# CITY ENGINEERING COLLEGE

**DODDAKALLASANDRA, KANAKAPURA ROAD**

**BENGALURU - 560062**

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**

**VI SEMESTER**

# SOFTWARE TESTING LABORATORY

## (SUBJECT CODE: 21ISL66)

**By**
**M Mathivanan**
**Asst.professor**

***LABORATORY MANUAL***

**1. 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.**

**Program**

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


}
```

Boundary values for the output range, near threshold points of $1000 and $1800

| Case | Locks | Stocks | Barrels | Sales | Commission | Comment |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 100 | 10 | Output minimum |
| 2 | 1 | 1 | 2 | 125 | 12.5 | Output minimum+ |
| 3 | 1 | 2 | 1 | 130 | 13 | Output minimum+ |
| 4 | 2 | 1 | 1 | 145 | 14.5 | Output minimum+ |
| 5 | 5 | 5 | 5 | 500 | 50 | Midpoint |
| 6 | 10 | 10 | 9 | 975 | 97.5 | Border point- |
| 7 | 10 | 9 | 10 | 970 | 97 | Border point- |
| 8 | 9 | 10 | 10 | 955 | 95.5 | Border point- |
| 9 | 10 | 10 | 10 | 1000 | 100 | Border point |
| 10 | 10 | 10 | 11 | 1025 | 103.75 | Border point+ |
| 11 | 10 | 11 | 10 | 1030 | 104.5 | Border point+ |
| 12 | 11 | 10 | 10 | 1045 | 106.75 | Border point+ |
| 13 | 14 | 14 | 14 | 1400 | 160 | Midpoint |
| 14 | 18 | 18 | 17 | 1775 | 216.25 | Border point- |
| 15 | 18 | 17 | 18 | 1770 | 215.5 | Border point- |
| 16 | 17 | 18 | 18 | 1755 | 123.25 | Border point- |
| 17 | 18 | 18 | 18 | 1800 | 220 | Border point |
| 18 | 18 | 18 | 19 | 1825 | 225 | Border point+ |
| 19 | 18 | 19 | 18 | 1830 | 226 | Border point+ |
| 20 | 19 | 18 | 18 | 1845 | 229 | Border point+ |
| 21 | 48 | 48 | 48 | 4800 | 820 | Midpoint |
| 22 | 70 | 80 | 89 | 7775 | 1415 | Output Maximum- |
| 23 | 70 | 79 | 90 | 7770 | 1414 | Output Maximum- |
| 24 | 69 | 80 | 90 | 7755 | 1411 | Output Maximum- |
| 25 | 70 | 80 | 90 | 7800 | 1420 | Output Maximum |

| Case | Locks | Stocks | Barrels | Sales | Commission | Comment |
|---|---|---|---|---|---|---|
| 1 | 10 | 11 | 9 | 1005 | 100.75 | Border point+ |
| 2 | 18 | 17 | 19 | 1795 | 219.25 | Border point- |
| 3 | 18 | 19 | 17 | 1805 | 221 | Border point+ |

**2. 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.**

A rifle salesperson in the former Arizona Territory sold rifle locks, stocks, and barrels made by a gunsmith in Missouri. Locks cost $45, stocks cost $30, and barrels cost $25. The salesperson had to sell at least one complete rifle per month, and production limits were such that the most the salesperson could sell in a month was 70 locks, 80 stocks, and 90 barrels. After each town visit, the salesperson sent a telegram to the Missouri gunsmith with the number of locks, stocks, and barrels sold in that town. At the end of a month, the salesperson sent a very short telegram showing –1 locks sold. The gunsmith then knew the sales for the month were complete and computed the salesperson's commission as follows:

1) 10% on sales upto and including $1000.

2) 15% of the next $800.

3) And 20% on any sales in excess of $1800

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

The valid classes of the input variables are as follows
F1 = {locks: 1 ≤ locks ≤ 70}
F2 = {locks = -1}
P1 = {stocks : 1 ≤ stocks ≤ 80}
B1 = {barrels : 1 ≤ barrels ≤ 90}
The corresponding invalid classes of the input variables are as follows:
F3 = {locks: locks = 0 or locks < -1}
F4 = {locks : locks > 70}
P2 = {stocks : stocks < 1}
P3 = {stocks :stocks > 80}
B2 = {barrels : barrels < 1}
B3 = {barrels : barrels > 90}

```
#include<stdio.h>
int main()
{
int locks, stocks, barrels, t_sales, flag = 0;
float commission;
printf("Enter the total number of locks");
scanf("%d",&locks);
if ((locks <= 0) || (locks > 70))
{
flag = 1;
}
printf("Enter the total number of stocks");
scanf("%d",&stocks);
if ((stocks <= 0) || (stocks > 80))
{
```

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


}
```

**1) & 2) Weak Normal & Strong Normal Equivalence Class:** Since the number of valid classes is equal to the number of independent variables, so we have exactly one weak normal equivalence class test case and again, it is identical to the strong normal equivalence class test case.

| Test Case ID | Locks | Stocks | Barrels | Expected Output for sales |
|---|---|---|---|---|
| WN1, SN1 | 35 | 40 | 45 | 3900 |

**3) Weak Robust Equivalence Class:** Test Cases falling under this category are as under

| Test Case ID | Locks | Stocks | Barrels | Expected Output for sales |
|---|---|---|---|---|
| WR1 | 10 | 10 | 10 | $100 |

| | | | | |
|---|---|---|---|---|
| WR2 | -1 | 40 | 45 | Program Terminates |
| WR3 | -2 | 40 | 45 | Locks out of range |
| WR4 | 71 | 40 | 45 | Locks out of range |
| WR5 | 35 | -1 | 45 | Stocks out of range |
| WR6 | 35 | 81 | 45 | Stocks out of range |
| WR7 | 35 | 40 | -1 | Barrels out of range |
| WR8 | 35 | 40 | 91 | Barrels out of range |

**4) Strong Robust Equivalence Class:** Test Cases falling under this category are

| Test Case ID | Locks | Stocks | Barrels | Expected Output for sales |
|---|---|---|---|---|
| SR1 | -2 | 40 | 45 | Value of Locks not in range 1 - 70 |
| SR2 | 35 | -1 | 45 | Value of Stocks not in range 1 - 80 |
| SR3 | 35 | 40 | -2 | Value of Barrels not in range 1 - 90 |
| SR4 | -2 | -1 | 45 | Value of Locks & Stocks are not in their ranges |
| SR5 | -2 | 40 | -1 | Value of Locks & Barrels are not in their ranges |
| SR6 | 35 | -1 | -1 | Value of Stocks & Barrels are not in their ranges |
| SR7 | -2 | -1 | -1 | Value of Locks, Stocks & Barrels are not in their ranges |

Improved output range equivalence class test cases

In order to calculate the commission of sales, consider the equivalence classes defined on the output range. Sale if a function of the number of locks, stocks, and barrels sold:

$$Sale=45 \text{ x } locks + 30 \text{ x } stocks + 25 \text{ x } barrels$$

Equivalence classes of three variables by commission ranges:

S1 = {<locks, stocks, barrels>:sales≤1000}

S1 = {<locks, stocks, barrels>:1000< sales ≤1800}

S1 = {<locks, stocks, barrels>:sales>1800}

| **Test Case ID** | **Locks** | **Stocks** | **Barrels** | **Sales** | **Commission** |
|---|---|---|---|---|---|
| OR1 | 5 | 5 | 5 | 500 | 50 |
| OR1 | 15 | 15 | 15 | 1500 | 175 |
| OR1 | 25 | 25 | 25 | 2500 | 360 |

**3. 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.**

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


 }
```

## INPUT DECISION TABLE

| | RULES | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Conditions** | **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 | | Commen |
|---|---|---|---|---|---|---|---|
| | | Locks | Stocks | Barrels | Sales | Com | |
| **IDT1** | Enter no. of locks=-1 | -1 | - | - | 0<br>Program Terminates | 0 | valid |
| **IDT2** | Enter the valid no. of locks and invalid values for stocks and barrels | 20 | 81 | 91 | 900<br>Invalid no.of stocks and barrels | 90 | valid |
| **IDT3** | Enter the valid values for locks, stocks and invalid value for barrels | 20 | 20 | 96 | 1500<br>Invalid no.of barrels | 175 | valid |
| **ID** | Enter the valid values for locks and barrels and invalid value for stocks | 20 | -1 | 20 | 1400<br>Invalid no.of stocks | 160 | valid |

| | | | | | | |
|---|---|---|---|---|---|---|
| T 4 | | | | | | |
| IDT5 | Enter the valid values for locks, stocks and barrels | 20 | 20 | 20 | 2000 260<br>Calculates sales and commission | valid |
| IDT6 | Enter the invalid values for locks, stocks and barrels | -2 | 81 | -1 | 0 0<br>Invalid no.of locks, Stocks andbarrels. | valid |
| IDT7 | Enter the valid value for stocks and invalid values for locks and barrels | -2 | 20 | 91 | 600 60<br>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<br>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<br>Invalid no.of locks | valid |

**COMMISSION CALCULATION DECISION TABLE**

| | **RULES** | **R1** | **R2** | **R3** |
|---|---|---|---|---|
| **Conditions** | **C1: Sales> 1801** | T | F | F |
| | **C2: Sales >1001 and sales <= 1800** | --- | T | F |
| | | --- | --- | 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 |

**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 boundary-value analysis, execute the**

**Program**

```
#include<stdio.h>
#include<math.h>
int main()
{
	int a,b,c,match;
	printf("Enter 3 integers which are sides of a triangle);
	scanf("%d%d%d",&a,&b,&c);
	printf("side A is %d\n",a);
	printf("side B is %d\n",b);
	printf("side C is %d\n",c);
	match=0;
	if(a==b)
		match=match+1;
	if(a==c)
		match=match+2;
	if(b==c)
		match=match+3;
	if(match==0)
	{
		if((a+b)<=c)
			printf("not a triangle\n");
		else if((b+c)<=a)
			printf("not a traingle\n");
		else if((a+c)<+b)
			printf("NOT A TRAINGLE\n");
		else
			printf("scalene");
	}
	if(match==1)
	{
		if((a+c)<=b)
			printf("not a triangle\n");
		else
			printf("isosceles");
	}
	else if(match==2)
	{
		if((a+c)<=b)
			printf("not a trianlge\n");
		else
```

```
                    printf("isosceles\n");
        }
        else if(match==3)
        {
                if((b+c)<=a)
                    printf("not a triangle\n");
                else
                    printf("isosceles\n");
        }
        else
                printf("Equilateral\n");
}
```

**Boundary-value analysis**

"The triangle program accepts three integers, a, b and c as input. These are taken to be the sides of a triangle. The integers a, b and c must satisfy the following conditions

C1: $1 \leq a \leq 200$

C2: $1 \leq b \leq 200$

C3: $1 \leq c \leq 200$

C4: $a < b+c$

C5: $b < a+c$

C6: $c < a+b$

The output of the program may be either of: Equilateral Triangle, Isosceles Triangle, Scalene or "Not a Traingle".

We know that our range is [1, 200] where 1 is the lower bound and 200 being the upper bound.

Also, we find that this program has three inputs like a, b and c.

Hence for our case number of inputs or n = 3

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

Now we can draw the following Table indicating all the 13 test-cases.

| Test Case ID | Side "a" | Side "b" | Side "c" | Expected Output |
|---|---|---|---|---|
| 1 | 100 | 100 | 1 | Isosceles Triangle |
| 2 | 100 | 100 | 2 | Isosceles Triangle |
| 3 | 100 | 100 | 100 | Equilateral Triangle |
| 4 | 100 | 100 | 199 | Isosceles Triangle |
| 5 | 100 | 100 | 200 | Not a Triangle |
| 6 | 100 | 1 | 100 | Isosceles Triangle |
| 7 | 100 | 2 | 100 | Isosceles Triangle |

| | | | | |
|---|---|---|---|---|
| 8 | 100 | 100 | 100 | Equilateral Triangle |
| 9 | 100 | 199 | 100 | Isosceles Triangle |
| 10 | 100 | 200 | 100 | Not a Triangle |
| 11 | 1 | 100 | 100 | Isosceles Triangle |
| 12 | 2 | 100 | 100 | Isosceles Triangle |
| 13 | 100 | 100 | 100 | Equilateral Triangle |
| 14 | 199 | 100 | 100 | Isosceles Triangle |
| 15 | 200 | 100 | 100 | Not a Triangle |

It may be noted that as explained above that we can have 13 test cases (4n + 1) for this problem. But instead of 13, now we have 15 test cases.

Moreover we can see that the test cases vide ID number 8 and 13 are redundant. Hence we can ignore them. However, we do not ignore test case ID number 3, as we must consider at least one test case out of these three. Thus it is evident that it is a mechanical activity.

Hence we can say that these 13 test cases are sufficient to test this program using BVA technique.

**5. 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.**

## C Program

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


}
```

**Dataflow Testing**

```
program  lock_stock_and_barrel
const
      lock_price    = 45.0;
      stock_price   = 30.0;
      barrel_price  = 25.0;
type
      STRING_30 = string[30];  (Salesman's Name)
var
      locks, stocks, barrels, num_locks, num_stocks,
      num_barrels, salesman_index, order_index :  INTEGER;
      sales, commission : REAL;
      salesman : STRING_30;

BEGIN (program lock_stock_and_barrel)
```

```
   FOR salesman_index := 1 TO 4 DO
      BEGIN
         READLN(salesman);
         WRITELN ('Salesman is ', salesman);
         num_locks := 0;
         num_stocks := 0;
         num_barrels := 0;
         READ(locks);
         WHILE locks <> -1 DO
           BEGIN
              READLN(stocks, barrels);
              num_locks := num_locks + locks;
              num_stocks := num_stocks + stocks;
              num_barrels := num_barrels + barrels;
              READ(locks);
           END; {WHILE locks}
      READLN;
       WRITELN('Sales for ',salesman);
       WRITELN('Locks sold: ', num_locks);
       WRITELN('Stocks sold: ', num_stocks);
       WRITELN('Barrels sold: ', num_barrels);
       sales := lock_price*num_locks + stock_price*num_stocks
                   + barrel_price*num_barrels;
       WRITELN('Total sales: ', sales:8:2);
      WRITELN;
       IF (sales > 1800.0) THEN
         BEGIN
             commission := 0.10 * 1000.0;
             commission := commission + 0.15 * 800.0;
             commission := commission + 0.20 * (sales-1800.0);
         END;
        ELSE IF (sales > 1000.0) THEN
         BEGIN
             commission := 0.10 * 1000.0;
             commission := commission + 0.15*(sales - 1000.0);
         END
        ELSE commission := 0.10 * sales;
        WRITELN('Commission is $',commission:6:2);
   END; {FOR salesman}
 END. {program lock_stock_and_barrel}
```

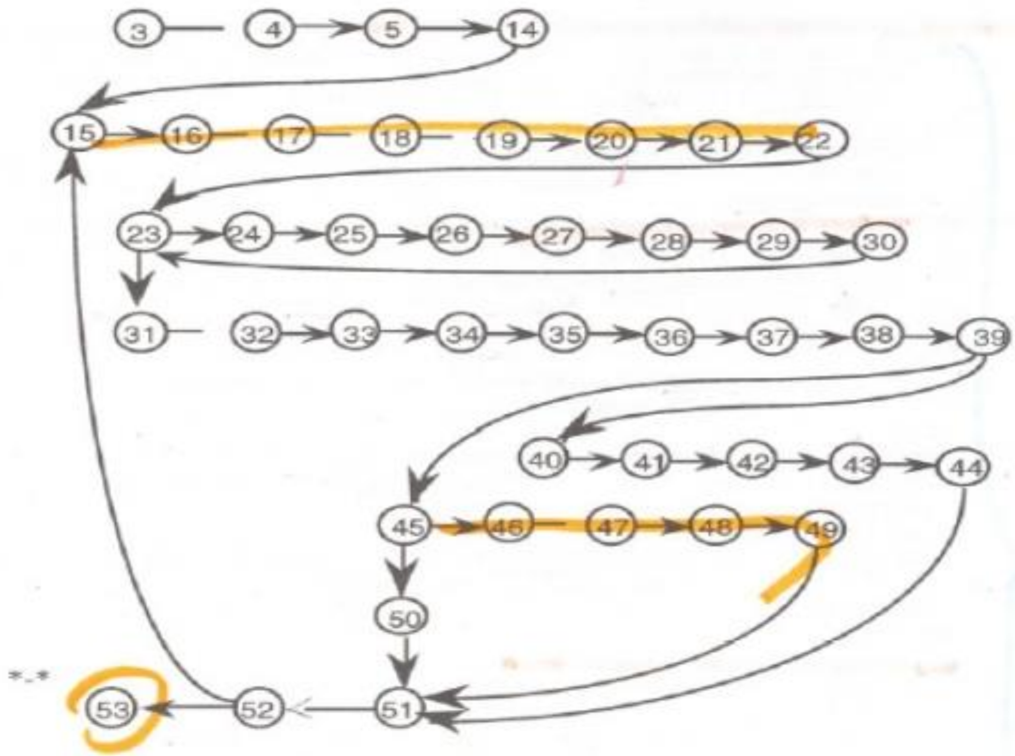



**Figure 10.1** Program Graph of the Commission Program

Table 1 DD-Paths in Figure 10.1

| DD-Path | Nodes |
|---|---|
| 1 | 14 |
| 2 | 15 - 22 |
| 3 | 23 |
| 4 | 24 - 30 |
| 5 | 31 - 39 |
| 6 | 40 - 44 |
| 7 | 45 |
| 8 | 46 - 49 |
| 9 | 50 |
| 10 | 51, 52 |
| 11 | 53 |

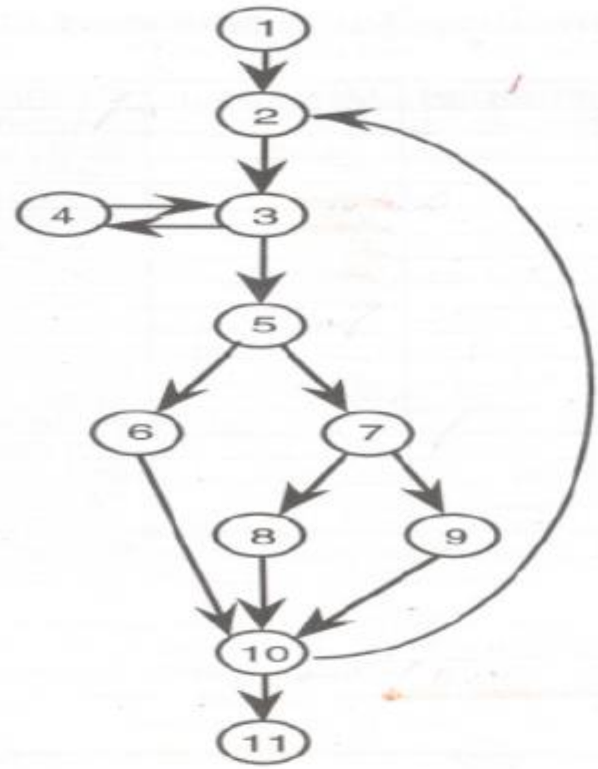



**Figure 10.2** DD-Path Graph of the Commission Program

- Stocks variable
  - DEF(stocks, 25)
  - USE(stocks, 27)
  - DuPath
    - P0=<25, 27>
  - Dc-Path
    - P0=<25,27>
- Locks variable
  - DEF(locks,22)
  - DEF(locks, 29)
  - USE(locks,23)
  - Use(locks,26)
  - Du-path
    - p1=<22,23>
    - p2=<22,23,24,25,26> (begin the loop)
    - P3=<29,30,23>
    - p4=<29,30,23,24,25,26> (repeat the loop)
    - p1’=<22,23,31> (by pass the loop)
    - p3’=<29,30,23,31> (exist the loop)
  - Dc-paths
    - p1,p2,p3,p4,p1’,p3’
  - Complete set of test cases for the WHILE-loop: p1’, p2, p3’, p4

Table 2: Define/Use Information for locks, stocks, and num_locks

| Variable | Defined at | Used at | Comment |
|---|---|---|---|
| locks | 9 | | (to compiler) |
| locks | 22 | | READ |
| locks | | 23 | predicate use |
| locks | | 26 | computation use |
| locks | 29 | | READ |
| stocks | 9 | | (to compiler) |
| stocks | 25 | | READ |
| stocks | | 27 | computation use |
| num_locks | 9 | | (to compiler) |
| num_locks | 19 | | assignment |
| num_locks | 26 | | assignment |
| num_locks | | 26 | computation use |
| num_locks | | 33 | WRITE |
| num_locks | | 36 | computation use |

## Table 3: Define/Use Information for Sales and Commission

| Variable | Defined at | Used at | Comment |
|---|---|---|---|
| sales | 11 | | (to compiler) |
| sales | 36 | | assignment |
| sales | | 37 | WRITE |
| sales | | 39 | predicate use |
| sales | | 43 | computation use |
| sales | | 45 | predicate use |
| sales | | 48 | computation use |
| sales | | 50 | computation use |
| commission | 11 | | (to compiler) |
| commission | 41 | | assignment |
| commission | 42 | | assignment |
| commission | | 42 | computation use |
| commission | 43 | | assignment |
| commission | | 43 | computation use |
| commission | 47 | | assignment |
| commission | 48 | | assignment |
| commission | | 48 | computation use |
| commission | 50 | | assignment |
| commission | | 51 | WRITE |

**DU-paths w.r.t. num_locks**

- DU-Paths w.r.t. num_locks
  - Used in computational uses (c-uses)
  - Defining nodes
    - DEF(num_locks,19)
    - DEF(num_locks,26)
  - Usage nodes
    - USE(num_locks,26)
    - USE(num_locks,33)
    - USE(num_locks,36)
  - DU-paths
    - P5=<19,20,21,22,23,24,25,26> (dc-path)
    - P6=<19,20,21,22,23,24,25,26, 26, 27,28,29,30,31,32,33> (NOT dc-path)
    - Corrected p6
      - ← P6=<19,20,21,22,23,24,25,26, 27,28,29,30,31,32,33>
    - p7 = <19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36)
    - p7 can also be rewritten as

P7= <p6,34,35,36> (NOT dc-path because it contains node 26)

- P8=<26,27,28,29,30,31,32,33> (subpath of p7; dc-path)
- P9=<26,27,28,29,30,31,32,33,34,35,36> (subpath of p7; dc-path)

**DU-paths w.r.t. sales**

- Du-paths w.r.t. sales
  - ONLY one defining node for sales (i.e. the paths are dc-paths)

p10=<36,37>
p11=<36,37,38,39>
p12=<36,37,38,39,40,41,42,43>

- Look at IF, ELSE IF (statements 39-50)

p13=<36,37,38,39,45,46,47,48>
p14=<36,37,38,39,45,50>

DU-paths w.r.t. Commission

- Du-paths w.r.t. Commission
  - Consider only du-paths that begin with three “real” defining nodes for commission
  - p15 =<43,51>
  - p16=<48,51>
  - p17=<50,51>
- Table 4 shows the full set of du-paths

Table 4: Du-Paths in Figure 10.1

| Du-Path | Variable | Def Node | Use Node |
|---|---|---|---|
| 1 | locks | 22 | 23 |
| 2 | locks | 22 | 26 |
| 3 | locks | 29 | 23 |
| 4 | locks | 29 | 26 |
| 5 | stocks | 25 | 27 |
| 6 | barrels | 25 | 28 |
| 7 | num_locks | 19 | 26 |
| 8 | num_locks | 19 | 33 |
| 9 | num_locks | 19 | 36 |
| 10 | num_locks | 26 | 33 |
| 11 | num_locks | 26 | 36 |
| 12 | num_stocks | 20 | 27 |
| 13 | num_stocks | 20 | 34 |
| 14 | num_stocks | 20 | 36 |
| 15 | num_stocks | 27 | 34 |
| 16 | num_stocks | 27 | 36 |
| 17 | num_barrels | 21 | 28 |
| 18 | num_stocks | 21 | 35 |
| 19 | num_stocks | 21 | 36 |
| 20 | num_stocks | 28 | 35 |
| 21 | num_stocks | 28 | 36 |
| 22 | sales | 36 | 37 |
| 23 | sales | 36 | 39 |
| 24 | sales | 36 | 43 |
| 25 | sales | 36 | 45 |
| 26 | sales | 36 | 48 |
| 27 | sales | 36 | 50 |
| 28 | commission | 41 | 42 |
| 29 | commission | 42 | 43 |
| 30 | commission | 43 | 51 |
| 31 | commission | 47 | 48 |
| 32 | commission | 48 | 51 |
| 33 | commission | 50 | 51 |

**6. 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**.

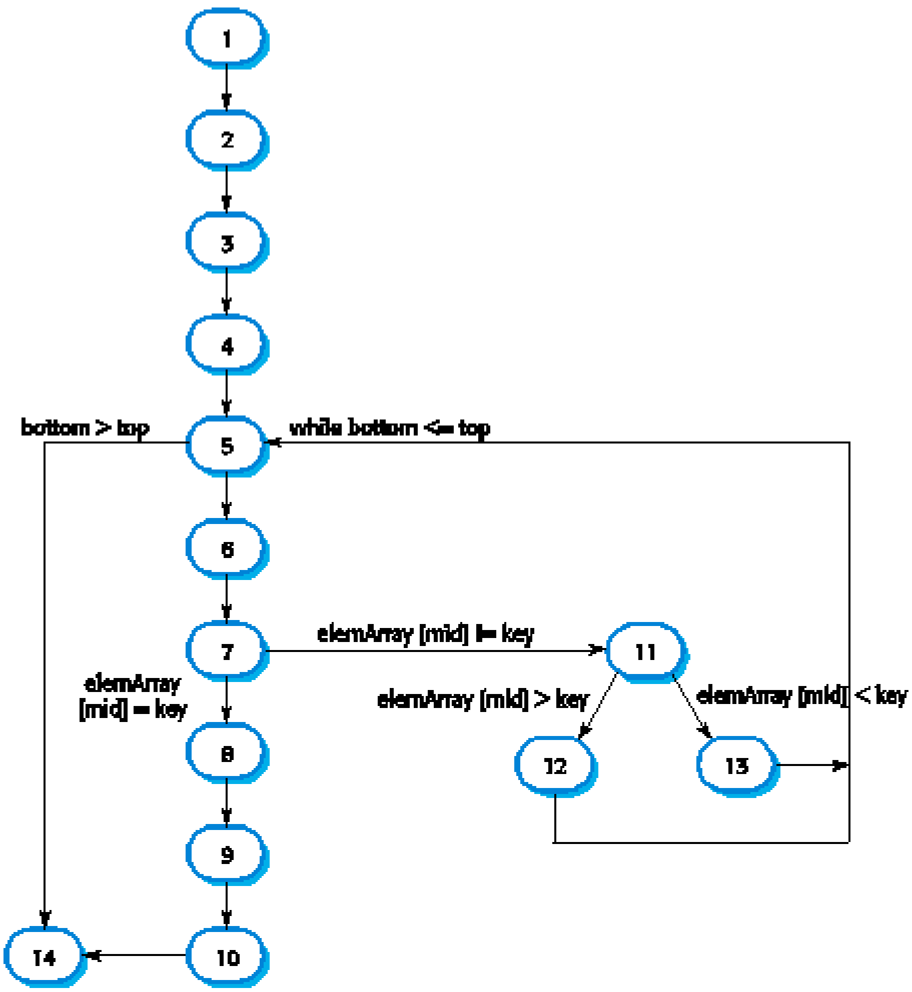
**Program**

```
# include<stdio.h>
int main()
        {
        int n,i,key,low,high,mid,f=0,pos;
        float a[10];
        printf("Enter the no. of elements\n");
        scanf("%d",&n);
        printf("enter the elements in  the ascending order\n");
        for(i=0;i<n;i++)
        scanf("%f",&a[i]);
        printf("Enter the key element to be searched\n");
        scanf("%d",&key);
        low=0;
        high=n-1;
        while(low<=high)
        {
        mid=(low+high)/2;
        if(key==a[mid])
        {
        f=1;
        pos=mid;
        break;
        }
        if(key>a[mid])
        low=mid+1;
        else
        high=mid-1;
        }
        if(f==1)
        {
        printf("search successfull\n");
        printf("key %d is at location %d",key,pos+1);
        }
    else
        printf("search unsuccessfull\nkey not found");
        return 0;
        }
```

/*

**output1:**
Enter the no. of elements
5
enter the elements in  the ascending order
10
20
30
40
50
Enter the key element to be searched
40
search successfull
key 40 is at location 4
output2:
Enter the no. of elements
5
enter the elements in  the ascending order
10
20
30
40
50
Enter the key element to be searched
60
search unsuccessfull
key not found*/

**Basis paths**




**Independent paths**

- **1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14**
- **1, 2, 3, 4, 5, 14**
- **1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …**
- **1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …**
- **Test cases should be derived so that all of these paths are executed**
- **A dynamic program analyser may be used to check that paths have been executed**

| **Test Case ID** | **Value of n** | **Array n elements** | **Enter key** | **Expected output** | **Test pass/fail** |
|---|---|---|---|---|---|
| **T1** | **5** | **50,40,30,20,10** | **10** | **Invalid input** | **fail** |
| **T2** | **5** | **10,20,30,40,50** | **10** | **Valid input, Key at 1st position** | **pass** |
| **T3** | **5** | **10,20,30,40,50** | **60** | **Key not found** | **pass** |
| **T4** | **5** | **50,40,30,20,10** | **10** | **Invalid input** | **fail** |
| **T5** | **5** | **10,20,30,40,50** | **10** | **Valid input, Key at 1st position** | **pass** |
| **T3** | **5** | **10,20,30,40,50** | **60** | **Key not found** | **pass** |
| | | | | | |

**Since V (G)=4. There are 4 paths**

**Path 1: 1,2,3,6,7,8**

**Path 2: 1,2,3,5,7,8**

**Path 3: 1,2,4,7,8**

**Path 4: 1,2,4,7,2,4,...7,8**

**Finally we derive test cases to exercise these paths.**