**Program 1: Triangle Problem Testing Technique: Boundary value analysis**

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

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

{

int a,b,c,c1,c2,c3;

do

{

printf("enter the sides of triangle\n"); scanf("%d%d%d",&a,&b,&c);

c1=((a>=1) && (a<=10)); 4

c2=((b>=1) && (b<=10)); 5

c3=((c>=1) && (c<=10)); 6

if(!c1)

printf("value of a is out of range"); if(!c2)

printf("value of b is out of range");

if(!c3)

printf("value of c is out of range");

}while(!c1 || !c2 || !c3); if((a+b)>c && (b+c)>a && (c+a)>b)

{

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

printf("Triangle is equilateral\n"); else if(a!=b && b!=c && c!=a)

printf("Triangle is scalene\n"); else

printf("Triangle is isosceles\n");

}

else

printf("Triangle cannot be formed \n"); getch( );

return 0;

}

**Boundary Value Analysis**

Boundary value analysis focuses on the boundary of the input and output space to identify test cases because errors tend to occur near the extreme values of an input variable. The basic idea is to use input variables at their minimum, just above minimum, nominal, just below their maximum and maximum.

Considering Triangle program, we have three variables a, b and c. Each variables value ranges from 1 to 10.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Variables** | **Min** | **Min+** | **Nom** | **Max-** | **Max** |
| **a** | 1 | 2 | 5 | 9 | 10 |
| **b** | 1 | 2 | 5 | 9 | 10 |
| **c** | 1 | 2 | 5 | 9 | 10 |

**Boundary Value Analysis** = **4n+1** test cases, where n is number of variables

In Triangle program for BVA, we start by taking nominal values for **a** and **b** variables then cross product it with values min, min-, nom, max- and max values of variable **c**. similarly keeping nominal values for variables **a** and **c**, we cross product it with min, min-, nom, max-, max values of variable **b**. Again keeping variable **b** and **c** as nominal combine with 5 values of **a**. By this we get 15 test cases in which a test case with all nominal values for **a**, **b** and **c** is repeated thrice, so we discard 2 duplicate such cases and finally we get 15-2=**13** test cases which is equal to BVA i.e., 4(3)+1=**13**.

**Test cases using Boundary value analysis for Triangle Program**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Output** | **Comments** |
| **A** | **B** | **C** |
| **BVA1** | Enter the values of a(nom),b(nom) and c(min) | 5 | 5 | **1** | Isosceles | Valid |
| **BVA 2** | Enter the values of a(nom),b(nom) and c(min+) | 5 | 5 | **2** | Isosceles | Valid |
| **BVA 3** | Enter the values of a(nom),b(nom) and c(nom) | 5 | 5 | **5** | Equilateral | Valid |
| **BVA 4** | Enter the values of a(nom),b(nom) and c(max-) | 5 | 5 | **9** | Isosceles | Valid |
| **BVA 5** | Enter the values of a(nom),b(nom) and c(max) | 5 | 5 | **10** | Triangle cannot be formed | Valid |
| **BVA 6** | Enter the values of a(nom),b(min) and c(nom) | 5 | **1** | 5 | Isosceles | Valid |
| **BVA 7** | Enter the values of a(nom),b(min+) and c(nom) | 5 | **2** | 5 | Isosceles | Valid |
| **BVA 8** | Enter the values of a(nom),b(max-) and c(nom) | 5 | **9** | 5 | Isosceles | Valid |
| **BVA 9** | Enter the values of a(nom),b(max) and c(nom) | 5 | **10** | 5 | Triangle cannot be formed | Valid |
| **BVA 10** | Enter the values of a(min),b(nom) and c(nom) | **1** | 5 | 5 | Isosceles | Valid |
| **BVA 11** | Enter the values of a(min+),b(nom) and c(nom) | **2** | 5 | 5 | Isosceles | Valid |
| **BVA 12** | Enter the values of a(max-),b(nom) and c(nom) | **9** | 5 | 5 | Isosceles | Valid |
| **BVA 13** | Enter the values of a(max),b(nom) and c(nom) | **10** | 5 | 5 | Triangle cannot be formed | Valid |

**Program 2: Commission Problem Testing Technique: Boundary value analysis**

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

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

{

int c1,c2,c3,temp;

int locks, stocks, barrels, totallocks, totalstocks, totalbarrels;

float lockprice,stockprice,barrelprice,locksales,stocksales,barrelsales,sales,com; lockprice=45.0;

stockprice=30.0; barrelprice=25.0; totallocks=0; totalstocks=0; totalbarrels=0; clrscr();

printf("Enter the number of locks and to exit press -1\n"); scanf("%d",&locks);

while(locks != -1)

{

c1=(locks<=0 || locks>70);

printf("\nEnter the number of stocks and barrels\n"); scanf("%d %d",&stocks,&barrels);

c2=(stocks<=0 || stocks>80); c3=(barrels<=0 || barrels>90);

if(c1)

printf("\nValue of locks are not in the range of 1. 70\n");

else

{

temp=totallocks+locks; if(temp>70)

printf("New totallocks = %d not in the range of 1. 70\n",temp);

else

totallocks=temp;

}

printf("Total locks = %d",totallocks); if(c2)

printf("\n Value of stocks not in the range of 1. 80\n");

else

{

temp=totalstocks+stocks;

if(temp>80)

printf("\nNew total stocks = %d not in the range of 1. 80",temp);

else

totalstocks=temp;

}

printf("\nTotal stocks = %d",totalstocks); if(c3)

printf("\n Value of barrels not in the range of 1. 90\n");

else

{

temp=totalbarrels+barrels; if(temp>90)

printf("\nNew total barrels = %d not in the range of 1. 90\n",temp);

else

totalbarrels=temp;

}

printf("\nTotal barrels=%d", totalbarrels);

printf("\nEnter the number of locks and to exit press -1\n"); scanf("%d",&locks);

}

printf("\n Total locks = %d",totallocks); prin tf("\n Total stocks = %d",totalstocks); printf("\n Total barrels = %d",totalbarrels);

locksales=totallocks\*lockprice; stocksales=totalstocks\*stockprice; barrelsales=totalbarrels\*barrelprice; sales=locksales+stocksales+barrelsales; printf("\n Total sales = %f",sales); if(sales>1800)

{

com=0.10\*1000;

com=com+(0.15\*800); com=com+0.20\*(sales-1800);

}

else if(sales>1000)

{

com=0.10\*1000;

com=com+0.15\*(sales-1000);

}

else

com=0.10\*sales; printf("\nCommission = %f",com); getch();

return 0;

}

Considering **Commission program**, we have three input variables **lock**, **stock** and **barrels**.

Range of value for **locks**= 1-70, **stocks**= 1-80 and **barrels**= 1-90

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Variables** | **Min** | **Min+** | **Nom** | **Max-** | **Max** |
| **locks** | 1 | 2 | 35 | 69 | 70 |
| **stocks** | 1 | 2 | 40 | 79 | 80 |
| **barrels** | 1 | 2 | 45 | 89 | 90 |

Considering output variable **sales** we have 3 slots for calculating **com**mission. i.e., if sales are below 1000, com is 10%, if sales are 1001 to 1800 then com is 15% and if sales are greater than 1801, com is 20%.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sales** | **Min**  **locks, stocks, barrrels** | **Min+**  **locks, stocks, barrrels** | **Nom**  **locks, stocks, barrrels** | **Max-**  **locks, stocks, barrrels** | **Max**  **locks, stocks, barrrels** |
| **1-1000** | 1,1,1 | 2,1,1  1,2,1  1,1,2 | 5,5,5 | 9,10,10  10,9,10  10,10,9 | 10,10,10 |
| **1001-1800** | 11,10,10  10,11,10  10,10,11 | 12,10,10  10,12,10  10,10,12 | 14,14,14 | 17,18,18  18,17,18  18,18,17 | 18,18,18 |
| **1801- above** | 19,18,18  18,19,18  18,18,19 | 20,18,18  18,20,18  18,18,20 | 35,40,45 | 69,80,90  70,79,90  70,80,89 | 70,80,90 |

**Test cases for commission program using INPUT Boundary value analysis**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Output** | | **Comm ents** |
| **Locks** | **Stocks** | **Barrels** | **Sales** | **Com** |
| **BVA1** | Enter the values for locks(nom), stocks(nom) and barrels(min) | 35 | 40 | **1** | 2800 | 420 | Valid |
| **BVA2** | Enter the values for locks(nom), stocks(nom) and barrels(min+) | 35 | 40 | **2** | 2825 | 425 | Valid |
| **BVA3** | Enter the values for locks(nom), stocks(nom) and barrels(nom) | 35 | 40 | **45** | 3900 | 640 | Valid |
| **BVA4** | Enter the values for locks(nom), stocks(nom) and barrels(max-) | 35 | 40 | **89** | 5000 | 860 | Valid |
| **BVA5** | Enter the values for locks(nom), stocks(nom) and barrels(max) | 35 | 40 | **90** | 5025 | 865 | Valid |
| **BVA6** | Enter the values for locks(nom), stocks(min) and barrels(nom) | 35 | **1** | 45 | 2730 | 406 | Valid |
| **BVA7** | Enter the values for locks(nom), stocks(min+) and barrels(nom) | 35 | **2** | 45 | 2760 | 412 | Valid |
| **BVA8** | Enter the values for locks(nom), stocks(max-) and barrels(nom) | 35 | **79** | 45 | 5070 | 874 | Valid |
| **BVA9** | Enter the values for locks(nom), stocks(max) and barrels(nom) | 35 | **80** | 45 | 5100 | 880 | Valid |
| **BVA10** | Enter the values for locks(min), stocks(nom) and barrels(nom) | **1** | 40 | 45 | 2370 | 334 | Valid |
| **BVA11** | Enter the values for locks(min+), stocks(nom) and barrels(nom) | **2** | 40 | 45 | 2415 | 343 | Valid |
| **BVA12** | Enter the values for locks(max-), stocks(nom) and barrels(nom) | **69** | 40 | 45 | 5430 | 946 | Valid |
| **BVA13** | Enter the values for locks(max), stocks(nom) and barrels(nom) | **70** | 40 | 45 | 5475 | 955 | Valid |

**Test cases for commission program using OUTPUT Boundary value analysis**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Output** | | **Comm ents** |
| **Locks** | **Stocks** | **Barrels** | **Sales** | **Com** |
| **BVA1** | Enter the values for locks(min), stocks(min) and  barrels(min) for the range 100 to 1000 | 1 | 1 | 1 | 100 | 10 | Valid |
| **BVA2** | Enter the values for locks(min+), stocks(min) and  barrels(min) for the range 100 to 1000 | 2 | 1 | 1 | 145 | 14.5 | Valid |
| **BVA3** | Enter the values for locks(min), stocks(min+) and barrels(min) for the range 100 to 1000 | 1 | 2 | 1 | 130 | 13 | Valid |
| **BVA4** | Enter the values for locks(min), stocks(min) and  barrels(min+) for the range 100 to 1000 | 1 | 1 | 2 | 125 | 12.5 | Valid |
| **BVA5** | Enter the values for locks(nom), stocks(nom) and  barrels(nom) for the range 100 to 1000 | 5 | 5 | 5 | 500 | 50 | Valid |
| **BVA6** | Enter the values for locks(max-), stocks(max) and  barrels(max) for the range 100 to 1000 | 9 | 10 | 10 | 955 | 95.5 | Valid |
| **BVA7** | Enter the values for locks(max), stocks(max-) and  barrels(max) for the range 100 to 1000 | 10 | 9 | 10 | 970 | 97.0 | Valid |
| **BVA8** | Enter the values for locks(max), stocks(max) and  barrels(max-) for the range 100 to 1000 | 10 | 10 | 9 | 975 | 97.5 | Valid |
| **BVA9** | Enter the values for locks(max), stocks(max) and  barrels(max) for the range 100 to 1000 | 10 | 10 | 10 | 1000 | 100 | Valid |
| **BVA10** | Enter the values for locks(min), stocks(min) and  barrels(min) for the range 1000 to 1800 | 11 | 10 | 10 | 1045 | 106.75 | Valid |
| **BVA11** | Enter the values for locks(min), stocks(min+) and barrels(min) for the range 1000 to 1800 | 10 | 11 | 10 | 1030 | 104.5 | Valid |
| **BVA12** | Enter the values for locks(min), stocks(min) and  barrels(min+) for the range 1000 to 1800 | 10 | 10 | 11 | 1025 | 103.75 | Valid |
| **BVA13** | Enter the values for locks(min+), stocks(min) and  barrels(min) for the range 1000 to 1800 | 12 | 10 | 10 | 1090 | 113.5 | Valid |
| **BVA14** | Enter the values for locks(min), stocks(min+) and  barrels(min) for the range 1000 to 1800 | 10 | 12 | 10 | 1060 | 109 | Valid |
| **BVA15** | Enter the values for locks(min), stocks(min) and  barrels(min+) for the range 1000 to 1800 | 10 | 10 | 12 | 1050 | 107.5 | Valid |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **BVA16** | Enter the values for locks(nom), stocks(nom) and  barrels(nom) for the range 1000 to 1800 | 14 | 14 | 14 | 1400 | 160 | Valid |
| **BVA17** | Enter the values for locks(max-), stocks(max) and  barrels(max) for the range 1000 to 1800 | 17 | 18 | 18 | 1755 | 213.25 | Valid |
| **BVA18** | Enter the values for locks(max), stocks(max-) and barrels(max) for the range 1000 to 1800 | 18 | 17 | 18 | 1770 | 215.5 | Valid |
| **BVA19** | Enter the values for locks(max), stocks(max) and  barrels(max-) for the range 1000 to 1800 | 18 | 18 | 17 | 1775 | 216.25 | Valid |
| **BVA20** | Enter the values for locks(max), stocks(max) and  barrels(max) for the range 1000 to 1800 | 18 | 18 | 18 | 1800 | 220 | Valid |
| **BVA21** | Enter the values for locks(min), stocks(min) and  barrels(min) for the range > 1800 | 19 | 18 | 18 | 1845 | 229 | Valid |
| **BVA22** | Enter the values for locks(min), stocks(min) and  barrels(min) for the range > 1800 | 18 | 19 | 18 | 1830 | 226 | Valid |
| **BVA23** | Enter the values for locks(min), stocks(min) and barrels(min) for the range > 1800 | 18 | 18 | 19 | 1825 | 225 | Valid |
| **BVA24** | Enter the values for locks(min+), stocks(min) and  barrels(min) for the range > 1800 | 20 | 18 | 18 | 1890 | 238 | Valid |
| **BVA25** | Enter the values for locks(min), stocks(min+) and  barrels(min) for the range > 1800 | 18 | 20 | 18 | 1860 | 232 | Valid |
| **BVA26** | Enter the values for locks(min), stocks(min) and  barrels(min+) for the range > 1800 | 18 | 18 | 20 | 1850 | 230 | Valid |
| **BVA27** | Enter the values for locks(nom), stocks(nom) and  barrels(nom) for the range > 1800 | 48 | 48 | 48 | 4800 | 820 | Valid |
| **BVA28** | Enter the values for locks(max-), stocks(max) and barrels(max) for the range > 1800 | 69 | 80 | 90 | 7755 | 1411 | Valid |
| **BVA29** | Enter the values for locks(max), stocks(max-) and  barrels(max) for the range > 1800 | 70 | 79 | 90 | 7770 | 1414 | Valid |
| **BVA30** | Enter the values for locks(max), stocks(max) and  barrels(max-) for the range > 1800 | 70 | 80 | 89 | 7775 | 1415 | Valid |
| **BVA31** | Enter the values for locks(max), stocks(max) and  barrels(max) for the range > 1800 | 70 | 80 | 90 | 7800 | 1420 | Valid |

**Program 3: Next date program Testing Technique: Boundary Value Analysis**

**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.**

#include<stdio.h>

int check(int day,int month)

{

if((month==4||month==6||month==9 ||month==11) && day==30) return 1;

else

}

return 0;

int isleap(int year)

{

if((year%4==0 && year%100!=0) || year%400==0) return 1;

else

return 0;

}

int main()

{

int day,month,year,tomm\_day,tomm\_month,tomm\_year; char flag;

do

{

flag='y';

printf("\n enter the today's date in the form of dd mm yyyy\n"); scanf("%d%d%d", &day, &month, &year);

tomm\_month=month; tomm\_year= year; if(day<1 || day>31)

{

printf("value of day, not in the range 1...31\n"); flag='n';

}

if(month<1 || month>12)

{

printf("value of month, not in the range 1. 12\n");

flag='n';

}

else if(check(day,month))

{

printf("value of day, not in the range day<=30"); flag='n';

}

if(year<=1812 || year>2015)

{

printf("value of year, not in the range 1812. 2015\n");

flag='n';

}

if(month==2)

{

if(isleap(year) && day>29)

{

printf("invalid date input for leap year"); flag='n';

}

else if(!(isleap(year))&& day>28)

{

printf("invalid date input for not a leap year"); flag='n';

}

}

}while(flag=='n');

switch (month)

{

case 1:

case 3:

case 5:

case 7:

case 8:

case 10:if(day<31)

tomm\_day=day+1;

else

{

}

tomm\_day=1; tomm\_month=month+1;

break; case 4:

case 6:

case 9:

case 11: if(day<30)

tomm\_day=day+1;

else

{

tomm\_day=1; tomm\_month=month+1;

}

break;

case 12: if(day<31)

tomm\_day=day+1;

else

{

tomm\_day=1; tomm\_month=1; if(year==2015)

{

printf("the next day is out of boundary value of year\n"); tomm\_year=year+1;

}

else

tomm\_year=year+1;

}

break;

case 2:

if(day<28)

tomm\_day=day+1;

else if(isleap(year)&& day==28)

tomm\_day=day+1; else if(day==28 || day==29)

{

tomm\_day=1; tomm\_month=3;

}

break;

}

printf("next day is : %d %d %d",tomm\_day,tomm\_month,tomm\_year); return 0;

}

Considering Date program, we have three variables day, month and year.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Variables** | **Min** | **Min+** | **Nom** | **Max-** | **Max** |
| **day** | 1 | 2 | 15 | 30 | 31 |
| **month** | 1 | 2 | 6 | 11 | 12 |
| **year** | 1812 | 1813 | 1914 | 2014 | 2015 |

**Test cases for Date program using Boundary Value Analysis**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Output** | **Comments** |
| **DD** | **MM** | **YY** |
| **BVA1** | Enter the values for day(nom),month(nom) and year(min) | 15 | 6 | **1812** | 16/6/1812 | Valid |
| **BVA2** | Enter the values for day(nom),month(nom) and year(min+) | 15 | 6 | **1813** | 16/6/1813 | Valid |
| **BVA 3** | Enter the values for day(nom),month(min) and year(nom) | 15 | 6 | **1914** | 16/6/1914 | Valid |
| **BVA4** | Enter the values for day(nom),month(nom) and year(max-) | 15 | 6 | **2014** | 16/6/2014 | Valid |
| **BVA5** | Enter the values for day(nom),month(nom) and year(max) | 15 | 6 | **2015** | 16/6/2015 | Valid |
| **BVA6** | Enter the values for day(nom),month(min) and year(nom) | 15 | **1** | 1914 | 16/1/1914 | Valid |
| **BVA7** | Enter the values for day(nom),month(min+) and year(nom) | 15 | **2** | 1914 | 16/2/1914 | Valid |
| **BVA8** | Enter the values for day(nom),month(max-) and year(nom) | 15 | **11** | 1914 | 16/11/1914 | Valid |
| **BVA9** | Enter the values for day(nom),month(max) and year(nom) | 15 | **12** | 1914 | 16/12/1914 | Valid |
| **BVA10** | Enter the values for day(min),month(nom) and year(nom) | **1** | 6 | 1914 | 2/6/1914 | Valid |
| **BVA11** | Enter the values for day(min+),month(nom) and year(nom) | **2** | 6 | 1914 | 3/6/1914 | Valid |
| **BVA12** | Enter the values for day(max-),month(nom) and year(nom) | **30** | 6 | 1914 | 1/7/1914 | Valid |
| **BVA13** | Enter the values for day(max),month(nom) and year(nom) | **31** | 6 | 1914 | Day out of range for the month | Valid |

**Program 4: Triangle Problem Testing Technique: Equivalence class partitioning**

**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 the upper limit for the size of any side is 10. Derive test cases for your program based on equivalence class partitioning, execute the test cases and discuss the results.**

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

{

int a,b,c,c1,c2,c3; do

{

printf("enter the sides of triangle\n"); scanf("%d%d%d",&a,&b,&c);

c1=((a>=1) && (a<=10));

c2=((b>=1) && (b<=10));

c3=((c>=1) && (c<=10));

if(!c1)

printf("value of a is out of range"); if(!c2)

printf("value of b is out of range");

if(!c3)

printf("value of c is out of range");

}while(!c1 || !c2 || !c3); if((a+b)>c && (b+c)>a && (c+a)>b)

{

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

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

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

printf("triangle is isosceles\n");

}

else

printf("triangle cannot be formed \n"); getch( );

return 0;

}

**Equivalence Class Test For The Triangle Program Output Equivalence Classes are as follows:**

**R1={<a,b,c>:the triangle with sides a,b and c is Equilateral} R2={<a,b,c>:the triangle with sides a,b and c is Isosceles} R3={<a,b,c>:the triangle with sides a,b and c is Scalene} R4={<a,b,c>:sides a,b and c do not form a Triangle}**

**Weak Normal /Strong Normal**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Expected output** | **Comments** |
| **A** | **B** | **C** |
| **WN/SN1** | Enter the valid values for a, b and c from output equivalence classes. | 5 | 5 | 5 | Equilateral | Valid |
| **WN/SN2** | Enter the valid values for a, b and c from output equivalence classes. | 5 | 5 | 3 | Isosceles | Valid |
| **WN/SN3** | Enter the valid values for a, b and c from output equivalence classes. | 5 | 3 | 4 | Scalene | Valid |
| **WN/SN4** | Enter the valid values for a, b and c from output equivalence classes. | 10 | 1 | 1 | Triangle cannot be formed | Valid |

**Weak Robust**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Expected output** | **Comments** |
| **A** | **B** | **C** |
| **WR 1** | Enter the valid values for b and c from output equivalence classes and invalid value for a. | -1 | 5 | 5 | Value of a is not in a range | Triangle cannot be formed |
| **WR 2** | Enter the valid values for a and c from output  equivalence classes and invalid value for b. | 3 | -1 | 4 | Value of b is not  in a range | Triangle cannot be  formed |
| **WR 3** | Enter the valid values for a and b from output equivalence classes and invalid value for c. | 10 | 10 | -1 | Value of c is not in a range | Triangle cannot be formed |
| **WR 4** | Enter the valid values for b and c from output equivalence classes and invalid value for a. | 11 | 3 | 3 | Value of a is not in a range | Triangle cannot be formed |
| **WR 5** | Enter the valid values for a and c from output  equivalence classes and invalid value for b. | 5 | 11 | 6 | Value of b is not  in a range | Triangle cannot be  formed |
| **WR 6** | Enter the valid values for a and b from output equivalence classes and invalid value for c. | 9 | 10 | 11 | Value of c is not in a range | Triangle cannot be formed |

**Strong Robust**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Expected output** | **Comments** |
| **A** | **B** | **C** |
| **SR 1** | Enter the valid value for b from output equivalence classes and invalid values for a and c. | -1 | 3 | -1 | Values of a and c are not in range | Triangle cannot be formed |
| **SR 2** | Enter the valid value for a from output equivalence classes and invalid values for b and c. | 5 | -1 | -1 | Values of b and c are not in range | Triangle cannot be formed |
| **SR 3** | Enter the valid value for c from output equivalence classes and invalid values for a and b. | -1 | -1 | 10 | Values of a and b are not in range | Triangle cannot be formed |
| **SR 4** | Enter the valid value for a from output equivalence classes and invalid values for b and c. | 7 | 11 | 11 | Values of b and c are not in range | Triangle cannot be formed |
| **SR 5** | Enter the valid value for c from output equivalence classes and invalid values for a and b. | 11 | 11 | 10 | Values of a and b are not in range | Triangle cannot be formed |
| **SR 6** | Enter the valid value for b from output equivalence classes and invalid values for a and c. | 11 | 5 | 11 | Values of a and c are not in range | Triangle cannot be formed |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SR 7** | Enter the valid values for b and c from output equivalence classes and invalid value for a. | -1 | 5 | 5 | Values of a is not in range | Triangle cannot be formed |
| **SR 8** | Enter the valid values for a and c from output equivalence classes and invalid value for b. | 10 | -1 | 10 | Values of b is not in range | Triangle cannot be formed |
| **SR 9** | Enter the valid values for a and b from output equivalence classes and invalid value for c. | 7 | 6 | -1 | Values of c is not in range | Triangle cannot be formed |
| **SR 10** | Enter the valid values for b and c from output equivalence classes and invalid value for a. | 11 | 5 | 4 | Values of a is not in range | Triangle cannot be formed |
| **SR 11** | Enter the valid values for a and c from output equivalence classes and invalid value for b. | 2 | 11 | 3 | Values of b is not in range | Triangle cannot be formed |
| **SR 12** | Enter the valid values for a and b from output equivalence classes and invalid value for c. | 3 | 4 | 11 | Values of c is not in range | Triangle cannot be formed |
| **SR 13** | Enter the invalid value for a, b and c. | 11 | 11 | 11 | Values of a, b and c are not in a  range | Triangle cannot be formed |
| **SR 14** | Enter the invalid value for a, b and c. | -1 | -1 | -1 | Values of a, b and c are not in a  range | Triangle cannot be formed |

**Program 5: Commission Problem Testing Technique: Equivalence class testing**

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

**Equivalence Class Test For The Commission Program Equivalence Classes are as follows:**

## L1={ locks: 1<=locks<=70} L2={ locks=-1}

**S1={ stocks: 1<=stocks<=80} B1={ barrels: 1<=barrels<=90}**

**Weak Normal /Strong Normal**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Expected output** | | **Comments** |
| **Locks** | **Stocks** | **Barrels** | **Sales** | **Com** |
| **WN/SN1** | Enter the valid value within range for locks, stocks and barrels | 35 | 45 | 65 | 4550 | 770 | valid |
| **WN/SN2** | Enter the value of locks=-1 and valid inputs for stocks and barrels | -1 | 40 | 65 | 2825 425  For Program termination | | valid |

**Weak Robust/Strong Robust**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **st cases** | **Description** | **Inputs** | | | **Expected output** | | **Comments** |
| **Locks** | **Stocks** | **Barrels** | **Sales** | **Com** |
| **WR1/SR 1** | Enter the value of lock less than -1 or equal to zero and valid values for stocks and  barrels. | -2 | 45 | 60 | 2850 430  Value of locks not in the range of 1..70 | | Valid |
| **WR2/SR 2** | Enter the value of stocks less than or equal to zero and valid values for locks and barrels. | 35 | -1 | 45 | 2700 400  Value of stocks not in the range of 1..80 | | Valid |
| **WR3/SR3** | Enter the value of barrels less than or equal to zero and valid values for locks and stocks. | 35 | 40 | -1 | 2775 415  Value of barrels not in the range of 1..90 | | Valid |
| **SR 4** | Enter the value of locks & stocks less than or equal to zero and valid values for barrels. | -2 | -1 | 45 | 1125 118.75  Value of locks and stocks not in  the range 1..70 & 1..80 resp | | Valid |
| **SR 5** | Enter the value of locks and barrels less than or equal to zero and valid values for stocks. | -2 | 40 | -1 | 1200 130  Value of locks and barrels not in range of 1..70 & 1..80 resp | | Valid |
| **SR 6** | Enter the value of stocks & barrels less than or equal to zero and valid values for locks. | 35 | -1 | -1 | 1575 186.25  Value of stocks and barrels not in range of 1..80 & 1..90 resp | | Valid |
| **SR 7** | Enter the value of locks, stocks and barrels less than or equal to zero. | -2 | -1 | -1 | Value of locks,stocks & barrels not in range of 1..70,1..80 &  1..90 resp | | Valid |
| **WR4/SR8** | Enter the value of locks greater than 70 and valid values for stocks and barrels. | 71 | 40 | 45 | 2325 325  Value of locks not in the range of 1..70 | | Valid |
| **WR5/SR 9** | Enter the value of stocks greater than 80 & valid values for locks and barrels. | 35 | 82 | 45 | 2700 400  Value of stocks not in the range of 1..80 | | Valid |
| **WR6/SR 10** | Enter the value of barrels greater than 90 and valid values for locks and stocks. | 35 | 40 | 93 | 2775 415  Value of barrels not in the range of 1..90 | | Valid |
| **SR 11** | Enter the value of locks, stocks greater than 70,80 and valid values for barrels. | 71 | 81 | 45 | 1125 118.75  Value of locks & stocks not in the range of 1..70 & 1..80 resp | | Valid |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SR 12** | Enter the value of stocks and barrels greater than 80,90 and valid values for locks. | 35 | 81 | 91 | 1575 186.25  Value of stocks & barrels not in the range of 1..80 & 1..90 resp | Valid |
| **SR 13** | Enter the value of locks and barrels greater than 70, 90 and valid values for stocks. | 71 | 40 | 91 | 1200 130  Value of locks & barrels not in range of 1..70 & 1..90 resp | Valid |
| **SR 14** | Enter the value of locks, stocks & barrels greater than 70, 80, 90. | 71 | 81 | 91 | Value of locks,stocks and barrels not in the range of 1..70,1..80 and 1..90 resp | Valid |

**Program 6: Next date program Testing Technique: Equivalence class testing**

**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.**

**Equivalence class testing for next date program**

## Equivalence Classes are as follows: D1= { Day/DD : 1<=DD<=31 }

**M1= { Month/MM : 1<=MM<=12 } Y1= { Year /YY: 1812<=YY<=2015 }**

**Weak Normal /Strong Normal**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Output** | **Comments** |
| **DD** | **MM** | **YY** |
| **WN/SN1** | Enter valid values for day, month and year from equivalence classes. | 12 | 2 | 1990 | 13/2/1990 | Valid |

**Weak Robust**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Output** | **Comments** |
| **DD** | **MM** | **YY** |
| **WR1** | Enter valid values for month and year from equivalence classes and invalid value for day. | -1 | 6 | 1992 | Day out of range | Valid |
| **WR2** | Enter valid values for day and year from equivalence classes and invalid value for month. | 15 | -1 | 1992 | Month out of range | Valid |
| **WR3** | Enter valid values for day and month from equivalence classes and invalid value for year. | 15 | 6 | 1811 | Year out of range | Valid |
| **WR4** | Enter valid values for month and year from equivalence classes and invalid value for day. | 32 | 6 | 1992 | Day out of range | Valid |
| **WR5** | Enter valid values for day and year from equivalence classes and invalid value for month. | 15 | 13 | 1992 | Month out of range | Valid |
| **WR6** | Enter valid values for day and month from equivalence classes and invalid value for year. | 15 | 6 | 2016 | Year out of range | Valid |

**Strong Robust**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Output** | **Comments** |
| **DD** | **MM** | **YY** |
| **SR1** | Enter valid values for month and year from equivalence classes and invalid value for day. | -1 | 6 | 1992 | Day out of range | Valid |
| **SR2** | Enter valid values for day and year from equivalence classes and invalid value for month. | 15 | -1 | 1992 | Month out of range | Valid |
| **SR3** | Enter valid values for day and month from equivalence classes and invalid value for year. | 15 | 6 | 1811 | Year out of range | Valid |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SR4** | Enter valid value for year from equivalence classes and invalid values for day and month. | -1 | -1 | 1992 | Day, Month out of range | Valid |
| **SR5** | Enter valid value month for from equivalence classes and invalid values for day and year. | -1 | 6 | 1811 | Day, Year out of range | Valid |
| **SR6** | Enter valid value for day from equivalence classes and invalid values for month and year. | 15 | -1 | 1811 | Month, Year out of range | Valid |
| **SR7** | Enter valid values for month and year from equivalence classes and invalid value for day. | 32 | 6 | 1992 | Day out of range | Valid |
| **SR8** | Enter valid values for day and year from equivalence classes and invalid value for month. | 15 | 13 | 1992 | Month out of range | Valid |
| **SR9** | Enter valid values for day and month from equivalence classes and invalid value for year. | 15 | 6 | 2016 | Year out of range | Valid |
| **SR10** | Enter valid value for year from equivalence classes and invalid values for day and month. | 32 | 13 | 1992 | Day, Month out of range | Valid |
| **SR11** | Enter valid value month for from equivalence classes and invalid values for day and year. | 32 | 6 | 2016 | Day, Year out of range | Valid |
| **SR12** | Enter valid value for day from equivalence classes and invalid values for month and year. | 15 | 13 | 2016 | Month, Year out of range | Valid |
| **SR13** | Enter invalid values for day, month and year. | -1 | -1 | 1811 | Day, Month, Year out of range | Valid |
| **SR14** | Enter invalid values for day, month and year. | 32 | 13 | 2016 | Day, Month, Year out of range | Valid |

**Program 7: Triangle Problem Testing Technique: Decision Table Approach 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.**

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

{

int a,b,c;

printf("enter the sides of triangle\n"); scanf("%d%d%d",&a,&b,&c); if((a+b)>c && (b+c)>a && (c+a)>b)

{

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

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

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

printf("triangle is isosceles\n");

}

else

printf("triangle cannot be formed\n"); return 0;

}

**Decision Table for Triangle Problem**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Stub**  **Conditions**  **Actions** | **Rules** | **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7** | **R8** | **R9** | **R10** | **R11** |
| **C1: a<b+c** | F | T | T | T | T | T | T | T | T | T | T |
| **C2: b<c+a** | -- | 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 | F | F | F | T | T | F |
| **C5: b=c** | -- | -- | -- | T | F | T | F | F | T | F | T |
| **C6: c=a** | -- | -- | -- | T | F | F | T | F | F | T | T |
| **A1: Not a triangle** | x | x | x |  |  |  |  |  |  |  |  |
| **A2: Equilateral** |  |  |  | x |  |  |  |  |  |  |  |
| **A3: Isosceles** |  |  |  |  | x | x | x |  |  |  |  |
| **A4: Scalene** |  |  |  |  |  |  |  | x |  |  |  |
| **A5: Impossible** |  |  |  |  |  |  |  |  | x | x | X |

**Test Cases using Decision Table for Triangle Program**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Output** | **Comments** |
| **A** | **B** | **C** |
| **Case 1** | Enter the values of a, b and c such that value of a is greater than sum of b and c. | 7 | 2 | 3 | Not a triangle | Valid |
| **Case 2** | Enter the values of a, b and c such that value of b is greater than sum of a and c. | 2 | 8 | 3 | Not a triangle | Valid |
| **Case 3** | Enter the values of a, b and c such that value of c is greater than sum of a and b. | 2 | 4 | 7 | Not a triangle | Valid |
| **Case 4** | Enter the values of a, b and c such that values of a,b and c are equal. | 5 | 5 | 5 | Equilateral | Valid |
| **Case 5** | Enter the values of a, b and c Such that value of a is equal to value of b. | 4 | 4 | 3 | Isosceles | Valid |
| **Case 6** | Enter the values of a, b and c such that value of b is equal to value of c. | 2 | 5 | 5 | Isosceles | Valid |
| **Case 7** | Enter the values of a, b and c such that value of a is equal to value of c. | 6 | 2 | 6 | Isosceles | Valid |
| **Case 8** | Enter the values of a, b and c such that values of a,b and c are different. | 2 | 3 | 4 | Scalene | Valid |
| **Case 9** | Enter the values of a, b and c such that value of a is equal to value of b and c but value of b is not equal to c. | ? | ? | ? | Impossible | Valid |
| **Case 10** | Enter the values of a, b and c such that value of b is equal to  value of c and value of c is equal to a but value of a not equal to b. | ? | ? | ? | Impossible | Valid |
| **Case 11** | Enter the values of a, b and c such that value of a is equal  to b and value of b is equal to value of c but value of a not equal to c. | ? | ? | ? | Impossible | Valid |

**Program 8: Commission Problem Testing Technique: Decision table-based testing**

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

**INPUT DECISION TABLE**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Conditions** | **RULES** | **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7** | **R8** | **R9** |
| **C1: 1<= locks <=70** | --- | T | T | T | T | F | F | F | F |
| **C2: 1<= stocks <=80** | --- | F | T | F | T | F | T | F | T |
| **C3: 1<= barrels <= 90** | --- | F | F | T | T | F | F | T | T |
| **C4: locks = -1** | T | T | T | T | T | T | T | T | T |
| **Actions** | **a1: Invalid lock input** |  |  |  |  |  | X | X | X | X |
| **a2: Invalid stock input** |  | X |  | X |  | X |  | X |  |
| **a3: Invalid barrels input** |  | X | X |  |  | X | X |  |  |
| **a4: Calculate totallocks, totalstocks and**  **totalbarrels** | X | X | X | X | X | X | X | X | X |
| **a5: Calculate sales** | X | X | X | X | X | X | X | X | X |

**Test cases for Commission program for Input Decision table.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Expected output** | | **Comments** |
| **Locks** | **Stocks** | **Barrels** | **Sales** | **Com** |
| **IDT1** | Enter no. of locks=-1 | -1 | - | - | 0 0  Program Terminates | | valid |
| **IDT2** | Enter the valid no. of locks and invalid values for stocks and barrels | 20 | 81 | 91 | 900 90  Invalid no.of stocks and barrels | | valid |
| **IDT3** | Enter the valid values for locks, stocks and invalid value for barrels | 20 | 20 | 96 | 1500 175  Invalid no.of barrels | | valid |
| **IDT4** | Enter the valid values for locks and barrels and invalid value for stocks | 20 | -1 | 20 | 1400 160  Invalid no.of stocks | | valid |
| **IDT5** | Enter the valid values for locks, stocks and barrels | 20 | 20 | 20 | 2000 260  Calculates sales and commission | | valid |
| **IDT6** | Enter the invalid values for locks, stocks and barrels | -2 | 81 | -1 | 0 0  Invalid no.of locks, Stocks and barrels. | | valid |
| **IDT7** | Enter the valid value for stocks and invalid values for locks and barrels | -2 | 20 | 91 | 600 60  Invalid no.of locks and barrels | | valid |
| **IDT8** | Enter invalid input for locks and stocks and valid input for barrels | 71 | -1 | 20 | 500 50  Invalid no.of locks and stocks | | valid |
| **IDT9** | Enter the invalid value for locks and valid values for stocks and barrels | -3 | 20 | 20 | 1100 115  Invalid no.of locks | | valid |

**COMMISSION CALCULATION DECISION TABLE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conditions** | **RULES** | **R1** | **R2** | **R3** |
| **C1: Sales> 1801** | T | F | F |
| **C2: Sales >1001 and sales <= 1800** | --- | T | F |
| **C3: Sales <=1000** | --- | --- | T |
| **Actions** | **a1: comm. = 10% \*1000 + 15%\*800 + (sales-1800) \* 20%** | X |  |  |
| **a2: comm. = 10% \*1000 + (sales-1000)\* 15%** |  | X |  |
| **a3: comm. = 10% \*sales** |  |  | X |

**Test cases for Commission program for Output Decision table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test cases** | **Description** | **Inputs** | | | **Expected output** | | **Comments** |
| **Locks** | **Stocks** | **Barrels** | **Sales** | **Com** |
| **IDT1** | Enter the value of locks,stocks and Barrels such that sales>1800. | 19 | 18 | 18 | 1845 | 229 | Valid |
| **IDT2** | Enter the value of locks,stocks and  Barrels such that sales>1000 and sales <=1800. | 14 | 14 | 14 | 1400 | 160 | Valid |
| **IDT3** | Enter the value of locks,stocks and Barrels such that sales<=1000. | 5 | 5 | 5 | 500 | 50 | Valid |

**Program 9: Commission Problem Testing Technique: Dataflow testing**

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

1. #include<stdio.h>
2. #include<conio.h>
3. int main()

**4** {

1. int c1,c2,c3,temp;
2. int locks, stocks, barrels, totallocks, totalstocks, totalbarrels;
3. float lockprice,stockprice,barrelprice,locksales,stocksales,barrelsales,sales,com;
4. lockprice=45.0;
5. stockprice=30.0; **10** barrelprice=25.0; **11** totallocks=0;
6. totalstocks=0;
7. totalbarrels=0;
8. clrscr();
9. printf("Enter the number of locks and to exit press -1\n");
10. scanf("%d",&locks);
11. while(locks != -1)

**18** {

**19** c1=(locks<=0 || locks>70);

1. printf("\nEnter the number of stocks and barrels\n");
2. scanf("%d %d",&stocks,&barrels);
3. c2=(stocks<=0 || stocks>80); **23** c3=(barrels<=0 || barrels>90); **24** if(c1)
4. printf("\nValue of locks are not in the range of 1. 70\n");
5. else

**27** {

1. temp=totallocks+locks;
2. if(temp>70)
3. printf("New totallocks = %d not in the range of 1. 70\n",temp);
4. else
5. totallocks=temp;

**33** }

1. printf("Total locks = %d",totallocks);
2. if(c2)
3. printf("\n Value of stocks not in the range of 1. 80\n");
4. else

**38** {

1. temp=totalstocks+stocks;
2. if(temp>80)
3. printf("\nNew total stocks = %d not in the range of 1. 80",temp);
4. else
5. totalstocks=temp;

**44** }

1. printf("\nTotal stocks = %d",totalstocks);
2. if(c3)
3. printf("\n Value of barrels not in the range of 1. 90\n");
4. else

**49** {

1. temp=totalbarrels+barrels;
2. if(temp>90)
3. printf("\nNew total barrels = %d not in the range of 1. 90\n",temp);
4. else
5. totalbarrels=temp;

**55** }

1. printf("\nTotal barrels=%d", totalbarrels);
2. printf("\nEnter the number of locks and to exit press -1\n");
3. scanf("%d",&locks);

**59** }

1. printf("\n Total locks = %d",totallocks);
2. printf("\n Total stocks = %d",totalstocks); **62** printf("\n Total barrels = %d",totalbarrels); **63** locksales=totallocks\*lockprice;
3. stocksales=totalstocks\*stockprice;
4. barrelsales=totalbarrels\*barrelprice;
5. sales=locksales+stocksales+barrelsales;
6. printf("\n Total sales = %f",sales);
7. if(sales>1800)

**69** {

**70** com=0.10\*1000;

**71** com=com+(0.15\*800);

**72** com=com+0.20\*(sales-1800);

**73** }

**74** else if(sales>1000)

**75** {

**76** com=0.10\*1000;

**77** com=com+0.15\*(sales-1000);

**78** }

1. else
2. com=0.10\*sales;
3. printf("\nCommission = %f",com);
4. getch();
5. return 0;

**84** }

**Define/ Use Nodes for variables in the commission problem**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Defined at node** | **Used at node** |
| lockprice | 8 | 63 |
| stockprice | 9 | 64 |
| barrelprice | 10 | 65 |
| totallocks | 11,32 | 28,34,60,63 |
| totalstocks | 12,43 | 39,45,61,64 |
| totalbarrels | 13,54 | 50,56,62,65 |
| Locks | 16,58 | 17,19,28 |
| stocks | 21 | 22,39 |
| barrels | 21 | 23,50 |
| locksales | 63 | 66 |
| stocksales | 64 | 66 |
| barrelsales | 65 | 66 |
| Sales | 66 | 67,68,72,74,77,80 |
| Com | 70,71,72,76,77,80 | 71,72,77,81 |

**Define /Use paths with definition clear status**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test id** | **Description** | **Variables**  **(beginning**  **,end nodes)** | **DU paths** | **DC**  **path ?** |
| 1 | Check for lockprice variable DEF(ocklprice,8)  And USE(lockprice,63) | <8,63> | 8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,  31,32,33,34,35,37,38,39,40,42,43,44,45,46,48,49  ,50,51,53,54,55,56,57,58,59,60,61,62,63 | **YES** |
| 2 | Check for stockprice variable DEF(stockprice,9)  And USE(stockprice,64) | <9,64> | 9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,  27,28,29,31,32,33,34,35,37,38,39,40,42,43,44,45,  46,48,49,50,51,53,54,55,56,57,58,59,60,61,62,63,64 | **YES** |
| 3 | Check for barrelprice variable DEF(barrelprice,10)  And USE(barrelprice,65) | <10,65> | 10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,  29,31,32,33,34,35,37,38,39,40,42,43,44,45,46,  48,49,50,51,53,54,55,56,57,58,59,60,61,62,63,64,65 | **YES** |
| 4 | Check for totallocks variable DEF(totallocks,11,32)  And USE(totallocks, 28,34,60,63) | <11,28>  <11,34>  <11,60>  <11,63>  <32,28> | 11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28  11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,31,32,33,34  11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,31,32,33,34,35,37  ,38,39 ,40,42,43 ,44,45,46,48,49,50,51,53,54,55,56,57,58,59,60  11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,31,32,33,34,35,37  ,38,39 ,40,42,43 ,44,45,46,48,49,50,51,53,54,55,56,57,58,59,60,61,62,63  32,33,34,35,37,38,39,40,42,43,44,45,46,48,49,50,51,53,54,55,56,57,58,17  ,18,19,20,21,22,23,24,26,27,28 | **YES**  **NO**  **NO**  **NO**  **YES** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | <32,34>  <32,60>  <32,63> | 32,33,34  32,33,34,35,37,38,39,40,42,43,44,45,46,48,49,50,51,53,54,  55,56,57,58,59, 60  32,33,34,35,37,38,39,40,42,43,44,45,46,48,49,50,51,  53,54,55,56,57,58,59 ,60,61,62,63 | **YES**  **YES**  **YES** |
| 5 | Check for totalstocks variable DEF(totalstocks,12,43)  And USE(totalstocks, 39,45,61,64) | <12,39>  <12,45>  <12,61>  <12,64>  <43,45> | 12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,31  ,32,33,34,35,37,38,39  12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29  ,31,32,33,34,35,37,38,39,40,42,43,44,45  12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29  ,31,32,33,34,35,37,38,39,40,42,43,44,45,46,48,49  ,50,51,53,54,55,56,57,58,59,60,61  12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29  ,31,32,33,34,35,37,38,39,40,42,43,44,45,46,48,49  ,50,51,53,54,55,56,57,58,59,60,61,62,63,64  43,44,45 | **YES**  **NO**  **NO**  **NO**  **YES** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | <43,61>  <43,64> | 43,44,45,46,48,49,50,51,53,54,55,56,57,58,59,60,61  43,44,45,46,48,49,50,51,53,54,55,56,57,58,59,60,61,62,63,64 | **YES**  **YES** |
| 6 | Check for totalbarrels variable DEF(totalbarrels,13,54)  And USE(totalbarrels, 50,56,62,65) | **<**13,50>  <13,56>  <13,62>  <13,65>  <54,56>  <54,62>  <54,65> | 13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29  ,31,32,33,34,35,37,38,39,40,42,43,44,45,46,48,49,50  13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29  ,31,32,33,34,35,37,38,39,40,42,43,44,45,46,48,49  ,50,51,53,54,55,56  13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,  31,32,33,34,35,37,38,39,40,42,43,44,45,46,48,49,  50,51,53,54,55,56,57,58,59,60,61,62  13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29,  31,32,33,34,35,37,38,39,40,42,43,44,45,46,48,49,  50,51,53,54,55,56,57,58,59,60,61,62,63,64,65  54,55,56  54,55,56,57,58,59,60,61,62  54,55,56,57,58,59,60,61,62,63,74,65 | **YES**  **NO**  **NO**  **NO**  **YES**  **YES**  **YES** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | Check for locks variable DEF(locks,16,58)  And USE(locks,17,19,28) | <16,17>  <16,19>  <16,28>  <58,17>  <58,19>  <58,28> | 16,17  16,17,18,19  16,17,18,19,20,21,22,23,24,26,27,28  58,17  58,17,18,19  58,17,18,19,20,21,22,23,24,26,27,28 | **YES YES YES YES YES**  **YES** |
| 8 | Check for stocks variable DEF(stocks,21)  And USE(stocks,22,39) | <21,22>  <21,39> | 21,22  21,22,23,24,26,27,28,29,31,32,33,34,35,37,38,39 | **YES YES** |
| 9 | Check for barrels variable DEF(barrels,21)  And USE(barrels,23,50) | <21,23>  <21,50> | 21,22,23  21,22,23,24,25,26,27,28,29,31,32,33,34,35,37,38,39,40,42,43,44,45,  46,47,48,49,50 | **YES** |
| 10 | Check for lockpsales variable DEF(locksales,63)  And USE(locksales,66) | <63,66> | 63,64,65,66 | **YES** |
| 11 | Check for stocksales variable DEF(stocksales,64)  And USE(stocksales,66) | <64,66> | 64,65,66 | **YES** |
| 12 | Check for barrelsales variable DEF(barrelsales,65)  And USE(barrelsales,66) | <65,66> | 65,66 | **YES** |
| 13 | Check for sales variable DEF(sales,66)  And USE(sales,67,68,72,74,77,8 0) | <66,67>  <66,68>  <66,72>  <66,74>  <66,77> | 66,67  66,67,68  66,67,68,69,70,71,72  66,67,68,74  66,67,68,74,75,76,77 | **YES YES YES YES**  **YES** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | <66,80> | 66,67,68,74,79,80 | **YES** |
| 14 | Check for commission variable DEF(com,70,71,72,76,77,8 0)  And USE(com,71,72,77,81) | <70,71>  <70,72>  <70,81>  <71,72>  <71,81>  <72,81>  <76,77>  <76,81>  <77,81>  <80,81> | 70,71  70,71,72  70,71,72,73,81  71,72  71,72,73,81  72,73,81  76,77  76,77,78,81  77,78,81  80,81 | **NO NO NO NO NO YES NO NO YES**  **YES** |

# Note

In above Du-Paths, some paths like

<70,77>,<71,71>,<71,77>,<72,71>,<72,72>,<72,77>,<76,71>,<76,72>,<77,71>,<77,71>,<77,77>,

<80,70>,<80,72>,<80,77>,<80,77> are not possible to be formed. So they are not considered in above table.

**Program 10: Binary Search Testing Technique: Basis paths**

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

#include<stdio.h>

1. **1** int binsrc(int x [ ],int low,int high,int key)

**2** {

**3** int mid;

1. **4** while(low<=high)
2. **5** {

**6** mid=(low+high)/2;

1. **7** if(x[mid]==key)

**I. 8** return mid;

1. **9** elseif(x[mid]<key)

**G. 10** low=mid+1;

**F. 11** else

**12** high=mid-1;

## H. 13 }

**J. 14** return -1;

## K. 15 }

int main()

{

int a[20],key,i,n,succ;

printf("Enter the n value up to max of 20"); scanf("%d",&n);

if(n>0)

{

printf("enter the elements in ascending order\n"); for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("enter the key element to be searched\n"); scanf("%d",&key);

succ=binsrc(a,0,n-1,key); if(succ>=0)

printf("Element found in position = %d\n",succ+1); else

printf("Element not found \n");

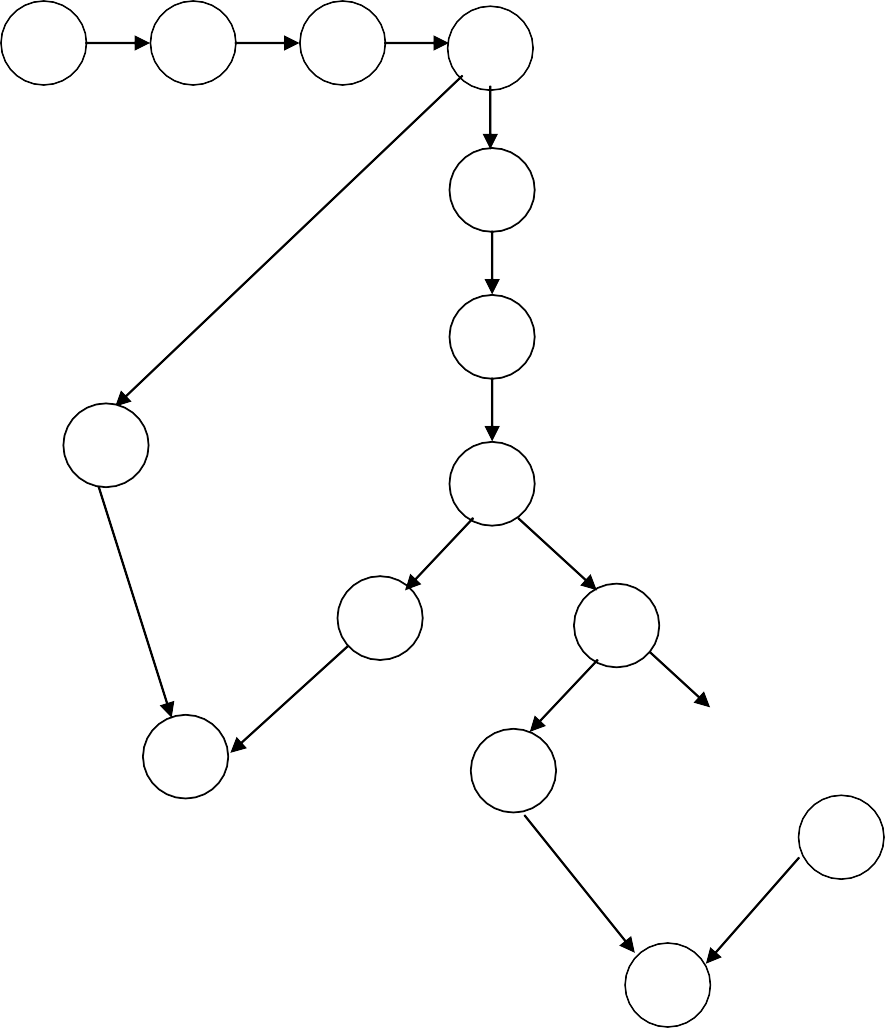
}

else

printf("Number of element should be greater than zero\n"); return 0;

}

**Program Graph**



1

2

3

4

5

6

14

7

8

9

11

15

10

12

13

**DD Path graph**

|  |  |
| --- | --- |
| **NODES** | **DD-paths** |
| **1-3** | **A** |
| **4** | **B** |
| **5,6** | **C** |
| **7** | **D** |
| **8** | **I** |
| **9** | **E** |
| **10** | **G** |
| **11,12** | **F** |
| **13** | **H** |
| **14** | **J** |
| **15** | **K** |

A

**B**

**C**

**D**

**J**

**I**

**E**

**K**

**G**

**F**

**H**

## McCabe’s Basis path method

Considering DD-Path graph of the program, first we need to find Baseline path. A baseline path consists of maximum number of decision nodes. Using Baseline path we start flipping each decision node for finding new paths.

## Considering Binary search program

Considering DD-Path graph of Binary search function, function starts at node A and Ends at node K. First, Base Line path is formed by considering all decision nodes as shown below.

**Baseline Path:** A **B** C **D E** F H **B** J K**.**

Nodes which are bold and large are decision nodes. Now start flipping each decision node.

**Flipping at B :** A B J K.

**Flipping at D :** A B C D I K.

**Flipping at E :** A B C D E G H B J.

**Cyclomatic Complexity**

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

Where,

**e** is number of edges in DD-Path graph.

**n** is number of nodes in DD-Path graph.

**p** is number of regions connected.(always 1)

Number of linearly independent paths for a given graph G **= 13-11+2(1)= 4** Test cases

**Test Cases for Binary Search Program**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Cases** | **Description** | **Input** | | | **Expected Output** | **Comment** |
| **N** | **Array elements** | **Key** |
| **TC1** | Enter the basis path consisting of all decision nodes **ABCDEFHBJK.** | 2  1 | {5,10}  {10} | 4  5 | Infeasible because low>high means from B to J then K which means no elements left which is  not true in any case. | Valid |
| **TC2** | Enter the basis path consisting of all decision nodes **ABJK.** | 0 | --------- | ---- | Infeasible because low>high means from B to J then K which  means no elements left which is not true in any case. | Invalid |
| **TC3** | Enter the basis path consisting of all decision nodes **ABCDIK.** | 2  3  5 | {5,10}  {5,10,15}  {5,10,15,20,25} | 10  10  15 | Element found in position 2 Element found in position 2 Element found in position 3 | Valid |
| **TC4** | Enter the basis path consisting of all decision nodes **ABCDEGHBJK.** | 2  1 | {5,10}  {10} | 15  12 | Infeasible because low>high means from B to J then K which means no elements left which is  not true in any case. | Invalid |

# Note

Path **B J K** indicates fail of while (low<=high) condition. Because when there is one element in the array, then low will b equal to high (i.e., low=high). Similarly when there are more than one elements in the array low will be greater than high (i.e., low>high). So low>high means there no elements in the array. So in above table paths containing **B J K** are considered as infeasible.

**Program 11: Quick Sort Testing Technique: Path Testing**

**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.**

#include<stdio.h>

1. **1** void quicksort(int x[10],int first,int last)
2. **2** {

**3** int temp,pivot,i,j;

1. **4** if(first<last)
2. **5** {
3. pivot=first;
4. i=first;
5. j=last;
6. **9** while(i<j)

## F. 10 {

1. **11** while(x[i]<=x[pivot] && i<last)

**H. 12** i++;

* 1. **13** while(x[j]>x[pivot])

**J. 14** j--;

**K. 15** if(i<j)

{

**L. 16 17 18 19 20**

temp=x[i]; x[i]=x[j];

x[j]=temp;

}

## M. 21 }

**N. 22** temp=x[pivot];

1. x[pivot]=x[j];
2. x[j]=temp;
3. quicksort(x,first,j-1);

**P. 26** quicksort(x,j+1,last);

## Q. 27 }

**O. 28** }

int main()

{

int a[20],i,key,n;

printf("enter the size of the array max of 20 elements"); scanf("%d",&n);

if(n>0)

{

printf("enter the elements of the array"); for(i=0;i<n;i++)

}

else

}

scanf("%d",&a[i]);

quicksort(a,0,n-1);

printf("the elements in the sorted array is:\n"); for(i=0;i<n;i++)

print f("%d\t",a[i]);

printf(“size of array is invalid\n”);

**Cyclomatic Complexity**

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

Where,

**e** is number of edges in DD-Path graph.

**n** is number of nodes in DD-Path graph.

**p** is number of regions connected.(always 1)

**or**

**(for closed closed graph)**

**V(G) =e-n+p**

Number of linearly independent paths (Test cases) for a given graph G **= 23-17+(1)**

**= 6+1**

**= 7** Test cases

**Program graph**

4

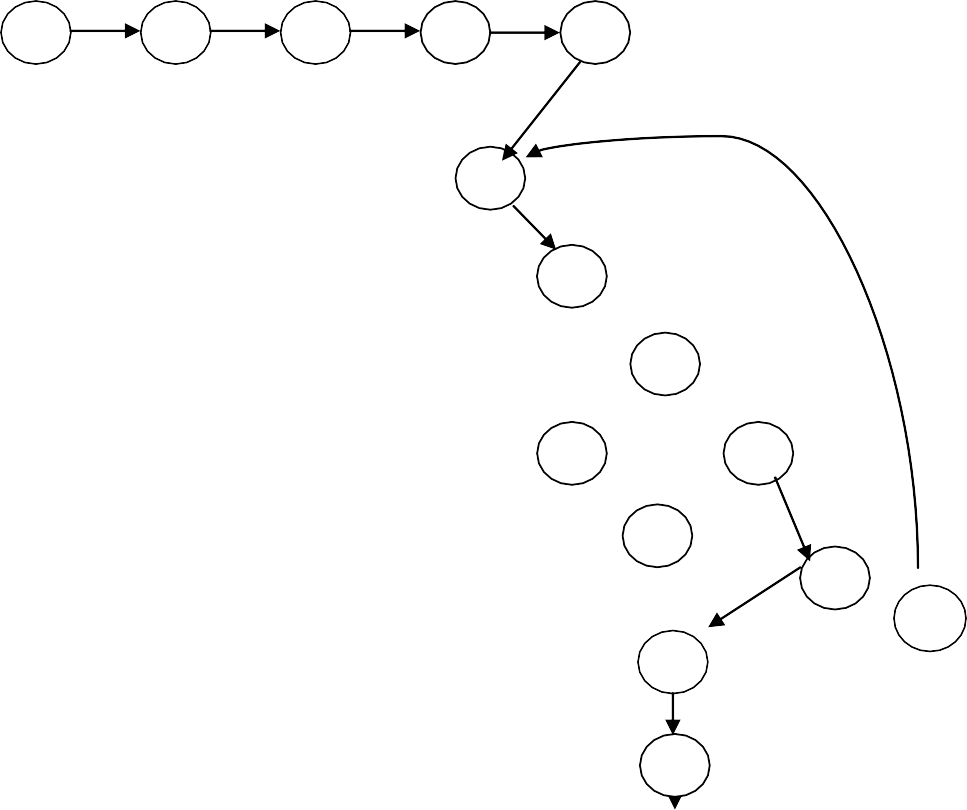
28

26



1

2



5

6

7

8

9

10

22

11

12

13

23

14

24

15

21

16

25

17

18

19

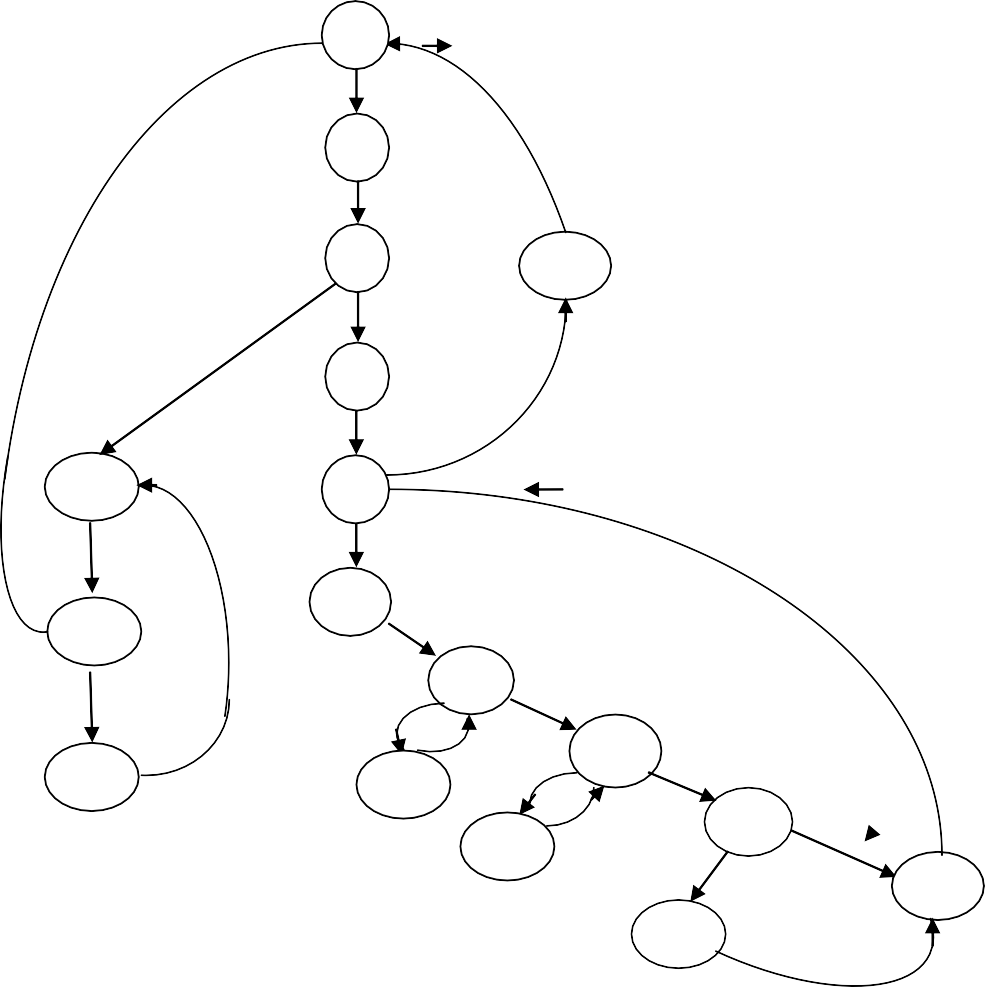
20

3

27

**DD path graph**

|  |  |
| --- | --- |
| **NODES** | **DDPATHS** |
| **1** | **A** |
| **2-3** | **B** |
| **4** | **C** |
| **5-8** | **D** |
| **9** | **E** |
| **10** | **F** |
| **11** | **G** |
| **12** | **H** |
| **13** | **I** |
| **14** | **J** |
| **15** | **K** |
| **16-20** | **L** |
| **21** | **M** |
| **22-25** | **N** |
| **26** | **P** |
| **27** | **Q** |
| **28** | **O** |



A

B

C

N

D

O

E

P

F

G

Q

I

H

J

K

M

L

## McCabe’s Basis path method

Considering DD-Path graph of the program, first we need to find Baseline path. A baseline path consists of maximum number of decision nodes. Using Baseline path we start flipping each decision node for finding new paths.

## Considering Quick Sort program

Considering DD-Path graph of Quick sort function, function starts at node A and Ends at node O. First, Base Line path is formed by considering all decision nodes as shown below.

**Baseline Path:** A B **C** D **E** F **G I K** M E N A B C O **P** A B C O.

Nodes which are bold and large are decision nodes. Now start flipping each decision node.

**Flipping at C :** A B C O.

**Flipping at E :** A B C D E N A B C O.

**Flipping at G :** A B C D E F G H G I K M E N A B C O. **Flipping at I :** A B C D E F G I J I K M E N A B C O. **Flipping at K :** A B C D E F G I K L M E N A B C O.

**Flipping at P :** A B C D E F G I K L M E N A B C O P A B C O.

**Test Cases for Quick Sort Program**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Cases** | **Description** | **Number of elements**  **(n)** | **Array Elements** | **Comment** |
| **TC1** | Enter the basis path consisting of all decision nodes **ABCDEFGIKMENABCOPABCO.** | ---- | Infeasible because path from G to I means no elements in array. | Invalid |
| **TC2** | Enter the basis path consisting of all decision nodes **ABCO.** | 1 | {9} | Valid |
| **TC3** | Enter the basis path consisting of all decision nodes **ABCDENABCO.** | ---- | Path C to D indicates if(first<last) is condition is true. So at first iteration while(x[i]<=x[pivot]&&i<last) condition also should be true and path E to F should be  present. But we have EN so Infeasible | Invalid |
| **TC4** | Enter the basis path consisting of all decision nodes **ABCDEFGHGIKMENABCO.** | 2 | {5,4 } | Valid |
| **TC5** | Enter the basis path consisting of all decision nodes **ABCDEFGIJIKMENABCO.** | ---- | Infeasible because path from G to I means no elements in array. | Invalid |
| **TC6** | Enter the basis path consisting of all decision nodes **ABCDEFGIKLMENABCO.** | ---- | Infeasible because path from G to I means no elements in array. | Invalid |
| **TC7** | Enter the basis path consisting of all decision nodes **ABCDEFGIKLMENABCOPABCO.** | ---- | Infeasible because path from G to I means no elements in array. | Invalid |

# Note

If given array contains a single element, then first=last, if(first<last) condition is true indicates there are more than one elements in the given array. Even when there will be single element While(x[i]<=x[pivot]&&i<last) condition will get executed at least once, because x[i]=x[pivot] is also considered. So path there should be one path G to H present for any feasible solution. So in above table paths containing G to I are all infeasible.

**Program 12: Absolute Letter Grading Testing Technique: Path Testing**

**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 cases , execute these test cases and discuss the test results.**

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

1. **1** int main()

**2** {

1. float per;
2. char grade;
3. printf("enter the percentage\n");
4. scanf("%f",&per);
5. **7** if(per>=90)
6. **8** grade='a';

**D. 9** else if((per>=80) && (per<90))

**E. 10** grade='b';

**F. 11** else if((per>=70) && (per<80))

1. **12** grade='c';

**H. 13** else if((per>=60) && (per<70))

* 1. **14** grade='d';

1. **15** else grade='e';
2. **16** switch(grade)

## L. 17 {

1. **18** case 'a':printf("excellent\n");

**19** break;

1. **20** case 'b':printf("very good\n");

**21** break;

1. **22** case 'c':printf("good\n");

**23** break;

1. **24** case 'd':printf("above average\n");

**25** break;

1. **26** case 'e':printf("satisfactory\n");

**27** break;

## R. 28 }

**S. 29** printf("the percentage is %f and the grade is %c\n",per,grade);

**30** return 0;

**31** }

**Cyclomatic Complexity**

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

Where,

**e** is number of edges in DD-Path graph.

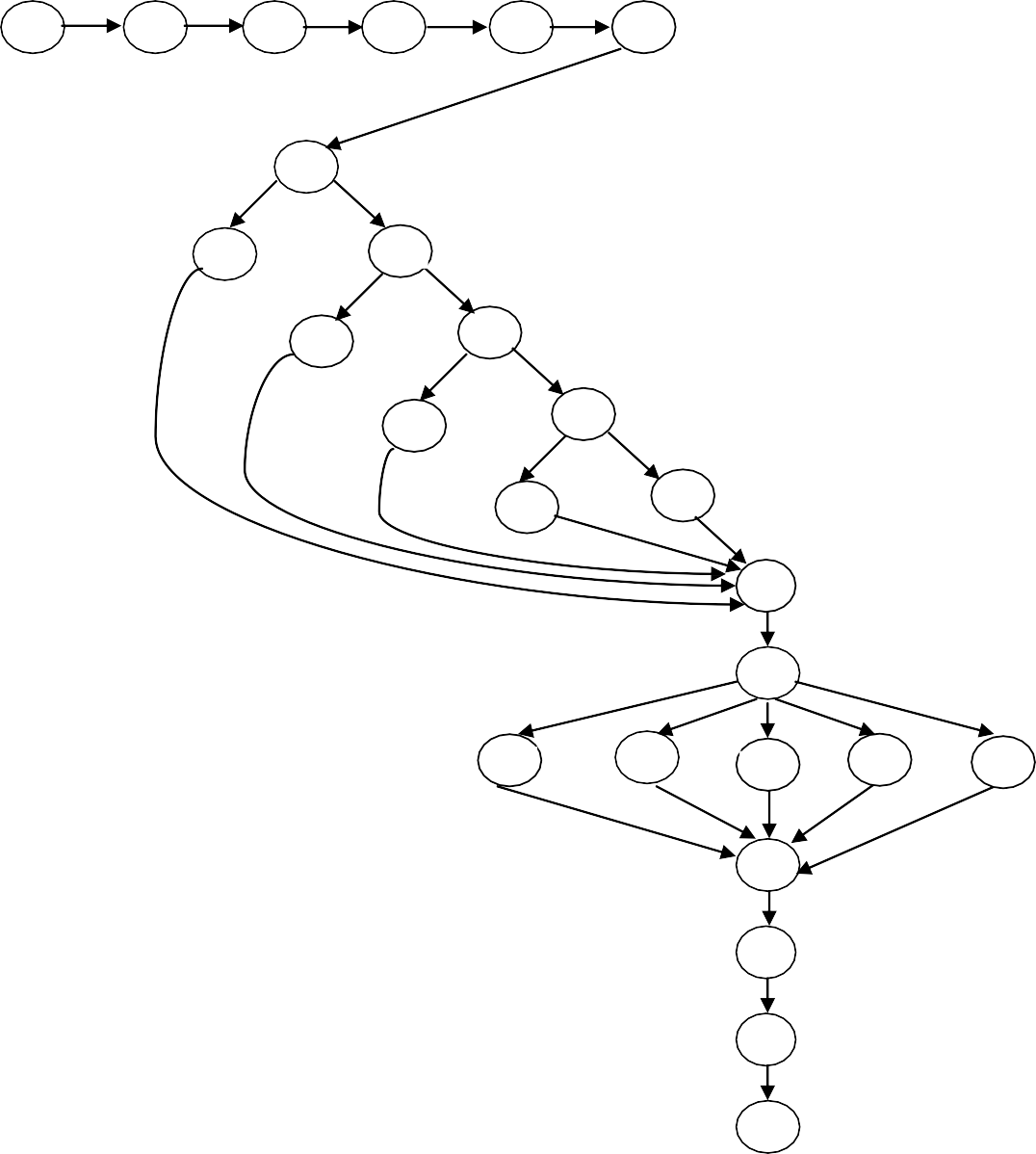
**n** is number of nodes in DD-Path graph.

**p** is number of regions connected.(always 1)

Number of linearly independent paths (Test cases) for a given graph G **= 26-19+2(1)**

**= 7+2**

**= 9** Test cases



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

20

22

24

26

28

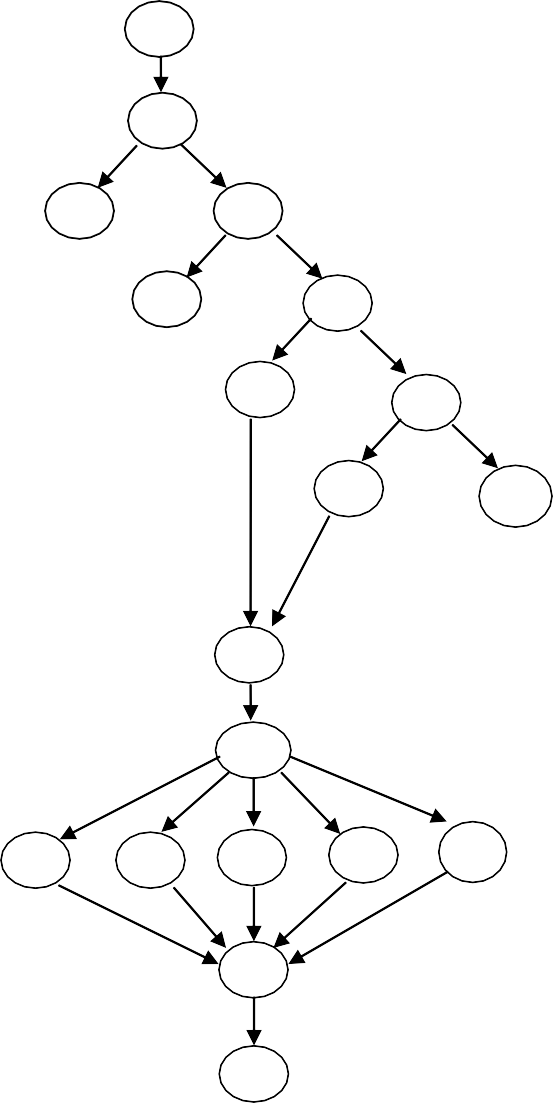
29

30

31

**Program graph**

**DD path graph**



A

B

C

D

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

|  |  |
| --- | --- |
| **Nodes** | **DD-Paths** |
| **1-6** | **A** |
| **7** | **B** |
| **8** | **C** |
| **9** | **D** |
| **10** | **E** |
| **11** | **F** |
| **12** | **G** |
| **13** | **H** |
| **14** | **I** |
| **15** | **J** |
| **16** | **K** |
| **17** | **L** |
| **18-19** | **M** |
| **20-21** | **N** |
| **22-23** | **O** |
| **24-25** | **P** |
| **26-27** | **Q** |
| **28** | **R** |
| **29-31** | **S** |

E

## Finding Basis paths for Letter Grading program using McCabe’s method

Considering DD-Path graph of Absolute Letter grading program, Baseline path is formed by considering all decision nodes as shown below.

**Baseline Path:** A **B D F H J** K **L** M R S.

Nodes which are bold and large are decision nodes. Now start flipping each decision node.

|  |  |  |
| --- | --- | --- |
| **Flipping** | **at** | **B :** A B C K L M R S. |
| **Flipping** | **at** | **D :** A B D E K L M R S. |
| **Flipping** | **at** | **F :** A B D F G K L M R S. |
| **Flipping** | **at** | **H :** A B D F H I K L M R S. |
| **Flipping** | **at** | **L :** A B D F H J K L N R S. |
|  |  | A B D F H J K L O R S. A B D F H J K L P R S.  A B D F H J K L Q R S. |

**Test Cases for Letter Grading Pogram**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Comment** |
| **TC1** | Enter the basis path containing the decision nodes **ABDFHJKLMRS** | 55 | Satisfactory | Node J indicates grade „e‟ so case „e‟  should be executed i.e., node Q. but there is no Q in this path so Infeasible | Invalid |
| **TC2** | Enter the basis path containing the decision nodes **ABCKLMRS** | 95 | Excellent | **Excellent** | Valid |
| **TC3** | Enter the basis path containing the decision nodes **ABDEKLMRS** | 85 | Very good | Node E indicates grade „b‟ so case „b‟ should be executed i.e., node N. but  there is no N in this path so Infeasible | Invalid |
| **TC4** | Enter the basis path containing the decision nodes **ABDFGKLMRS** | 75 | Good | Node F indicates grade „c‟ so case „c‟  should be executed i.e., node O. but there is no O in this path so Infeasible | Invalid |
| **TC5** | Enter the basis path containing the decision nodes **ABDFHIKLMRS** | 65 | Above average | Node H indicates grade „d‟ so case „d‟ should be executed i.e., node P. but  there is no P in this path so Infeasible | Invalid |
| **TC6** | Enter the basis path containing the decision nodes **ABDFHJKLNRS** | 55 | Satisfactory | Node J indicates grade „e‟ so case „e‟ should be executed i.e., node Q. but  there is no Q in this path so Infeasible | Invalid |
| **TC7** | Enter the basis path containing the decision nodes **ABDFHJKLORS** | 55 | Satisfactory | Node J indicates grade „e‟ so case „e‟ should be executed i.e., node Q. but  there is no Q in this path so Infeasible | Invalid |
| **TC8** | Enter the basis path containing the decision nodes **ABDFHJKLPRS** | 55 | Satisfactory | Node J indicates grade „e‟ so case „e‟  should be executed i.e., node Q. but there is no Q in this path so Infeasible | Invalid |
| **TC9** | Enter the basis path containing the decision nodes **ABDFHJKLQRS** | 55 | Satisfactory | **Satisfactory** | Valid |

In above table we got test cases containing only **Excellent** and **satisfactory** type outputs. As we have five types of outputs in our program, three types of outputs ( i.e., **good, very good** and **above average**) are left untested. So for completeness we add three more tests for left out cases as shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TC10** | Enter the basis path containing the decision nodes  **ABDEKLNRS** | 85 | Very good | **Very good** | Valid |
| **TC11** | Enter the basis path containing the decision nodes  **ABDFGKLORS** | 75 | Good | **Good** | Valid |
| **TC12** | Enter the basis path containing the decision nodes  **ABDFHIKLPRS** | 65 | Above average | **Above average** | Valid |

**ALL THE VERY BEST…**