Program – 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 and execute test cases for your program based on boundary-value analysis and decision-table approach. Compare and contrast the testing techniques used here.

Requirements:

- **R1:**
  The system should accept 3 positive integer numbers (a, b, c) which represent 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 R2 is satisfied, then the system should determine the type of triangle, which can be:
  - **Equilateral** (all three sides are equal)
  - **Isosceles** (two sides are equal)
  - **Scalene** (all three sides are unequal)
- **R4:**
  Limits for the size of any side is - $1 \leq \text{size} \leq 10$.

Algorithm:

- **Step 1:** Input a, b, c as three integer values which represent three sides of the triangle.
- **Step 2:**
  If (a < (b + c)) and (b < (a + c)) and (c < (a + b)), then go to Step 3.
  Else, print **"Not a triangle"**, go to Step 6.
- **Step 3:** If (a == b) and (b == c), then print **"Equilateral triangle"**, go to Step 6.
- **Step 4:** If (a==b) or (b == c) or (a == c), then print **"Isosceles triangle"**, go to Step 6.
- **Step 5:** Print **"Scalene triangle"**.
- **Step 6:** Stop.

Implementation:

Python Code –

```
def verify(a, b, c):
   if a + b > c and b + c > a and c + a > b:
      if a == b == c:
         print("Equilateral triangle")
      elif a == b or b == c or a == c:
         print("Isosceles triangle")
```

```
        else:
            print("Scalene triangle")
    else:
        print("Triangle cannot be formed")

a, b, c = map(int, input("Input the sides of triangle: ").split())

if 1 <= a <= 10 and 1 <= b <= 10 and 1 <= c <= 10:
    verify(a, b, c)
else:
    print("Out of range")
```

C++ Code –

```
#include <iostream>
using namespace std;

void verify(int a, int b, int c) {
    if (a + b > c && b + c > a && c + a > b) {
        if (a == b && b == c) {
            cout << "Equilateral triangle" << endl;
        } else if (a == b || b == c || a == c) {
            cout << "Isosceles triangle" << endl;
        } else {
            cout << "Scalene triangle" << endl;
        }
    } else {
        cout << "Triangle cannot be formed" << endl;
    }
}

int main() {
    int a, b, c;
    cout << "Input the sides of the triangle: ";
    cin >> a >> b >> c;

    if (a >= 1 && a <= 10 && b >= 1 && b <= 10 && c >= 1 && c <= 10) {
        verify(a, b, c);
    } else {
        cout << "Out of range" << endl;
    }
    return 0;
}
```

Boundary Value Analysis

Testing (Test Plan):

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		a	b	c			
BVA_1	Valid inputs, does not form a triangle	1	2	3	Invalid Triangle	Invalid Triangle	Pass
BVA_2	Valid isosceles triangle	10	8	8	Isosceles Triangle	Isosceles Triangle	Pass
BVA_3	Valid equilateral triangle	5	5	5	Equilateral Triangle	Equilateral Triangle	Pass
BVA_4	All sides are zero (invalid input)	0	0	0	Invalid Triangle	Out of range	Fail
BVA_5	Input value is out of range	0	1	2	a is out of range	Out of range	Fail
BVA_6	Input value is out of range	10	11	3	b is out of range	Out of range	Fail
BVA_7	Input value is out of range	1	2	11	c is out of range	Out of range	Fail
BVA_8	Negative Input	-1	2	8	Negative Input	Out of Range	Fail
BVA_9	Less number of Inputs	5	4	-	Value Error	Invalid Triangle	Fail
BVA_10	Character input	a	5	4	Invalid Input	Invalid T	Fail
BVA_11	String Input	abc	5	4	Invalid Input	Invalid T	Fail
BVA_12	Decimal input	1.2	5	4	Invalid Input	Invalid T	Fail
BVA_13	Extra inputs beyond three values	3	4	5	Many number of inputs	Scalene T	Fail
		6	7	5			

Testing Report:

- Total number of Test Cases executed: 13
- Total number of Test Cases Passed: 3
- Total number of Test Cases Failed: 10

Decision Table Approach

Conditions:

C1 – Given input sides forms a Triangle or not
C2 – a=b
C3 – a=c
C4 – b=c

A1 – Not a Triangle
A2 – Scalene Triangle
A3 – Isosceles Triangle
A4 – Equilateral Triangle
A5 - Impossible

Decision Table:

C1	F	T	T	T	T	T	T	T	T
C2		T	T	T	T	F	F	F	F
C3		T	T	F	F	T	T	F	F
C4		T	F	T	F	T	F	T	F
A1	✓								
A2									✓
A3					✓		✓	✓	
A4		✓							
A5			✓	✓		✓			

Testing (Test Plan):

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		a	b	c			
DT_1	Valid inputs, does not form a triangle	1	2	3	Invalid Triangle	Invalid Triangle	Pass
DT_2	Valid isosceles triangle	10	8	8	Isosceles Triangle	Isosceles Triangle	Pass
DT_3	Valid equilateral triangle	5	5	5	Equilateral Triangle	Equilateral Triangle	Pass
DT_4	Input value is out of range	0	1	2	a is out of range	Out of range	Fail
DT_5	Character input	a	5	4	Invalid Input	Invalid T	Fail
DT_6	String Input	abc	5	4	Invalid Input	Invalid T	Fail
DT_7	Negative Input	-1	2	8	Negative Input	Out of Range	Fail
DT_8	Decimal input	1.2	5	4	Invalid Input	Invalid T	Fail
DT_9	Less number of Inputs	5	4	-	Value Error	Invalid Triangle	Fail

Testing Report:

- Total number of Test Cases executed: 9
- Total number of Test Cases Passed: 3
- Total number of Test Cases Failed: 6

Program – 2

Design and develop a program in C / C++ to implement the NextDate function. Analyze it from the perspective of boundary value testing, decision table testing and equivalence class partitioning testing techniques. Derive test cases for each technique, execute them and discuss the test results.

Requirements:

- R1: The system should accept 3 integer inputs: month, day, and year, representing a calendar date.
- R2: The system should validate that:
    - $1 \leq \text{month} \leq 12$
    - $1 \leq \text{day} \leq 31$
    - $1812 \leq \text{year} \leq 2012$
- R3: The system should verify that the combination of month, day, and year forms a valid calendar date (e.g., not June 31) and also correctly account for leap years when validating February 29.
- R4: If the input date is valid, the system should return the next calendar date else return "Invalid Input Date."

Algorithm:

- Step 1: Input Validation
    - Accept three integer inputs: day, month, and year.
    - Check if year is in the valid range [1812, 2012].
    - Check if month is in the valid range [1, 12].
    - Check if day is in the valid range [1, 31].
- Step 2: Leap Year Determination
  Determine if the given year is a leap year:
  If (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0), it is a leap year.
  Set February's days to 29 in a leap year, otherwise 28.
- Step 3: Identify the Number of Days in the Current Month
  Define an array for days in each month:
  {31, 28 (or 29 for leap year), 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
  Assign the number of days for the given month.
- Step 4: Compute the Next Date
  If day < max_days_in_month, increment the day (day = day + 1).
  If day == max_days_in_month:
  Set day = 1.
  If month < 12, increment the month (month = month + 1).
  If month == 12, reset the month to 1 and increment the year (year = year + 1).
- Step 5: Return the Next Date
  Output the next date as (day, month, year).

Implementation:

```
#include <bits/stdc++.h>
int main()
{
   int month[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
   int d, m, y, nd, nm, ny, ndays;

   printf("Enter the day, month, year: ");
   scanf("%d %d %d", &d, &m, &y);

   ndays = month[m - 1];

   if (y < 1812 || y > 2012 || d < 1 || d > ndays || m < 1 || m > 12)
   {
      printf("Invalid input date");
      exit(0);
   }
   if (m == 2)
   {
      if (y % 100 != 0)
      {
         if (y % 4 == 0)
            ndays = 29;
      } else
      {
         if (y % 400 == 0)
            ndays = 29;
      }
   }
   nd = d + 1;
   nm = m;
   ny = y;
   if (nd > ndays)
   {
      nd = 1;
      nm++;
   }
   if (nm > 12)
   {
      nm = 1;
      ny++;
   }
   printf("The given date is: %d : %d : %d\n", d, m, y);
   printf("The next day's date is: %d : %d : %d\n", nd, nm, ny);
   return 0;
}
```

Boundary Value Analysis

Testing (Test Plan):

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		dd	mm	yyyy			
BVA_1	Valid inputs	14	6	2004	15:6:2004	15:6:2004	Pass
BVA_2	Valid Leap Year	28	2	2004	29:2:2004	29:2:2004	Pass
BVA_3	Invalid Leap Year	28	2	2006	1:3:2006	1:3:2006	Pass
BVA_4	Invalid Date upper bound	32	4	2004	Input Day is out of range	Invalid Input Date	Fail
BVA_5	Invalid Date lower bound	0	3	2004	Input Day is out of range	Invalid Input Date	Fail
BVA_6	Invalid Month upper bound	28	13	2004	Input Month is out of range	Invalid Input Date	Fail
BVA_7	Invalid Month lower bound	7	0	2004	Input Month is out of range	Invalid Input Date	Fail
BVA_8	Invalid Year upper bound	10	11	2013	Input Year is out of range	Invalid Input Date	Fail
BVA_9	Invalid Year lower bound	8	8	1810	Input Year is out of range	Invalid Input Date	Fail
BVA_10	Character input	a	6	2004	Invalid Input	No output	Fail
BVA_11	String Input	abc	8	2004	Invalid Input	No output	Fail
BVA_12	Decimal input	1.2	5	2004	Invalid Input	No output	Fail
BVA_13	Negative Input	-6	5	2004	Invalid Input	Invalid Input Date	Fail

Testing Report:

- Total number of Test Cases executed: 13
- Total number of Test Cases Passed: 3
- Total number of Test Cases Failed: 10

Decision Table Approach

Conditions & Actions:

C1 – Month in {M1 (30 days), M2 (31 days), M3 (December), M4 (February)}
C2 – Day in {D1 (<27), D2 (=28), D3 (=29), D4 (=30), D5 (=31)}
C3 – Year in {Y1 (Leap year), Y2 (Not a leap year)}

A1 – Impossible
A2 – Increment Day
A3 – Reset Day
A4 – Increment Month
A5 – Reset Month
A6 - Increment Year

Decision Table:

C1	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2
C2	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5
C3	-	-	-	-	-	-	-	-	-	-
A1					✓					
A2	✓	✓	✓			✓	✓	✓	✓	
A3				✓						✓
A4				✓						✓
A5										
A6										

C1	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4
C2	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
C3	-	-	-	-	-	-	Y1	Y2	Y1	Y2	-	-
A1										✓	✓	✓
A2	✓	✓	✓	✓		✓	✓					
A3					✓			✓	✓			
A4								✓	✓			
A5					✓							
A6					✓							

Testing (Test Plan):

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		dd	mm	yyyy			
DT_1	Valid inputs	14	6	2004	15:6:2004	15:6:2004	Pass
DT_2	Valid Leap Year	28	2	2004	29:2:2004	29:2:2004	Pass
DT_3	Valid Month Change	31	5	2004	1:6:2004	1:6:2004	Pass
DT_4	Invalid Date	32	4	2004	Input Day is out of range	Invalid Input Date	Fail
DT_5	Invalid Month	28	13	2004	Input Month is out of range	Invalid Input Date	Fail
DT_6	Invalid Year	10	11	2013	Input Year is out of range	Invalid Input Date	Fail
DT_7	Invalid Date & Month	30	13	2004	Input Date & Month is out of range	Invalid Input Date	Fail
DT_8	Negative Input	-6	5	2004	Invalid Input	Invalid Input Date	Fail

Testing Report:

- Total number of Test Cases executed: 8
- Total number of Test Cases Passed: 3
- Total number of Test Cases Failed: 5

Equivalence Class Partitioning

Equivalence Classes:

- D1 – {DD: 1<=DD<=31}
- M1 – {MM: 1<=MM<=12}
- Y1 – {YYYY: 1812 <= YYYY <= 2012}

Weak Normal & Strong Normal

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		dd	mm	yyyy			
WN1, SN1	Valid inputs	14	6	2004	15:6:2004	15:6:2004	Pass

Weak Robust

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		dd	mm	yyyy			
WR1	Invalid Date upper bound	32	4	2004	Input Day is out of range	Invalid Input Date	Fail
WR2	Invalid Date lower bound	0	3	2004	Input Day is out of range	Invalid Input Date	Fail
WR3	Invalid Month upper bound	28	13	2004	Input Month is out of range	Invalid Input Date	Fail
WR4	Invalid Month lower bound	7	0	2004	Input Month is out of range	Invalid Input Date	Fail
WR5	Invalid Year upper bound	10	11	2013	Input Year is out of range	Invalid Input Date	Fail
WR6	Invalid Year lower bound	8	8	1810	Input Year is out of range	Invalid Input Date	Fail

Strong Robust

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		dd	mm	yyyy			
SR1	Invalid Date & Month	32	13	2004	Input Day & Month is out of range	Invalid Input Date	Fail
SR2	Invalid Month & Year	14	13	2024	Input Month & Year is out of range	Invalid Input Date	Fail
SR3	Invalid Date & Year	32	3	1804	Input Date & Year is out of range	Invalid Input Date	Fail
SR4	Invalid Inputs	32	13	1800	Input values are out of range	Invalid Input Date	Fail

Testing Report:

- Total number of Test Cases executed: 12
- Total number of Test Cases Passed: 2
- Total number of Test Cases Failed: 10

Program – 3

Design and develop a program in C/ C++ to solve the commission problem. Analyze it from the perspective of boundary value, decision table testing and equivalence class partitioning testing techniques. Derive test cases for each technique, execute them and discuss the test results.

Requirements

- **R1:** The system shall accept three non-negative integers representing the number of locks, stocks, and barrels sold in a town.
- **R2:** The cost of each item is fixed as:
  - Lock = $45
  - Stock = $30
  - Barrel = $25
- **R3:** The input for each type of item must not exceed monthly production limits:
  - Maximum Locks = 70
  - Maximum Stocks = 80
  - Maximum Barrels = 90
- **R4:** The input sequence ends when the number of locks sold is -1, which signals the end of the month and triggers commission calculation.
- **R5:** The system shall calculate total monthly sales value as:
  Total Sales = (Locks×45) + (Stocks×30) + (Barrels×25)
- **R6:** The salesperson's monthly commission shall be calculated based on the total sales as follows:
  - 10% on the first $1000 (inclusive)
  - 15% on the next $800 (i.e., from $1001 to $1800)
  - 20% on any amount above $1800
- **R7:** The system shall output:
  - Total number of locks, stocks, and barrels sold
  - Total dollar value of the sales
  - Total commission earned
- **R8:** The system must ensure that at least one complete rifle (1 lock + 1 stock + 1 barrel) is sold in the entire month. Otherwise, it should produce an error or warning message.

Algorithm

- Step I: Define lockPrice=45.0, stockPrice = 30.0, barrelPrice=25.0
- Step 2: Input locks
- Step 3: while(locks != -1), input device uses -l 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 in $”, commission)
- Step 12: exit

Implementation

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

   printf("Enter the total number of stocks: ");
   scanf("%d", &stocks);
   if ((stocks < 1) || (stocks > 80)) flag = 1;

   printf("Enter the total number of barrels: ");
   scanf("%d", &barrels);
   if ((barrels < 1) || (barrels > 90)) flag = 1;

   if (flag == 1) {
      printf("Invalid input\n");
      return 0;
   }
   t_sales = (locks * 45) + (stocks * 30) + (barrels * 25);

   if (t_sales <= 1000) {
      commission = 0.10 * t_sales;
   } else if (t_sales < 1800) {
      commission = 0.10 * 1000;
      commission += 0.15 * (t_sales - 1000);
   } else {
      commission = 0.10 * 1000;
      commission += 0.15 * 800;
      commission += 0.20 * (t_sales - 1800);
   }
   printf("The total sales is %d and the commission is %.2f\n", t_sales, commission);
   return 0;
}
```

Boundary Value Analysis

Testing (Test Plan):

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		l	s	b			
BVA_1	Valid inputs	10	10	10	100	100	Pass
BVA_2	Valid locks	1	8	8	48.50	48.50	Pass
BVA_3	Valid stocks	5	1	5	38.0	38.0	Pass
BVA_4	Valid barrels	40	45	1	495.0	495.0	Pass
BVA_5	Invalid locks lower bound	0	5	8	locks is out of range	Invalid Input	Fail
BVA_6	Invalid locks upper bound	71	40	45	locks are out of range	Invalid Input	Fail
BVA_7	Invalid stocks lower bound	10	0	45	stocks are out of range	Invalid Input	Fail
BVA_8	Invalid stocks upper bound	10	81	11	stocks are out of range	Invalid Input	Fail
BVA_9	Invalid barrels lower bound	40	45	0	barrels are out of range	Invalid Input	Fail
BVA_10	Invalid barrels upper bound	40	44	91	barrels are out of range	Invalid Input	Fail
BVA_11	Character Input	5	4	a	Non-integer input given	Invalid Input	Fail
BVA_12	Decimal input	40	25	4.5	Non-integer input given	Invalid Input	Fail
BVA_13	Negative Input	-3	4	5	Negative Input	Invalid Input	Fail

Testing Report:

- Total number of Test Cases executed: 13
- Total number of Test Cases Passed: 3
- Total number of Test Cases Failed: 10

Equivalence Class Partitioning

Equivalence Classes:

- L1 – {locks: 1 <= locks <= 70}
- S1 – {stocks: 1 <= stocks <= 80}
- B1 – {barrels: 1 <= barrels <= 90}

Weak Normal & Strong Normal

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		l	s	b			
WN1, SN1	Valid inputs	35	30	40	555.0	555.0	Pass

Weak Robust

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		l	s	b			
WR1	Invalid locks lower bound	0	5	8	locks is out of range	Invalid Input	Fail
WR2	Invalid locks upper bound	71	40	45	locks are out of range	Invalid Input	Fail
WR3	Invalid stocks lower bound	10	0	45	stocks are out of range	Invalid Input	Fail
WR4	Invalid stocks upper bound	10	81	11	stocks are out of range	Invalid Input	Fail
WR5	Invalid barrels lower bound	40	45	0	barrels are out of range	Invalid Input	Fail
WR6	Invalid barrels upper bound	40	44	91	barrels are out of range	Invalid Input	Fail

Strong Robust

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		l	s	b			
SR1	Invalid locks & stocks	71	81	45	Input locks & stocks is out of range	Invalid Input	Fail
SR2	Invalid stocks & barrels	45	81	0	Input stocks & barrels is out of range	Invalid Input	Fail
SR3	Invalid locks & barrels	71	30	91	Input locks & barrels is out of range	Invalid Input	Fail
SR4	Invalid Inputs	71	0	91	Input values are out of range	Invalid Input	Fail

Testing Report:

- Total number of Test Cases executed: 12
- Total number of Test Cases Passed: 2
- Total number of Test Cases Failed: 10

Program – 4

Design and develop a program in C/ C++ to implement an absolute letter grading procedure, making suitable assumptions. Determine the basis paths and using them derive different test cases, execute the test cases and discuss the test results.

Requirements:

- R1: Input six subjects marks in range 1 to 100.
- R2: If R1 is satisfied then compute Total, Percentage and depending on percentage display the grade.

Design:

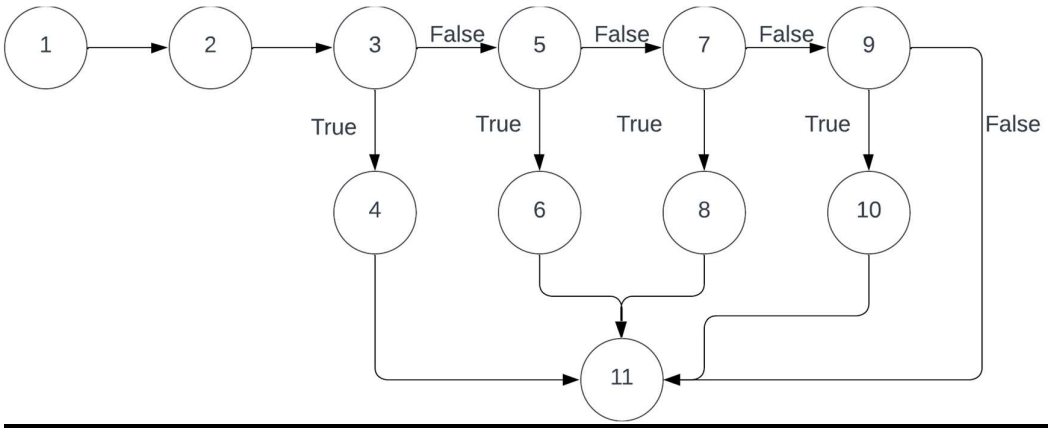
- Grade A – 81 <= per <= 100
- Grade B – 61 <= per <= 80
- Grade C – 41 <= per <= 60
- Fail – 1 <= per <= 40

Implementation (Code):

```
#include<stdio.h>
void grade(int s1,int s2,int s3,int s4,int s5,int s6,float per);
void main()
{
   int s1,s2,s3,s4,s5,s6;
   float total, per;
   printf("Absolute Letter Grading\n");
   printf("Enter marks of Subject - 1:\n");
   scanf("%d",&s1);
   printf("Enter marks of Subject - 2:\n");
   scanf("%d",&s2);
   printf("Enter marks of Subject - 3:\n");
   scanf("%d",&s3);
   printf("Enter marks of Subject - 4:\n");
   scanf("%d",&s4);
   printf("Enter marks of Subject - 5:\n");
   scanf("%d",&s5);
   printf("Enter marks of Subject - 6:\n");
   scanf("%d",&s6);
   total = s1+s2+s3+s4+s5+s6;
   per = (total/600)*100;
   printf("\n The Total Marks secured: %f",total);
   printf("\n The Percentage secured: %f",per);
   grade(s1,s2,s3,s4,s5,s6,per);
}
void grade(int s1,int s2,int s3,int s4,int s5,int s6,float per)
{
   if(s1<=40 || s2<=40 || s3<=40 || s4<=40 ||s5<=40 || s6<=40)
```

```
        printf("\n Fail");
    else if(per>80 && per<=100)
        printf("\n Grade A");
    else if(per>60 && per<=80)
        printf("\n Grade B");
    else if(per>40 && per<=60)
        printf("\n Grade C");
}
```

DD Path Graph:



Cyclomatic Complexity:

- No. of Nodes – 11
- No. of Edges – 14
- No. of Paths – (E – N) + 2
  (14 – 11) + 2
  3 + 2
  5
- Thus, Total Number of Paths – 5

Paths:

- P1: 1 → 2 → 3 → 4 → 11
- P2: 1 → 2 → 3 → 5 → 6 → 11
- P3: 1 → 2 → 3 → 5 → 7 → 8 → 11
- P4: 1 → 2 → 3 → 5 → 7 → 9 → 10 → 11
- P5: 1 → 2 → 3 → 5 → 7 → 9 → 11

Testing (Test Plan)

Test ID	Description	Inputs						Excepted Output	Actual Output	Test Result
		S1	S2	S3	S4	S5	S6			
TC1	Testing Path 1	85	86	87	88	89	32	Fail	Fail	Pass
TC2	Testing Path 2	85	86	87	88	89	90	Grade A	Grade A	Pass
TC3	Testing Path 3	65	66	67	68	69	70	Grade B	Grade B	Pass
TC4	Testing Path 4	45	46	47	48	49	50	Grade C	Grade C	Pass
TC5	Testing Path 5	25	26	27	28	29	30	Fail	Fail	Pass

Testing Report:

- Total number of Test Cases executed: 5
- Total number of Test Cases Passed: 5
- Total number of Test Cases Failed: 0

Program – 5

Design and develop a program in C/ C++ to implement the binary search algorithm. Determine the basis paths and using them derive different test cases, execute the test cases and discuss the test results.

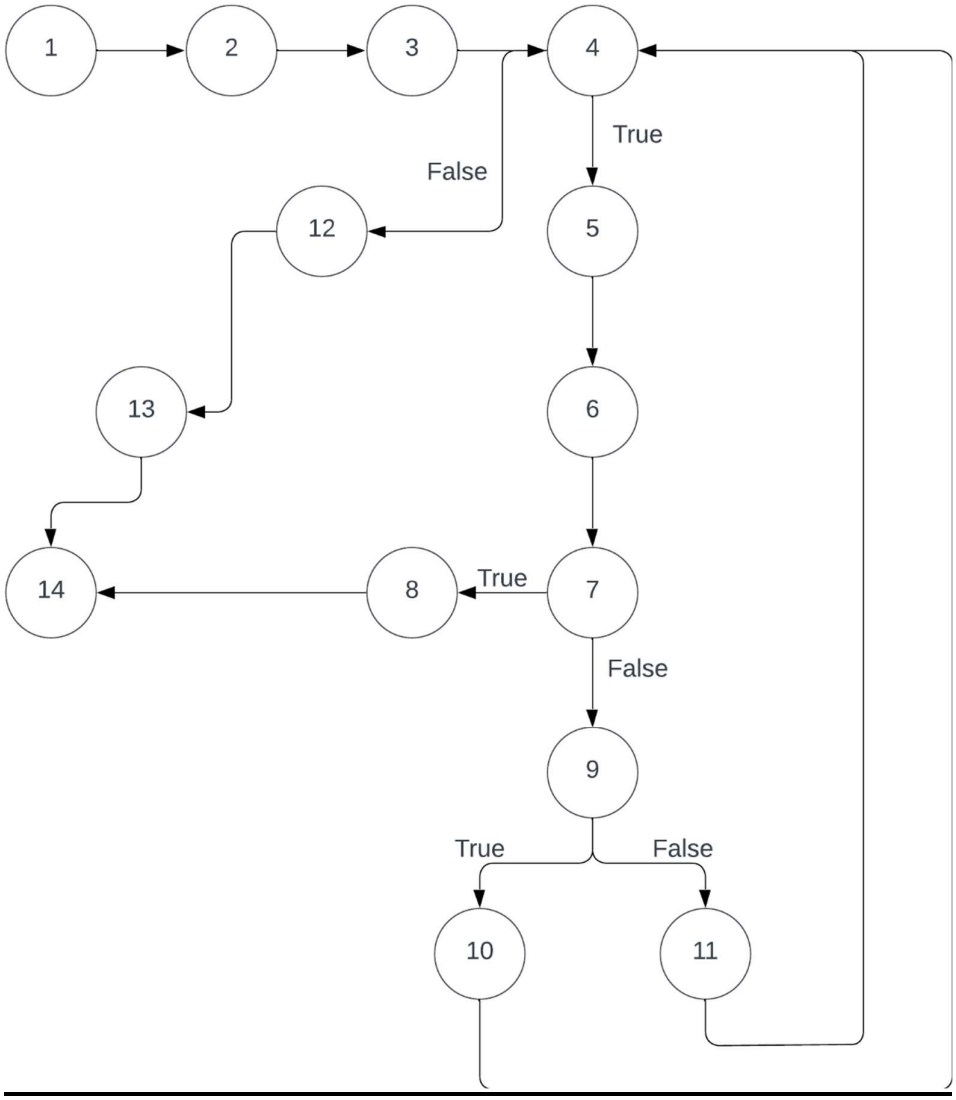
Requirements:

- R1: The system should accept n number of elements and key element that is to be searched among the n elements
- R2: Check if the key element is present in the array and display the position if present otherwise print unsuccessful search

Implementation (Code):

```
#include <stdio.h>
int binarySearch(int arr[], int size, int target)
{
   int low = 0, high = size - 1;
   while (low <= high)
   {
      int mid = (low + high) / 2;
      if (arr[mid] == target)
         return mid;
      else if (arr[mid] < target)
         low = mid + 1;
      else
         high = mid - 1;
   }
   return -1;
}
int main()
{
   int arr[100], n, i, key;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter %d sorted elements:\n", n);
   for (i = 0; i < n; i++)
      scanf("%d", &arr[i]);
   printf("Enter the element to search: ");
   scanf("%d", &key);
   int result = binarySearch(arr, n, key);
   if (result != -1)
      printf("Element found at index %d\n", result);
   else
      printf("Element not found in the array.\n");
   return 0;
}
```

DD Path Graph:



Cyclomatic Complexity:

- No. of Nodes – 14
- No. of Edges – 16
- No. of Paths – (E – N) + 2
  (16 – 14) + 2
  2 + 2
  4
- Thus, Total Number of Paths – 4

Paths:

- P1: 1 → 2 → 3 → 4 → 12 → 13 → 14
- P2: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 14
- P3: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 9 → 10 → 4 → 12 → 13 → 14
- P4: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 9 → 11 → 4 → 12 → 13 → 14

Testing (Test Plan):

Test ID	Description	Inputs			Excepted Output	Actual Output	Test Result
		Size	Array Elements	Key			
TC1	Testing Path P1	0	-	5	-1	-1	Pass
TC2	Testing Path P2	5	12, 44, 64, 87, 99	87	3	3	Pass
TC3	Testing Path P3	4	1,15,35,65	25	-1	-1	Pass
TC4	Testing Path P4	4	1, 3, 5, 8	2	-1	-1	Pass

Testing Report:

- Total number of Test Cases executed: 4
- Total number of Test Cases Passed: 4
- Total number of Test Cases Failed: 0