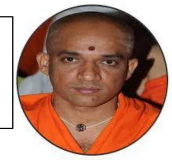
VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"JNANA SANGAMA" BELAGAVI - 590018



SOFTWARE TESTING LAB MANUAL
[18ISL66]

Prepared by By:

Mr.YOGARAJA G S R
Mr.ANAND T

S.J.C.INSTITUTE OF TECHNOLOGY
DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING
CHICKBALLAPUR -562101

During the Year: 2020-21

SOFTWARE TESTING LABORATORY
SEMESTER – VI

Subject Code 18ISL66	**IA Marks 40**
Number of Lecture Hours/Week 0:2:2	**Exam Marks 60**
Total Number of Lecture Hours 36	**Exam Hours 03**

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

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

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

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

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

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

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

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

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

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

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

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

PROGRAM OUTCOMES:

PO1	Ability to apply the knowledge of Mathematics, Science and Engineering Fundamentals with the domain knowledge of Information science to solve engineering problems
PO2	Identify, formulate and analyze engineering problems using the knowledge of mathematics, science and Engineering with the domain knowledge of Information science to arrive at substantiated conclusions.
PO3	Ability to Design and develop computer based solutions for practical engineering problems under realistic constraints.
PO4	Ability to design, conduct experiment, analyze, interpret data and arrive at valid conclusions.
PO5	Ability to select and use modern tools and techniques for computing practice.
PO6	Ability to reason by assessing the societal, health, safety, legal, cultural issues and exhibit responsibility relevant to professional engineering practice.
PO7	Ability to Understand the impact of Engineering solutions in societal and Environmental aspects and exhibit sustainable development.
PO8	Ability to apply ethical principles and commit to norms of professional engineering practice.
PO9	Ability to function effectively as an individual and as a team member or leader in diverse teams and multidisciplinary settings.
PO10	Ability to Communicate effectively, design good documentations and make clear presentations.
PO11	Ability to understand and apply the engineering, finance and management principles to manage multidisciplinary projects.
PO12	Ability to recognize the need and engage in lifelong learning for professional Growth.

Course outcomes:

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

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

Requirements:-

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

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

R3. If the requirement R1 satisfied then the system should determine the type of triangle, can be

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

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

Design

Algorithm:-

Step 1: Input a, b and c i.e three integer values which represents three side of the triangle

Step 2: if (a< (b+c)) and (b<(a+c)) and (c<(a+b)) then do step 3
else
Print not a triangle. Do step 6.

Step 3: if (a=b) and (b=c) then

Print triangle formed is equilateral. Do step 6.

Step 4: if (a! = b) and (a! =c) and (b! =c) then

Print triangle formed is scalene. Do step 6.

Step 5: print triangle formed is Isosceles.

Step 6: stop.

Program Code:

```
print("Enter the three sides of the triangle")
a=int(input("Enter the value of a"))
b=int(input("Enter the value of b"))
c=int(input("Enter the value of c"))
if(a<=10)or(b<=10)or(c<=10)or(a<1)or(b<1)or(c<1):
    print("Out of Range ")
    exit()
if(a<b+c)and(b<a+c)and(c<a+b):
    if(a==b)and(b==c):
        print("Equilateral triangle")
    elif(a!=b)and(a!=c)and(b!=c):
        print("Scalene triangle")
    else:
        print("Isosceles triangle")
else:
    print("Triangle cannot be formed")
```

Testing

1. Technique used : Boundary value analysis
2. Test case design

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

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

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

$1 \leq a \leq 10$

$1 \leq b \leq 10$

$1 \leq c \leq 10$

Equivalence classes for a:

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

Equivalence classes for b:

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

Equivalence classes for c:

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

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

TC ID	Test Cases description	Input data			Expected output	Actual output	Status
		a	b	c			
1	Input of a is min	1	5	5	Isosceles triangle		
2	Input of a is min+	2	5	5	Isosceles triangle		
3	Input of a is nom	5	5	5	Equilateral triangle		
4	Input of a is max-	9	5	5	Isosceles triangle		
5	Input of a is max	10	5	5	Not a triangle		
6	Input of b is min	5	1	5	Isosceles triangle		
7	Input of b is min+	5	2	5	Isosceles triangle		
8	Input of b is max-	5	9	5	Isosceles triangle		
9	Input of b is max	5	10	5	Not a triangle		
10	Input of c is min	5	5	1	Isosceles triangle		
11	Input of c is min+	5	5	2	Isosceles triangle		
12	Input of c is max-	5	5	9	Isosceles triangle		
13	Input of c is max	5	5	10	Not a triangle		

Execution and Result Discussion

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

Test Report :

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed :

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

Requirements Specification

Problem definition: The commission problem includes a sales person in the former Arizona territory sold the rifle locks, stocks and barrels made by gunsmith in Missouri. Cost includes

Locks = 45 $

Stocks = 30 $

Barrels = 25 $

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

After each town visit, the sales person sent a telegram to the Missouri gunsmith with the number of locks, stocks and barrels sold in the town. At the end of the month, the sales person sent a very short telegram showing -1 lock sold. The gunsmith then knew the sales for the month were complete and computed the sales person's commission as follows.

On sales up to (and including) $ 1000 = 10 %

On sales up to (and includes) $ 1800 = 15 %

On the sales in excess of $ 1800 = 20 %

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

Design

Algorithm

Step 1: Define lockprice=45.0, stockprice=30.0, barrelprice=25.0

Step 2: Input locks

Step 3: while (locks! = -1) ' input device uses -1 to indicate end of data ' goto Step 12

Step 4: input (stocks, barrels)

Step 5: Compute locksales, stocksales, barrelsales and sales

Step 6: output ("Total sales", sales)

Step 7: if (sales > 1800.0) go to Step 8 else go to Step 9

Step 8: commission = 0.10 * 1000.0; commission = commission+0.15*800.0; commission = commission+0.20*(sales-1800.0);

Step 9: if (sales>1000.0) goto Step 10 else goto Step 11

Step 10: commission = 0.10 * 1000.0; commission = commission+0.15*(sales-1000.0);

Step 11: output ("commission is $", commission);

Step 12: exit

Program Code

```
flag=0
locks=int(input("Enter the total number of locks"))
stocks=int(input("Enter the total number of stocks"))
barrels=int(input("Enter the total number of barrels"))
if locks<0 or locks>70 or stocks<0 or stocks>80 or barrels<0 or
barrels>90:
    flag=1
if flag==1:
    print("invalid input")
    exit()
totalsales=(locks*45.0)+(stocks*30.0)+(barrels*25.0)
if totalsales<1000:
    commission=0.10*totalsales
elif totalsales<1800:
    commission=0.10*1000
    commission=commission+(0.15*(totalsales-1000))
else:
    commission=0.10*1000
    commission=commission+(0.15*800)
    commission=commission+(0.20*(totalsales-1800))
print("The totalsale is %d",totalsales)
print("The commission is %f",commission)
```

Testing

Technique used: Boundary value analysis

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

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

Test cases design

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

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

The boundary value test cases can be generated over output by using following constraint and these constraints generated over commission:

C1: sales up to (and including) $1000 = 10% commission

C2: sales up to (and includes) $1800 = 15% commission

C3: sales in excess of $1800 = 20% commission

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

Equivalence class for 10% commission:

E1: sales less than 1000.

E2: sales equals to 1000.

Equivalence class for 15% commission:

E3: sales greater than 1000 and less than 1800.

E4: sales equals to 1800.

Equivalence class for 20% commission:

E5: sales greater than to 1800.

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

TC ID	Test cases description	Input data			Sales	Expected output (commission)	Actual output	Status
		Locks	Stocks	Barrels				
1	Input test cases for locks=1, stocks=1, barrels=1	1	1	1	100	10		
2	Input test cases for locks=1, stocks=1, barrels=2	1	1	2	125	12.5		
3	Input test cases for locks=1, stocks=2, barrels=1	1	2	1	130	13		
4	Input test cases for locks=2, stocks=1, barrels=1	2	1	1	145	14.5		
5	Input test cases for locks=5, stocks=5, barrels=5	5	5	5	500	50		
6	Input test cases for locks=9, stocks=10, barrels=10	9	10	10	955	95.5		
7	Input test cases for locks=10, stocks=9, barrels=10	10	9	10	970	97		
8	Input test cases for locks=10, stocks=10, barrels=9	10	10	9	975	97.5		
9	Input test cases for locks=10, stocks=10, barrels=10	10	10	10	1000	100		
10	Input test cases for locks=10, stocks=10, barrels=11	10	10	11	1025	103.75		
11	Input test cases for locks=10, stocks=11, barrels=10	10	11	10	1030	104.5		
12	Input test cases for locks=11, stocks=10, barrels=10	11	10	10	1045	106.75		
13	Input test cases for locks=14, stocks=14, barrels=14	14	14	14	1400	160		
14	Input test cases for locks=17, stocks=18, barrels=18	17	18	18	1755	213.25		
15	Input test cases	18	17	18	1770	215.5		

	for locks=18, stocks=17, barrels=18							
16	Input test cases for locks=18, stocks=18, barrels=17	18	18	17	1775	216.25		
17	Input test cases for locks=18, stocks=18, barrels=18	18	18	18	1800	220		
18	Input test cases for locks=18, stocks=18, barrels=19	18	18	19	1825	225		
19	Input test cases for locks=18, stocks=19, barrels=18	18	19	18	1830	226		
20	Input test cases for locks=19, stocks=18, barrels=18	19	18	18	1845	229		
21	Input test cases for locks=48, stocks=48, barrels=48	48	48	48	4800	820		
22	Input test cases for locks=69, stocks=80, barrels=90	69	80	90	7755	1411		
23	Input test cases for locks=70, stocks=79, barrels=90	70	79	90	7770	1414		
24	Input test cases for locks=70, stocks=80, barrels=89	70	80	89	7775	1415		
25	Input test cases for locks=70, stocks=80, barrels=90	70	80	90	7800	1420		

Execution and Result Discussion

Execute the program and test the test cases in Table 1 against the designed test cases and complete the table for actual output columns and status column.

Test Report:

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed :

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

Requirement

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

The constraints are :

C1 : $1 \leq month \leq 12$

C2 : $1 \leq day \leq 31$

C3 : $1812 \leq year \leq 2012$

If any one condition out of C1, C2 and C3 fails, then this function produces an output "Value of month not in range".

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

Design :

Algorithm

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

Step 2 : If MM is 01,03,05,07,08,10 do Step 3 else Step 6

Step 3 : If DD < 31 then do Step 4 else if DD=31 do Step 5 else output (invalid date).

Step 4 : tomorrowday = DD+1 goto Step 18

Step 5 : tomorrowday=1; tomorrowmonth=month+1 goto Step 18

Step 6 : if MM is 04 ,06,09,11 do Step 7

Step 7 : if DD<30 then do to Step 4 else if DD=30 do Step =5 else output (invalid date)

Step 8 : if MM is 12.

Step 9 : if DD < 31 then Step 4 else Step 10.

Step 10 : tomorrowday=1, tomorrowmonth=1, tomorrowyear=year+1; goto Step 18.

Step 11 : if MM is 2

Step 12 : if DD<28 do Step 4 else do Step 13.

Step 13 : if DD=28 & YYYY is a leap year do Step 14 else Step 15.

Step 14 : tomorrowday=29 goto Step 18.

Step 15 : tomorrowday=1; tomorrowmonth=3, goto Step 18.

Step 16 : if DD=29 then do Step 15 else Step 17.

Step 17 : ouput("cannot have feb", DD) Step 19

Step 18 : output(tomorrowday,tomorrowmonth,tomorrowyear);

Step 19 : exit.

Program Code :

```
months=[31,28,31,30,31,30,31,31,30,31,30,31]
day=int(input("enter the day"))
month=int(input("enter the month"))
year=int(input("enter the year"))
num_days=months[month-1]
if day<1 or day>num_days:
    print("invalid day")
    exit()
if month<1 or month>12:
    print("invalid month")
    exit()
if year<1812 or year>2012:
    print("invalid year")
    exit()
if month==2:
    if year%100==0:
        if year%400==0:
            num_days=29
        elif year%4==0:
            num_days=29;
numday=day+1
nummonth=month
numyear=year
if numday>num_days:
    numday=1
    nummonth+=1
if nummonth>12:
    nummonth=1
    numyear+=1
print("Given date is %d:%d:%d",day,month,year)
print("Next date is %d:%d:%d",numday,nummonth,numyear)
```

Testing

Technique used: Boundary value Analysis

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

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

Test case Design

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

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

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

C1: 1≤MM≤12
C2: 1≤DD≤31

C3: 1812≤YYYY≤2012

The following equivalence classes can be generated for each variable:

Equivalence classes for MM:

E1 : values lessthan 1

E2 : values in the range

E3 : values greater than 12

Equivalence classes for DD:

E4 : values lessthan 1

E5 : values in the range

E6 : values greater than 31

Equivalence classes for YYYY:

E7 : values lessthan 1812

E8 : values in the range

E9 : values greater than 2012

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

TC ID	Description	Input			Expected Output	Actual Output	Status
		MM	DD	YYYY			
1	Input of month is min	01	15	1912	01/16/1912		
2	Input of month is min+	02	15	1912	02/16/1912		
3	Input of month is nom	06	15	1912	06/16/1912		
4	Input of month is max-	11	15	1912	11/16/1912		
5	Input of month is max	12	15	1912	12/16/1912		
6	Input of day is min	06	01	1912	06/02/1912		
7	Input of day is min+	06	02	1912	06/03/1912		
8	Input of day is max-	06	30	1912	07/01/1912		
9	Input of day is max	06	31	1912	invalid day		
10	Input of year is min	06	15	1812	06/16/1812		
11	Input of year is min+	06	15	1813	06/16/1813		
12	Input of year is max-	06	15	2011	06/16/2011		
13	Input of year is max-	06	15	2012	06/16/2012		

Test Report:

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed :

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

Requirements:-

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

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

R3. If the requirement R1 satisfied then the system should determine the type of triangle, can be

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

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

Design

Algorithm:-

Step 1: Input a, b and c i.e three integer values which represents three side of the triangle

Step 2: if(a<(b+c)) and (b<(a+c)) and (c<(a+b)) then do step 3 else
Print not a triangle. Do step 6. Step

3: if (a=b) and (b=c) then

Print triangle formed is equilateral. Do step 6.

Step 4: if (a != b) and (a!=c) and (b!=c) then

Print triangle formed is scalene. Do step 6.

Step 5: print triangle formed is Isosceles.

Step 6: stop.

Program Code:-

```
print("Enter the three sides of the triangle")
a=int(input("Enter the value of a"))
b=int(input("Enter the value of b"))
c=int(input("Enter the value of c"))
if(a<=10)or(b<=10)or(c<=10)or(a<1)or(b<1)or(c<1):
    print("Out of Range ")
    exit()
```

```
if(a<b+c)and(b<a+c)and(c<a+b):
    if(a==b)and(b==c):
        print("Equilateral triangle")
    elif(a!=b)and(a!=c)and(b!=c):
        print("Scalene triangle")
    else:
        print("Isosceles triangle")
else:
    print("Triangle cannot be formed")
```

Testing :

1. Technique used : Equivalence class partition
2. Test cases design.

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

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

$1 \le a \le 10$
$1 \le b \le 10$
$1 \le c \le 10$

First Attempt

Weak normal equivalence class: In the problem statement, we note that four possible outputs can occur: Not a Triangle, Scalene, Isosceles, and Equilateral. We can use these to identify output (range) equivalence class 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 Isoselecs}
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 from a triangle }

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

TC ID	Test cases description	Input data			Expected output	Actual output	Status
		a	b	c			
1	WN1	5	5	5	Equilateral triangle		
2	WN2	2	2	5	Isosceles triangle		
3	WN3	3	4	5	Scalene triangle		
4	WN4	4	1	2	Not a triangle		
5	WR1	-1	5	5	Value of a is not in the range of		

					permitted values		
6	WR2	5	-1	5	Value of b is not in the range of permitted values		
7	WR3	5	5	-1	Value of c is not in the range of permitted values		
8	WR4	11	5	5	Value of a is not in the range of permitted values		
9	WR5	5	11	5	Value of b is not in the range of permitted values		
10	WR6	5	5	11	Value of c is not in the range of permitted values		

Table 1 : Weak normal and weak robust test cases for triangle problem

Second attempt

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

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

D2={<a,b,c> : a=b, a≠c}

D3={<a,b,c> : a=c, a≠b}

D4={<a,b,c> : b=c, a≠b}

D5={<a,b,c> :a≠b,a≠c,b≠c}

D6={<a,b,c> :a>=b+c}

D7={<a,b,c> : b>=a+c}

D8={<a,b,c> : c>=a+b}

TC ID	Test cases description	Input data			Expected output	Actual output	Status
		a	b	c			
1	SR1	-1	5	5	Value of a is not in the range of permitted values		
2	SR2	5	-1	5	Value of b is not in the range of permitted values		

3	SR3	5	5	-1	Value of c is not in the range of permitted values		
4	SR4	-1	-1	5	Value of a, b is not in the range of permitted values		
5	SR5	5	-1	-1	Value of b, c is not in the range of permitted values		
6	SR6	-1	5	-1	Value of a, c is not in the range of permitted values		
7	SR7	-1	-1	-1	Value of a, b, c is not in the range of permitted values		

Table 2: Strong robust test cases for triangle problem

Execution and Result Discussion

Execute the program and test the test cases in Table 1 and Table 2 against the designed test cases and complete the table for actual output columns and status column.

Test Report:

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :

Number of TC's failed :

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

Requirements Specification

Problem definition: The commission problem includes a sales person in the former Arizona territory sold the rifle locks, stocks and barrels made by gunsmith in Missouri. Cost includes

Locks = 45 $

Stocks = 30 $

Barrels = 25 $

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

After each town visit, the sales person sent a telegram to the Missouri gunsmith with the number of locks, stocks and barrels sold in the town. At the end of the month, the sales person sent a very short telegram showing -1 lock sold. The gunsmith then knew the sales for the month were complete and computed the sales person's commission as follows.

On sales up to (and including) $ 1000 = 10 %

On sales up to (and includes) $ 1800 = 15 %

On the sales in excess of $ 1800 = 20 %

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

Design

Algorithm

Step 1: Define lockprice=45.0, stockprice=30.0, barrelprice=25.0

Step 2: Input locks

Step 3: while (locks!= -1) ' input device uses -1 to indicate end of data ' goto Step 12

Step 4: input (stocks, barrels)

Step 5: Compute locksales, stocksales, barrelsales and sales

Step 6: output ("Total sales", sales)

Step 7: if (sales > 1800.0) goto Step 8 else goto Step 9

Step 8: commission = 0.10 * 1000.0; commission = commission+0.15*800.0; commission = commission+0.20*(sales-1800.0);

Step 9: if (sales>1000.0) goto Step 10 else goto Step 11

Step 10: commission = 0.10 * 1000.0; commission = commission+0.15*(sales-1000.0);

Step 11: output ("commission is $", commission);

Step 12: exit

Program Code

```
flag=0
locks=int(input("Enter the total number of locks"))
stocks=int(input("Enter the total number of stocks"))
barrels=int(input("Enter the total number of barrels"))
if locks<0 or locks>70 or stocks<0 or stocks>80 or barrels<0 or
barrels>90:
    flag=1
if flag==1:
    print("invalid input")
    exit()
totalsales=(locks*45.0)+(stocks*30.0)+(barrels*25.0)
if totalsales<1000:
    commission=0.10*totalsales
elif totalsales<1800:
    commission=0.10*1000
    commission=commission+(0.15*(totalsales-1000))
else:
    commission=0.10*1000
    commission=commission+(0.15*800)
    commission=commission+(0.20*(totalsales-1800))
print("The totalsale is %d",totalsales)
print("The commission is %f",commission)
```

Testing

Technique used: Equivalence Class Testing

Test selection using equivalence partitioning allows a tester to subdivide the input domain into relatively small number of sub domains.

Test case design

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

The valid classes of the input variables are:

L1 = {locks: 1≤locks≤70}

L2 = {locks = -1}{occurs if locks = -1 is used to control input iteration}

S1 = {stocks: 1≤stocks≤80}

B1 = {barrels: 1≤barrels≤90}

The corresponding invalid classes of the input variables are:

L3 = {locks: locks=0 or locks < -1}

L4 = {locks: locks > 70}

S2 = {stocks: stocks <1}

S3 = {stocks: stocks > 80}

B2 = {barrels: barrels < 1}

B3 = {barrels: barrels > 90}

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

First attempt

Weak robust test cases

TC ID	Test cases description	Input data			Sales	Expected output (commission)	Actual output	Status
		Locks	Stocks	Barrels				
1	WR1	10	10	10	100	10		
2	WR2	-1	40	45	Program terminates			
3	WR3	-2	40	45	Value of locks not in the range 1….70			
4	WR4	71	40	45	Value of locks not in the range 1….70			
5	WR5	35	-1	45	Value of stocks not in the range 1….80			
6	WR6	35	81	45	Value of stocks not in the range 1….80			
7	WR7	35	40	-1	Value of barrels not in the range 1….90			
8	WR8	35	40	91	Value of barrels not in the range 1….90			

Second Attempt

Strong robust test cases

TC ID	Test cases description	Input data			Sales	Expected output (commission)	Actual output	Status
		Locks	Stocks	Barrels				
1	SR1	-2	40	45	Value of locks not in the range 1….70			
2	SR2	35	-1	45	Value of stocks not in the range 1….80			
3	SR3	35	40	-1	Value of barrels not in the range 1….90			
4	SR4	-2	-1	45	Value of locks not in the range 1….70 Value of stocks not in the range 1….80			
5	SR5	-2	40	-1	Value of locks not in the range 1….70 Value of barrels not in the range 1….90			
6	SR6	35	-1	-1	Value of stocks not in the range 1….80 Value of barrels not in the range 1….90			
7	SR7	-2	-1	-1	Value of locks not in the range 1….70 Value of stocks not in the range 1….80 Value of barrels not in the range 1….90			

Test Report:

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed:

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

Requirement

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

The constraints are:

C1: $1 \leq month \leq 12$

C2: $1 \leq day \leq 31$

C3: $1812 \leq year \leq 2012$

If any one condition out of C1, C2 and C3 fails, then this function produces an output "Value of month not in range".

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

Design:

Algorithm

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

Step 2: If MM is 01, 03, 05, 07, 08, 10 do Step 3 else Step 6

Step 3: If DD < 31 then do Step 4 else if DD=31 do Step 5 else output (invalid date).

Step 4: tomorrowday = DD+1 goto Step 18

Step 5: tomorrowday=1; tomorrowmonth=month+1 goto Step 18

Step 6: if MM is 04, 06, 09, 11 do Step 7

Step 7: if DD<30 then do to Step 4 else if DD=30 do Step =5 else output (invalid date)

Step 8: if MM is 12.

Step 9: if DD < 31 then Step 4 else Step 10.

Step 10: tomorrowday=1, tomorrowmonth=1, tomorrowyear=year+1; goto Step 18.

Step 11: if MM is 2

Step 12: if DD<28 do Step 4 else do Step 13.

Step 13: if DD=28 & YYYY is a leap year do Step 14 else Step 15.

Step 14: tomorrowday=29 goto Step 18.

Step 15: tomorrowday=1; tomorrowmonth=3, goto Step 18.

Step 16: if DD=29 then do Step 15 else Step 17.

Step 17: ouput("cannot have feb", DD) Step 19

Step 18: output(tomorrowday,tomorrowmonth,tomorrowyear);

Step 19: exit.

Program Code:

```
months=[31,28,31,30,31,30,31,31,30,31,30,31]
day=int(input("enter the day"))
month=int(input("enter the month"))
year=int(input("enter the year"))
num_days=months[month-1]
if day<1 or day>num_days:
    print("invalid day")
    exit()
if month<1 and month>12:
    print("invalid month")
    exit()
if year<1812 and year>2012:
    print("invalid year")
    exit()
if month==2:
    if year%100==0:
        if year%400==0:
            num_days=29
        elif year%4==0:
            num_days=29;
numday=day+1
nummonth=month
numyear=year
if numday>num_days:
    numday=1
    nummonth+=1
if nummonth>12:
    nummonth=1
    numyear+=1
print("Given date is %d:%d:%d",day,month,year)
print("Next date is %d:%d:%d",numday,nummonth,numyear)
```

Testing:

Test cases design

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

First attempt

A first attempt at creating an equivalence relation might produce intervals such as ;'

Valid intervals

M1={month : 1≤month≤12}

D1={day : 1≤day≤31}

Y1={year : 1812≤year≤2012}

Invalid intervals

M2 = {month : month < 1}

M3 = {month : month > 12}

D2 = {day : day < 1}

D3 ={day : day > 31}

Y2 = {year : year < 1812}

Y3 = {year : year > 2012}

At a first glance it seems that everything has been taken into account and our day, month, and year intervals have been defined well. Using these intervals we produce a test cases using the four different types of equivalence class testing.

Weak and Strong normal

TC ID	Description	Input			Expected output	Actual output	Status
		MM	DD	YYYY			
1	WN1,SN1	06	15	1990	06/16/1990		

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

Weak robust

TC	Description	Inputs	Expected Output	Actual	Status

ID		MM	DD	YYYY		Output	
1	WR1	6	15	1912	6/16/1912		
2	WR2	-1	15	1912	value of month not in the range 1…12		
3	WR3	13	15	1912	value of month not in the range 1…12		
4	WR4	6	-1	1912	value of day not in the range 1…31		
5	WR5	6	32	1912	value of day not in the range 1…31		
6	WR6	6	15	1811	value of year not in the range 1812…2012		
7	WR7	6	15	2013	value of year not in the range 1812…2012		

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

Strong Robust

TC ID	Description	Inputs			Expected Output	Actual Output	Status
		MM	DD	YYYY			
1	SR1	-1	15	1912	value of month not in the range 1…12		
2	SR2	6	-1	1912	value of day not in the range 1…31		
3	SR3	6	15	1811	value of year not in the range 1812..2012		
4	SR4	-1	-1	1912	value of month not in the range 1…12 value of day not in the range 1…31		
5	SR5	6	-1	1811	value of day not in the range 1…31 value of year not in the range 1812…2012		
6	SR6	-1	15	1811	value of month not in the range 1…12 value of year not in the range		

					1812…2012		
7	SR7	-1	-1	1811	value of month not in the range 1…12 value of day not in the range 1…31 value of year not in the range 1812…2012		

Second Attempt

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

M1 = {month : month has 30 days}

M2 = {month : month has 31 days}

M3 = {month : month is February}

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

D1 = {day : 1≤day≤28}

D2 = {day : day=29}

D3 = {day : day=30}

D4 = {day : day=31}

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

Y1 = {year : year = 2000}

Y2 = {year : year is a leap year}

Y3 = {year ; year is a common year}

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

Weak normal

TC ID	Description	Input			Expected output	Actual output	Status
		MM	DD	YYYY			
1	WN1	6	14	2000	6/15/2000		
2	WN2	7	29	1996	7/30/1996		
3	WN3	2	30	2002	invalid input date		
4	WN4	6	31	2000	Invalid input date		

Strong normal

TC ID	Description	Inputs			Expected Output	Actual Output	Status
		MM	DD	YYYY			
1	SN1	6	14	2000	6/15/2000		
2	SN2	6	14	1996	6/15/1996		
3	SN3	6	14	2002	6/15/2002		
4	SN4	6	29	2000	6/30/2000		
5	SN5	6	29	1996	6/30/1996		
6	SN6	6	29	2002	6/30/2002		
7	SN7	6	30	2000	Invalid Input Date		
8	SN8	6	30	1996	Invalid Input Date		
9	SN9	6	30	2002	Invalid Input Date		
10	SN10	6	31	2000	Invalid Input Date		
11	SN11	6	31	1996	Invalid Input Date		
12	SN12	6	31	2002	Invalid Input Date		
13	SN13	7	14	2000	7/15/2000		
14	SN14	7	14	1996	7/15/1996		
15	SN15	7	14	2002	7/15/2002		
16	SN16	7	29	2000	7/30/2000		
17	SN17	7	29	1996	7/30/1996		
18	SN18	7	29	2002	7/30/2002		
19	SN19	7	30	2000	7/31/2000		
20	SN20	7	30	1996	7/31/1996		
21	SN21	7	30	2002	7/31/2002		
22	SN22	7	31	2000	8/1/2000		
23	SN23	7	31	1996	8/1/1996		
24	SN24	7	31	2002	8/1/2002		
25	SN25	2	14	2000	2/15/2000		
26	SN26	2	14	1996	2/15/1996		
27	SN27	2	14	2002	2/15/2002		

28	SN28	2	29	2000	3/1/2000		
29	SN29	2	29	1996	3/1/1996		
30	SN30	2	29	2002	Invalid Input Date		
31	SN31	2	30	2000	Invalid Input Date		
32	SN32	2	30	1996	Invalid Input Date		
33	SN33	2	30	2002	Invalid Input Date		
34	SN34	2	31	2000	Invalid Input Date		
35	SN35	2	31	1996	Invalid Input Date		
36	SN36	2	31	2002	Invalid Input Date		

Test Report:

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed :

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

Requirements:-

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

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

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

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

Design:-

Form the given requirements we can draw the following conditions :

C1: a < b+c ?
C2: b < a+c ?
C3: c < a+b ?
C4: a=b ?
C5: a=c ?
C6: b=c ?

According to the property of the triangle, if any one of the three conditions C1,C2,C3 are not satisfied then triangle cannot be constructed. So only when C1,C2 and C3 are true the triangle can be formed, then depending on conditions C4,C5 and C6 we can decide which type of triangle will formed. (i.e requirement 2)

Algorithm:-

Step 1: Input a, b and c i.e three integer values which represents three side of the triangle

Step 2: if(a<(b+c)) and (b<(a+c)) and (c<(a+b)) then do step 3 else
Print not a triangle. Do step 6.

Step 3: if (a=b) and (b=c) then
Print triangle formed is equilateral. Do step 6.
Step 4: if (a != b) and (a!=c) and (b!=c) then
Print triangle formed is scalene. Do step 6.

Step 5: print triangle formed is Isosceles.

Step 6: stop.

Program Code:-

```
print("Enter the three sides of the triangle")
a=int(input("Enter the value of a"))
b=int(input("Enter the value of b"))
c=int(input("Enter the value of c"))
if(a<b+c)and(b<a+c)and(c<a+b):
    if(a==b)and(b==c):
        print("Equilateral triangle")
    elif(a!=b)and(a!=c)and(b!=c):
        print("Scalene triangle")
    else:
        print("Isosceles triangle")
else:
    print("Triangle cannot beformed")
```

Testing:

Technique Used: Decision Table Approach

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

	Conditions	**Conditions Entry (Rules)**										
		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
C1	: a<b+c ?	F	T	T	T	T	T	T	T	T	T	T
C2	: b<a+c ?	--	F	T	T	T	T	T	T	T	T	T
C3	: c<a+b ?	--	--	F	T	T	T	T	T	T	T	T
C4	: a==b ?	--	--	--	F	T	T	T	F	F	T	F
C5	: a ==c ?	--	--	--	T	F	T	F	T	F	T	F
C6	: b==c ?	--	--	--	T	T	F	F	F	T	T	F
Actions		**Action Entries**										
A1	: not a triangle	X	X	X								
A2	: Scalene											X
A3	: Isosceles							X	X	X		
A4	: equilateral										X	
A5	: impossible				X	X	X					

The '--' symbol in the table indicates don't care values. The table shows the six conditions and five actions. All the conditions in the decision table are binary value. Hence it is called as Limited entry decision table.

Deriving test cases using Decision table approach:

Test cases:

TC ID	Test cases description	a	b	C	Expected output	Actual output	Status
1	Testing for Req 1	4	1	2	Not a triangle		
2	Testing for Req 1	1	4	2	Not a triangle		
3	Testing for Req 1	1	2	4	Not a triangle		
4	Testing for Req 2	5	5	5	Equilateral triangle		
5	Testing for Req 2	2	2	3	Isosceles triangle		
6	Testing for Req 2	2	3	2	Isosceles triangle		
7	Testing for Req 2	3	2	2	Isosceles triangle		
8	Testing for Req 2	3	4	5	Scalene		

Execution and Result Discussion

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

Test Report:

1. Number of TC Executed :08
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed :

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

Requirements

R1: The system should read the number of locks, stocks and barrels sold in a month.

(ie 1≤locks≤70)

(ie 1≤stocks≤80)

(ie 1≤barrels≤90)

R2: If R1 satisfied the system should compute the sale's person commission depending on the total number of locks, stocks and barrels sold else it should display suitable error message. Following is the percentage of commission for the sales done:

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

15% on next $800

20% on any sales in excess of $1800

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

Design

For given requirements we can draw the following conditions.

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

C2: 1≤stocks≤80?

C3: 1≤barrels≤90?

C4: sales > 1800?

C5: sales > 1000?

C6: sales ≥ 1000?

Algorithm

Step 1: Define lockprice=45.0, stockprice=30.0, barrelprice=25.0

Step 2: Input locks

Step 3: while (locks! = -1) ' input device uses -1 to indicate end of data ' goto Step 12

Step 4: input (stocks, barrels)

Step 5: Compute locksales, stocksales, barrelsales and sales

Step 6: output ("Total sales", sales)

Step 7: if (sales > 1800.0) go to Step 8 else go to Step 9

Step 8: commission = 0.10 * 1000.0; commission = commission+0.15*800.0; commission = commission+0.20*(sales-1800.0);

Step 9: if (sales>1000.0) goto Step 10 else goto Step 11

Step 10: commission = 0.10 * 1000.0; commission = commission+0.15*(sales-1000.0);

Step 11: output ("commission is $", commission);

Step 12: exit

Program Code

```
flag=0
locks=int(input("Enter the total number of locks"))
stocks=int(input("Enter the total number of stocks"))
barrels=int(input("Enter the total number of barrels"))
if locks<0 or locks>70 or stocks<0 or stocks>80 or barrels<0 or
barrels>90:
    flag=1
if flag==1:
    print("invalid input")
    exit()
totalsales=(locks*45.0)+(stocks*30.0)+(barrels*25.0)
if totalsales<1000:
    commission=0.10*totalsales
elif totalsales<1800:
    commission=0.10*1000
    commission=commission+(0.15*(totalsales-1000))
else:
    commission=0.10*1000
    commission=commission+(0.15*800)
    commission=commission+(0.20*(totalsales-1800))
print("The totalsale is %d",totalsales)
print("The commission is %f",commission)
```

Testing

Technique used: Decision table approach

Conditions	Condition Entries (Rule)					
C1 : 1≤locks≤70	F	T	T	T	T	T
C2 : 1≤stocks≤80 ?	--	F	T	T	T	T
C3 : 1≤barrels≤90 ?	--	--	F	T	T	T
C4 : Sales >1800 ?	--	--	--	T	F	F
C5 : Sales >1000?	--	--	--	--	T	F
C6 : Sales ≤ 1000?	--	--	--	--	--	T
Action	**Action entries**					
A1: com1=0.10*sales						X
A2 : com2=com1+0.15*(sales-1000)					X	
A3 : com3=com2+0.20*(sales-1800)				X		
A4 : out of range	X	X	X			

Deriving test cases using decision table approach:

Test cases

TC ID	Test cases description	Input data			Sales	Expected output (commission)	Actual output	Status
		Locks	Stocks	Barrels				
1	Testing for Req1 Condition C1	-2	40	45	Out of range			
2	Testing for Req1 Condition C1	90	40	45	Out of range			
3	Testing for Req1 Condition C2	35	-2	45	Out of range			
4	Testing for Req1 Condition C2	35	90	45	Out of range			
5	Testing for Req1 Condition C3	35	40	-1	Out of range			
6	Testing for Req1 Condition C3	35	45	100	Out of range			
7	Testing for Req2	5	5	5	500	50		

8	Testing for Req2	15	15	15	1500	175		
9	Testing for Req2	25	25	25	2500	360		

Test Report:

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed :

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

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

Requirements Specification

Problem definition: The commission problem includes a sales person in the former Arizona territory sold the rifle locks, stocks and barrels made by gunsmith in Missouri. Cost includes

Locks = 45 $

Stocks = 30 $

Barrels = 25 $

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

After each town visit, the sales person sent a telegram to the Missouri gunsmith with the number of locks, stocks and barrels sold in the town. At the end of the month, the sales person sent a very short telegram showing -1 lock sold. The gunsmith then knew the sales for the month were complete and computed the sales person's commission as follows.

On sales up to (and including) $ 1000 = 10 %

On sales up to (and includes) $ 1800 = 15 %

On the sales in excess of $ 1800 = 20 %

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

Design

Algorithm

Step 1: Define lockprice=45.0, stockprice=30.0, barrelprice=25.0

Step 2: Input locks

Step 3: while (locks!= -1) ' input device uses -1 to indicate end of data ' goto Step 12

Step 4: input (stocks, barrels)

Step 5: Compute locksales, stocksales, barrelsales and sales

Step 6: output ("Total sales", sales)

Step 7: if (sales > 1800.0) goto Step 8 else goto Step 9

Step 8: commission = 0.10 * 1000.0; commission = commission+0.15*800.0; commission = commission+0.20*(sales-1800.0);

Step 9: if (sales>1000.0) goto Step 10 else goto Step 11

Step 10: commission = 0.10 * 1000.0; commission = commission+0.15*(sales-1000.0);

Step 11: output ("commission is $", commission);

Step 12: exit

Program Code

```
1.flag=0
2.locks=int(input("Enter the total number of locks"))
3.stocks=int(input("Enter the total number of stocks"))
4.barrels=int(input("Enter the total number of barrels"))
5.if locks<0 or locks>70 or stocks<0 or stocks>80 or barrels<0 or
barrels>90:
6.    flag=1
7.if flag==1:
8.    print("invalid input")
9.    exit()
10.totalsales=(locks*45.0)+(stocks*30.0)+(barrels*25.0)
11.if totalsales<1000:
12.    commission=0.10*totalsales
13.elif totalsales<1800:
14.    commission=0.10*1000
15.    commission=commission+(0.15*(totalsales-1000))
16.else:
17.    commission=0.10*1000
18.    commission=commission+(0.15*800)
19.    commission=commission+(0.20*(totalsales-1800))
20.print("The totalsale is %d",totalsales)
21.print("The commission is %f",commission)
```

Testing Technique: Dataflow Testing

A structural testing technique

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

Variable definition

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

Variable Uses

Occurrences of a variable where a variable is given not a new value (Variable declaration is not its use).

P-uses (predicate uses)

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

C-uses (computation uses)

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

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

Test case definition based on four groups of coverage's

- All definitions
- All c-uses
- All p-uses
- All du-paths

Data flow testing: Key Steps

Given a code (program or pseudo code)

1. Number the lines.

2. List the variables.

3. List occurrences and assign a category to each variable.

4. Identify du-pairs and their use (p-uses and c-uses)

5. Define test cases, depending on the required coverage.

Step 3

Line No	Category		
	Definition	**c-use**	**p-use**
2,3,4	Locks, stocks, barrels		
5			Locks, stocks, barrels
6	Flag		
7			flag
10	totalsales	Locks, stocks, barrels	
11			totalsales
12	Commission	totalsales	
13			totalsales
14	Commission		
15	Commission	Commission, totalsales	
17	Commission		
18	Commission	Commission	
19	Commission	Commission, totalsales	
20,21		Commission, totalsales	

Table 1 : list occurrences and assign a category to each variable

Step 4

Definition-use pair	variables	
Start-line →end-line	**c-use**	**p-use**
2,3,4→5		Locks, stocks, barrels
6→7		Flag
2,3,4→10	Locks, stocks, barrels	
10→11		totalsales
10→13		totalsales
12→21	Commission	
14→15	Commission	
15→21	Commission	
17→18	Commission	
18→19	Commission	
19→21	Commission	

Table 2: Identify du-pairs and their use (p-uses and c-uses

Step 5

Variables	Du-pair	Sub-path	Inputs			Expected output	
			Locks	**Stocks**	**Barrels**	**t_sales**	**commission**
Locks, Stocks, Barrels	2,3,4→10	8,9,18	10	10	10	1000	100
Locks, Stocks, Barrels	2,3,4→ 5	8,9	5	-1	22	Invalid input	
Flag	6→7	11,12,13	-1	5	99	Invalid input	
totalsales	10→11	18,19	5	5	5	500	50
totalsales	10→13	18,19,23	15	15	15	1500	175
commission	12→21	21,34	5	5	5	500	50
commission	15→21	26,34	15	15	15	1500	175
commission	19→21	32,34	25	25	25	2500	260

Test Report:

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed :

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

Requirements

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

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

Design

Algorithm

Step 1: Input value of n. Enter n integer numbers in array int mid;

Step 2: Initialize low=0, high=n-1;

Step 3: until(low<=high) do

mid=(low+high)/2;

if (a[mid]==key) then do Step 5

else if(a[mid]>key) then do

high=mid-1

else

low=mid+1

Step 4: Print unsuccessful search do Step 6

Step 5: Print successful search, Element found at position mid+1

Step 6: stop

Program Code

```
n = int(input("Enter the size of the list: "))
flag=0
sortedlist = []
for i in range(n):
    sortedlist.append(input("Enter %dth element: "%i))
x = input("Enter the number to search: ")
low=0
high=n-1
while(low <= high):
        mid = int((low + high)/2)
        if (x == sortedlist[mid]):
            flag=1
            break
        elif(x < sortedlist[mid]):
```

```
            high = mid - 1
        else:
            low = mid + 1
if (flag==1):
    print("Successfull search entered number %s is present at position :%d"%(x,mid))
else:
    print("unsuccessful search entered number %s is not present at position"%x)
```

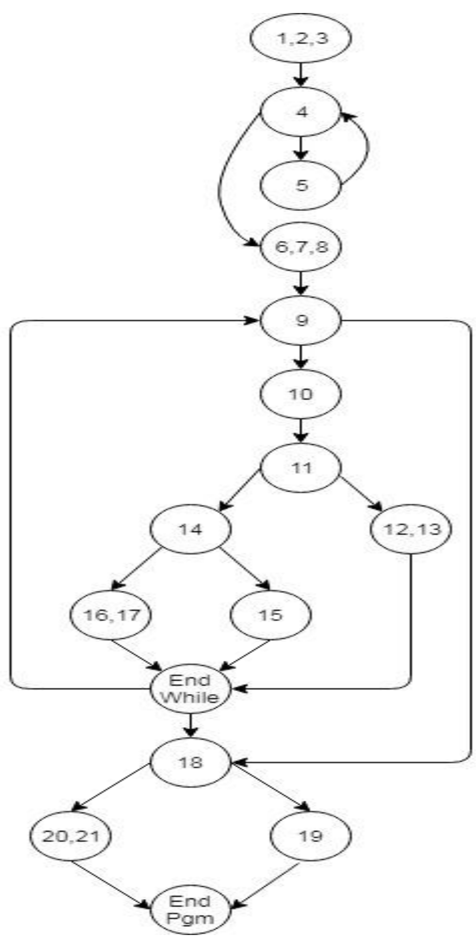
Testing

Technique used: Basis Path Testing

Basis path testing is a form of structural testing (white box testing). The method devised by McCabe to carry out basis path testing has four steps: these are

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

Step 1: Program graph for binary search.



Using the program graph we derive (Decision to Decision) DD path graph for binary search program.

DD path name	Program graph nodes
A	1,2,3
B	4
C	5
D	6,7,8
E	9
F	10
G	11
H	12,13
I	14
J	15
K	16,17
L	18
M	19
N	20,21