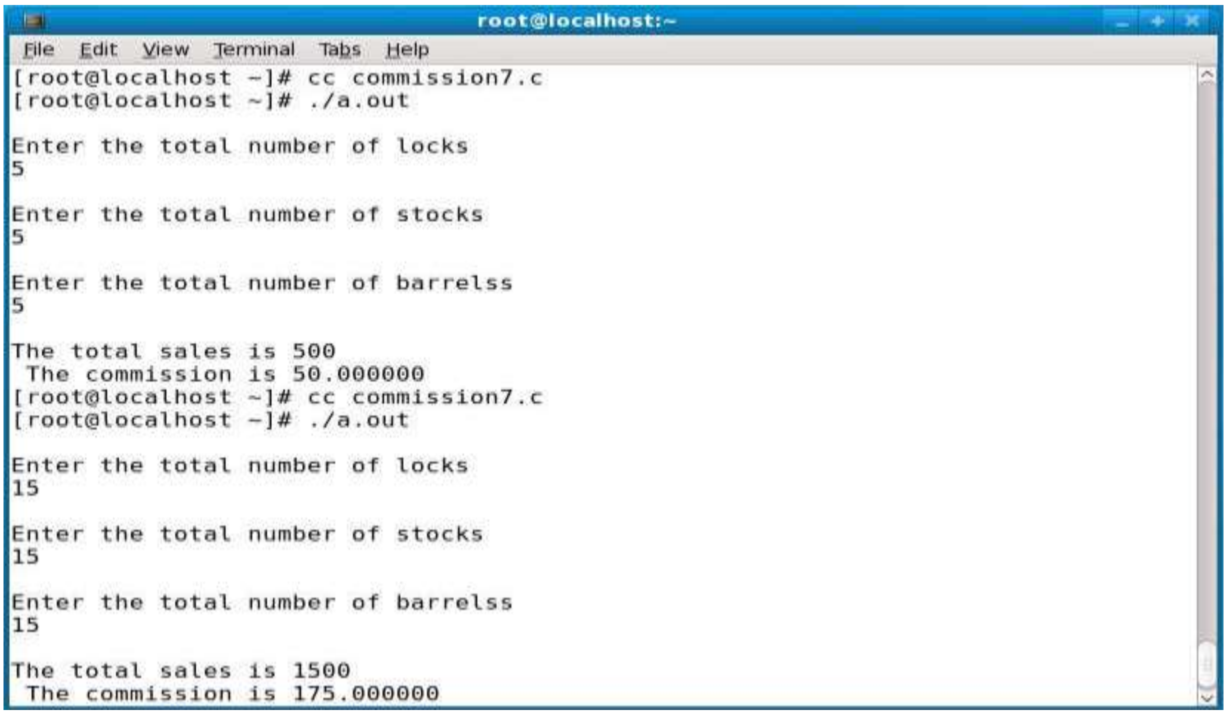
3. No of TC's Pass:

4. No of TC's Failed:

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.

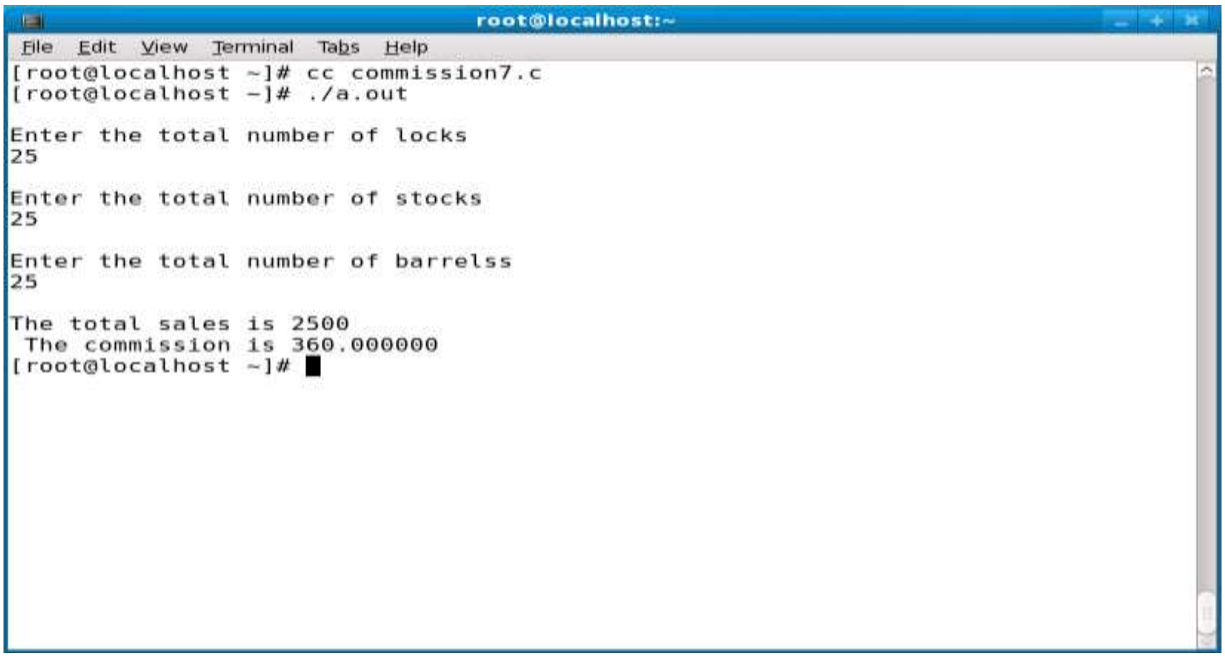
8.6 SNAPSHOTS:

1. Snapshot for Total sales and commission when total sales are within 1000 and 1800



```
[root@localhost ~]# cc commission7.c
[root@localhost ~]# ./a.out

Enter the total number of locks
5

Enter the total number of stocks
5

Enter the total number of barrelss
5

The total sales is 500
 The commission is 50.000000
[root@localhost ~]# cc commission7.c
[root@localhost ~]# ./a.out

Enter the total number of locks
15

Enter the total number of stocks
15

Enter the total number of barrelss
15

The total sales is 1500
 The commission is 175.000000
```

2. Snapshot when the inputs all are 25.



```
[root@localhost ~]# cc commission7.c
[root@localhost ~]# ./a.out

Enter the total number of locks
25

Enter the total number of stocks
25

Enter the total number of barrelss
25

The total sales is 2500
 The commission is 360.000000
[root@localhost ~]#
```

8.7 REFERENCES:

1. Requirement Specification
2. Assumptions

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 REQUIREMENT SPECIFICATION

Problem Definition: The Commission Problem includes a salesperson in the former Arizona Territory sold rifle locks, stocks and barrels made by a gunsmith in Missouri. Cost includes

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 sales person sent a telegram to the Missouri gunsmith with the number of locks, stocks and barrels sold in the town. At the end of the 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:

On sales up to(and including) $1000= 10% On
the sales up to(and includes) $1800= 15% On the
sales in excess of $1800= 20%
The commission program produces a monthly sales report that gave the total number of locks, stocks and barrels sold, the salesperson’s total dollar sales and finally the commission

9.2 DESIGN

Algorithm:

```
Step 1: Define lockPrice=45.0, stockPrice=30.0, barrelPrice=25.0
Step2: Input locks
Step3: while(locks!=-1) ‘input device uses -1 to indicate end of data goto
```

```
Step 12
Step 4:input (stocks, barrels)
Step 5: compute lockSales, stockSales, barrelSales and sales
Step 6: output("Total sales:" sales)
Step 7: if (sales > 1800.0)  goto Step 8 else goto Step 9
Step 8: commission=0.10*1000.0; commission=commission+0.15 * 800.0;
    commission = commission + 0.20 * (sales-1800.0)
Step 9: if (sales > 1000.0) goto Step 10 else goto Step 11
Step10: commission=0.10* 1000.0; commission=commission + 0.15 *
(sales-1000.0)
Step 11: Output("Commission is $", commission)
Step12: exit
```

9.3 PROGRAM CODE:

```c
#include<stdio.h>
#include<conio.h>
int main()
{
        int locks, stocks, barrels, t_sales, flag = 0;
        float commission;
        clrscr();
        printf("Enter the total number of locks");
        scanf("%d",&locks);
        if ((locks <= 0) || (locks > 70))
        {
                flag = 1;
        }
        printf("Enter the total number of stocks");
        scanf("%d",&stocks);
        if ((stocks <= 0) || (stocks > 80))
        {
                flag = 1;
        }
        printf("Enter the total number of barrelss");
        scanf("%d",&barrels);
        if ((barrels <= 0) || (barrels > 90))
        {
                flag = 1;
        }
        if (flag == 1)
        {
                printf("invalid input");
                getch();
                exit(0);
        }
        t_sales = (locks * 45) + (stocks * 30) + (barrels * 25);
        if (t_sales <= 1000)
        {
                commission = 0.10 * t_sales;
        }
        else if (t_sales < 1800)
        {

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

9.4 TESTING TECHNIQUE: DATAFLOW TESTING

A structural testing technique

• Aims to execute sub-paths from points where each variable is defined to points where it is referenced. These sub-paths are called definition-use pairs, or du-pairs (du-paths, du-chains) Data flow testing is centred on variables (data) Data flow testing follows the sequences of events related to a given data item with the objective to detect incorrect sequences It explores the effect of using the value produced by every and each computation.

Variable definition

Occurrences of a variable where a variable is given a new value (assignment, input by the user, input from a file, etc.) Variable DECLARATION is NOT its definition !!!

Variable uses

Occurences of a variable where a variable is not given a new value (variable DECLARATION is NOT its use)

p-uses (predicate uses)

Occur in the predicate portion of a decision statement such as if-then-else, while-do etc.

c-uses (computation uses)

All others, including variable occurrences in the right hand side of an assignment statement, or an output statement

du-path: A sub-path from a variable definition to its use.

Test case definitions based on four groups of coverage

– All definitions.

– All c-uses.

– All p-uses.

– All du-paths.

DATA FLOW TESTING: KEY Steps

Given a code (program or pseudo-code).

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

line	**catogary**		
	Definition	**c-use**	**p-use**
1			
2			
3			
4			
5			
6			
7			
8			
9	locks, stocks, barrels		
10			locks, stocks, barrels
11			
12	Flag		
13			
14			flag
15			
16			
17			

18			
19			
20	t_sales	locks, stocks, barrels	
21			t_sales
22			
23	commission	t_sales	
24			
25			t_sales
26			
27	commission		
28	commission	commission, t_sales	
29			
30			
31			
32	commission		
33	commission	commission	
34	commission	commission, t_sales	
35			
36		commission	
37			
38			
39			

Table: list occurrences & assign a category to each variable

definition - use pair	**variables()**	
start line → end line	c-use	p-use
9→10		locks
9→10		stocks
9→10		barrels
9→20	locks	
9→20	stocks	
9→20	barrels	
12→14		flag
20→21		t_sales
20→23	t_sales	
20→25		t_sales
20→28	t_sales	
20→34	t_sales	
23→36	commission	
27→28	commission	
28→36	commission	
32→33	commission	
33→34	commission	
34→36	commission	

Table: Define test cases

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.

			Inputs			**Expected output**	
Variable(s)	**du-pair**	**sub-path**	**locks**	**stocks**	**barrels**	**t_sales**	**commission**
locks, stocks, barrels	9→20	9,10,20	10	10	10	1000	
locks, stocks, barrels	9→10	9→10	5	-1	22	Invalid Input	
Flag	12→14	12→14	-1	40	45	Invalid Input	

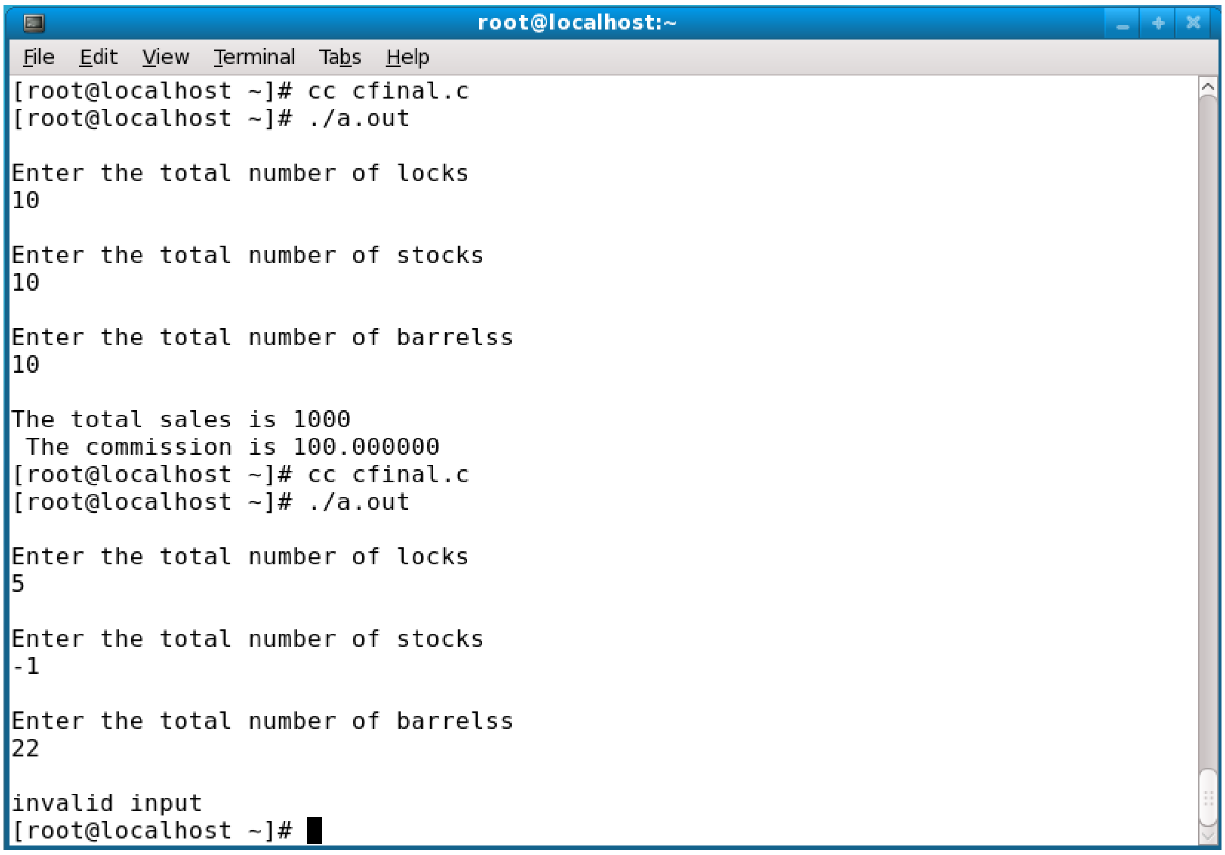
t_sales	20→21	20,21	5	5	5	500	
t_sales	20→25	20,21,25	15	15	15	1500	
commission	23→36	23→36	5	5	5		50
commission	27→36	27,28,36	15	15	15		175
commission	32→36	32,33,34, 36	25	25	25		360

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.

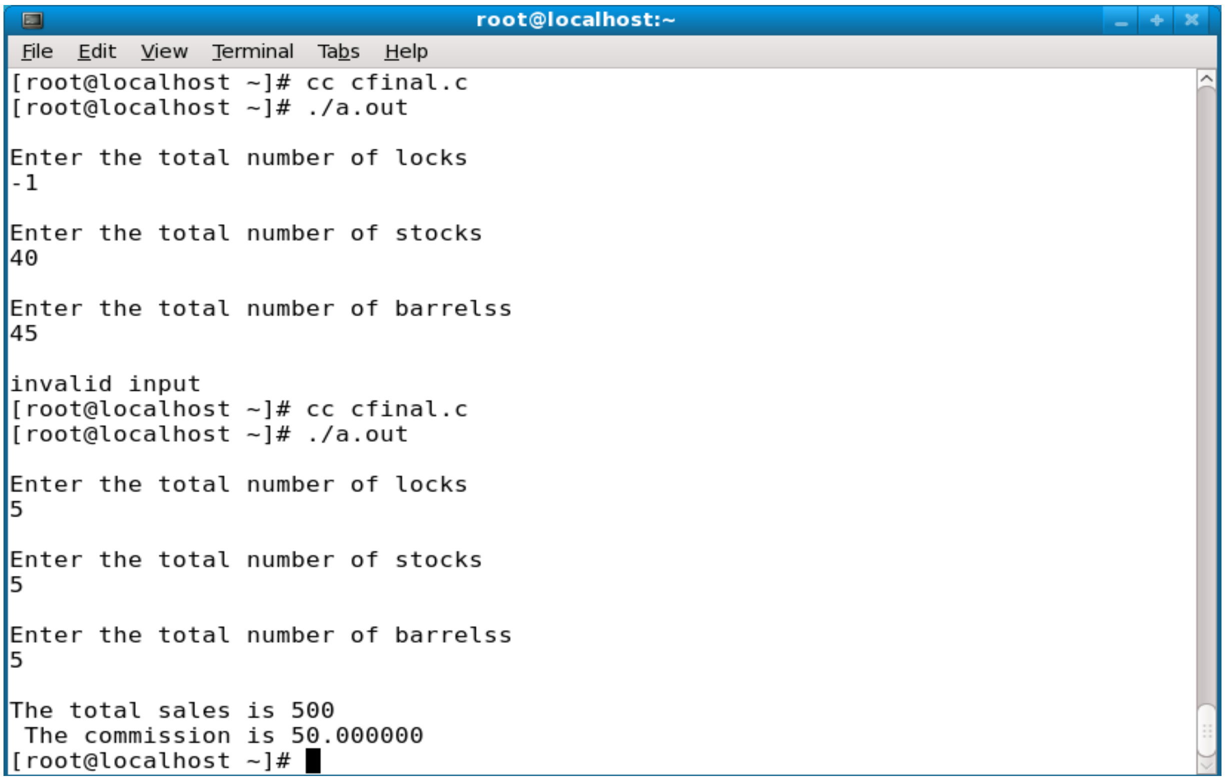
9.6 SNAPSHOTS:

1. Snapshot for Total sales and commission when total sales are within 1000 and Invalid input



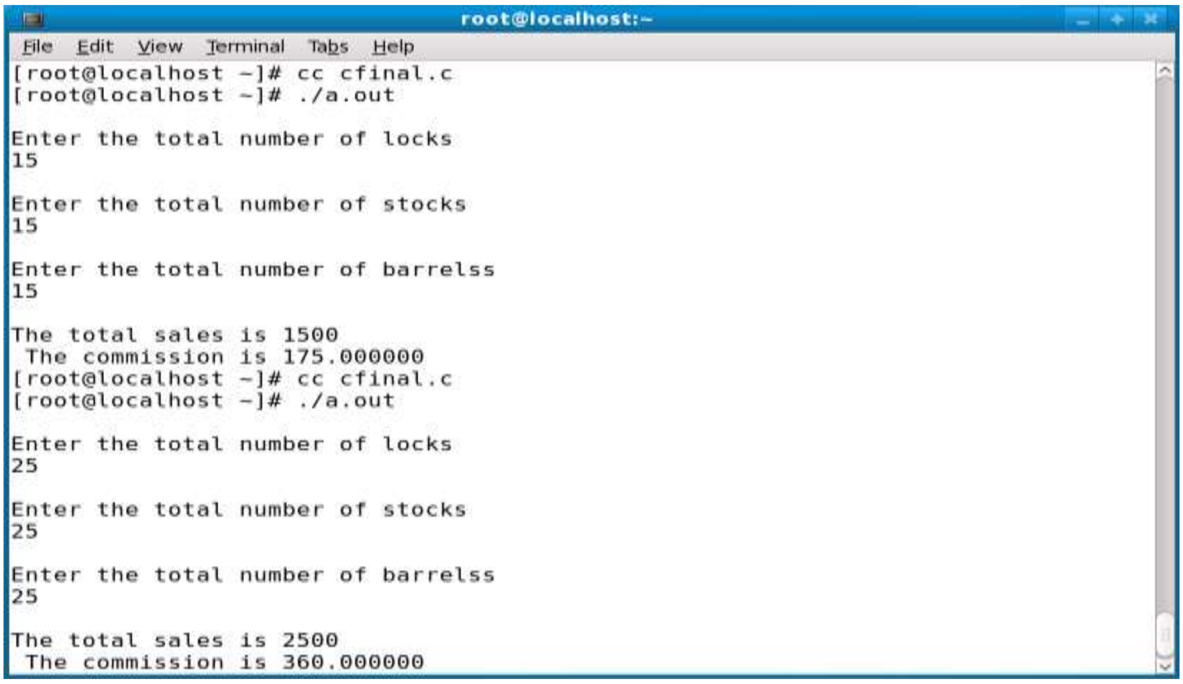
```
[root@localhost ~]# cc cfinal.c
[root@localhost ~]# ./a.out

Enter the total number of locks
10

Enter the total number of stocks
10

Enter the total number of barrelss
10

The total sales is 1000
 The commission is 100.000000
[root@localhost ~]# cc cfinal.c
[root@localhost ~]# ./a.out

Enter the total number of locks
5

Enter the total number of stocks
-1

Enter the total number of barrelss
22

invalid input
[root@localhost ~]# 
```

2. Invalid Input and Total sales and commission when total sales are within 1000



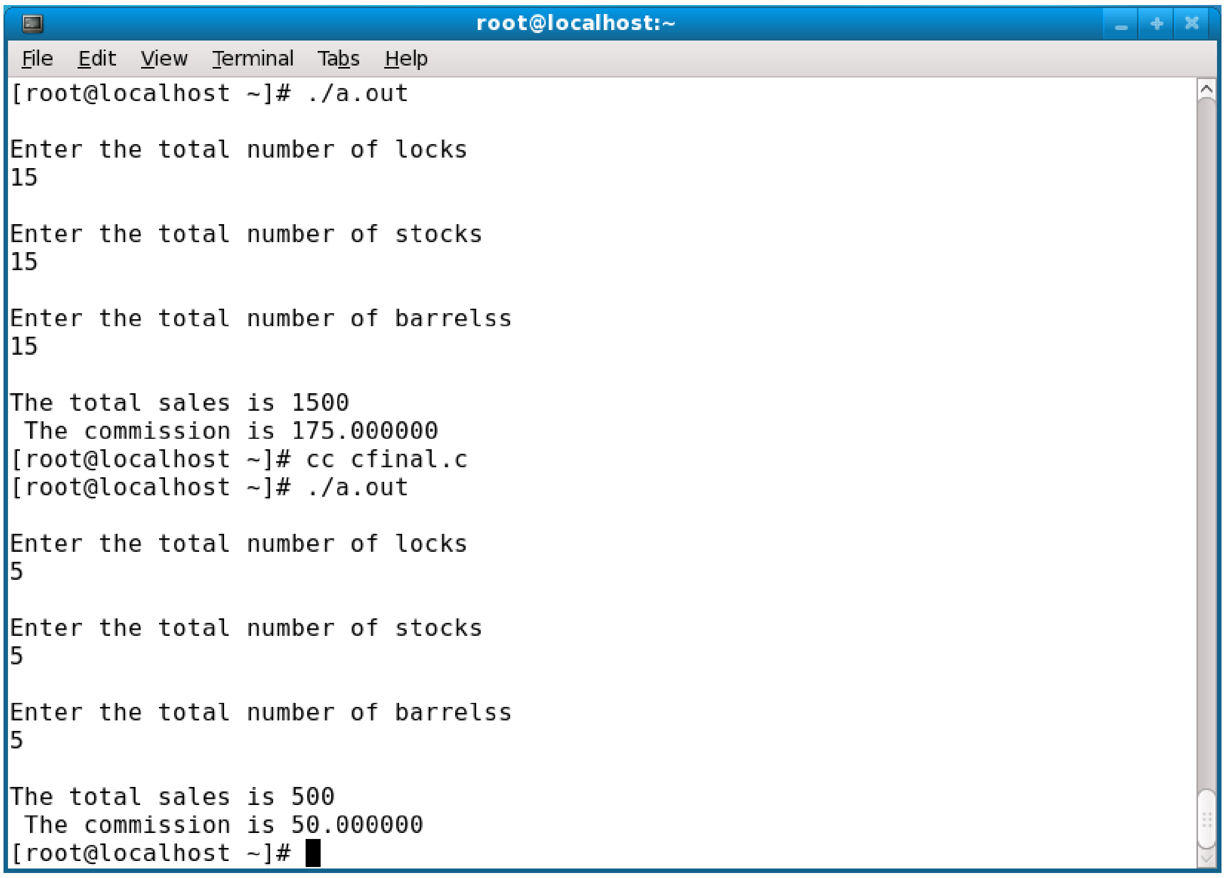
```
[root@localhost ~]# cc cfinal.c
[root@localhost ~]# ./a.out

Enter the total number of locks
-1

Enter the total number of stocks
40

Enter the total number of barrelss
45

invalid input
[root@localhost ~]# cc cfinal.c
[root@localhost ~]# ./a.out

Enter the total number of locks
5

Enter the total number of stocks
5

Enter the total number of barrelss
5

The total sales is 500
 The commission is 50.000000
[root@localhost ~]# 
```

3. Snapshot for for Total sales and commission when total sales are within 1800 and to find out the total commission 360



```
[root@localhost ~]# cc cfinal.c
[root@localhost ~]# ./a.out

Enter the total number of locks
15

Enter the total number of stocks
15

Enter the total number of barrelss
15

The total sales is 1500
 The commission is 175.000000
[root@localhost ~]# cc cfinal.c
[root@localhost ~]# ./a.out

Enter the total number of locks
25

Enter the total number of stocks
25

Enter the total number of barrelss
25

The total sales is 2500
 The commission is 360.000000
```

4. Snapshot for total sales and commission



```
[root@localhost ~]# ./a.out

Enter the total number of locks
15

Enter the total number of stocks
15

Enter the total number of barrelss
15

The total sales is 1500
 The commission is 175.000000
[root@localhost ~]# cc cfinal.c
[root@localhost ~]# ./a.out

Enter the total number of locks
5

Enter the total number of stocks
5

Enter the total number of barrelss
5

The total sales is 500
 The commission is 50.000000
[root@localhost ~]#
```

9.7 REFERENCES

1. Requirement Specification.
2. Assumptions.

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 SPECIFICATION

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

We use integer array as a data structure 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 CODE:

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

Int a[20],n,low,high,mid,key,I;
int flag=0;
clrscr();
printf("Enter the value of n:\n");
scanf("%d",&n);
if(n>0)
{
        printf("Enter %d elements in ASCENDING order\n",n);
        for(i=0;i<n;i++)
        {
        scanf("%d",&a[i]);
        }
        printf("Enter the key element to be searched\n");
        scanf("%d",&key);
        low-=0;
        high=n-1;
while(low<=high)
        {
          mid=(low+high)/2;
          if(a[mid]==key)
        {
           flag=1;
            break;
 }
           else if(a[mid]<key)
        {
 low=mid+1;

          }
          else
          {
           high=mid-1;
 }
        }
        if(flag==1)
        printf("Successful   search\nElement   found   at Location
              %d\n",mid+1);
       else
        printf("Key element is not found");
  }
        else
         printf("Wrong input");
        getch();
        return 0;
 }
```

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:

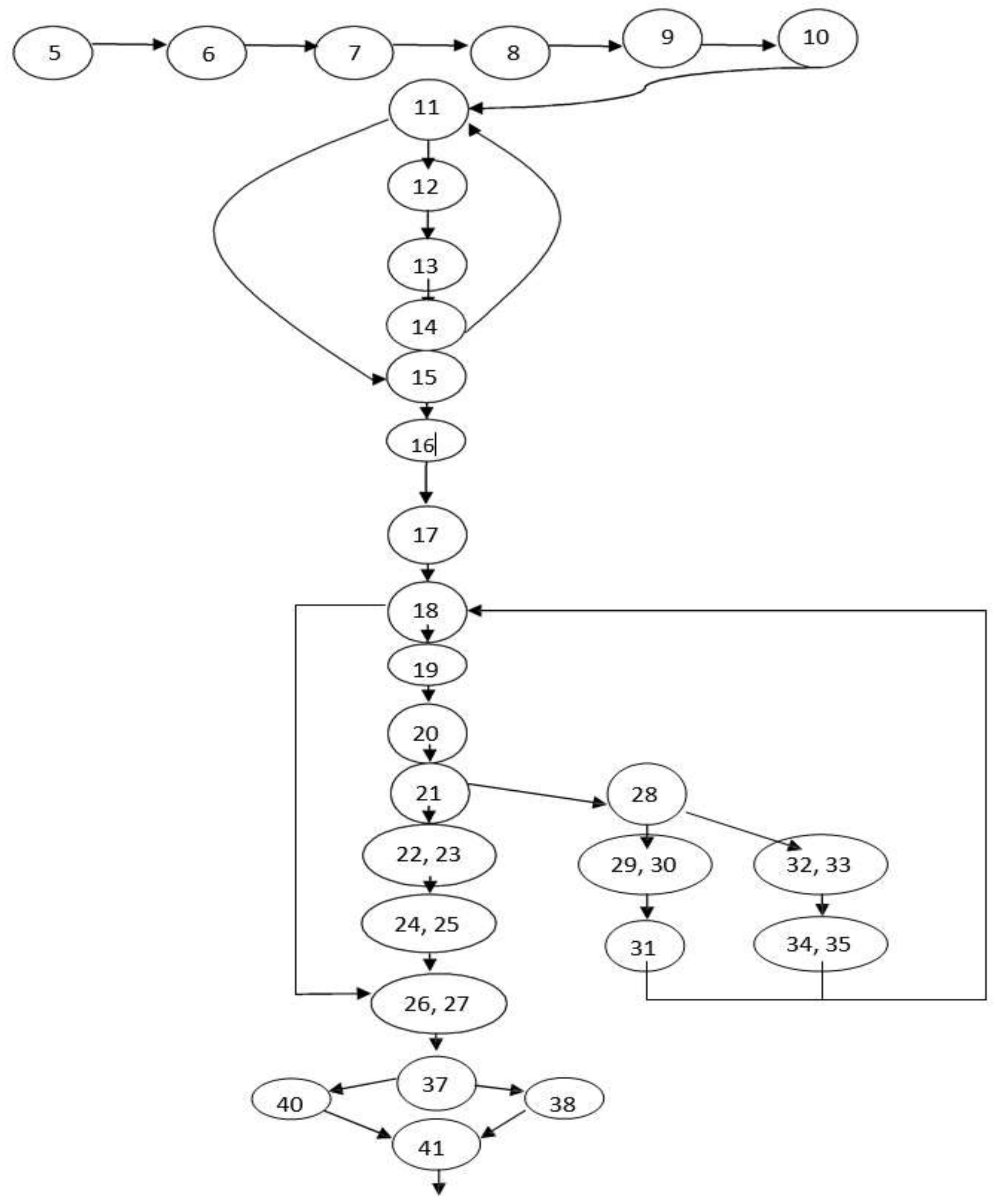
1. Compute the program graph.

2. Calculate the cyclomatic complexity.

3. Select a basis set of paths.

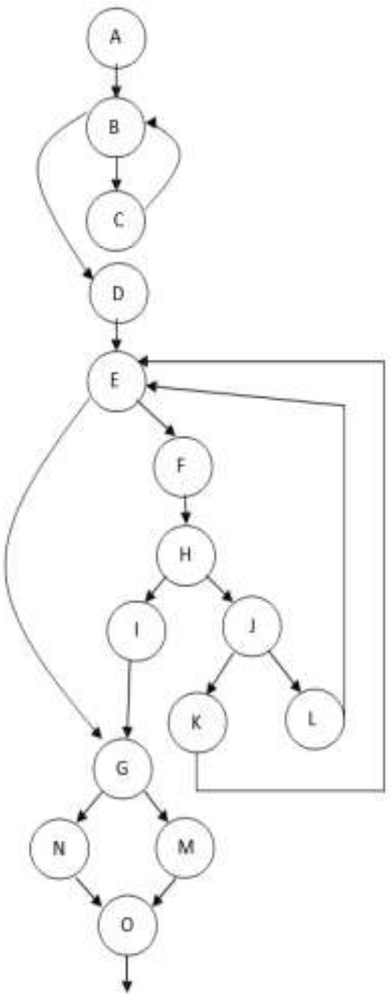
4. Generate test cases for each of these paths.

Below is the program graph of binary search code.



Using the program graph we derive (Decision-to-Decision) DD path graph for Binary search program

Program Graph Nodes	DD – Path Name
First	5
A	6,7,8,9,10
B	11
C	12,13,14
D	15,16,17
E	18
F	19,20
G	37
H	21
I	22,23,24,25,26,27
J	28
K	29.30,31
L	32,33,34,35
M	38
N	40
O	41



The cyclomatic complexity of a connected graph is provided by the formula V(G) = e – n + 2p. The number of edges is represented by e, the number of nodes by n and the number of connected regions by p. If we apply this formula to the graph given below, the number of linearly independent circuits is:

Number of edges = 21

Number of nodes = 15

Number of connected regions = 1

21 – 15 + 2(1) = 4.

Here we are dealing code level dependencies, which are absolutely incompatible with the latent assumption, that basis path are independent. McCabe's procedure successfully identifies basis path that are topologically independent, but when these contradict semantic dependencies, topologically possible paths are seen to be logically infeasible. One solution to this problem is to always require that flipping a decision result in a semantically feasible path. For this problem we identify some of the rules:

If node C not traversed, then node M should be traversed.

If node E and node G is traversed, then node M should be traversed.

If node I is traversed, then node N should be traversed.

Taking into consideration the above rules, next step is to find the basis paths.

According to cyclomatic complexity 4 feasible basis path exists:

P1: A, B, D, E, G, N, O	if n value is 0.
P2: A, B, C, B, D, E, F, H, I, G, M, O	key element found.
P3: A, B, C B, D, E, F, H, J, K, E, F, H, J, K, E, G, N, O	key element not found.
P4: A, B, C, B, D, E, F, H, J, L, E, F, H, J, L, E, G, N, O	key element not found.

DERIVING TEST CASES USING BASIS PATH TESTING

The last step is to devise test cases for the basis paths.

TEST CASES

TC ID	Test Case Description	Value of 'n'	array elements	key	Expected Output	Actual Output	Status
1	Testing for requirement 1 Path P1	0	--	5	key not found		
2	Testing for requirement 2 Path P2	4	2,3,5,6,7	5	Key found at position 3		
3	Testing for requirement 2 Path P3	3	1,2,5	6	key not found		
4	Testing for requirement 2 Path P4	3	1,2,5	1	key not found		
5	Testing for requirement 2 Path P4+P2-P1	5	1,2,4,6,7	2	Key found at position 2		
6	Testing for requirement 2 Path P3+P2-P1	5	4,5,7,8,9	8	key found at position		

10.5 EXECUTION & RESULT DISCUSION:

Execute the program against the designed test cases and complete the table for

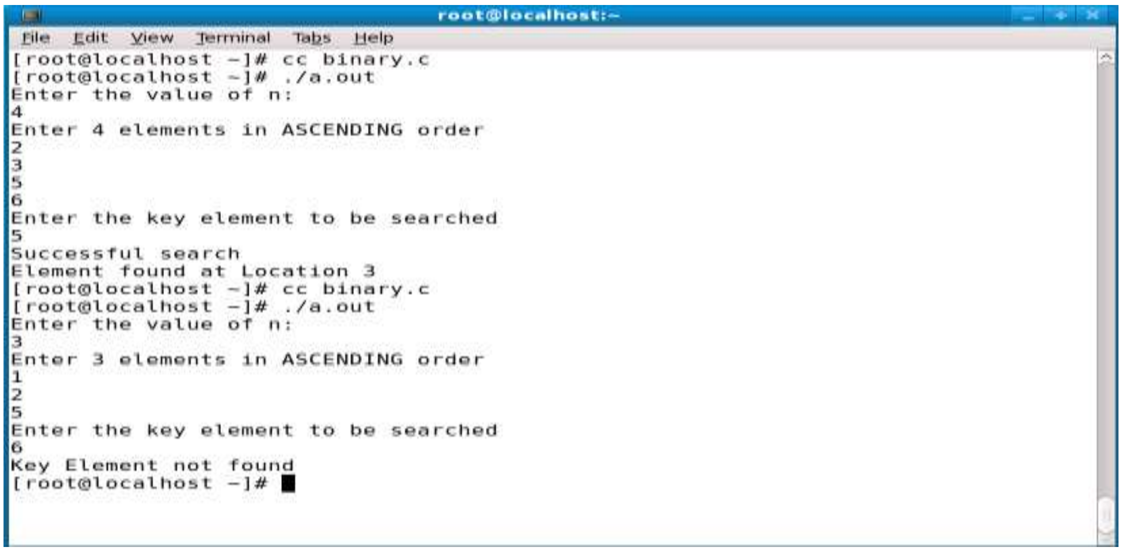
Actual output column and status column.

Test Report:

1. No of TC's Executed: **06**
2. No of Defects Raised:
3. No of TC's Pass:
4. No of TC's Failed:

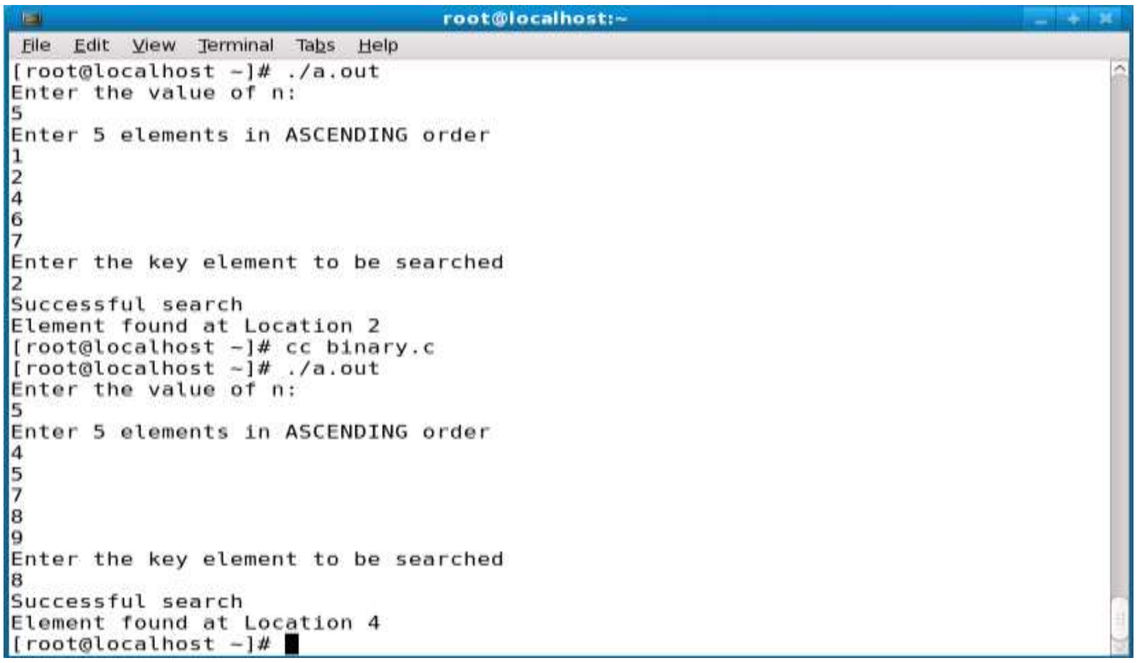
10.6 SNAPSHOTS:

1. Snapshot to check successful search and not found key element.



```
[root@localhost ~]# cc binary.c
[root@localhost ~]# ./a.out
Enter the value of n:
4
Enter 4 elements in ASCENDING order
2
3
5
6
Enter the key element to be searched
5
Successful search
Element found at Location 3
[root@localhost ~]# cc binary.c
[root@localhost ~]# ./a.out
Enter the value of n:
3
Enter 3 elements in ASCENDING order
1
2
5
Enter the key element to be searched
6
Key Element not found
[root@localhost ~]# █
```

2.Snapshot to check successful search and not found key element.



```
[root@localhost ~]# ./a.out
Enter the value of n:
5
Enter 5 elements in ASCENDING order
1
2
4
6
7
Enter the key element to be searched
2
Successful search
Element found at Location 2
[root@localhost ~]# cc binary.c
[root@localhost ~]# ./a.out
Enter the value of n:
5
Enter 5 elements in ASCENDING order
4
5
7
8
9
Enter the key element to be searched
8
Successful search
Element found at Location 4
[root@localhost ~]# █
```

10.7 REFERENCES:

1. Requirement Specification
2. Assumptions

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. discuss the test results.

11.1 REQUIREMENTS SPECIFICATION

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.

11.2 DESIGN

We use integer array as a data structure to store ‘n’ number of elements.

Iterative programming technique is used.

11.3 PROGRAM CODE:

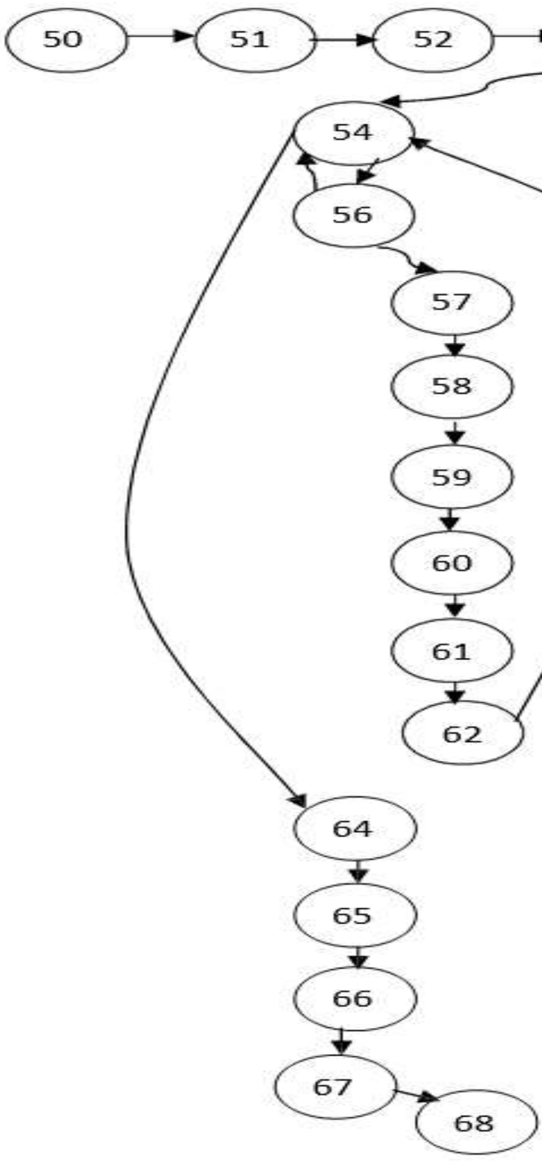
```
// An iterative implementation of quick
sort
1       #include <stdio.h>

    // A utility function to swap two elements
2       void swap ( int* a, int* b )
{
 3      int t = *a;
4       *a = *b;
5       *b = t;
6       }

/* This function is same in both iterative and recursive*/
7       int partition (int arr[], int l, int h)
8       {
9       int x = arr[h]; int i = (l - 1),j;
```

```
11          for (j = l; j <= h- 1; j++)
12        {
13          if (arr[j] <= x)
14    {
15         i++;
16         swap (&arr[i], &arr[j]);
17   }
18         }
19         swap (&arr[i + 1], &arr[h]);
20        return (i + 1);
 21   }

/* A[] --> Array to be sorted, l  --> Starting index, h  --> Ending index */

22       void quickSortIterative (int arr[], int l, int h)
23   {
   // Create an auxiliary
24      int  stack[10],p;

// initialize top of
25      int top = -1;

// push initial values of l and h to
26     stack stack[ ++top ] = l;
27     stack[ ++top ] = h;

// Keep popping from stack while is not empty
 28    while ( top >= 0 )
 29   {
      // Pop h and l
30    h = stack[ top-- ];
31    l = stack[ top-- ];
```

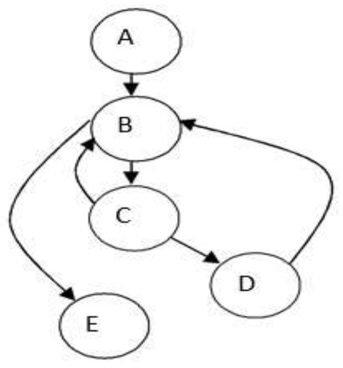
```
        // Set pivot element at its correct position in sorted
        array
      32   p = partition( arr, l, h );

      // If there are elements on left side of pivot, then push left
      // side to stack
      33   if ( p-1 > l )
      34   {
      35     stack[ ++top ] = l;
      36     stack[ ++top ] = p - 1;
      37   }

      // If there are elements on right side of pivot, then push right
      // side to
      38   stack if ( p+1 < h )
       39    {
       40      stack[ ++top ] = p + 1;
       41       stack[ ++top ] = h;
       42   }
       43      }
       44         }
// Driver program to test above
functions int main()
{
   int arr[20],n,i;
   clrscr();
   printf("Enter the size of the array");
   scanf("%d",&n);
   printf("Enter %d elements",n);
   for(i=0;i<n;i++)
         scanf("%d",&arr[i]);
         quickSortIterative( arr, 0, n - 1 );
         printf("Elements of the array are;");
        for(i=0;i<n;i++)
        printf("%d",arr[i]);
       getch();
       return 0;
}
```

11.4 TESTING

Program Graph for partition:



DD Path Graph:



Using program graph we derive DD path graph for partition()

DD Path Names	Program Graph
A	50,51,52,53
B	54
C	56
D	57,58,59,60,61,62
E	64,65,66,67,68

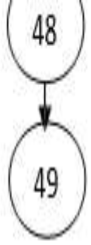
Cyclomatic complixity
No. of edges =6
No. of
nodes=5
e-n+2
6-5+2 =3
No. of predicate nodes +1 (i.e., node B and node C)
2+1=3
No. of region + 1
R1 and R2 are two regions
2+1=3

According to cyclomatic complexity 3 basis path exists. They are,

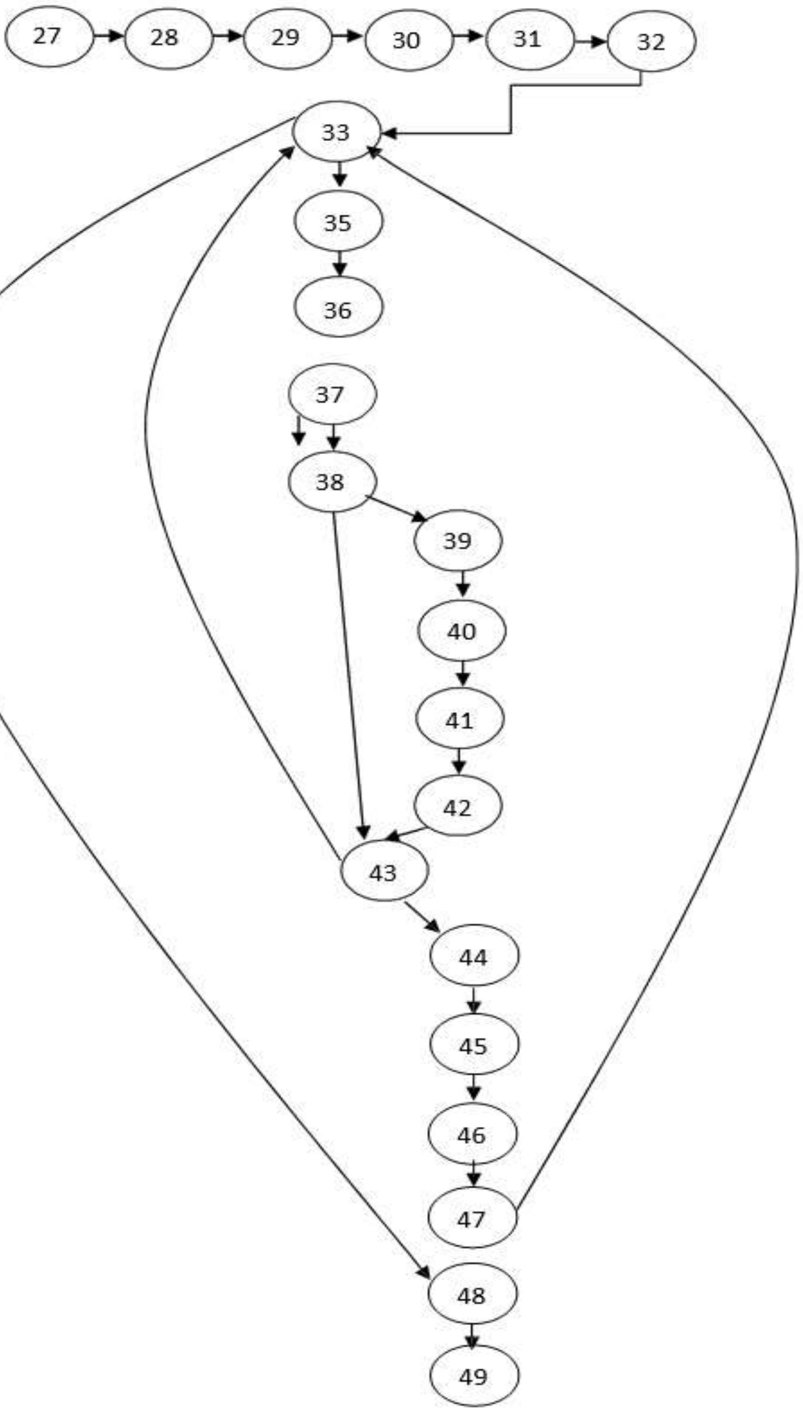
P1: A, B, E
P2: A, B, C,
D, B, E P3:
A, B, C, B, E

Deriving test cases using basis path testing: Test Cases

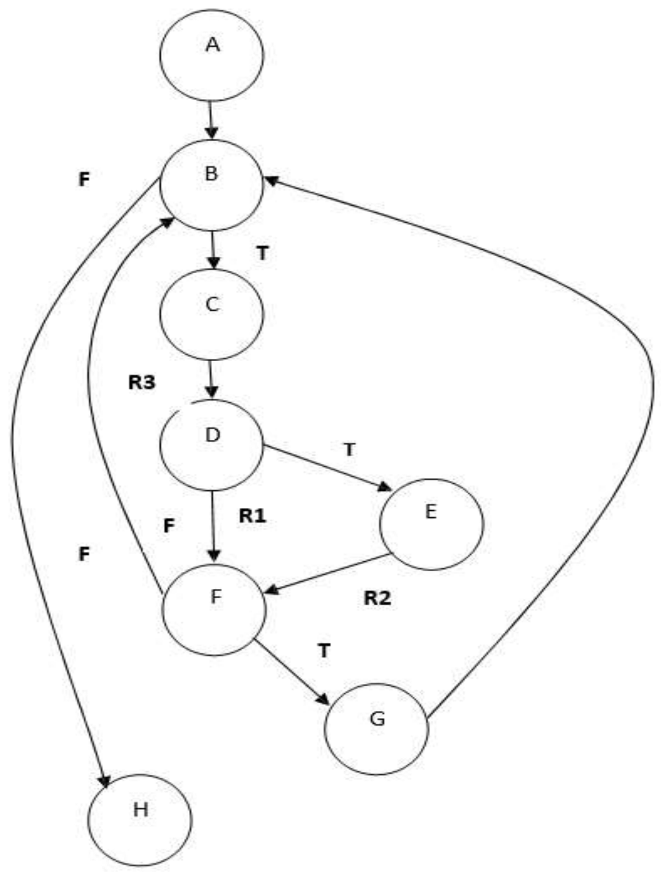
TC ID	Test Case Descriptio	Array element s	Expected output		Actual output	Status
			Array	Value of i		
1	Testing for path P1	5	5	0		
2	Testing for path	5, 4, 6, 2, 7	5, 4, 6, 2, 7	4		
3	Testing for path	5, 4, 6, 7, 5	5, 4, 6, 7, 5	0		



Program Graph for Quick sort()



DD Path Graph



CYCLOMATIC COMPLEXITY

No. of nodes = 8
No. of nodes =10
e-n+2
10-8+2 =4
No. of predicate nodes + 1
3+1=4 (i.e., node B, D & F)
No. of regions+1
3+1=4 (i.e., Region R1, R2 & R3)

According to cyclomatic complexity 4 basis path exists. They are
P1: A, B, C, D, E,
F, G, B, H P2: A,
B, C, D, E, F, B,H
P3: A, B, C, D, F, G, B, H
P4: A, B, C, D, F, B, H

Deriving test cases using basis path testing
Test cases:

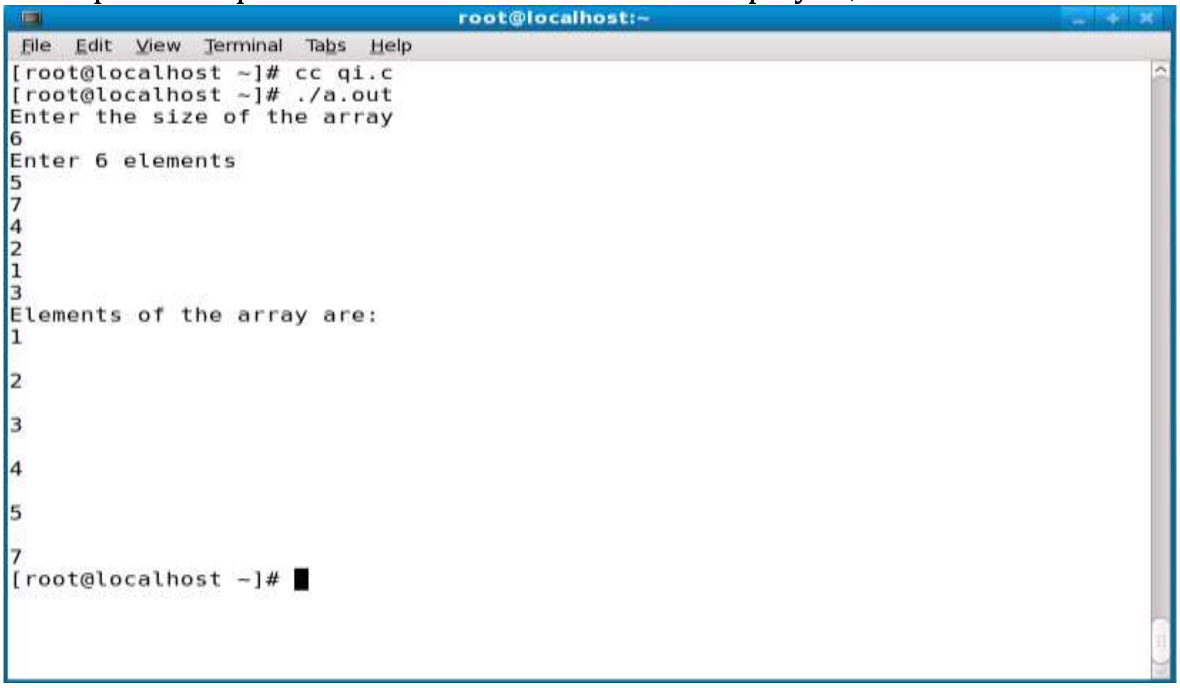
TC ID	Test Case Description	Array elements	Expected output	Actual output	Status
1	Testing for path 1	5, 7, 4, 2, 1, 3	2, 1, 3, 5, 7, 4		
2	Testing for path 2	5, 4, 8, 2, 7	5, 4, 2, 7, 8		
3	Testing for path 3	5, 4, 6, 7, 3	3, 4, 6, 7, 5		

11.5 EXECUTION

Compile the program and enter inputs Test above table array elements for test cases.

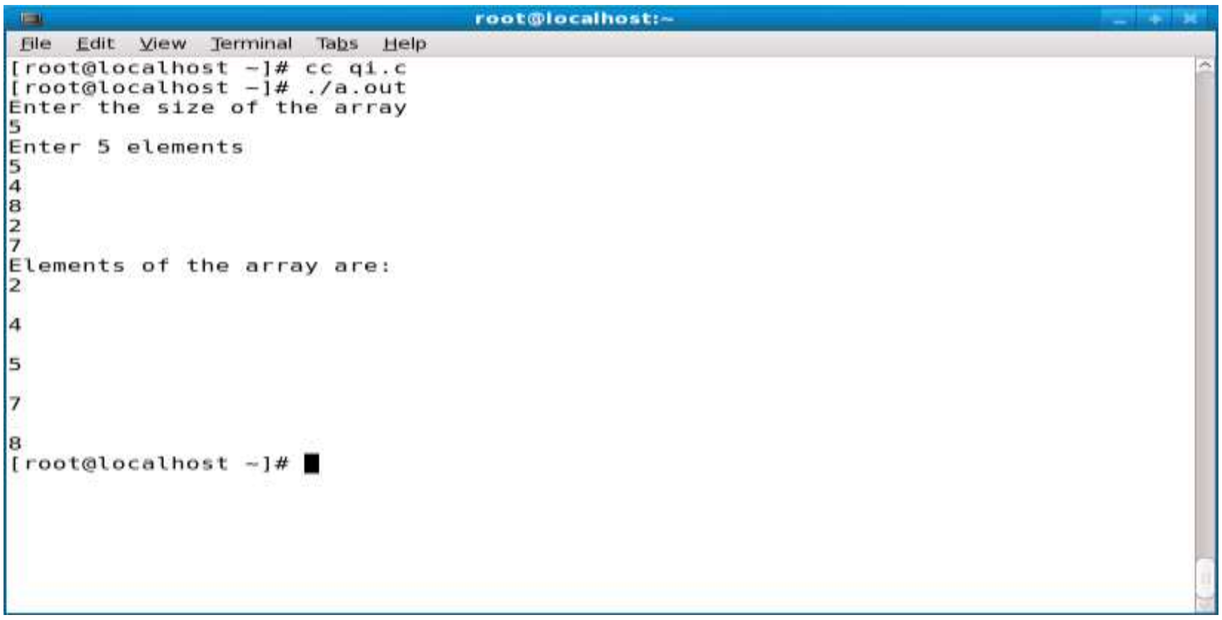
11.6 SNAPSHOTS:

1. Snapshot of quick sort sorted elements are displayed, when the n=6



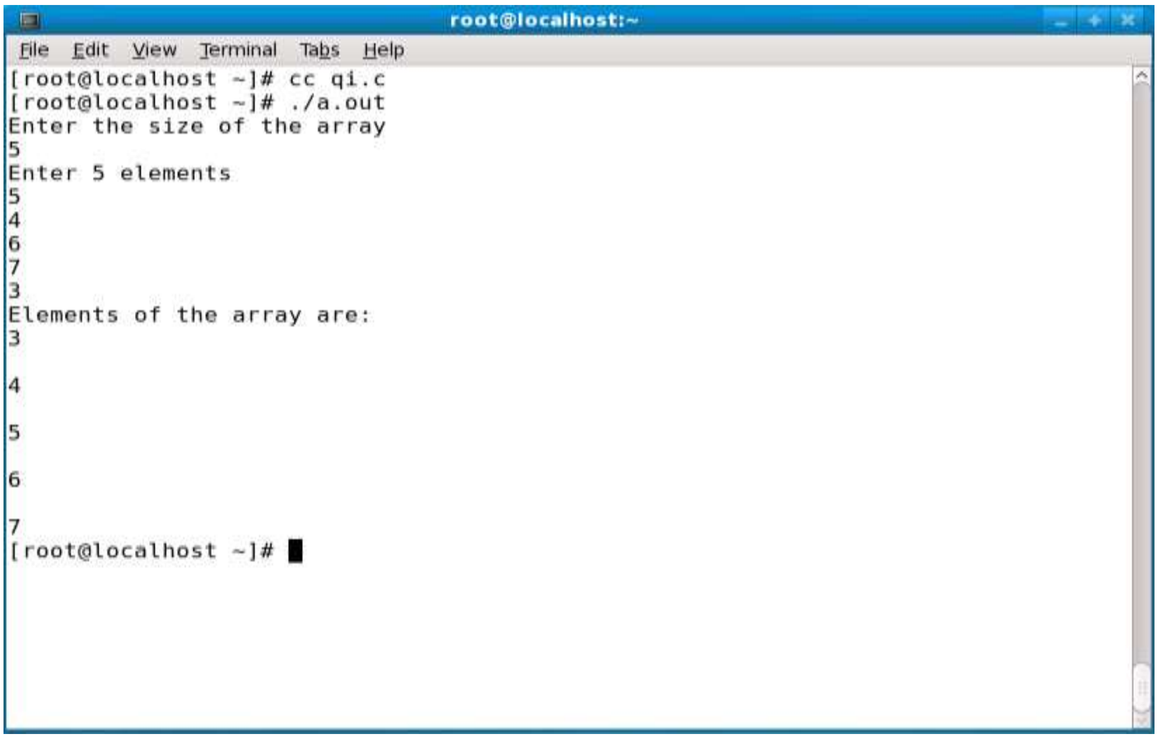
```
[root@localhost ~]# cc qi.c
[root@localhost ~]# ./a.out
Enter the size of the array
6
Enter 6 elements
5
7
4
2
1
3
Elements of the array are:
1

2

3

4

5

7
[root@localhost ~]# 
```

2. Snapshot of quick sort sorted elements are displayed, when the n=5



```
[root@localhost ~]# cc qi.c
[root@localhost ~]# ./a.out
Enter the size of the array
5
Enter 5 elements
5
4
8
2
7
Elements of the array are:
2

4

5

7

8
[root@localhost ~]# 
```

3. Snapshot of quick sort sorted elements are displayed, when the n=5



```
[root@localhost ~]# cc qi.c
[root@localhost ~]# ./a.out
Enter the size of the array
5
Enter 5 elements
5
4
6
7
3
Elements of the array are:
3

4

5

6

7
[root@localhost ~]# 
```

11.7 REFERENCES:

1. Requirement Specification
2. Assumptions

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 SPECIFICATION:

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

i.e., for example, 1<=marks<=100
1<=kannada<=100
1<=maths<=100 etc.

R2: If R1 is satisfied compute average of marks scored and percentage of the same and depending on percentage display the grade.

12.2 DESIGN:

We use the total percentage of marks to grade the student marks.

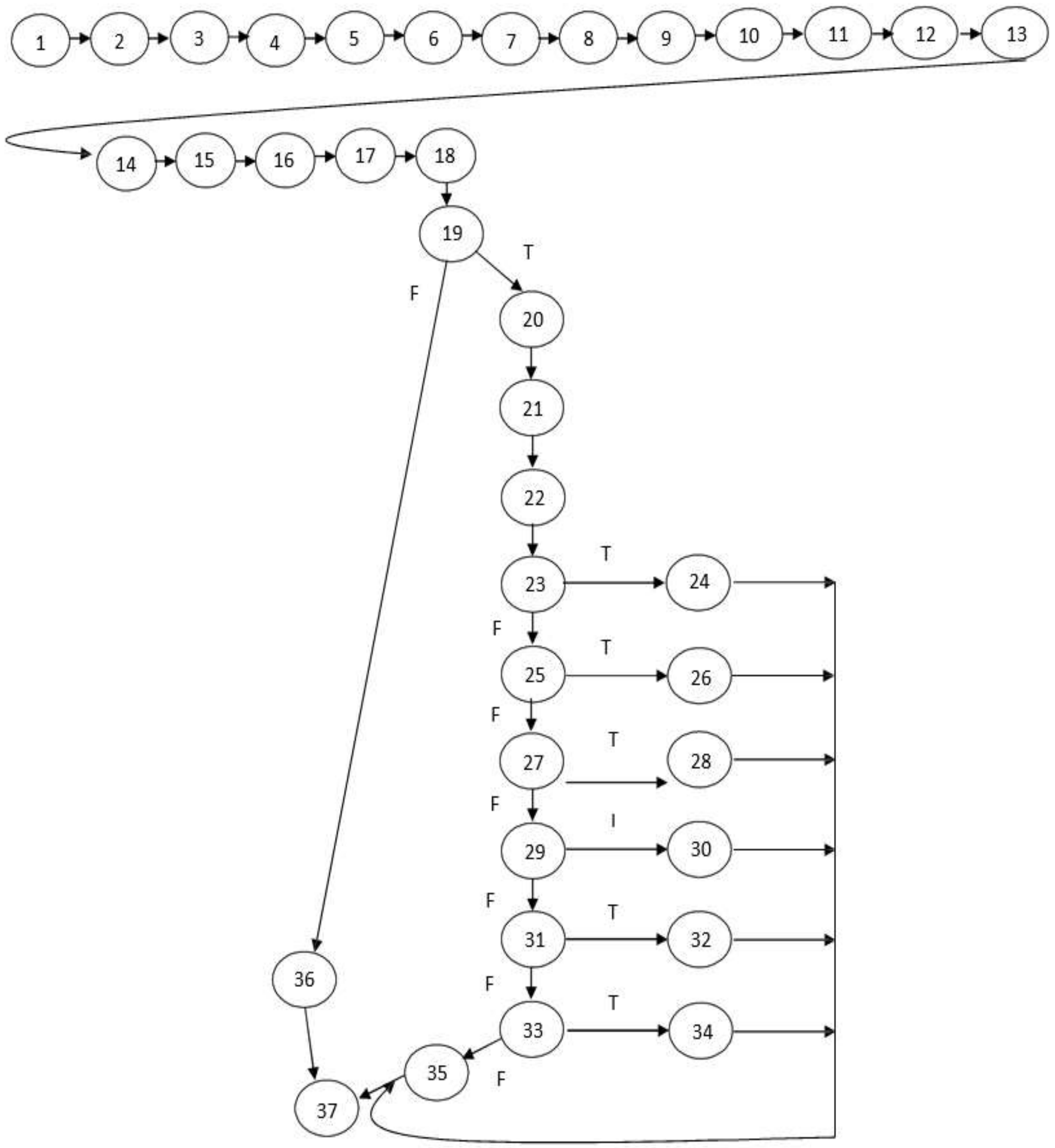
<35 && >0 of percentage make it as FAIL
avmar<=40 && avmar>35 make it as Grade C
avmar<=50 && avmar>40 make it as Grade
C+ avmar<=60 && avmar>50 make it as
Grade B avmar<=70 && avmar>60 make it
as Grade B+ avmar<=80 && avmar>70 make
it as Grade A avmar<=100 && avmar>80
make it as Grade A+

12.3 PROGRAM CODE:

```
#include<stdio.h>
main()
1     {
2     float  kan,eng,hindi,maths,science, sst,avmar;
3     printf("Letter Grading\n");
4      printf("SSLC Marks Grading\n");
5      printf("Enter the marks for 6  Kannada:");
 6    scanf("%f",&kan);
printf("enter the marks for English:");
scanf("%f",&eng);
printf("enter the marks for Hindi:");
scanf("%f",&hindi);
printf("enter the marks for Maths");
scanf("%f",&maths);
printf("enter the marks for Science:");
scanf("%f",&science);
printf("enter the marks for Social Science:");
scanf("%f",&sst);
avmar=(kan+eng+hindi+maths+science+sst)/6.25;
printf("the average marks are=%f\n",avmar);
 if((avmar<35)&&(avmar>0))
 printf("fail");
 elseif((avmar<=40)&&(avar>35))
printf("Grade C");
elseif((avmar<=50)&&(avmar>40))
printf("Grade C+");
elseif((avmar<=60)&&(avmar>50))
printf("Grade B");
elseif((avmar<=70)&&(avmar>60))
printf("Grade B+");
elseif((avmar<=80)&&(avmar>70))
printf("Grade A");
elseif((avmar<=100)&&(avmar>80))
printf("Grade A+");
else
printf(“Invalid”);
}
```

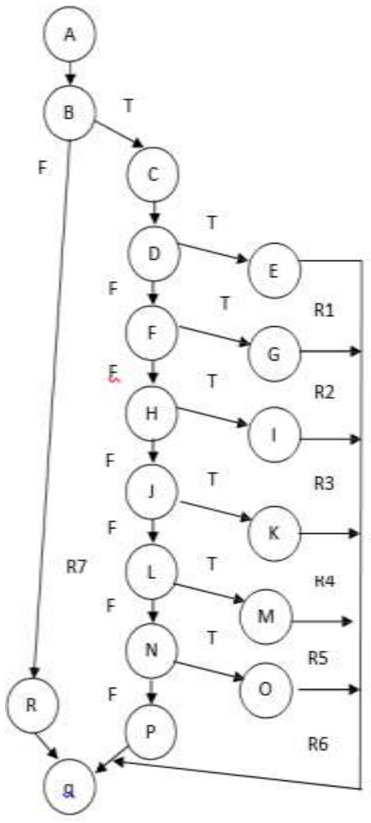
12.4 TESTING

PROGRAM GRAPH:



Using the program graph derive DD path graph

DD path Names	Program Graph Nodes
A	1, 2, 3, 4, 5, 6, 7, 8 . . . 18
B	19
C	20, 21, 22
D	23
E	24
F	25
G	26
H	27
I	28
J	29
K	30
L	31
M	32
N	33
O	34
P	35
Q	37
R	36



CYCLOMATIC COMPLEXITY
No. of nodes = 18
No. of edges = 24
e-n+2
24-18+2=8

No. of predicate nodes + 1
7 + 1 = 8 (i.e., B, D, F, H, J, L, N)

No. of regions + 1
7 + 1 = 8 (i.e., Regions R1, R2, R3, R4, R5, R6, R7)

According to cyclomatic complexity we can derive 8 basis path.
P1: A, B, R q
P2: A, B, C, D, E, q
P3: A, B, C, D, F, G, q
P4: A, B, C, D, F, H, I, q
P5: A, B, C, D, F, H, J, K, q
P6: A, B, C, D, F, H, J, L, M, q
P7: A, B, C, D, F, H, J, L, N, O, q
P8: A, B, C, D, F, H, J, L, N, P, q

Test Cases:

TC ID	Test Description	Input	Expected Output	Actual Output	Status
1	Testing for path P1	K=50 E=50 H=50 M=50 S=50 SST=150	Invalid Input		

2	Testing for path P2	K=30 E=30 H=30 M=35 S=35 SST=35 Avg=32.5	Fa il		
3	Testing for path P3	K=40 E=38 H=37 M=40 S=40 SST=38 Avg=38.8	Grade C		
4	Testing for path P4	K=45 E=47 H=48 M=46 S=49 SST=50 Avg=47.5	Grade C+		
5	Testing for path P5	K=55 E=58 H=60 M=56 S=57 SST=60 Avg=57.66	Grade B		
6	Testing for path P6	K=65 E=65 H=65 M=65 S=65 SST=65 Avg=65.0	Grade B+		

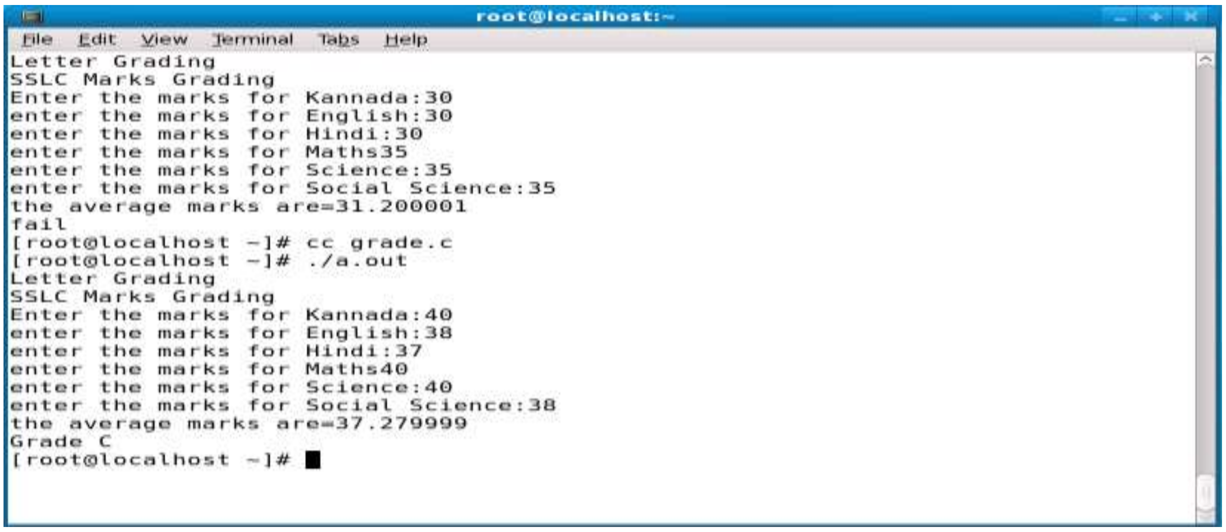
7	Testing for path P7	K=75 E=72 H=78 M=75 S=80 SST=80 Avg=76.6	Grade A		
8	Testing for path P8	K=85 E=90 H=80 M=95 S=85 SST=85 Avg=86.6	Grade A+		

12.5 EXECUTION

Compile the program and enter inputs for subject marks, then it will display the Total percentage, depending on the percentage it will shows the Grade and test the test cases for above table.

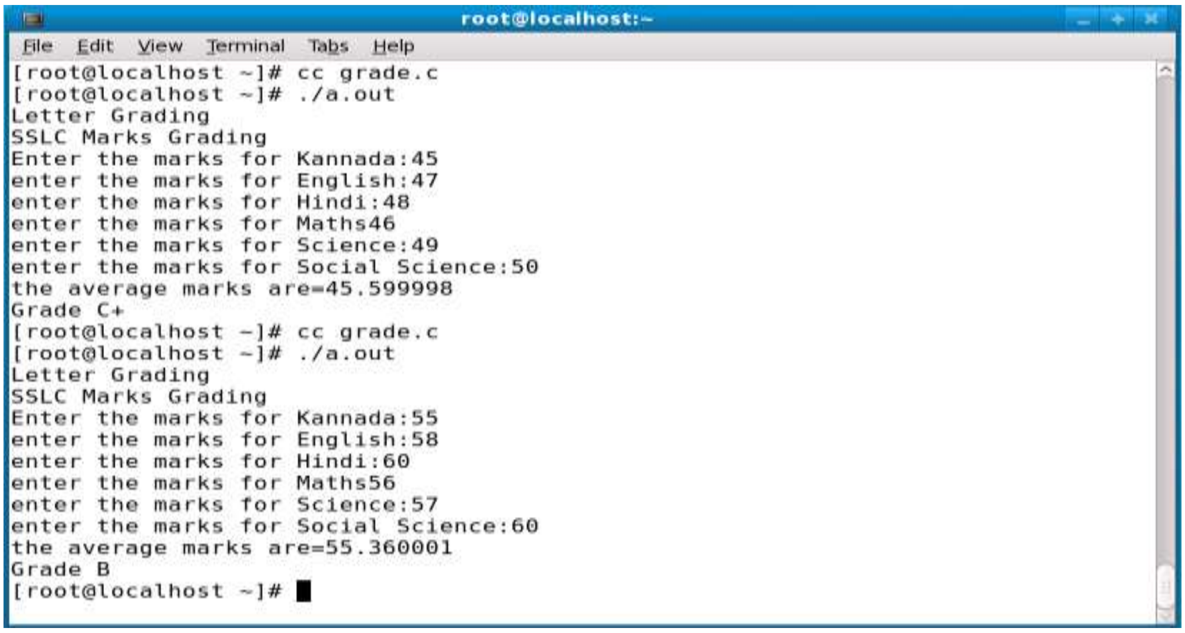
12.6 SNAPSHOTS:

1. Snapshot to Show Fail and Grade C



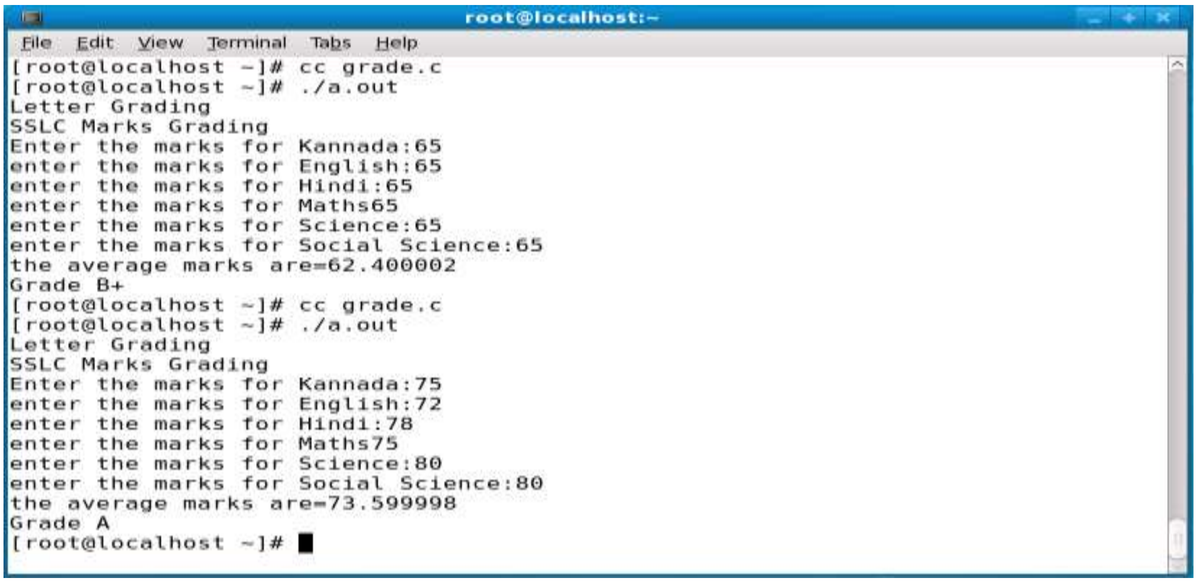
```
Letter Grading
SSLC Marks Grading
Enter the marks for Kannada:30
enter the marks for English:30
enter the marks for Hindi:30
enter the marks for Maths35
enter the marks for Science:35
enter the marks for Social Science:35
the average marks are=31.200001
fail
[root@localhost ~]# cc grade.c
[root@localhost ~]# ./a.out
Letter Grading
SSLC Marks Grading
Enter the marks for Kannada:40
enter the marks for English:38
enter the marks for Hindi:37
enter the marks for Maths40
enter the marks for Science:40
enter the marks for Social Science:38
the average marks are=37.279999
Grade C
[root@localhost ~]# 
```

2. Snapshot to show Grade B and Grade C+



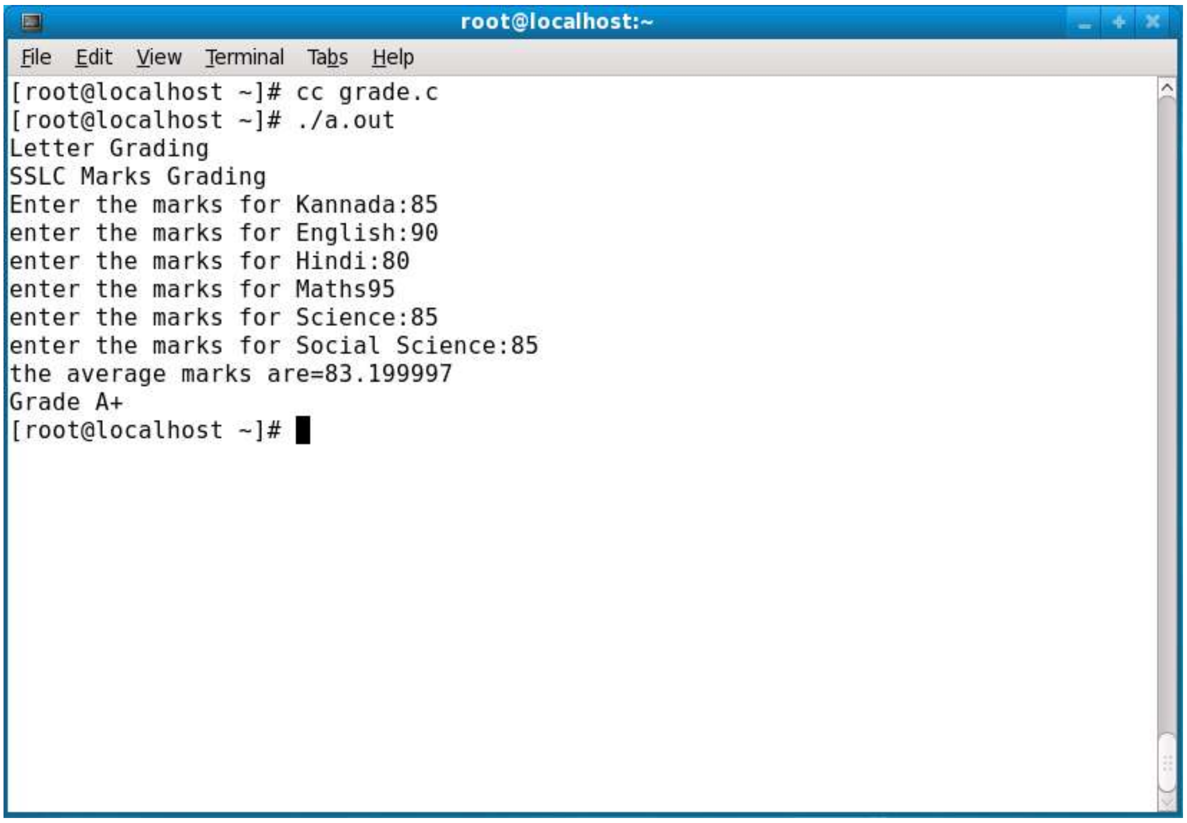
```
[root@localhost ~]# cc grade.c
[root@localhost ~]# ./a.out
Letter Grading
SSLC Marks Grading
Enter the marks for Kannada:45
enter the marks for English:47
enter the marks for Hindi:48
enter the marks for Maths46
enter the marks for Science:49
enter the marks for Social Science:50
the average marks are=45.599998
Grade C+
[root@localhost ~]# cc grade.c
[root@localhost ~]# ./a.out
Letter Grading
SSLC Marks Grading
Enter the marks for Kannada:55
enter the marks for English:58
enter the marks for Hindi:60
enter the marks for Maths56
enter the marks for Science:57
enter the marks for Social Science:60
the average marks are=55.360001
Grade B
[root@localhost ~]# █
```

3. Snapshot to show the Grade A and Grade B+



```
[root@localhost ~]# cc grade.c
[root@localhost ~]# ./a.out
Letter Grading
SSLC Marks Grading
Enter the marks for Kannada:65
enter the marks for English:65
enter the marks for Hindi:65
enter the marks for Maths65
enter the marks for Science:65
enter the marks for Social Science:65
the average marks are=62.400002
Grade B+
[root@localhost ~]# cc grade.c
[root@localhost ~]# ./a.out
Letter Grading
SSLC Marks Grading
Enter the marks for Kannada:75
enter the marks for English:72
enter the marks for Hindi:78
enter the marks for Maths75
enter the marks for Science:80
enter the marks for Social Science:80
the average marks are=73.599998
Grade A
[root@localhost ~]# █
```

4. Snapshot to show the Grade A+



```
[root@localhost ~]# cc grade.c
[root@localhost ~]# ./a.out
Letter Grading
SSLC Marks Grading
Enter the marks for Kannada:85
enter the marks for English:90
enter the marks for Hindi:80
enter the marks for Maths95
enter the marks for Science:85
enter the marks for Social Science:85
the average marks are=83.199997
Grade A+
[root@localhost ~]# 
```

12.7 REFERENCES:

1. Requirement Specification
2. Assumptions

EXECUTION STEPS IN LINUX

1. Open Terminal
2. Then open **VI –Editor** using the filename, following command will shows that

   [root@localhost ~]# vi   Triangle.c

3. Write the Suitable code for the given program
4. Then compile and execute the program using the command;

   [root@localhost ~]# cc triangle.c

5. Then execute the command;

   [root@localhost ~]# ./a.out

6. Enter the suitable input for the program.
7. Then will get the suitable output.