INSTITUTIONAL MISSION AND VISION

MISSION:

To foster an intellectual and ethical environment in which both skill and spirit will thrive so as to impart high quality education, training, and service with an international outlook. To create and develop technocrats and business leaders who will strive to improve the quality of life.

VISION:

To become a world-class center in providing globally relevant higher education in the field of management, technology and applied science embedded with human values.

DEPARTMENT VISION AND MISSION

MISSION:

Don Bosco Bangalore, Department of Computer Science and Engineering would like to nurture thedevelopment of skill sets among the students in the area of software engineering, computer networking, database, computer graphics, operating systems, artificial intelligence, numerical and symbolic computation with the collaborative efforts of participants and industry

VISION:

Don Bosco Bangalore, Department languages with an orientation of Computer Science and Engineering looks forward to be a quality center of computation by imparting knowledge on various algorithms and data structures, programming methodology and of computer architecture as per the standards of the industry with cohesive efforts of its students and staffs.

SOFTWARE TESTING LABORATORY
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2019 - 2020)
SEMESTER – VI

Subject Code	18CSL67	IA Marks	40
Number of Lecture Hours/Week	01I + 02P	Exam Marks	60
Total Number of Lecture Hours	40	Exam Hours	03

CREDITS – 02

Description (If any):

Design, develop, and implement the specified algorithms for the following problems using any language of your choice under LINUX /Windows environment.

Lab Experiments:

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

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 equivalenceclass partitioning, execute the test cases and discuss the results.

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

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

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

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

9. Design, develop, code and run the program in any suitable language to solve the commission problem. Analyze it from the perspective of dataflow testing, derive different test cases, execute these test cases and discuss the test results.
10. Design, develop, code and run the program in any suitable language to implement the binary search algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.
11. Design, develop, code and run the program in any suitable language to implement the quicksort algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.
12. Design, develop, code and run the program in any suitable language to implement an absolute letter grading procedure, making suitable assumptions. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results

Study Experiment / Project:

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

Course outcomes: The students should be able to:

- Understand requirements for the given problem
- Design and implement the solution for given problem in any programming language(C,C++,JAVA)
- Discuss test cases for any given problem
- Apply the appropriate technique for the design of flow graph.
- Create appropriate document for the software artefact.

Conduction of Practical Examination:

1. All laboratory experiments are to be included for practical examination.
2. Students are allowed to pick one experiment from the lot.
3. Strictly follow the instructions as printed on the cover page of answer script for breakup of marks
4. Procedure + Conduction + Viva: **15 + 70 + 15 (100)**
5. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero

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 anyside is 10. Derive test cases for your program based on boundary-value analysis, execute the testcases and discuss the results.

AIM: The aim of the triangle problem is to check whether a given triangle is equilateral, isosceles, and scalene or not a triangle

SCOPE: To check whether the given three values of sides of a triangle form an equilateral, isosceles, scalene or not a triangle using boundary value testing approach.

1. TEST FUNCTION(PROGRAM)

```
#include<stdio.h>
main()
{
  int a,b,c;
  a=b=c=0;     //initialization

  printf("enter three integers which are sides of the triangle : \n");
  scanf("%d%d%d", &a,&b,&c);
  printf("Side A is:%d\t Side B is:%d\t Side C is:%d\n", a,b,c);

 //checks the ranges of the integers a, b and c
  if(!((1<=a) && (a<=10)) && (!((1<=b) && (b<=10))) && (!((1<=c)&&(c<=10))))
        printf("a, b & c is not in the range\n")
  else if(!((1<=a) && (a<=10)) && (!((1<=b) && (b<=10))))
        printf("a, b not in the range\n");
  else if(!((1<=b) && (b<=10)) && (!((1<=c) && (c<=10))))
        printf("b, c not in the range\n");
  else if(!((1<=a) && (a<=10)) && (!((1<=c) && (c<=10))))
        printf("a, c not in the range\n");
  else if(!((1<=a) && (a<=10)))
        printf("a is not in a range\n");
  else if(!((1<=b) && (b<=10)))
        printf("b is not in a range\n");
  else if(!((1<=c) && (c<=10)))
        printf("c is not in a range\n");

 //check the conditions whether three integers will form triangle or not
  else if((a<b+c) && (b<a+c) && (c<a+b))
        if((a==b) && (b==c))
              printf("\n--------Equilateral triangle          \n");
        else if((a!=b) && (a!=c) && (b!=c))
              printf("----------Scalene triangle             \n");
        else  printf("----------Isosceles triangle            \n");

   else

        printf("-----------Not a Triangle               \n");
        }
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: triangle1.2

b. Features to be tested:

- The given three integer values (ranges from 0 to 32768) form an equilateral triangle
- The given three integer values (ranges from 0 to 32768) form an isosceles triangle
- The given three integer values (ranges from 0 to 32768) form a scalene triangle
- The given three integer values (ranges from 0 to 32768) doesn't form a triangle
- The given values are invalid inputs.

c. Approach Refinement:

i. **Selection of specific test techniques:** Boundary value analysis

1. **Reasons for technique selection:** it makes use of the fact that the inputs and outputs of the components under test can be partitioned into order sets with identifiable boundaries.

ii. **Method(s) for results analysis :** Manual testing with appropriate test case

iii. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

	Min	**Min+**	**Normal**	**Max-**	**Max**
a, b, & c values	1	2	5	9	10

Features	**Test case ID**	**Test case Description**
equilateral	Triangle1.2.1	If all the three sides of a triangle are equal
isosceles	Triangle1.2.2	If any two sides of a triangle are equal
scalene	Triangle1.2.3	If all the three sides of the triangle are unequal
Not a triangle	Triangle1.2.4	If the given three sides don not form a triangle
Invalid inputs	Triangle1.2.5	Other than integer values.

3. TEST CASE:

Triangle problem Boundary Value analysis test cases

Test Case ID	Input values			Expected output	Actual output
	a	b	c		
Triangle1.2.2	5	5	1	isosceles	
Triangle1.2.2	5	5	2	isosceles	
Triangle1.2.1	5	5	5	equilateral	
Triangle1.2.2	5	5	9	isosceles	
Triangle1.2.4	5	5	10	Not a triangle	
Triangle1.2.2	5	1	5	isosceles	
Triangle1.2.2	5	2	5	isosceles	
Triangle1.2.1	5	5	5	equilateral	

Triangle1.2.2	5	9	5	isosceles	
Triangle1.2.4	5	10	5	Not a triangle	
Triangle1.2.2	1	5	5	isosceles	
Triangle1.2.2	2	5	5	isosceles	
Triangle1.2.1	5	5	5	equilateral	
Triangle1.2.2	9	5	5	isosceles	
Triangle1.2.4	10	5	5	Not a triangle	

Triangle problem Boundary Value analysis test cases for worst case

Test Case ID	Input values			Expected output	Actual output
	a	b	c		
Triangle1.2.1	1	1	1	equilateral	
Triangle1.2.4	1	1	2	Not a triangle	
Triangle1.2.4	1	1	5	Not a triangle	
Triangle1.2.4	1	1	9	Not a triangle	
Triangle1.2.4	1	1	10	Not a triangle	
Triangle1.2.4	1	2	1	Not a triangle	
Triangle1.2.2	1	2	2	isosceles	
Triangle1.2.4	1	2	5	Not a triangle	
Triangle1.2.4	1	2	9	Not a triangle	
Triangle1.2.4	1	2	10	Not a triangle	
Triangle1.2.4	1	5	1	Not a triangle	
Triangle1.2.4	1	5	2	Not a triangle	
Triangle1.2.2	1	5	5	isosceles	
Triangle1.2.4	1	5	9	Not a triangle	
Triangle1.2.4	1	5	10	Not a triangle	
Triangle1.2.4	1	9	1	Not a triangle	
Triangle1.2.4	1	9	2	Not a triangle	
Triangle1.2.4	1	9	5	Not a triangle	
Triangle1.2.2	1	9	9	isosceles	
Triangle1.2.4	1	9	10	Not a triangle	
Triangle1.2.4	1	10	1	Not a triangle	
Triangle1.2.4	1	10	2	Not a triangle	
Triangle1.2.4	1	10	5	Not a triangle	
Triangle1.2.4	1	10	9	Not a triangle	
Triangle1.2.2	1	10	10	Isosceles	

4. **TEST PROCEDURE :**
    1. **Store the triangle program in your system.**
    2. **Compile and run the program**
    3. **Enter three integer values (excluding negative values).**
    4. **Check for features to be tested using test cases.**
5. **TEST REPORT**

- Using boundary value analysis, we tested all the test cases specified in the test case (triangle1.2.1 to 1.2.5).
- All the feature test cases are tested and got the expected 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.

AIM: The aim of the commission program is to find out the commission value by using user given criteria.
SCOPE: To check whether the given values of locks, stocks and barrels are within the range and calculate the commission value.

1. TEST FUNCTION (PROGRAM)

```
#include<stdio.h>
main()
{
int  lock,stocks,barrels;
int tlocks,tstocks,tbarrels;
float lprice,sprice,bprice,lsales,bsales,ssales,sales,commission;
lprice=45.0,sprice=30.0,bprice=25.0,tlocks=0,tstocks=0,tbarrels=0;
printf("\n Enter the value of locks:\n");
scanf("%d",&lock);
while(!(lock==-1))
{
printf("Enter the value of stocks and barrels:\n");
scanf("%d%d",&stocks,&barrels);

//check the range of locks, stocks and barrels
if((!((1<=lock) && (lock<=70))) && (!((1<=stocks) && (stocks<=80))) && (!((1<=barrels) &&
(barrels<=90))))
{
printf("\n values of locks, stocks and barrels are not in the range\n"); exit(0);}
else if((!((1<=lock) && (lock<=70))) && (!((1<=stocks) && (stocks<=90))))
{
printf("\n values of locks and stocks are not in the range\n"); exit(0);}

else if((!((1<=stocks) && (stocks<=80))) && (!((1<=barrels) && (barrels<=90))))
{
printf("\n values of stocks and barrels are not in the range\n"); exit(0);}

else if((!((1<=lock) && (lock<=70))) && (!((1<=barrels) && (barrels<=90))))
{
printf("\n values of locks and barrels are not in the range\n"); exit(0);}
else if(!((1<=lock) && (lock<=70)))
{
printf("\n values of locks not in the range\n"); exit(0);}

else if(!((1<=stocks) && (stocks<=80)))
{
printf("\n values of stocks not in the range\n"); exit(0);}
else if(!((1<=barrels) && (barrels<=90)))
{printf("\n values of barrels not in the range\n"); exit(0);}
tlocks=tlocks+lock;
tstocks=tstocks+stocks;
tbarrels=tbarrels+barrels;
printf("\n Enter the value of locks:\n");
scanf("%d",&lock);
}
printf("lock sold:%d\n",tlocks);
printf("stock sold:%d\n",tstocks);
printf("barrels sold:%d\n",tbarrels);

//calculation of sales and commission
lsales=lprice*tlocks;
ssales=sprice*tstocks;
bsales=bprice*tbarrels;
sales=lsales+ssales+bsales;
printf("\n total sales:%f",sales);

//depending on sales calculate the commission
if(sales>1800.0)
{
commission=0.10*1000.0;
commission=commission+0.15*800.0;
commission=commission+0.20*(sales-1800.0);
}
else if(sales>1000.0)
{
commission=0.10*1000.0;
commission=commission+0.15*(sales-1000.0);
}
else
commission=0.10*sales;
printf("\n commission is $%f\n",commission);
}
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: com2.2

b. Features to be tested:

- The given integer values of stock, lock and barrels (ranges from 0 to 32768)
- The given values are invalid inputs.

c. Approach Refinement:

i. **Selection of specific test techniques:** Boundary value analysis

1. **Reasons for technique selection:** there is it makes use of the fact that the inputs and outputs of the components under test can be partitioned into order sets with identifiable boundaries.

ii. **Method(s) for results analysis :** Manual testing with appropriate test case

iii. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

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

Features	Test case ID	Test case Description
1<=Locks, stocks and barrels>=10	Com2.2.1	10% of commission on sales<=$1000
10<=Locks, stocks and barrels<=18	Com2.2.2	15% of commission on sales<=$1800
18<=locks<=70 18<=locks<=80 18<=locks<=90	Com2.2.3	20% of commission on sales>$1800

3. TEST CASE:

Test case ID	Locks	Stocks	Barrels	Sales	Commission	Comments
Com2.2.1	1	1	1	100	10	Output minimum
Com2.2.1	1	1	2	125	12.5	Output minimum+
Com2.2.1	1	2	1	130	13	Output minimum+
Com2.2.1	2	1	1	145	14.5	Output minimum+
Com2.2.1	5	5	5	500	50	Midpoint
Com2.2.1	10	10	9	975	97.5	Border point-
Com2.2.1	10	9	10	970	97	Border point-
Com2.2.1	9	10	10	955	95.5	Border point-
Com2.2.1	10	10	10	1000	100	Border point
Com2.2.2	10	10	11	1025	103.75	Border point+
Com2.2.2	10	11	10	1030	104.5	Border point+
Com2.2.2	11	10	10	1045	106.75	Border point+
Com2.2.2	14	14	14	1400	160	Midpoint
Com2.2.2	18	18	17	1775	216.25	Border point-
Com2.2.2	18	17	18	1770	215.5	Border point-
Com2.2.2	17	18	18	1755	213.25	Border point-
Com2.2.2	18	18	18	1800	220	Border point
Com2.2.3	18	18	19	1825	225	Border point+
Com2.2.3	18	19	18	1830	226	Border point+
Com2.2.3	19	18	18	1845	229	Border point+
Com2.2.3	48	48	48	4800	820	Midpoint
Com2.2.3	70	80	89	7775	1415	Output maximum-
Com2.2.3	70	79	90	7770	1414	Output maximum-
Com2.2.3	69	80	90	7755	1411	Output maximum-
Com2.2.3	70	80	90	7800	1420	Output maximum

4. TEST PROCEDURE:

1. **Store the triangle program in your system.**
2. **Compile and run the program**
3. **Enter three integer values for locks, stocks and barrels to find out commission (excluding negative values).**
4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using boundary value analysis, we tested all the test cases specified in the test case (com2.2.1 to 2.2.3).
- All the feature test cases are tested and got the expected 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.

AIM: The aim of the Next date program is to check whether a given date is valid, leap year, common year and to find next date of the given date.
SCOPE: To find next date of the given date, using boundary value analysis.

1. TEST FUNCTION (PROGRAM)

```
#include<stdio.h>
main()
{
  int tday=0, tmonth=0, tyear=0, day, month, year,valid;
  do
  {
// Do While Loop is repeated until the user enters the valid date
            valid=0;
            printf("\nenter the today's date in the form MM DD YYYY:");
            scanf("%d%d%d", &month, &day, &year);

//check the range of the day (1 to 31), month (1 to 12) and year (1812 to 2013)
            if(!((1<=day) && (day<=31)))
                        printf("the value of the day is not in the range of 1 to 31\n"); else valid++;
            if(!((1<=month) && (month<=12)))
                        printf("the value of the month is not in the range of 1 to 12\n"); else valid++;
            if(!((1812<=year) && (year<=2013)))
                        printf("the value of year is not in the range\n");  else valid++;
  }while(!((1<=day) && (day<=31) && ((1<=month) && (month<=12)) && ((1812<=year) &&
(year<=2013))));

//initialize tomorrow's date to today's date
  tday=day; tmonth=month; tyear=year;
                        //printf("valid= %d", valid);
  switch(month)
  {         //month's having 31 days except december (normal dates)
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10: if(day<31)
                                    tday= day+1;

                        else
                        {           tday = 1; tmonth = month+1;

                        }
                        break;


//month's having 30 days (normal dates)
            case 4:
```

case 6:

```
        case 9:
        case 11: if(day<30)
                        tday= day+1;
                else if(day==30)
                {
                        tday = 1; tmonth = month+1;
                }
                else
                {
                        printf("\ninvalid date\n"); exit(0);
                }
                break;


//december month special case to handle 31st dec of any year
        case 12: if(day<31)
                        tday=day+1;
                else
                {
                        tday=1; tmonth=1;
                        if(year==2013)
                        {
                                printf("\ninvalid input date\n"); exit(0);
                        }
                        else tyear=tyear+1;
                }
                break;

//february month special case to handle leap year and normal years
        case 2: if(day<28)
                        tday=day+1;
                else if(day==28)
                {
                        if(year%4==0) tday=29; //leap year
                        else {
                        tday=1; tmonth=3; }
                }
                else if(day==29) //leap year
                {if(year%4==0)
                        {tday=1; tmonth=3;}
                }
                else if(day>=29)
                        { printf("\ninvalid input date\n"); exit(0); }
                break;
 }
 if(valid==3)
 printf("\nTommorrow's date is : %d %d %d\n", tmonth, tday, tyear);
}
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: Nextdate6.1

b. Features to be tested:

- The given valid date is leap year.
- The given valid date is not a leap year.
- The given valid date is common year.
- Invalid date.

c. Approach Refinement:

i) **Selection of specific test techniques:** Equivalence class partitioning.

1. Reasons for technique selection: it makes use of the fact that the inputs and outputs of the components under test can be partitioned into order sets with identifiable boundaries.

ii) **Method(s) for results analysis :** Manual testing with appropriate test case

iii) **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

	Min	**Min+**	**Normal**	**Max-**	**Max**
Month	1	2	6	11	12
Day	1	2	15	30	31
year	1812	1813	1912	2012	2013

Features	**Test case ID**	**Test case Description**
Valid date	Nextdate6.1.1	Display NextDate
Invalid date	Nextdate6.1.2	Display invalid input date

3. TEST CASE:

Test case ID	**Input**			**Expected output**			**Actual output**		
	Month	**Day**	**Year**	**Month**	**Day**	**Year**	**Month**	**Day**	**Year**
Nextdate6.1.1	1	1	1812	1	2	1812			
Nextdate6.1.1	1	1	1813	1	2	1813			
Nextdate6.1.1	1	1	1912	1	2	1912			
Nextdate6.1.1	1	1	2012	1	2	2012			
Nextdate6.1.1	1	1	2013	1	2	2013			
Nextdate6.1.1	1	2	1812	1	3	1812			
Nextdate6.1.1	1	2	1813	1	3	1813			
Nextdate6.1.1	1	2	1912	1	3	1912			
Nextdate6.1.1	1	2	2012	1	3	2012			
Nextdate6.1.1	1	2	2013	1	3	2013			
Nextdate6.1.1	1	15	1812	1	16	1812			
Nextdate6.1.1	1	15	1813	1	16	1813			
Nextdate6.1.1	1	15	1912	1	16	1912			
Nextdate6.1.1	1	15	2012	1	16	2012			
Nextdate6.1.1	1	15	2013	1	16	2013			
Nextdate6.1.1	1	30	1812	1	31	1812			
Nextdate6.1.1	1	30	1813	1	31	1813			
Nextdate6.1.1	1	30	1912	1	31	1912			
Nextdate6.1.1	1	30	2012	1	31	2012			
Nextdate6.1.1	1	30	2013	1	31	2013			
Nextdate6.1.1	1	31	1812	2	1	1812			

Nextdate6.1.1	1	31	1813	2	1	1813			
Nextdate6.1.1	1	31	1912	2	1	1912			
Nextdate6.1.1	1	31	2012	2	1	2012			
Nextdate6.1.1	1	31	2013	2	1	2013			

3. **TEST PROCEDURE :**
   1. **Store the triangle program in your system.**
   2. **Compile and run the program**
   3. **Enter the valid and invalid date to find the NextDate.**
   4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using boundary value analysis, we tested all the test cases specified in the test case (NextDate6.1.1 to 6.1.2).
- All the feature test cases are tested and got the expected 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.

AIM: The aim of the triangle problem is to check whether a given triangle is equilateral, isosceles, and scalene or not a triangle

SCOPE: To check whether the given three values of sides of a triangle form an equilateral, isosceles, scalene or not a triangle using equivalence class partitioning testing approach.

1. TEST FUNCTION (PROGRAM)

```
#include<stdio.h>
main()
{
  int a,b,c;
  a=b=c=0;     //initialization

  printf("enter three integers which are sides of the triangle : \n");
  scanf("%d%d%d", &a,&b,&c);
  printf("Side A is:%d\t Side B is:%d\t Side C is:%d\n", a,b,c);

 //checks the ranges of the integers a, b and c
  if(!((1<=a) && (a<=10)) && (!((1<=b) && (b<=10))) && (!((1<=c)&&(c<=10))))
        printf("a,b & c is not in the range\n")
  else if(!((1<=a) && (a<=10)) && (!((1<=b) && (b<=10))))
        printf("a, b not in the range\n");
  else if(!((1<=b) && (b<=10)) && (!((1<=c) && (c<=10))))
        printf("b,c not in the range\n");
  else if(!((1<=a) && (a<=10)) && (!((1<=c) && (c<=10))))
        printf("a,c not in the range\n");
  else if(!((1<=a) && (a<=10)))
        printf("a is not in a range\n");
  else if(!((1<=b) && (b<=10)))
        printf("b is not in a range\n");
  else if(!((1<=c) && (c<=10)))
        printf("c is not in a range\n");

 //check the conditions whether three integers will form triangle or not
  else if((a<b+c) && (b<a+c) && (c<a+b))
        if((a==b) && (b==c))
                printf("\n--------Equilateral triangle          \n");
        else if((a!=b) && (a!=c) && (b!=c))
                printf("----------Scalene triangle              \n");
        else    printf("----------Isosceles triangle            \n");

   else

        printf("-----------Not a Triangle                \n");
        }
```

2. TEST DESIGN SPECIFICATION:

b. Test Design Specification Identifier:

This example test design specification has the ID: triangle1.3

c. **Features to be tested:**

- The given three integer values (ranges from 0 to 32768) form an equilateral triangle
- The given three integer values (ranges from 0 to 32768) form an isosceles triangle
- The given three integer values (ranges from 0 to 32768) form a scalene triangle
- The given three integer values (ranges from 0 to 32768) doesn't form a triangle
- The given integer values are invalid.

d. **Approach Refinement:**

i) **Selection of specific test techniques:** Equivalence class partitioning.

1. Reasons for technique selection: This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed. An advantage of this approach is reduction in the timerequired for testing software due to lesser number of test cases.

ii) **Method(s) for results analysis :** Manual testing with appropriate test cases

iii) **Relationship of the test items/features to the levels of testing:** unit level testing.

e. **Test Identification:**

Features	**Test case ID**	**Test case Description**
equilateral	Triangle1.3.1	If all the three sides of a triangle are equal
isosceles	Triangle1.3.2	If any two sides of a triangle are equal
scalene	Triangle1.3.3	If all the three sides of the triangle are unequal
Not a triangle	Triangle1.3.4	If the given three sides don not form a triangle
Invalid inputs	Triangle1.3.5	Other than integer values.
Variable nature	Triangle1.3.6	a is not in the range
Variable nature	Triangle1.3.7	b is not in the range
Variable nature	Triangle1.3.8	c is not in the range
Variable nature	Triangle1.3.9	a & b is not in the range
Variable nature	Triangle1.3.10	a & c is not in the range
Variable nature	Triangle1.3.11	b & c is not in the range
Variable nature	Triangle1.3.12	a , b & c is not in the range

3. **TEST CASE:**

Triangle problem equivalence class test cases for weak normal

Test case	input			Expected output	Actual output
	a	b	c		
Triangle1.3.1	5	5	5	equilateral	
Triangle1.3.2	2	2	3	isosceles	
Triangle1.3.3	3	4	5	scalene	
Triangle1.3.4	4	1	2	Not a triangle	

Triangle problem equivalence class test cases for weak robust

Test case	input			Expected output	Actual output
	a	b	c		
Triangle1.3.6	-1	5	5	Value of a is not in the range	
Triangle1.3.7	5	-1	5	Value of b is not in the range	

Triangle1.3.8	5	5	-1	Value of c is not in the range	
Triangle1.3.6	11	5	5	Value of a is not in the range	
Triangle1.3.7	5	11	5	Value of b is not in the range	
Triangle1.3.8	5	5	11	Value of c is not in the range	

Triangle problem equivalence class test cases for strong robust

Test case	input			Expected output	Actual output
	a	b	c		
Triangle1.3.6	-1	5	5	Value of a is not in the range	
Triangle1.3.7	5	-1	5	Value of b is not in the range	
Triangle1.3.8	5	5	-1	Value of c is not in the range	
Triangle1.3.9	-1	-1	5	Value of a and b is not in the range	
Triangle1.3.11	5	-1	-1	Value of b and c is not in the range	
Triangle1.3.10	-1	5	-1	Value of a and c is not in the range	
Triangle1.3.12	-1	-1	-1	Value of a , b and c is not in the range	

Triangle problem equivalence class test cases for strong robust

Test case	input			Expected output	Actual output
	a	b	c		
Triangle1.3.6	-1	5	5	Value of a is not in the range	
Triangle1.3.7	5	-1	5	Value of b is not in the range	
Triangle1.3.8	5	5	-1	Value of c is not in the range	
Triangle1.3.9	-1	-1	5	Value of a and b is not in the range	
Triangle1.3.11	5	-1	-1	Value of b and c is not in the range	
Triangle1.3.10	-1	5	-1	Value of a and c is not in the range	
Triangle1.3.12	-1	-1	-1	Value of a , b and c is not in the range	

4. TEST PROCEDURE :

1. **Store the triangle program in your system.**
2. **Compile and run the program**
3. **Enter three integer values (excluding negative values).**
4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using Equivalence class partitioning, we tested all the test cases specified in the test case (triangle1.3.1 to 1.3.12).
- All the feature test cases are tested and got the expected 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.

AIM: The aim of the commission program is to find out the commission value by using user given criteria.

SCOPE: To check whether the given values of locks, stocks and barrels are within the range and calculate the commission value.

1. TEST FUNCTION (PROGRAM)

```
#include<stdio.h>
main()
{
int  lock,stocks,barrels;
int tlocks,tstocks,tbarrels;
float lprice,sprice,bprice,lsales,bsales,ssales,sales,commission;
lprice=45.0,sprice=30.0,bprice=25.0,tlocks=0,tstocks=0,tbarrels=0;
printf("\n Enter the value of locks:\n");
scanf("%d",&lock);
while(!(lock==-1))
{
printf("Enter the value of stocks and barrels:\n");
scanf("%d%d",&stocks,&barrels);

//check the range of locks, stocks and barrels
if((!((1<=lock) && (lock<=70))) && (!((1<=stocks) && (stocks<=80))) && (!((1<=barrels) &&
(barrels<=90))))
{
printf("\n values of locks, stocks and barrels are not in the range\n"); exit(0);
}
else if((!((1<=lock) && (lock<=70))) && (!((1<=stocks) && (stocks<=90))))
{
printf("\n values of locks and stocks are not in the range\n"); exit(0);}

else if((!((1<=stocks) && (stocks<=80))) && (!((1<=barrels) && (barrels<=90))))
{
printf("\n values of stocks and barrels are not in the range\n"); exit(0);
}

else if((!((1<=lock) && (lock<=70))) && (!((1<=barrels) && (barrels<=90))))
{
printf("\n values of locks and barrels are not in the range\n"); exit(0);
}
else if(!((1<=lock) && (lock<=70)))
{
printf("\n values of locks not in the range\n"); exit(0);
}

else if(!((1<=stocks) && (stocks<=80)))
{
printf("\n values of stocks not in the range\n"); exit(0);
}
else if(!((1<=barrels) && (barrels<=90)))
{printf("\n values of barrels not in the range\n"); exit(0);
}
tlocks=tlocks+lock;
tstocks=tstocks+stocks;
tbarrels=tbarrels+barrels;
printf("\n Enter the value of locks:\n");
scanf("%d",&lock);
}
printf("lock sold:%d\n",tlocks);
printf("stock sold:%d\n",tstocks);
printf("barrels sold:%d\n",tbarrels);

//calculation of sales and commission
lsales=lprice*tlocks;
ssales=sprice*tstocks;
bsales=bprice*tbarrels;
sales=lsales+ssales+bsales;
printf("\n total sales:%f",sales);

//depending on sales calculate the commission
if(sales>1800.0)
{
commission=0.10*1000.0;
commission=commission+0.15*800.0;
commission=commission+0.20*(sales-1800.0);
}
else if(sales>1000.0)
{
commission=0.10*1000.0;
commission=commission+0.15*(sales-1000.0);
}
else
commission=0.10*sales;
printf("\n commission is $%f\n",commission);}
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: com2.3

b. Features to be tested:

- The given integer values of stock, lock and barrels (ranges from 0 to 32768)
- The given values are invalid inputs.

c. Approach Refinement:

i. **Selection of specific test techniques:** equivalence class partitioning

1. **Reasons for technique selection:** This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed. An advantage of this approach is reduction in the time required for testing software due to lesser number of test cases.

ii. **Method(s) for results analysis :** Manual testing with appropriate test case

iii. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

Features	**Test case ID**	**Test case Description**
1<=Locks, stocks and barrels>=10	Com2.3.1	10% of commission on sales<=$1000
10<=Locks, stocks and barrels<=18	Com2.3.2	15% of commission on sales<=$1800
18<=locks<=70 18<=locks<=80 18<=locks<=90	Com2.3.3	20% of commission on sales>$1800
Locks=-1	Com2.3.4	Program terminates.
Variable range	Com2.3.5	Value of locks not in the range1….70
Variable range	Com2.3.6	Value of stocks not in the range1….80
Variable range	Com2.3.7	Value of barrels not in the range1….90
Variable range	Com2.3.8	Value of locks not in the range1….70 Value of stocks not in the range1….80
Variable range	Com2.3.9	Value of locks not in the range1….70 Value of barrels not in the range1….90
Variable range	Com2.3.10	Value of stocks not in the range1….80 Value of barrels not in the range1….90
Variable range	Com2.3.11	Value of locks not in the range1….70 Value of stocks not in the range1….80 Value of barrels not in the range1….90

3. TEST CASE:

Valid:
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}
Invalid:
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}

Weak robust

Test case ID	Input values			Expected output	Actual output
	Locks	Stocks	Barrels		
Com2.3.1	10	10	10	$100	
Com2.3.4	-1	40	45	Program terminates	
Com2.3.5	-2	40	45	Value of locks not in the range1….70	
Com2.3.5	71	40	45	Value of locks not in the range1….70	
Com2.3.6	35	-1	45	Value of stocks not in the range1….80	
Com2.3.6	35	81	45	Value of stocks not in the	

				range1….80	
Com2.3.7	35	40	-1	Value of barrels not in the range1….90	
Com2.3.7	35	40	91	Value of barrels not in the range1….90	

Strong robust

Test case ID	Input values			Expected output	Actual output
	Locks	Stocks	Barrels		
Com2.3.5	-2	40	45	Value of locks not in the range1….70	
Com2.3.6	35	-1	45	Value of stocks not in the range1….80	
Com2.3.7	35	40	-2	Value of barrels not in the range1….90	
Com2.3.9	-2	-1	45	Value of locks not in the range1….70 Value of stocks not in the range1….80	
Com2.3.9	-2	40	-1	Value of locks not in the range1….70 Value of barrels not in the range1….90	
Com2.3.10	35	-1	-1	Value of stocks not in the range1….80 Value of barrels not in the range1….90	
Com2.3.11	-2	-1	-1	Value of locks not in the range1….70 Value of stocks not in the range1….80 Value of barrels not in the range1….90	

4. TEST PROCEDURE:

1. **Store the triangle program in your system.**
2. **Compile and run the program**
3. **Enter three integer values for locks, stocks and barrels to find out commission (excluding negative values).**
4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using Equivalence class analysis, we tested all the test cases specified in the test case (com2.3.1 to 2.3.11).
- All the feature test cases are tested and got the expected results.

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

AIM: The aim of the Next date program is to check whether a given date is valid, leap year, common year and to find next date of the given date.

SCOPE: To find next date of the given date, using boundary value analysis.

1. TEST FUNCTION (PROGRAM)

```
#include<stdio.h>
main()
{
  int tday=0, tmonth=0, tyear=0, day, month, year,valid;
  do
  {
// Do While Loop is repeated until the user enters the valid date
          valid=0;
          printf("\nenter the today's date in the form MM DD YYYY:");
          scanf("%d%d%d", &month, &day, &year);

//check the range of the day (1 to 31), month (1 to 12) and year (1812 to 2013)
          if(!((1<=day) && (day<=31)))
                    printf("the value of the day is not in the range of 1 to 31\n"); else valid++;
          if(!((1<=month) && (month<=12)))
                    printf("the value of the month is not in the range of 1 to 12\n"); else valid++;
          if(!((1812<=year) && (year<=2013)))
                    printf("the value of year is not in the range\n");  else valid++;
  }
while(!((1<=day) && (day<=31) && ((1<=month) && (month<=12)) && ((1812<=year) &&
(year<=2013))));

//initialize tomorrow's date to today's date
  tday=day; tmonth=month; tyear=year;
                    //printf("valid= %d", valid);
  switch(month)
  {       //month's having 31 days except december (normal dates)
          case 1:
          case 3:
          case 5:
          case 7:
          case 8:
          case 10: if(day<31)
                            tday= day+1;
                    else
                    {         tday = 1; tmonth = month+1;

                    }
                    break;


//month's having 30 days (normal dates)
```

case 4:

```
        case 6:
        case 9:
        case 11: if(day<30)
                        tday= day+1;
                else if(day==30)
                {
                        tday = 1; tmonth = month+1;
                }
                else
                {
                        printf("\ninvalid date\n"); exit(0);
                }
                break;


//december month special case to handle 31st dec of any year
        case 12: if(day<31)
                        tday=day+1;
                else
                {
                        tday=1; tmonth=1;
                        if(year==2013)
                        {
                                printf("\ninvalid input date\n"); exit(0);
                        }
                        else tyear=tyear+1;
                }
                break;

//february month special case to handle leap year and normal years
        case 2: if(day<28)
                        tday=day+1;
                else if(day==28)
                {
                        if(year%4==0) tday=29; //leap year
                        else {
                        tday=1; tmonth=3; }
                }
                else if(day==29) //leap year
                {if(year%4==0)
                        {tday=1; tmonth=3;}
                }
                else if(day>=29)
                        { printf("\ninvalid input date\n"); exit(0); }
                break;
 }
 if(valid==3)
 printf("\nTommorrow's date is : %d %d %d\n", tmonth, tday, tyear);
}
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: Nextdate6.2

b. Features to be tested:

- The given valid date is leap year.
- The given valid date is not a leap year.
- The given valid date is common year.
- Invalid date.

c. Approach Refinement:

i) **Selection of specific test techniques:** Equivalence class partitioning.

1. **Reasons for technique selection:** This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed. An advantage of this approach is reduction in the time required for testing software due to lesser number of test cases.

ii) **Method(s) for results analysis:** Manual testing with appropriate test cases.

iii) **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

M1={month:1<=month<=12}, D1={day:1<=day<=31},Y1={1812<=Year<=2013}

Features	**Test case ID**	**Test case Description**
Valid Date	Nextdate6.2.1	Display NextDate
Invalid date	Nextdate6.2.2	Display invalid input date
Variable nature	Nextdate6.2.3	Display month is not in the range
Variable nature	Nextdate6.2.4	Display day is not in the range
Variable nature	Nextdate6.2.5	Display year is not in the range
Variable nature	Nextdate6.2.6	Display month & day are not in the range
Variable nature	Nextdate6.2.7	Display month & year are not in the range
Variable nature	Nextdate6.2.8	Display day & year are not in the range
Variable nature	Nextdate6.2.9	Display month, day & year not in the range

Equivalence class test cases for next date function for weak robust

Test case	input			Expected output	Actual output
	a	b	c		
Nextdate6.2.1	6	15	1912	6/6/1912	
Nextdate6.2.3	-1	15	1912	Value of month not in range	
Nextdate6.2.3	13	15	1912	Value of month not in range	
Nextdate6.2.4	6	-1	1912	Value of day not in range	
Nextdate6.2.4	6	32	1912	Value of day not in range	
Nextdate6.2.5	6	15	1811	Value of year not in range	
Nextdate6.2.5	6	15	2013	Value of year not in range	

Equivalence class test cases for next date function strong robust

Test case	input			Expected output	Actual output
	a	b	c		
Nextdate6.2.3	-1	15	1912	Value of month not in range	
Nextdate6.2.4	6	-1	1912	Value of day not in range	
Nextdate6.2.5	6	15	1811	Value of year not in range	
Nextdate6.2.6	-1	-1	1912	Value of month & day not in range	
Nextdate6.2.8	6	-1	1811	Value of day and year not in range	
Nextdate6.2.7	-1	15	1811	Value of month & year not in range	

Nextdate6.2.8	-1	-1	1811	Value of month, day & year not in range	

Equivalence class test cases for next date function for weak normal

Test case	input			Expected output	Actual output
	a	b	c		
Nextdate6.2.1	6	14	2000	6/15/2000	
Nextdate6.2.1	7	29	1996	7/30/1996	
Nextdate6.2.2	2	30	2002	Invalid input date	
Nextdate6.2.2	6	31	2000	Invalid input date	

Equivalence class test cases for next date strong normal

Test case	input			Expected output	Actual output
	a	b	c		
Nextdate6.2.1	6	14	2000	6/15/2000	
Nextdate6.2.1	6	14	1996	6/15/1996	
Nextdate6.2.1	6	14	2002	6/15/2002	
Nextdate6.2.1	6	29	2000	6/30/2000	
Nextdate6.2.1	6	29	1996	6/30/1996	
Nextdate6.2.1	6	29	2002	6/30/2002	
Nextdate6.2.2	6	30	2000	Invalid input date	
Nextdate6.2.2	6	30	1996	Invalid input date	
Nextdate6.2.2	6	30	2002	Invalid input date	
Nextdate6.2.2	6	31	2000	Invalid input date	
Nextdate6.2.2	6	31	1996	Invalid input date	
Nextdate6.2.2	6	31	2002	Invalid input date	
Nextdate6.2.1	7	14	2000	7/15/2000	
Nextdate6.2.1	7	14	1996	7/15/1996	
Nextdate6.2.1	7	14	2002	7/15/2002	
Nextdate6.2.1	7	29	2000	7/30/2000	
Nextdate6.2.1	7	29	1996	7/30/1996	
Nextdate6.2.1	7	29	2002	7/30/2002	
Nextdate6.2.1	7	30	2000	7/31/2000	
Nextdate6.2.1	7	30	1996	7/31/1996	
Nextdate6.2.1	7	30	2002	7/31/2002	
Nextdate6.2.1	7	31	2000	8/1/2000	
Nextdate6.2.1	7	31	1996	8/1/1996	
Nextdate6.2.1	7	31	2002	8/1/2002	
Nextdate6.2.1	2	14	2000	2/15/2000	
Nextdate6.2.1	2	14	1996	2/15/2000	
Nextdate6.2.1	2	14	2002	2/15/1996	
Nextdate6.2.2	2	29	2000	Invalid input date	
Nextdate6.2.1	2	29	1996	3/1/1996	
Nextdate6.2.2	2	29	2002	Invalid input date	
Nextdate6.2.2	2	30	2000	Invalid input date	
Nextdate6.2.2	2	30	1996	Invalid input date	
Nextdate6.2.2	2	30	2002	Invalid input date	
Nextdate6.2.2	2	31	2000	Invalid input date	
Nextdate6.2.2	2	31	1996	Invalid input date	
Nextdate6.2.2	2	31	2002	Invalid input date	

4. TEST PROCEDURE:

1. **Store the triangle program in your system.**
2. **Compile and run the program**
3. **Enter the valid and invalid date to find the NextDate.**
4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using Equivalence class partitioning, we tested all the test cases specified in the test case (Nextdate6.2.1 to 6.2.9)
- All the feature test cases are tested and got the expected 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.

AIM: The aim of the triangle problem is to check whether a given triangle is equilateral, isosceles, and scalene or not a triangle

SCOPE: To check whether the given three values of sides of a triangle form an equilateral, isosceles, scalene or not a triangle using Decision based approach testing approach.

1 TEST FUNCTION(PROGRAM)

```
#include<stdio.h>
main()
{
  int a,b,c;
  a=b=c=0;     //initialization

  printf("enter three integers which are sides of the triangle : \n");
  scanf("%d%d%d", &a,&b,&c);
  printf("Side A is:%d\t Side B is:%d\t Side C is:%d\n", a,b,c);

 //checks the ranges of the integers a, b and c
  if(!((1<=a) && (a<=10)) && (!((1<=b) && (b<=10))) && (!((1<=c)&&(c<=10))))
        printf("a, b & c is not in the range\n")
  else if(!((1<=a) && (a<=10)) && (!((1<=b) && (b<=10))))
        printf("a, b not in the range\n");
  else if(!((1<=b) && (b<=10)) && (!((1<=c) && (c<=10))))
        printf("b, c not in the range\n");
  else if(!((1<=a) && (a<=10)) && (!((1<=c) && (c<=10))))
        printf("a, c not in the range\n");
  else if(!((1<=a) && (a<=10)))
        printf("a is not in a range\n");
  else if(!((1<=b) && (b<=10)))
        printf("b is not in a range\n");
  else if(!((1<=c) && (c<=10)))
        printf("c is not in a range\n");

 //check the conditions whether three integers will form triangle or not
  else if((a<b+c) && (b<a+c) && (c<a+b))
        if((a==b) && (b==c))
              printf("\n--------Equilateral triangle          \n");
        else if((a!=b) && (a!=c) && (b!=c))
              printf("----------Scalene triangle              \n");
        else    printf("----------Isosceles triangle            \n");

   else

        printf("-----------Not a Triangle                \n");
        }
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: triangle1.1

b. Features to be tested:

- The given three integer values (ranges from 0 to 32768) form an equilateral triangle
- The given three integer values (ranges from 0 to 32768) form an isosceles triangle
- The given three integer values (ranges from 0 to 32768) form a scalene triangle
- The given three integer values (ranges from 0 to 32768) doesn't form a triangle
- The given values are invalid inputs.

c. Approach Refinement:

i. **Selection of specific test techniques:** Decision Based table testing

1. **Reasons for technique selection:** decision tables are declarative; there is no particular order for condition and actions to occur. These table guarantees that we consider every possible combination of possible values.
   This property is called **"completeness property".**

ii. **Method(s) for results analysis :** Manual testing with appropriate test case

iii. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

Features	**Test case ID**	**Test case Description**
equilateral	Triangle1.1.1	If all the three sides of a triangle are equal
isosceles	Triangle1.1.2	If any two sides of a triangle are equal
scalene	Triangle1.1.3	If all the three sides of the triangle are unequal
Not a triangle	Triangle1.1.4	If the given three sides don not form a triangle
Impossible	Triangle1.1.5	Other than integer values.

Conditions	1	2	3	4	5	6	7	8	9	10	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?	-	-	-	T	T	T	T	F	F	F	F
C5: a=c?	-	-	-	T	T	F	F	T	T	F	F
C6: b=c?	-	-	-	T	F	F	F	T	F	T	F
A1:not a triangle	X	X	X								
A2:scalene											X
A3:isosceleus							X		X	X	
A4:equilateral				X							
A5:impossible					X	X		X			

3. TEST CASE:

Triangle problem decision table-based test cases

Test case ID	Input values			Expected output	Actual output
	a	b	c		
Triangle1.1.4	4	1	2	Not a triangle	
Triangle1.1.4	1	4	2	Not a triangle	
Triangle1.1.4	1	2	4	Not a triangle	
Triangle1.1.1	5	5	5	Equilateral	
Triangle1.1.5	?	?	?	Impossible	
Triangle1.1.5	?	?	?	Impossible	
Triangle1.1.2	2	2	3	Isosceles	
Triangle1.1.5	?	?	?	Impossible	
Triangle1.1.2	2	3	2	Isosceles	
Triangle1.1.2	3	2	2	Isosceles	
Triangle1.1.1	3	4	5	Scalene	

4. TEST PROCEDURE :

1. **Store the triangle program in your system.**
2. **Compile and run the program**
3. **Enter three integer values (excluding negative values).**
4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using Decision based approach, we tested all the test cases specified in the test case (triangle1.1.1 to 1.1.5)
- All the feature test cases are tested and got the expected 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.

AIM: The aim of the commission program is to find out the commission value by using user given criteria.

SCOPE: To check whether the given values of locks, stocks and barrels are within the range and calculate the commission value.

1. TEST FUNCTION (PROGRAM)

```
#include<stdio.h>
main()
{
int  lock,stocks,barrels;
int tlocks,tstocks,tbarrels;
float lprice,sprice,bprice,lsales,bsales,ssales,sales,commission;
lprice=45.0,sprice=30.0,bprice=25.0,tlocks=0,tstocks=0,tbarrels=0;
printf("\n Enter the value of locks:\n");
scanf("%d",&lock);
while(!(lock==-1))
{
printf("Enter the value of stocks and barrels:\n");
scanf("%d%d",&stocks,&barrels);

//check the range of locks, stocks and barrels
if((!((1<=lock) && (lock<=70))) && (!((1<=stocks) && (stocks<=80))) && (!((1<=barrels) &&
(barrels<=90))))
{
printf("\n values of locks, stocks and barrels are not in the range\n"); exit(0);
}
else if((!((1<=lock) && (lock<=70))) && (!((1<=stocks) && (stocks<=90))))
{
printf("\n values of locks and stocks are not in the range\n"); exit(0);}

else if((!((1<=stocks) && (stocks<=80))) && (!((1<=barrels) && (barrels<=90))))
{
printf("\n values of stocks and barrels are not in the range\n"); exit(0);
}

else if((!((1<=lock) && (lock<=70))) && (!((1<=barrels) && (barrels<=90))))
{
printf("\n values of locks and barrels are not in the range\n"); exit(0);
}
else if(!((1<=lock) && (lock<=70)))
{
printf("\n values of locks not in the range\n"); exit(0);
}

else if(!((1<=stocks) && (stocks<=80)))
{
printf("\n values of stocks not in the range\n"); exit(0);
}
else if(!((1<=barrels) && (barrels<=90)))
{printf("\n values of barrels not in the range\n"); exit(0);
}
tlocks=tlocks+lock;
tstocks=tstocks+stocks;
tbarrels=tbarrels+barrels;
printf("\n Enter the value of locks:\n");
scanf("%d",&lock);
}
printf("lock sold:%d\n",tlocks);
printf("stock sold:%d\n",tstocks);
printf("barrels sold:%d\n",tbarrels);

//calculation of sales and commission
lsales=lprice*tlocks;
ssales=sprice*tstocks;
bsales=bprice*tbarrels;
sales=lsales+ssales+bsales;
printf("\n total sales:%f",sales);

//depending on sales calculate the commission
if(sales>1800.0)
{
commission=0.10*1000.0;
commission=commission+0.15*800.0;
commission=commission+0.20*(sales-1800.0);
}
else if(sales>1000.0)
{
commission=0.10*1000.0;
commission=commission+0.15*(sales-1000.0);
}
else
commission=0.10*sales;
printf("\n commission is $%f\n",commission);
}
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: com2.4

b. Features to be tested:

- The given integer values of stock, lock and barrels (ranges from 0 to 32768)
- The given values are invalid inputs.

c. Approach Refinement:

i. **Selection of specific test techniques:** decision based table testing

1. **Reasons for technique selection:** Decision tables are declarative; there is no particular order for condition and actions to occur. These table guarantees that we consider every possible combination of possible values.
This property is called **"completeness property".**

ii. **Method(s) for results analysis :** Manual testing with appropriate test case

iii. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

Decision based table testing

	1	2	3	4	5	6	7	8	9
C1:locks=-1	T	F	F	F	F	F	F	F	F
C2: 1<=locks<=70	-	T	T	T	T	F	F	F	F
C3:1<=stocks<=80	-	T	T	F	F	T	T	F	F
C4:1<=barrels<=90	-	T	F	T	F	T	F	T	F
Action									
Terminate the program	X								
Calculate total locks, stocks & barrels		X	X	X	X	X	X	X	X
Calculate sales & commission	X								
Locks value out of range						X	X	X	X
stocks value out of range				X	X			X	X
Barrels value out of range			X		X		X		X

Features	Test case ID	Test case Description
1<=Locks, stocks and barrels>=10	Com2.4.1	10% of commission on sales<=$1000
10<=Locks, stocks and barrels<=18	Com2.4.2	15% of commission on sales<=$1800
18<=locks<=70 18<=locks<=80 18<=locks<=90	Com2.4.3	20% of commission on sales>$1800
Locks=-1	Com2.4.4	Program terminates.
Variable range	Com2.4.5	Value of locks not in the range1….70
Variable range	Com2.4.6	Value of stocks not in the range1….80
Variable range	Com2.4.7	Value of barrels not in the range1….90
Variable range	Com2.4.8	Value of locks not in the range1….70 Value of stocks not in the range1….80
Variable range	Com2.4.9	Value of locks not in the range1….70 Value of barrels not in the range1….90
Variable range	Com2.4.10	Value of stocks not in the range1….80 Value of barrels not in the range1….90
Variable range	Com2.4.11	Value of locks not in the range1….70 Value of stocks not in the range1….80 Value of barrels not in the range1….90

4. TEST CASE:

Test case ID	Input values			Expected output	Actual output
	Locks	Stocks	Barrels		
Com2.4.4	-1	40	45	Program terminates	
Com2.4.1	10	10	10	$100	
Com2.4.7	35	40	91	Value of barrels not in the range1….90	
Com2.4.6	35	-1	45	Value of stocks not in the range1….80	
Com2.4.10	35	-1	-1	Value of stocks not in the range1….80 Value of barrels not in the range1….90	
Com2.4.5	-2	40	45	Value of locks not in the range1….70	
Com2.4.9	-2	40	-1	Value of locks not in the range1….70	

				Value of barrels not in the range1….90	
Com2.4.8	-2	-1	45	Value of locks not in the range1….70 Value of stocks not in the range1….80	
Com2.4.11	-2	-1	-1	Value of locks not in the range1….70 Value of stocks not in the range1….80 Value of barrels not in the range1….90	

4. TEST PROCEDURE:

1. **Store the triangle program in your system.**
2. **Compile and run the program**
3. **Enter three integer values for locks, stocks and barrels to find out commission (excluding negative values).**
4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using decision based table testing; we tested all the test cases specified in the test case (com2.4.1 to 2.4.11).
- All the feature test cases are tested and got the expected 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.

AIM: The aim of the commission program is to find out the commission value by using user given criteria.
SCOPE: To check whether the given values of locks, stocks and barrels are within the range and calculate the commission value.

1. TEST FUNCTION (PROGRAM)

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

24      lsales = lprice*tlocks;
25       ssales=sprice*tstocks;
26       bsales=bprice*tbarrels;
27      sales=lsales+ssales+bsales;
28      printf("\nthe total sales=%f\n",sales);
29       if(sales > 1800.0)
30       {
31       comm=0.10*1000.0;
32       comm=comm+0.15*800;
33       comm=comm+0.20*(sales-1800.0);
        }
34      else if(sales > 1000)
35      {
36       comm =0.10*1000;
37      comm=comm+0.15*(sales-1000);
        }
38      else
39      comm=0.10*sales;
40      printf("the commission is=%f\n",comm);
41        return 0;
42       }
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: com2.1

b. Features to be tested:

- The given integer values of stock, lock and barrels (ranges from 0 to 32768)
- The given values are invalid inputs.

c. Approach Refinement:

ii. **Selection of specific test techniques:** Dataflow testing

1. **Reasons for technique selection:** Data flow testing focuses on the variables used within a program. Variables are defined and used at different points within the program; data flow testing allows the tester to chart the changing values of variables within the program.

iii. **Method(s) for results analysis :** Manual testing with appropriate test case

iv. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

Define/use nodes for variables:

Variable	Defined at node	Used at node
lprice	7	24
sprice	8	25
bprice	9	26
tlocks	10,16	16,21,24
tstocks	11,17	17,22,25
tbarrels	12,18	18,23,26
locks	13,19	14,16
stocks	15	17
barrels	15	18
lsales	24	27
ssales	25	27
bsales	26	27
sales	27	28,29,33,34,37,39
comm	31,32,33,36,,37,39	32,33,37,42

Variable	Path(beginning, end) nodes	Definition-clear
lprice	7,24	yes
sprice	8,25	yes
bprice	9,26	yes
tlocks	10,16 <10,11,12,13,14,15,16>	Yes
	10,21<10,11,12,13,14,15,16,17,18,19,20,14,21>	No
	10,24<10,11,12,13,14,15,16,17,18,19,20,14,21,22,23,24>	No
	16,16	yes
	16,21 <16,17,18,19,20,14,21>	Yes
	16,24 <16,17,18,19,20,14,21,22,23,24>	Yes
tstocks	11,17 <11,12,13,14,15,16,17>	Yes
	11,22 <11,12,13,14,15,16,17,18,19,20,14,21,22>	No
	11,25<11,12,13,14,15,16,17,18,19,20,14,21,22,23,24,25>	No
	17,17	yes
	17,22 <17,18,19,20,14,21,22>	No
	17,25 <17,18,19,20,14,21,22,23,24,25>	No
locks	13,14 <13,14>	yes
	13,16 <13,14,15,16>	yes
	19,14 <19,20,14>	yes
	19,16 <19,20,14,15,16>	yes
Sales	27,28 <27,28>	yes
	27,29 <27,28,29>	yes
	27,33 <27,28,29,30,31,32,33>	yes
	27,34 <27,28,29,34>	yes
	27,37 <27,28,29,34,35,36,37>	yes
	27,39 <27,28,29,34,38,39>	yes
commission	31,32 <31,32>	Yes
	31,33 <31,32,33>	No
	31,37 (not feasible)	Not applicable
	31,42 <31,32,33,40,41,42>	no
	32,32	yes
	32,33 <32,33>	yes
	32,37 (not feasible)	Not applicable
	32,42 <32,33,40,41,42>	No
	33,32 (not feasible)	Not applicable
	33,33	yes
	33,37 (not feasible)	Not applicable
	33,42 <33,40,41,42>	yes
	36,32 (not feasible)	Not applicable
	36,33 (not feasible)	Not applicable
	36,37 <36,37>	yes
	36,42 <36,37,40,41,42>	No
	37,32 (not feasible)	Not applicable
	37,33 (not feasible)	Not applicable
	37,37	yes
	37,42 <37,40,41,42>	yes
	38,32 (not feasible)	Not applicable
	38,33 (not feasible)	Not applicable
	38,37 (not feasible)	Not applicable
	38,42 <38,39,40,41,42>	yes

3. TEST CASE:

Test case ID	**Variables**	**Input Values**			**Expected output**	
	Sales	locks	stocks	barrels	sales	commission
com2.1.1		**18**	**18**	**19**	**1825**	**225**
com2.1.2		**10**	**10**	**11**	**1025**	**103.755**
com2.1.3		**5**	**5**	**5**	**500**	**50**

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.

AIM: To search an element in a given array this should be in ascending order
SCOPE: To search an element.

TEST PROGRAM (Function)

```
#include<stdio.h>
Int main()
{
int  a[20],n,low,high,mid,key,i,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<=high)
{ mid=(low+high)/2;
if(a[mid]==key)
{ flag=1; break; }
else if(a[mid]
}
}
if(flag==1)
printf("Successful search\n Element found at Location %d\n",mid+1);
else printf("Key Element not found\n");
} else printf("Wrong input");
getch();
return 0;
}
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: bin3.1

b. Features to be tested:

- To search the given element in array.

c. Approach Refinement:

i. **Selection of specific test techniques:** Basis path testing.

1. Reasons for technique selection: the basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

ii. **Method(s) for results analysis :** Manual testing with appropriate test case

iii. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

Binary search flow graph

PATHS = E-N+2(P)

=14-11+2(1) =5

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.

AIM: To sort the elements in the given array.

SCOPE: Sort the elements in the ascending order.

1. TEST PROGRAM (Function)

```
#include<stdio.h>
void quicksort(int x[10],int first,int last)
{
int temp,pivot,i,j;
if(first<last)
{
pivot=first;
i=first;
j=last;
while(i<j)
{
while(x[i]<=x[pivot] && i<last)
i++;
while(x[j]>x[pivot])
j--;
if(i<j)
{
temp=x[i];
x[i]=x[j];
x[j]=temp;
}
}
temp=x[pivot];
x[pivot]=x[j];
x[j]=temp;
quicksort(x,first,j-1);
quicksort(x,j+1,last);
}
}
// main program
int main()
{
int a[20],i,key,n;
printf("enter the size of the array");
scanf("%d",&n);
if(n>0)
{
printf("enter the elements of the array");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
quicksort(a,0,n-1);
printf("the elements in the sorted array is:\n");
```

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

Quick sort function with line number

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

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: quick4.1

b. Features to be tested:

- To sort the given elements in the ascending order.

c. Approach Refinement:

i. **Selection of specific test techniques:** Basis path testing.

1. **Reasons for technique selection:** the basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are

guaranteed to execute every statement in the program at least one time during testing.

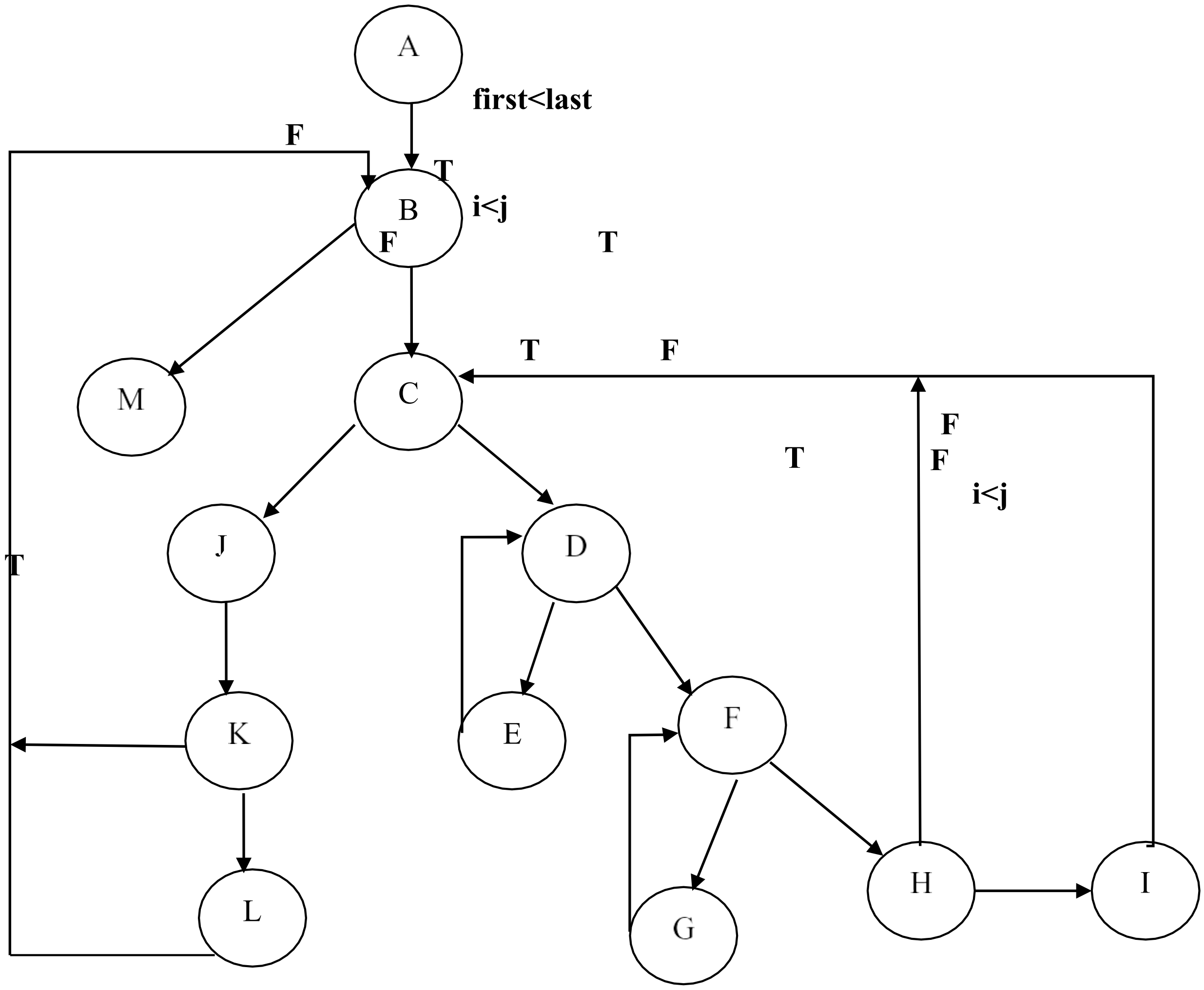
ii. **Method(s) for results analysis :** Manual testing with appropriate test case

iii. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

Sl. No	Program line no.	Flow graph nodes
1	1	A
2	2	B
3	3,4,5	C
4	7	D
5	8	E
6	9	F
7	10	G
8	11	H
9	12,13,14	I
10	15,16,17	J
11	18	K
12	19	L
13	20	M

Test case ID	Test description.
quick4.1.1	The input should follow the path P1: A-B-M
quick41.2	The input should follow the path P2: A-B-C-J-K-B
quick4.1.3	The input should follow the path P3: A-B-C-J-K-L-B
quick4.1.4	The input should follow the path P4: A-B-C-D-F-H-C
quick4.1.5	The input should follow the path P5: A-B-C-D-F-H-I-C
quick4.1.6	The input should follow the path P6: A-B-C-D-E-D-F-H
quick4.1.7	The input should follow the path P7: A-B-C-D-F-G-F-H



3. TEST CASE:

Test case ID	N	A[0]	A[1]	A[2]	A[3]	A[4]	Expected output	Actual output
quick4.1.1	1	6					Sorted	
quick41.2	2	6	4				Sorted	
quick4.1.3	3	3	2	1			Sorted	
quick4.1.4	5	1	2	3	4	5	Sorted	
quick4.1.5	5	5	4	3	2	1	Sorted	
quick4.1.6	5	1	4	3	2	5	Sorted	
quick5.1.7	5	5	2	3	1	4	Sorted	

4. TEST PROCEDURE:

1. **Store the triangle program in your system.**
2. **Compile and run the program**
3. **Enter the integer value to search an element in the given array.**
4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using basis path testing; we tested all the test cases specified in the test case (quick4.1.1 to 4.1.6).
- All the feature test cases are tested and got the expected results.

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

AIM: Depending on the given percentage value finding the grade of that value.

SCOPE: to find absolute letter grading.

1. TEST PROGRAM (Function)

```
#include<stdio.h>
void main()
  {
1.  float per;
2.  printf("Enter the percentage\n");
3.  scanf("%f",&per);
4.  if(per>=80&&per<=100)
5.  printf("A grade\n");
6.  else if(per>=70)
7.  printf("B grade\n");
8.  else if(per>=60)
9.  printf("C grade\n");
10. else if(per>=50)
11. printf("D grade\n");
12. else if(per>=35&&per<=50)
13. printf("E grade\n");
14. else
15. printf("Fail\n");
16. }
```

2. TEST DESIGN SPECIFICATION:

a. Test Design Specification Identifier:

This example test design specification has the ID: grade5.1

b. Features to be tested:

- To find the absolute letter grading depending on the percentage value.

c. Approach Refinement:

i. **Selection of specific test techniques:** Basis path testing.

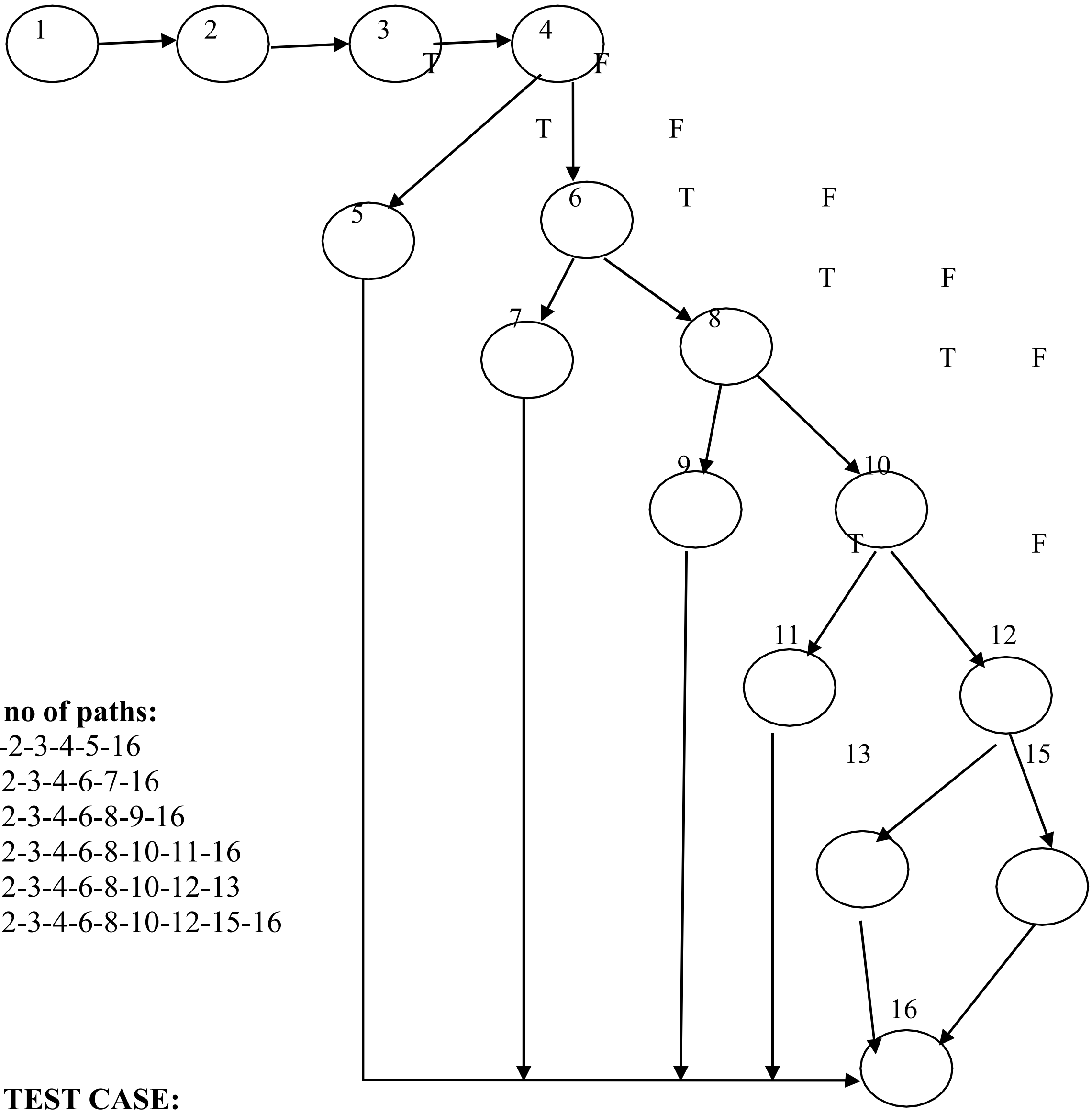
1. Reasons for technique selection: the basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

ii. **Method(s) for results analysis :** Manual testing with appropriate test case

iii. **Relationship of the test items/features to the levels of testing:** unit level testing.

d. Test Identification:

Paths=Edges-nodes+2(p)
=19-15+2=6



Total no of paths:

P1= 1-2-3-4-5-16
P2=1-2-3-4-6-7-16
P3=1-2-3-4-6-8-9-16
P4=1-2-3-4-6-8-10-11-16
P5=1-2-3-4-6-8-10-12-13
P6=1-2-3-4-6-8-10-12-15-16

3. TEST CASE:

Test case ID	N	Expected output	Actual output
grade5.1.1	95	A grade	
grade5.1.2	75	B grade	
grade5.1.3	62	C grade	
grade5.1.4	56	D grade	
grade5.1.5	35	E grade	
grade5.1.6	27	Fail	

4. TEST PROCEDURE:

1. **Store the triangle program in your system.**
2. **Compile and run the program.**
3. **Enter the percentage value to find out Grade.**
4. **Check for features to be tested using test cases.**

5. TEST REPORT

- Using basis path testing; we tested all the test cases specified in the test case (grade5.1.1 to 5.1.6).
- All the feature test cases are tested and got the expected results.