Experiment No: 1 Date: 12-09-2025

Decision Table Approach for Solving Triangle Problem

Problem Statement:

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

Theory

The Triangle problem can be solved using decision table testing, which is a black-box test design technique. It is useful when different combinations of conditions lead to different actions.

Triangle Property Rule

* Each side must be less than the sum of the other two sides

$$a < b + c$$

$$b < a + c$$

$$c < a + b$$

* If this condition is not satisfied, it is not a triangle.

Types of Triangles

i) Equilateral: All three sides are equal (a == b == c)

ii) Isosceles: Exactly two sides are equal (a == b || b == c || a == c)

iii) Scalene: All three sides are different (a != b && b != c && a != c)

Input data decision Table

Rules		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
Conditions	C1: a < b+c	F	T	T	T	T	T	T	T	T	T	
	C2: b < a+c	-	F	T	T	T	T	T	T	T	T	T
	C3: c < a+b	-	-	F	T	T	T	T	T	T	T	T
	C4: a = b	-	-	-	T	T	T	T	F	F	F	F
	C5: a = c	-	-	-	T	T	F	F	T	T	F	F
	C6: b = c	-	-	-	T	F	T	F	T	F	T	F
Actions	a1: Not a triangle	X	X	X								
	a2: Scalene triangle											X
	a3: Isosceles triangle							X		X	X	
	a4: Equilateral triangle				X							
	a5: Impossible					X	X		X			

Triangle Problem - Decision Table Test cases for input data

Case Id	Description	Input Data			Expected Output	Actual Output	Status	Comments
		A	B	C				
1	Enter the value of a, b and c such that a is not less than sum of two sides	20	5	5	Message should be displayed can't form a triangle	Can't form a triangle	Pass	R1 (a < b+c condition fails)
2	Enter the value of a, b and c such that b is not less than sum of two sides and a is less than sum of other two sides	3	15	11	Message should be displayed can't form a triangle	Can't form a triangle	Pass	R2
3	Enter the value of a, b and c such that c is not less than sum of two sides and a and b is less than sum of other two sides	4	5	20	Message should be displayed can't form a triangle	Can't form a triangle	Pass	R3
4	Enter the value a, b and c satisfying precondition and a=b, b=c and c = a	5	5	5	Message should be displayed ~~displayed~~ Equilateral triangle	Equilateral triangle	Pass	R4 (a = b = c)
5	Enter the value a, b and c satisfying precondition and a = b and b ≠ c	10	10	9	Message should be displayed Isosceles triangle	Isosceles triangle	Pass	R7 (a = b)
6	Enter the value a, b and c satisfying precondition and a ≠ b, b ≠ c and c ≠ a	5	6	7	Message should be displayed Scalene triangle	Scalene triangle	Pass	R11 (a != b != c)

Source Code

```
#include <stdio.h>
int main(){
  int a,b,c;
  char istriangle;

  printf("Enter A B C :");
  scanf("%d %d %d",&a,&b,&c);

  if(a<b+c && b<a+c && c<a+b) istriangle = 'y';
  else istriangle = 'n';

  if(istriangle == 'y'){
     if((a==b) && (b==c)) printf("Equilateral Triangle\n");
     else if((a!=b)&&(a!=c)&&(b!=c)) printf("Scalen Triangle\n");
     else printf("Isosceles Triangle\n");
  } else printf("Not a Triangle\n");

  return 0;

}
```

Conclusion:

Using the Decision Table Approach, we tested all possible cases of the triangle problem. The program successfully identifies whether the given sides form a valid triangle and classifies it into equilateral, isosceles or scalene.

References:

* Roger S. Pressman, Software Engineering: A Practitioner's Approach.
* Ian Sommerville, SOFTWARE ENGINEERING, Addison-Wesley, 9th Edition.

Experiment No: 2 Date: 26-09-2025

Boundary Value Analysis for solving Triangle Problem

Problem Statement:

Design and develop program to solve triangle problem defined as follows: Accept three integers as three sides of a triangle and determine if the three values represent an equilateral, isosceles or 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 cases and discuss the results.

Theory:

Boundary Value Analysis is a software testing technique where test cases are designed based on boundary value of input domains rather than arbitrary values. Errors often occur at input boundaries rather than within the middle of input ranges.

Triangle Properly Rule

* Each side must be less than the sum of the other two sides

$$a < b + c$$
$$b < a + c$$
$$c < a + b$$

* If this condition is not satisfied, it is not a triangle.

Type of Triangle

i) Equilateral: All three are equal (a == b == c)

ii) Isosceles: Exactly two sides are equal (a==b || b==c || a==c)

iii) Scalene: All three sides are different (a!=b && b!=c && a!=c)

Triangle Problem - Boundary value Test cases for input data

Case ID	Description	Input Data			Expected Output	Actual Output	Status	Comments
		a	b	c				
1	Enter the min value for a, b and c	1	1	1	Should display Equilateral triangle	Equilateral triangle	PASS	a=b=c
2	Enter min value for 2 items and min+1 for any one	1	1	2	Should display can't form a triangle	can't form triangle	PASS	c<a+b
3	Enter min value for 2 items and min+1 for any one	1	2	1	Should display can't form a triangle	can't form triangle	PASS	b<a+c
4	Enter min value for 2 items and min+1 for any one	2	1	1	Should display can't form a triangle	can't form triangle	PASS	a<b+c
5	Enter normal value for 2 items and 1 item is min value	5	5	1	Should display Isosceles triangle	Isoceles triangle	PASS	a=b
6	Enter normal value for 2 items and 1 item is min value	5	1	5	Should display Isosceles triangle	Isosceles triangle	PASS	a=c
7	Enter normal value for 2 items and 1 item is min value	1	5	5	Should display Isosceles triangle	Isosceles triangle	PASS	b=c
8	Enter normal value for a, b and c	5	5	5	Should display Equilateral triangle	Equilateral triangle	PASS	a=b=c
9	Enter normal value for 2 items and 1 item is max value	5	5	10	Should display Not a triangle	Not a triangle	PASS	c<a+b
10	Enter normal value for 2 items and 1 item is max value	5	10	5	Should display Not a triangle	Not a triangle	PASS	b<a+c
11	Enter normal value for 2 items and 1 item is max value	10	5	5	Should display Not a triangle	Not a triangle	PASS	a<b+c
12	Enter max value for 2 items and max-1 for any one	10	10	9	Should display Isosceles triangle	Isosceles triangle	PASS	a=b
13	Enter max value for 2 items and max-1 for any one	10	9	10	Should display Isosceles triangle	Isosceles triangle	PASS	a=c
14	Enter max value for 2 items and max-1 for any one	9	10	10	Should display Isosceles triangle	Isosceles triangle	PASS	b=c
15	Enter max value for a, b and c	10	10	10	Should display Equilateral triangle	Equilateral triangle	PASS	a=b=c

Source Code

```
# Include <stdio.h>

int main() {
    int a,b,c, valid = 1;
    printf("Enter triangle sides");
    scanf("%d %d %d", &a, &b, &c);

    if (a < 1 || a > 10) valid = 0;
    if (b < 1 || b > 10) valid = 0;
    if (c < 1 || c > 10) valid = 0;
    if (!valid) return 0;

    if (a+b > c && a+c > b && b+c > a) {
        if (a == b && b == c) printf("Equilateral Triangle");
        else if (a == b || b == c || a == c) printf("Isosceles Triangle");
        else printf("Scaline Triangle");
    } else {
        printf("Not a Triangle");
    }

    return 0;
}
```

Conclusion:

Using Boundary Value Analysis, we tested the program with values at and around the input boundaries (1 and 10). The program correctly validates inputs, rejects out-of-range values and applies the triangle inequality theorm to classify triangles.

References:

* Roger S Pressman, Software Engineering: A Practitioner's Approach.
* Ian Sommerville, SOFTWARE ENGINEERING, Addison-Wesly, 9th Edition.

Experiment No: 3 Date :

Data Flow Testing for Commission Problem

Problem Statement

Design, develop and analyze a program for commission calculation using Data Flow Testing, derive suitable test cases, execute them and discuss the results.

Theory

Data Flow Testing (DFT) is a white-box testing technique that focuses on the flow of data within a program. It examines how data are defined, used and destroyed during the program's execution.

The main goal is to identify:

* Variables that are used before being initialized
* Variables that are defined but never used
* Incorrect updates or redundant operations on variables

It uses the concept of:

* DEF (Definition): point where a variable is assigned a value.
* USE (use): point where the variable's value is accessed or utilized.

Selected Define/Use Paths for Commission problem

Test Case Id	Description	Variables Path (Beginning, End)	Du Path	Definition clear	comments
1	check for lock price variable (lprice,7) and USE (lprice,24)	(7,24)	<7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24>	Yes	
2	check for stock price variable DEF(sprice,8) and USE (sprice,25)	(8,25)	<8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25>	Yes	
3	check for barrel price variable DEF(bprice,9) and USE (bprice,26)	(9,26)	<9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26>	Yes	
4	check for total locks variables DEF(tlock,10) and DEF (tlock,16) and 3 usage node(use(tlocks,16), USE(tlock,21), USE(tlocks,24))	(10,16)	<10-11-12-13-14-15-16>	Yes	
		(10,21)	<10-11-12-13-14-15-16-17-18-19-20-14-21>	No	
		(10,24)	<10-11-12-13-14-15-16-17-18-19-20-14-21-22-23-24>	No	
		(16,16)	<16-16>	Yes	
		(16,21)	<16-17-18-19-14-21>	No	
		(16,24)	<16-17-18-19-20-14-21-22-23-24>	No	
5	check for total stocks variables DEF((tstocks,11) and DEF(tstocks,17)) and 3usage node (USE (tstocks,17), USE (tstocks,22), USE (tstocks,25))	(11,17)	<11-12-13-14-15-16-17>	Yes	
		(11,22)	<11-12-13-14-15-16-17-18-19-20-21-14-21>	No	
		(11,25)	<11-12-13-14-15-16-17-18-19-20-21-14-21-23-24-25>	No	
		(17,17)	<17-17>	Yes	
		(17,22)	<17-18-19-20-14-21-22>	No	
		(17,25)	<17-18-19-20-14-21-22-23-24-25>	No	
6	check for locks variable (DEF (locks,13), DEF (locks,14) and USE (locks,14), USE (locks,16)	(13,14)	<13-14>	Yes	Begin the loop
		(13,16)	<13-14-15-16>	Yes	
		(19,14)	<19-20-14>	Yes	
		(19,16)	<19-20-14-15-16>	Yes	Repeat the loop
7	check for stocks variable (DEF (stock,15) and USE (stocks,17)	(15,17)	<15-16-17>	Yes	
8	check for sales DEF(sales,27) and USE (sales,28) USE (sales,29), USE(sales,33), USE (sales,34), USE (sales,37), USE (sales,39)	(27,28)	<27-28>	Yes	
		(27,29)	<27-28-29>	Yes	
		(27,33)	<27-28-29-30-31-32-33>	Yes	
		(27,34)	<27-28-29-34>	Yes	
		(27,37)	<27-28-29-34-35-36-37>	Yes	
		(27,39)	<27-28-29-34-38-39>	Yes	
9	check for commission variable DEF(comm,31, 32, 33), DEF (comm,34,35) and DEF (comm,39) and USE (comm,40)	((31,32,33),40)	<31-32-33-40>	Yes	
		((34,35),40)	<34-35-40>	Yes	
		(39,40)	<39-40>	Yes	

Source Code

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

```
31  printf("the total sales = %f ", sales);
32  if (sales > 1800.0)
33  {
34      comm = 0.10 * 1000.0;
35      comm = comm + 0.15 * 800;
36      comm = comm + 0.20 * (sales - 1800.0);
37  }
38  else if (sales > 1000)
39  {
40      comm = 0.10 * 1000;
41      comm = comm + 0.15 * (sales - 1000);
42  }
43  else
44      comm = 0.10 * sales;
45  printf("the commission is = %f ", comm);
46  return 0;
47 }
```

Conclusion:

Data Flow Testing for commission Problem verifies that all variables are correctly initialized, used and updated.

References:

* Roger S. Pressman, Software Engineering : A practitioner's Approach.
* Ian Sommerville, SOFTWARE ENGINEERING, Addison-Wesly, 9th Edition.

Experiment No: 4 Date:

Boundary Value Testing for Commission Problem

problem statement

Design, Develop, code and run a program to calculate the sales commission based on the number of locks, stocks and barrels sold in a month and to analyze it using the Boundary Value Testing.

Theory:

Boundary Value Testing is a black-box testing technique used to check the behavior of software at the boundaries of input domains. Instead of testing only typical values, we test extreme values - the minimum, maximum and values just inside or outside these limits.

The rationale is that most program errors occur at boundaries, not in the middle of input ranges.

Commission Problem Output Boundary Value Analysis

Case Id	Description	Input Data			Expected Output		Actual Output		Status	Comment
		total Locks	Total stocks	Total Barrels	sales	Comm-ission	sales	Comm-ission	status	Comment
1	Enter the min value for locks, stocks and barrels	1	1	1	100	10	100	10.00	PASS	output minimum
2	Enter the min value for 2 items and min+1 for any one item	1	1	2	125	12.5	125	12.50	PASS	output minimum+
3		1	2	1	130	13	130	13.00	PASS	output min+
4		2	1	1	145	14.5	145	14.50	PASS	output min+
5	Enter the values sales approximately mid value between 100 to 1000	5	5	5	500	50	500	50.00	PASS	Midpoint
6	Enter the values to calculate the commission for sales nearly less than 1000	10	10	9	975	97.5	975	97.50	PASS	Border point -
7		10	9	10	970	97	970	97.00	PASS	Border point -
8		9	10	10	955	95.5	955	95.50	PASS	Border point -
9	Enter values sals exactly equal to 1000	10	10	10	1000	100	1000	100.00	PASS	Border point
10	Enter the values to calculate the commission for sales nearly greater then 1000	10	10	11	1025	103.75	1025	103.75	PASS	Borders point +
11		10	11	10	1030	104.5	1030	104.50	PASS	Borders point +
12		11	10	10	1045	106.75	1045	106.75	PASS	Border point +
13	Enter the value sales appox mid value between 1000 to 1800	14	14	14	1400	160	1400	160.00	PASS	Midpoint
14	Enter the value to calculate the commission for sales nearly less than 1800	18	18	17	1775	216.25	1775	216.25	PASS	Borders point -
15		18	17	18	1770	215.5	1770	215.50	PASS	Border point -
16		17	18	18	1775	213.25	1775	213.25	PASS	Border point -
17	Enter values sales exactly equal to 1800	18	18	18	1800	220	1800	220.00	PASS	Border point
18	Enter the values to calculate the commission for sales nearly greater than 1800	18	18	19	1825	225	1825	225.00	PASS	Border point +
19		18	19	18	1830	226	1830	226.00	PASS	Border point +
20		19	18	18	1845	229	1845	229.00	PASS	Border point +
21	Enter the values normal value for lock, stock and barrel	48	48	48	4800	820	4800	820.00	PASS	Midpoint
22	Enter the max value for 2 items and max-1 for any one item	70	80	89	7775	1415	7775	1415.00	PASS	output max -
23		70	79	90	7750	1414	7750	1414.00	PASS	output max -
24		69	80	90	7755	1411	7755	1411.00	PASS	output max -
25	Enter the max value for locks, stocks and barrels	70	80	90	7800	1420	7800	1420.00	PASS	output max

Source Code:

```
#include <stdio.h>
int main(){
    int locks, stocks, barrels, tlocks=0, tstocks=0, tbarrels=0;
    float lprice=45.0, sprice=30.0, bprice=25.0, sales, comm;

    printf("Enter number of locks (-1 to stop)");
    scanf("%d", &locks);

    while(locks!=-1){
     int c1= (locks<=0 || locks>70);
     printf("Enter number of stocks and barrels:");
     scanf("%d %d", &stocks, &barrels);
     int c2 = (stocks<=0 || stocks>80);
     int c3 = (barrels<=0 || barrels>90);

     if(!c1 && tlocks+locks<=70) tlocks+=locks;
     else if(c1) printf("Invalid locks (1-70)\n");
     if(!c2 && tstocks+stocks<=80) tstocks+=stocks;
     else if(c2) printf("Invalid stocks (1-80)\n");
     if(!c3 && tbarrels+barrels<=90) tbarrels+=barrels;
     else if(c3) printf("Invalid barrels (1-90)\n");

     printf("Totals-> locks:%d, stocks:%d, barrels:%d\n", tlocks, tstocks, tbarrels);
     printf("\n Enter number of locks (-1 to stop):");
     scanf("%d", &locks);
    }
```

```
sales = lprice * locks + sprice * stocks + bprice * barrels;
printf ("total sales = %.2f \n", sales);

if (locks > 0 && stocks > 0 && barrels > 0) {
    if (sales > 1800) comm = 0.10 * 1000 + 0.15 * 800 + 0.20 * (sales - 1800);
    else if (sales > 1000) comm = 0.10 * 1000 + 0.15 * (sales - 1000);
    else comm = 0.10 * sales;
    printf ("Comission = %.2f \n", comm);
} else {
    printf ("Commission cannot be be calculated. \n");
}

return 0;
}
```

Conclusion:

Boundary Value Testing proved to be effective method for identifying errors near input and output limits.

In this Commission Problem, the program:

* Handled valid and invalid input ranges accurately.
* calculated Correct Commission at all boundary points.

References:

* Roger S. Pressman, Software Engineering: A PractiHoner's Approach.
* Ian Sommerville, SOFTWARE ENGINEERING, Addison-Wesly, 9th EditHon

Experiment No: 5 Date:

Equivalence class Testing for Commission Problem

problem Statement

To design and test a commission calculation program using Equivalence class Testing - by dividing input data into valid and invalid classes and verifying correct behavior for each.

Theory:

Equivalence Class Testing is a black-box testing technique that divides the input data of a program into equivalence classes or partitions. Each partition represents a set of valid or invalid input values that are expected to behave similarly.

Weak Robustness Equivalence Class

Case Id	Description	Input Data			Expected Output	Actual Output	Status	Comment
		Locks	Stocks	Barrels				
WR1	Enter the value locks = -1	-1	40	45	Terminates Input loop and proceed to calculate sales Commission	Terminates loop	PASS	
WR2	Enter value less than -1 or equal to zero for locks and other valid inputs	0	40	45	Value of locks not in the range (1-70)	Value of locks not in range	PASS	
WR3	Enter value greater than 70 for locks and other valid inputs	71	40	45	Value of locks not in the range (1-70)	Value of locks not in range	PASS	
WR4	Enter value less than or equal than 0 for stocks and other valid inputs	35	0	45	Value of stocks not in the range (1-80)	Value of stocks not in range	PASS	
WR5	Enter value greater than 80 for stocks and other valid inputs.	35	81	45	Value of stocks not in the range (1-80)	Value of stocks not in range	PASS	
WR6	Enter value less than or equal to 0 for barrels and other valid inputs.	35	40	0	Value of Barrels not in the range (1-90)	Value of Barrels not in range	PASS	
WR7	Enter the value greater than 90 for barrels and other valid inputs	35	40	91	Value of Barrels not in the range (1-90)	Value of Barrels not in range	PASS	

Strong Robustness Equivalence Class

Case Id	Description	Input Data			Expected Output	Actual Output	Status	Comment
		Locks	Stocks	Barrels				
SR1	Enter the value less than -1 for locks and other valid inputs	-2	40	45	Value of locks not in range (1-70)	value of locks not in range	PASS	
SR2	Enter value less than or equal than 0 for stocks and other valid inputs	35	-1	45	Value of stocks not in range (1-80)	value of stocks not in range	PASS	
SR3	Enter value less than or equal, 0 for barrels and other valid inputs	35	40	-2	Value of Barrels not in range (1-90)	value of Barrels not in range	PASS	
SR4	Enter the locks and stocks less than or equal to 0 and other valid inputs	-2	-1	45	Value of locks not in range (1-70)	value of locks not in range	PASS	
					Value of stocks not in range (1-80)	value of stocks not in range	PASS	
SR5	Enter locks and barrels less than or equal to 0 and other valid inputs	-2	40	-1	Value of locks not in range (1-70)	value of locks not in range	PASS	
					Value of Barrels not in range (1-90)	value of barrels not in range	PASS	
SR6	Enter the stocks and barrel less than or equal to 0 and other valid inputs	35	-1	-1	Value of stocks not in range (1-80)	value of stocks not in range	PASS	
					Value of Barrels not in range (1-90)	value of barrels not in range	PASS	
SR7	Enter the stocks and barrels less than or equal to 0 and other valid inputs	-2	-2	-2	Value of locks not in range (1-70)	value of locks not in range	PASS	
					Value of stocks not in range (1-80)	value of stocks not in range	PASS	
					Value of Barrels not in range (1-90)	value of barrels not in range	PASS	

Source Code:

```
#include <stdio.h>
int main(){
    int locks, stocks, barrels, tlocks=0, tstocks=0, tbarrels=0;
    float lprice=45.0, sprice=30.0, bprice=25.0, sales, comm;

    printf("Enter number of locks (-1 to stop)");
    scanf("%d", &locks);

    while (locks!=-1){
        int c1 = (locks<=0 || locks>70);
        printf("Enter stocks and barrels:");
        scanf("%d %d", &stocks, &barrels);
        int c2 = (stocks<=0 || stocks>80);
        int c3 = (barrels<=0 || barrels>90);

        if(!c1 && tlocks+locks<=70)tlocks+=locks;
        else if(c1) printf("Invalid locks (1-70)\n");
        if(!c2 && tstocks+stocks<=80) tstocks+=stocks;
        else if(c2) printf("Invalid stocks (1-80)\n");
        if(!c3 && tbarrels+barrels<=90)tbarrels+=barrels;
        else if(c3) printf("Invalid barrels (1-90)\n");

        printf("total -> locks: %d, stocks: %d, Barrels: %d\n", tlocks, tstocks, tbarrels);
        printf("\n Enter number of locks (-1 to stop)");
        scanf("%d", &locks);
    }
```

```
sales = lprice * hlocks + sprice * hstocks + bprice * hbarrels;
printf ("total sales = %.2f\n", sales);

if (hlocks > 0 && hstocks > 0 && hbarrels > 0 ){
  if (sales > 1800) comm = 0.10*1000 + 0.15*800 + 0.20*(sales - 1800);
  else if(sales > 1000) comm = 0.10*1000 + 0.15*(sales-1000);
  else comm = 0.10*sales;
  printf ("commission = %.2f\n", comm);
} else {
    printf ("commission can't be calculated\n");
}

return 0;
}
```

Conclusion:

Equivalence Class Testing for the Commission Problem effectively validated the correctness of the program by dividing the input domain into valid and invalid classes for locks, stocks and barrels. This technique minimized the number of test cases while ensuring through coverage of all possible input conditions.

References:

* Roger S Pressman, Software Engineering: A Practitioner's Approach.
* Ian Sommerville, SOFTWARE ENGINEERING, Addison-Wesley, 9th Edition.

Experiment No : 6 Date :

Decision Test case for Commission Problem

Problem Statement

Design, develop and execute test cases for the Commission Problem using the Decision Table-Based Testing Technique, and to analyze the correctness of the commission calculation based on different input conditions and decision.

Theory:

Decision Table - Based Testing is a black-box Testing technique that represents complex business logic or system behavior in a tabular form based on combinations of inputs and corresponding actions. It is particularly useful when the system under test involves multiple logic conditions that determine different outcomes.

Input data decision Table

	RULES	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Conditions	c1 : Locks = -1	T	F	F	F	F	F	F	F	F	T
	c2 : 1 ≤ Locks ≤ 70	-	T	T	F	T	F	F	T	F	T
	c3 : 1 ≤ Stocks ≤ 80	-	T	F	T	F	T	F	T	F	T
	c4 : 1 ≤ Barrels ≤ 90	-	F	T	T	F	F	T	T	F	T
Actions	a1 : Terminate the loop	X									X
	a2 : Invalid locks input				X		X	X		X	
	a3 : Invalid stocks input			X		X		X		X	
	a4 : Invalid barrels input		X			X	X			X	
	a5 : Calculate Sales	X									X
	a6 : Proceed to Commision decision table	X									X
	a7 : Calculate total locks, Stocks and barrels		X	X	X	X	X	X	X		

Commission calculation Decision Table (Precondition : lock = -1)

	RULES	R1	R2	R3	R4
Conditions	c1 : locks > 0 AND stocks > 0 AND barrels > 0	T	T	T	F
	c2 : Sales > 0 AND Sales ≤ 1000	T	F	F	
	c3 : Sales > 1001 AND Sales ≤ 1800		T	F	
	c4 : Sales ≥ 1801			T	
Actions	a1 : Can't calculate Commission				X
	a2 : Comm = 10 % * Sales	X			
	a3 : Comm = 10% * 1000 + (Sales - 1000) * 15%		X		
	a4 : Comm = 10 % * 1000 + 15 % * 800 + (Sales - 1800) * 20%			X	

Commission Problem - Decision Table Test cases for Commission calculation (Precondition locks=1)

Case Id	Description	Input Data			Expected Output		Actual Output		Status	Comments
		Locks	stocks	barrels	Expected Sales	Expected Commission	Actual Sales	Actual Commission		
1	check value of all equal 0	0	0	0	0	0	0	0	PASS	
2	if sales value with in these range (sales>0 AND sales <= 1000)	10	9	10	970	97	970	97.00	PASS	
3	if sales value with in these range (sales > 1000 AND sales <= 1800)	15	15	15	2470	354	247	354.00	PASS	
4	if sales value with in these range (sales > 1800)	20	30	40	5270	914	5270	914.00	PASS	

Source Code:

```
#include <stdio.h>
int main(){
    int locks, stocks, barrels, tlocks=0, tstocks=0, tbarrels=0;
    float lprice=45.0, sprice=30.0, bprice=25.0, sales, comm;

    printf("Enter no. of locks (-1 to stop)");
    scanf("%d", &locks);

    while(locks != -1){
        int c1 = (locks<=0 || locks>70);
        printf("Enter number of stocks and barrels:");
        scanf("%d %d", &stocks, &barrels);
        int c2 = (stocks<=0 || stocks>80);
        int c3 = (barrels<=0 || barrels>90);

        if(!c1 && tlocks+locks<=70) tlocks+=locks;
        else if(c1) printf("Invalid locks (1-70)\n");
        if(!c2 && tstocks+stocks<=80) tstocks+=stocks;
        else if(c2) printf("Invalid stocks (1-80)\n");
        if(!c3 && tbarrels+barrels<=90) tbarrels+=barrels;
        else if(c3) printf("Invalid barrels (1-90)\n");

        printf("total -> locks:%d, stocks:%d, Barrels:%d", tlocks, tstocks, tbarrels);
        printf("Enter no. of locks (-1 to stop):");
        scanf("%d", &locks);
    }
```

```
sales = lprice * locks + sprice * stocks + bprice * barrels;
printf("total sales = %.2f \n", sales);

if(locks > 0 && stocks > 0 && barrels > 0){
    if (sales > 1800) comm = 0.10 * 1000 + 0.15 * 800 + 0.20 * (sales - 1800);
    else if (sales > 1000) comm = 0.10 * 1000 + 0.15 * (sales - 1000);
    else comm = 0.10 * sales;
    printf("commission = %.2f \n", comm);
} else {
    printf("commission can't be calculated");
}

return 0;
}
```

Conclusion:

Decision Table-Based Testing for Commission Problem provided a structured approach to validate all possible combinations of inputs and their corresponding outputs. By organizing conditions and actions into a decision table, it ensured complete coverage of commission calculation rules and error-handling scenarios.

References:

* Roger S. Freesman, Software Engineering: A Practitioner's Approach.
* Ian Sommerville, SOFTWARE ENGINEERING, Addison-Wesley, 9th Edition.

Experiment No : 7 Date :

Path testing

Problem statement

Design, develop and execute a C program that implements the Binary Search algorithm. The program should be analyzed using Path Testing to determine the independent paths in the control flow graph of the algorithm. Using these paths, suitable test cases must be derived and executed to verify that all possible logical paths in the binary search code are covered and that the program functions correctly for all conditions.

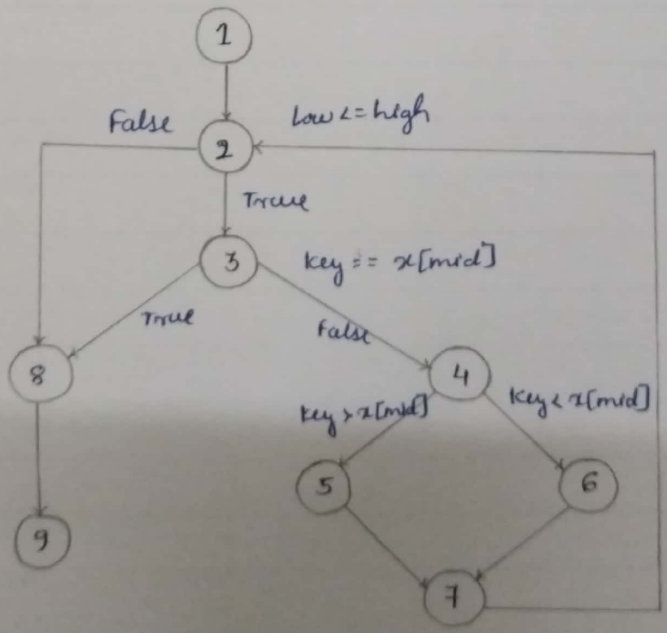
Theory

Path Testing is a white-box Testing technique used to ensure that all possible execution paths in a program are executed at least once. It is based on the program's control flow graph (CFG), where each node represents a statement or block of code, and edges represent the control flow between them. The goal of Path Testing is to identify basis paths, which are independent paths that ensure complete branch coverage.

Binary Search function with line number

```
0   int binsrc (int x[], int low, int high, int key)
    {
1       int mid;
2       while (low <= high)
        {
            mid = (low + high)/2;
3           if (key == x[mid])
8                   return mid;
4           else if (key > x[mid])
5                   low = mid+1;
6           else
                high = mid-1;
7       }
8       return -1;
9   }
```

Program Graph - for Binary Search



Pre-Conditions / Issues:

Array has Elements in Ascending order : T/F
key element is in the Array : T/F
Array has ODD number of Elements: T/F

Path	Inputs		Expected output	Remarks
	x[]	key		
P1: 1-2-3-8-9	{10, 20, 30, 40, 50}	30	Success	key ∈ x[] and key == x[mid]
P2: 1-2-3-4-5-7-2	{10, 20, 30, 40, 50}	20	Repeat and Success	key < x[mid] search 1st Half
P3: 1-2-3-4-6-7-2	{10, 20, 30, 40, 50}	40	Repeat and Success	key > x[mid] search 2nd Half
P4: 1-2-8-9	{10, 20, 30, 40, 50}	60 OR 05	Repeat and Failure	key ∉ x[]
P5: 1-2-8-9	Empty	Any key	Failure	Empty List

Source Code:

```
#include <stdio.h>
int binsrc (int arr[], int n, int k) {
    int low = 0, high = n-1, mid;
    while (low <= high) {
      mid = (low + high) / 2;
      if (arr[mid] == key) return mid;
      else if (key < arr[mid]) high = mid - 1;
      else low = mid + 1;
    }

    return -1;
}

int main() {
   int arr[20], n, key, pos;
   print ("Enter size ");
   scanf ("%d", &n);

   if (n <= 0) {
      printf ("Num of elements must be greater than zero.");
      return 0; }
   printf ("Enter elements");
   for(int i=0; i<n ; i++) { scanf ("%d", &arr[i]); }
   printf ("Enter key "); scanf ("%d", &key);
   pos = binsrc (arr, n, key);
   (pos != -1) ? printf ("Element found at %d", pos+1) :
              printf ("Element not found");

   return 0;
}
```

Conclusion:

Path Testing applied to the Binary Search program effectively validated all independent execution paths of the algorithm. Each path represented a different decision outcome, ensuring that all conditions : key found, key not found and boundary scenarios were tested.

References:

* Roger S. Freedman, Software Engineering : A Practitioner's Approach.
* Ian Sommerville , SOFTWARE ENGINEERING , Addison-Wesley , 9th Edition.

Experiment No: 8 Date

Implementation of Quick Sort Algorithm and Path Testing

problem Statement

Design, develop and execute a program to implement the quick sort algorithm. Using the program's control flow, identify basis path and derive independent test case using path testing.

Theory:

Path Testing

Path Testing is a white-box-testing technique used to ensure that every independent path in the program's control flow has been tested.

i. Draw the control Flow Graph (CFG)

Nodes = Statements

Edges = flow of control

ii. Calculate Cyclomatic Complexity (V(G))

Using formula:

$V(G) = E - N + 2$

where

E : number of edges

N : number of nodes

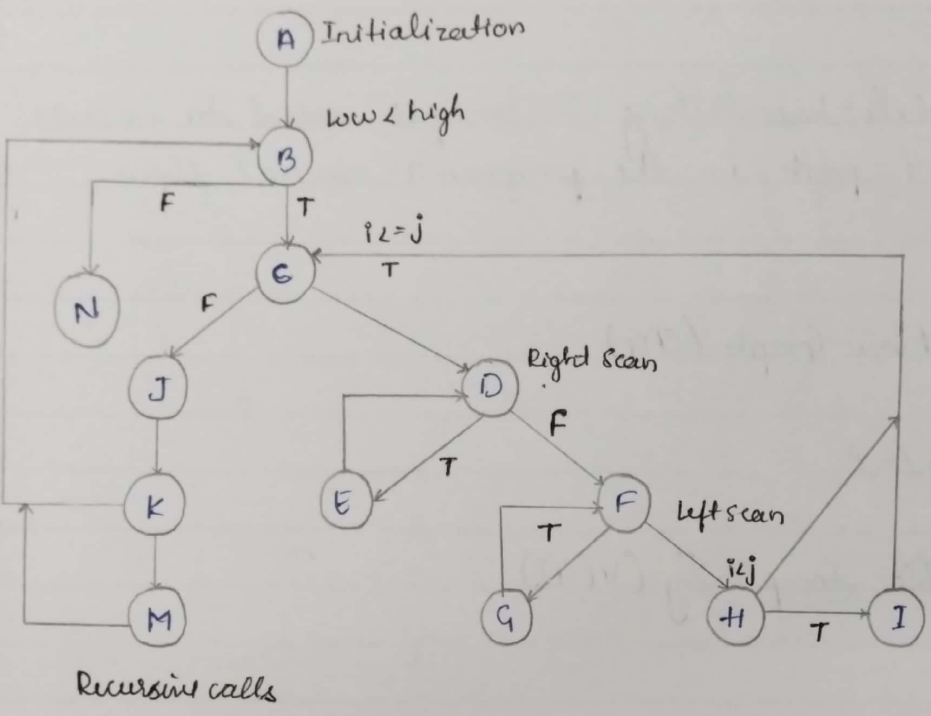
iii. Identify Bases Paths

Each independent path = one unique test scenario

iv. Design Test cases to cover each path.

- All decision are tested.
- Logic errors are minimized.
- Program reliability increases.

Program Graph



Independent paths

Edges = 18 Nodes = 13 P = 1

$$V(G) = E - N + 2P$$
$$= 18 - 13 + 2$$
$$= 7$$

Independent Path

P_1: A-B-N

P_2: A-B-C-J-K-B

P_3: A-B-C-J-K-M-B

P_4: A-B-C-D-F-H-C

P_5: A-B-C-D-F-H-I-C

P_6: A-B-C-D-E-D-F-H

P_7: A-B-C-D-F-G-F-H

Paths	Inputs		Expected Output	Remarks
	X[]	First, last		
P1: A-B-N	5	1,1	Sorted	only one Element
P2: A-B-C-J-K-B	5,4	1,2	Repeated & Sorted	Two elements
P3: A-B-C-J-K-M-B	1,2,3 OR 3,2,1	1,3	Repeated & Sorted	Three elements
P4: A-B-C-D-F-H-C	1,2,3,4,5	1,5	Repeated & Sorted	ASC Sequence
P5: A-B-C-D-F-H-I-C	5,4,3,2,1	1,5	Repeated & Sorted	DSC Sequence
P6: A-B-C-D-E-D-F-H	1,4,3,2,5 OR 2,2,2,2,2	1,5	Repeated & Sorted	Pivot in MID
P7: A-B-C-D-F-G-F-H	5,2,3,1,4	1,5	Repeated & Sorted	Pivot is MAX

Source Code:

```
#include <stdio.h>
void swap (int *a, int *b){
    int t = *a
        *a = *b
        *b = t;
}
int partition (int arr[], int low, int high){
    int pivot = arr[high], i = low-1;
    for (int j= low ; j< high ; j++){
        if (arr[j] <= pivot ){
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i+1], &arr[high]);
    return i+1;
}
void quickSort (int arr[], int low, int high){
    if (low < high){
        int pi = partition (arr, low, high);
        quickSort (arr, low, pi-1);
        quickSort (arr, pi+1, high );
    }
}
int main (){
 int arr[100], n;
 printf(" Enter n :"); scanf("%d",&n);
 for (int i=0; i<n; i++){ scanf(" %d", &arr[i]);}
 quickSort (arr, 0, n-1);
 for (int i=0; i<n ; i++){ printf(" %d", arr[i]);}
 return 0;
}
```

Conclusion:

The Quick Sort algorithm was successfully implemented and tested using basis path testing. By analyzing the control flow and desiring cyclomatic complexity, four independent paths were identified.

References:

* Roger. S Pressman, Software Engineering : A Practitioner's Approach
* Ian Sommerville, SOFTWARE ENGINEERING, Addison - Wesley 9th Edition

Experiment No: 9 Date:

Next Date Function - Boundary Value Analysis Testing

Problem Statement

Design, develop and execute program to compute the next calendar date for a given valid input date. Analyze the program using Boundary Value Analysis (BVA) to derive test cases, execute them.

Theory

Boundary Value Analysis (BVA)

Boundary Value Analysis is a black-box testing technique based on the observation that most bugs occur at the edges of input ranges, not in the middle.

Boundaries for the Next Date Program

i) Day Boundaries
- Lower : 1, 2
- Upper : 30, 31
- February Boundaries : 28, 29

ii) Month Boundaries
- Lower : 1, 2
- Upper : 11, 12

iii) Year Boundaries
- Lower : 1900 , 1901
- Upper : 2099 , 2100

Next date Output Boundary Value Analysis Cases

Case ID	Description	Input Data			Expected Output			Actual Output			Status	Comment
		month	day	year	month	day	year	month	day	year		
1	Enter the min value month, day and year	1	1	1812	1	2	1812	1	2	1812	pass	
2	Enter the min+1 value of for year and min for month and day	1	1	1813	1	2	1813	1	2	1813	pass	
3	Enter the normal value for year and min for month and day	1	1	1912	1	2	1912	1	2	1912	pass	
4	Enter the max-1 value for year and min for month and day	1	1	2012	1	2	2012	1	2	2012	pass	
5	Enter the max value of for year and min for month and day	1	1	2013	1	2	2013	1	2	2013	pass	
6	Enter the min+1 value of day and min for month and year	1	2	1812	1	3	1812	1	3	1812	pass	
7	Enter the min+1 value for day and year and min for month	1	2	1813	1	3	1813	1	3	1813	pass	
8	Enter the min+1 value for day, normal value for year and min value for month	1	2	1912	1	3	1912	1	3	1912	pass	
9	Enter the min+1 value for day, max-1 value for year and min value for month	1	2	2012	1	3	2012	1	3	2012	pass	
10	Enter the min+1 value for day, max value for month	1	2	2013	1	3	2013	1	3	2013	pass	
11	Enter the normal value of day and min for year and month	1	15	1812	1	16	1812	1	16	1812	pass	
12	Enter the normal value for day and min+1 for year and min for month	1	15	1813	1	16	1813	1	16	1813	pass	
13	Enter the normal value for day normal value for year and min value for month	1	15	1912	1	16	1912	1	16	1912	pass	
14	Enter the normal value for day, max-1 value for year and min value for month	1	15	2012	1	16	2012	1	16	2012	pass	
15	Enter the normal value for day, max value for year and min value for month	1	15	2013	1	16	2013	1	16	2013	pass	
16	Enter the max-1 value of day and min for day and year.	1	30	1812	1	31	1812	1	31	1812	pass	
17	Enter the max-1 value for day and min for month and min+1 for year	1	30	1813	1	31	1813	1	31	1813	pass	
18	Enter the max-1 value for day, normal value for year and min value for month.	1	30	1912	1	31	1912	1	31	1912	pass	
19	Enter the max-1 value for day, max-1 value for year and min value for month	1	30	2012	1	31	2012	1	31	2012	pass	
20	Enter the max-1 value for day, max value for year and min value for month	1	30	2013	1	31	2013	1	31	2013	pass	
21	Enter the max value of day and min for year and month	1	31	1812	2	1	1812	2	1	1812	pass	
22	Enter the max value for day and min for month and min+1 for year	1	31	1813	2	1	1813	2	1	1813	pass	
23	Enter the max value for day, normal value for year and min value for month	1	31	1912	2	1	1912	2	1	1912	pass	
24	Enter the max value for day, max-1, value for year and min value for month	1	31	2012	2	1	2012	2	1	2012	pass	
25	Enter the max value for day, max value for year and min value for month	1	31	2013	2	1	2013	2	1	2013	pass	

Source code

```
#include <stdio.h>
int isLeap(int y){return (y%400==0)||(y%4==0&&y%100!=0);}
int daysInMonth(int m, int y){
    if(m==2) return isLeap(y)?29:28;
    if(m==4||m==6||m==9||m==11) return 30;
    return 31;
}
int main(){
    int d,m,y;
    printf("enter day month year"); scanf("%d %d %d",&d,&m,&y);
    if(m<1||m>12||y<1900||y>2100){
        printf("Invalid date"); return 0;
    }
    int dim = daysInMonth(m,y);
    if(d<1||d>dim){
        printf("Invalid date"); return 0;
    }
    if(d<dim) d++;
    else{
        d=1;
        if(m==12){m=1;y++;}
        else m++;
    }
    printf("%02d-%02d-%04d",d,m,y);
    return 0;
}
```

Conclusion

Boundary Value Analysis helped identify important edge conditions such as month transitions, year transitions and leap year scenarios. Testing these boundary values ensured that the Next Date program correctly handles all critical cases and is robust near extreme input conditions.

References:

* Roger S. Pressman, Software Engineering : A Practitioner's Approach.
* Ian Sommerville, SOFTWARE ENGINEERING, Addison-Wesley, 9th Edition.

Experiment No : 10 Date :

Next Date Program Using Equivalence Class Analysis

Problem Statement

Implement a program that calculates the next date for a given day, month and year. Using Equivalence Class Analysis (ECA), divide the input domain into valid and invalid classes, derive test cases, execute them and discuss the results.

Theory

Equivalence Class Analysis (ECA)

Equivalence class Analysis is a black-box testing technique where the input domain is divided into classes of inputs that behave similarly.

From each class, only one representative value needs to be tested.

purpose:

- Reduce total number of test cases
- Maintain complete coverage
- Identify both valid and invalid inputs

Weak and strong Normal Equivalence Class

Case ID	Description	Input Data			Expected Output			Actual Output			Status	Comment
		month	day	year	month	day	year	month	day	year		
WN1, SN1	Enter M1, D1 and Y1 valid cases	6	15	1912	6	16	1912	6	16	1912	pass	

Weak Robustness Equivalence Class

Case Id	Description	Input Data			Expected Output			Actual Output			Status	Comment
		month	day	year	month	day	year	month	day	year		
WR1	Enter M1, D1 & Y1	6	15	1912	6	16	1912	6	16	192	pass	
WR2	Enter M2, D1 & Y1	-1	15	1912	should display message value of month not in range			month not in range			pass	
WR3	Enter M3, D1 and Y1	13	15	1912	should display message vale of month not in range			month not in range			pass	
WR4	Enter M1, D2 and Y1	6	-1	1912	should display message value of day not in range			day not in range			pass	
WR5	Enter M1, D3 and Y1	6	32	1912	should display message value of day not in range			day not in range			pass	
WR6	Enter M1, D1 and Y2	6	15	1811	should display message value of year not in range			year not in range			pass	
WR7	Enter M1, D1 and Y3	6	15	2014	should display message value of year not in range			year not in range			pass	

Strong Robustness Equivalence Class

Case Id	Description	Input Data			Expected Output	Actual Output	Status	Comment
		month	day	year				
SR1	Enter M1, D1 and Y1	-1	15	1912	display message value of month not in range	month not in range	pass	
SR2	Enter M1, D2 and Y1	6	-1	1912	display message value of day not in range	day not in range	pass	
SR3	Enter M1, D1 and Y2	-6	15	1817	display message value of year not in range	year not in range	pass	
SR4	Enter M2, D2 and Y1	-9	-1	1912	display message value of month not in range	month not in range	pass	
					display message value of day not in range	day not in range	pass	
SR5	Enter M1, D2 and Y2	-6	-1	1811	display message value of day not in range	day not in range	Pass	
					display message value of year not in range	year not in range	Pass	
SR6	Enter M2, D1 and Y2	-1	15	1811	display message value of month not in range	month not in range	pass	
					display message value of year not in range	year not in range	pass	
SR7	Enter M2, D2 and Y2	-1	-1	1811	display message value of month not in range	month not in range	pass	
					display message value of day not in range	day not in range	pass	
					display message value of year not in range	year not in range	pass	

Source Code

```
#include <stdio.h>
int isLeap(int y){return (y%400==0) || (y%4==0 && y%100!=0);}
int daysInMonth(int m, int y){
    if(m==2) return isLeap(y)?29:28;
    if(m==4 || m==6 || m==9 || m==11) return 30;
    return 31;
}

int main(){
  int d, m, y;
  printf("Enter date"); scanf("%d %d %d", &d, &m, &y);
  if(m<1 || m>12 || y<1811 || y>2012){
    printf("Invalid date"); return 0;
  }

  int dim = daysInMonth(m,y);
  if(d<1 || d>dim){
    printf("Invalid date"); return 0;
  }

  if(d<dim) d++;
  else{
    d=1;
    if(m==12){m=1; y++;}
    else m++;
  }


  printf("%02d-%02d-%04d", d, m, y);
  return 0;

}
```

Conclusion

Equivalence Class Analysis allowed us to divide the input space of the Next Date function into well-defined valid and invalid classes. By selecting one representative value from each class, we significantly reduced the number of test ~~cases~~ cases while still maintaining complete functional Coverage.

References

* Roger S. Pressman, Software Engineering : A Practitioner's Approach.
* Ian Sommerville , SOFTWARE ENGINEERING, Addison-Wesly, 9th Edition