Parul® University
VADODARA, GUJARAT

NAAC A++
ACCREDITED UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

Software Testing & Quality Assurance Laboratory
(303105378)

SEMESTER VII
Department of
Artificial Intelligence & Data Science
(A.I & D.S)

Laboratory Manual

CERTIFICATE

This is to certify that

Ms. ***Gopi Kota*** *with Enrollment No.* **2303031247002** has *successfully completed his laboratory experiments in the* ***Software Testing & Quality Assurance Lab (303105378)*** *from the department of* **Artificial Intelligence & Data Science (A.I & D.S)** *during the academic year* ***2025-2026.***

Date of Submission …..…… **Staff In charge …..……**

Head of Department……..……

TABLE OF CONTENTS

PRACTICAL: 01

Aim: Create test cases using boundary value analysis.

Objective: To design and execute test cases using Boundary Value Analysis (BVA) technique for identifying potential edge-case errors in software inputs.

Description (Theory in brief):

- **Boundary Value Analysis (BVA)** is a black-box testing technique where test cases are created using the boundaries of input domains. Errors often occur at the edges rather than the center of input ranges. BVA typically includes values:
    - **Minimum (min)**
    - **Minimum + 1 (min+1)**
    - **Maximum - 1 (max-1)**
    - **Maximum (max)**
    - And a **nominal (mid)** value

  This helps ensure software behaves correctly at extreme input ranges.

Examples (with Test cases):

1. **Example 1 : A college library allows students to borrow between 1 to 5 books at a time.**

   `min = 1, max = 5, mid = 3`

Valid Test Cases	Invalid Test Cases
1	0
3	6
5	

2. **Example 2 : An electric kettle can be set between 40°C to 100°C.**
   `min = 40, max = 100, mid = 70`

Valid Test Cases	Invalid Test Cases
40	39
70	101
50	
100	

3. **Example 3 : Building Height Limit Near Indian Airports**
   Within 5 km of the airport, maximum height allowed is 45 meters.

   `min = 0` **m** `max = 45` **m** `mid = 22` **m (approx.)**

Valid Test Cases	Invalid Test Cases
0	-1(negative input)
1	46
22	46
45	50

Conclusion:

- Boundary Value Analysis helps detect errors at input boundaries efficiently. It minimizes test cases while maximizing coverage of edge conditions.

PRACTICAL: 02

Aim: Create test cases using equivalence partitioning.

Objective: To design effective and minimal test cases by dividing input data into equivalence classes and selecting representative values from each class.

Description (Theory in brief):

- **Equivalence Partitioning** is a black-box testing technique where input data is divided into **valid and invalid partitions**. Each partition represents a set of values that should be treated the same by the system. The idea is: if one value in a class works, others will too.
    - **Valid partition**: Inputs that should be accepted.
    - **Invalid partition**: Inputs that should be rejected.

    From each partition, only one value is selected to reduce the total number of test cases.

Guidelines for equivalence partitioning

1. **Range Input Condition:**
    - Define 1 valid and 2 invalid equivalence classes.
    - *Example:*
2. **Specific (Fixed) Value Input:**
    - Define 1 valid and 2 invalid equivalence classes.
    - *Example:*
3. **Set Membership Input:**
    - Define 1 valid and 1 invalid equivalence class.
    - *Example:*

4. **Boolean Input Condition:**
    - Define 1 valid and 1 invalid equivalence class.
    - *Example:*

Examples (with Test cases):

1. **Range Type Input**
    - *Example*: Elevator weight sensor only allows between **50 kg and 250 kg**.

Invalid	Valid	Invalid
0	100	300

    - **Valid Input:** 100
    - **Valid Class:** 50 to 250
    - **Invalid Input:** 0 , 300
    - **Invalid class :** less than 50(too light)
      More than 250(too heavy)

2. **Fixed Value Type Input**
    - *Example*: A contest only accepts the **secret code 9999**.

Invalid	Valid	Invalid
8888	9999	10000

    - **Valid Input:** 9999
    - **Valid Class:** Exact match with 9999
    - **Invalid Input:** 8888, 10000
    - **Invalid Class:** Any number ≠ 9999

3. **Set Membership Type Input**
   - *Example*: Allowed file uploads: **PDF**, **DOCX**, **PNG**

Invalid	Valid
BAT	PDF

   - **Valid Input:** PDF
   - **Valid Class:** {"PDF", "DOCX", "PNG"}
   - **Invalid Input:** EXE, BAT
   - **Invalid Class:** File types not in allowed set

4. **Boolean Type Input**
   - *Example*: User must **accept terms (Yes/No)** before submitting

Invalid	Valid
NO	YES

   - **Valid Input:** YES
   - **Valid Class:** YES **(**accepted terms)
   - **Invalid Input:**NO
   - **Invalid Class:** Anything other than “YES”

Conclusion :

- Equivalence Partitioning helps reduce the number of test cases while ensuring effective test coverage. By dividing inputs into valid and invalid classes based on input type (range, fixed, set, boolean), testers can identify representative test cases and improve testing efficiency with minimal effort.

PRACTICAL: 03

AIM: Design independent paths by calculating cyclomatic complexity using date problem

Cyclomatic Complexity: Is software metric used to measure the complexity of a program. Thomas J. McCabe developed this metric in 1976. McCabe interprets a computer program as a set of a strongly connected directed graph. Nodes represent parts of the source code having no branches and arcs represent possible control flow transfers during the program execution. The notion of a program graph has been used for this measure, and it is used to measure and control the number of paths through a program. The complexity of a computer program can be correlated with the topological complexity of a graph.

Create a cyclomatic complexity graph and identify the independent paths in it.

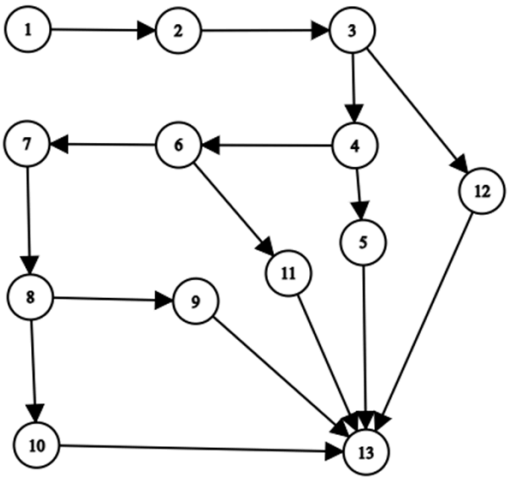
Problem: When entered a month number i.e.2 for example the output should be '30' number of days the input should be given by user

Code:

```
#include <stdio.h>

int main() {

    int month, year;

    printf("Enter the month number: ");

    scanf("%d", &month);

    if (month < 13 && month > 0) {

    if (month == 1 || month == 3 || month == 5 ||

    month == 7 || month == 8 || month == 10 || month == 12) {

    printf("NO. OF DAYS = 31\n");

    }

    else if (month == 2) {

    printf("Enter the year: ");

    scanf("%d", &year);
```

```
if ((year % 400 == 0) || (year % 4 == 0 && year % 100 != 0)) {


        printf("NO. OF DAYS = 29\n");
}
else {
        printf("NO. OF DAYS = 28\n");
}
}
else {
printf("NO. OF DAYS = 30\n");
}
} else {
         printf("Invalid month number.\n");
}
return 0;   }
```

Control Flow Graph



Cyclomatic complexity for the graph:

Method 1:

- **E-N+2**
- Nodes (N) = 13
- Edges (E) = 15

= 15-13+2

= 4

- **Method 2:**
  - **P+1**
  - Predicate nodes = 3
  - 3+1
  - =4

Method 3:

Total regions = 4

(V(G) = 4 **)**

Identification of independent paths(Basic path set) from the control flow graph:

1) 1 → 2 → 3 → 4 → 5 → 13
2) 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → 13
3) 1 → 2 → 3 → 4 → 6 → 7 → 8 → 10 → 13
4) 1 → 2 → 3 → 4 → 6 → 11 → 13