**Practical No 1.**

**Title:** Setting up a company that sells testing services to software houses

**Problem Statement:**

1. What is the testing process that will be followed in the company?
2. What is the focus of the testing services?
3. What kind of people are you going to hire as staff for the company?
4. How are you going to validate that a testing project carried out in the company has been beneficial to the customer?
5. What kind of automated tools will the company use?

**Solution:**

**Answer 1:**

The testing process that will be followed by the company is: a) Prepare for testing a software system.

1. Plan the tests that will be conducted on the software system.
2. Execute the steps as defined in the test plan.
3. Conduct acceptance testing by the software system users. (Note: This testing may be assisted by the IT independent test group.)
4. Analyse test results and report them to the appropriate software system stakeholders.
5. Test the installation of the software into the operational environment, and test changes made to the software after it is placed into the operational environment.
6. Conduct a post-implementation analysis to evaluate the effectiveness and efficiency of the test process.

**Answer 2:**

The focus of testing services will be on:

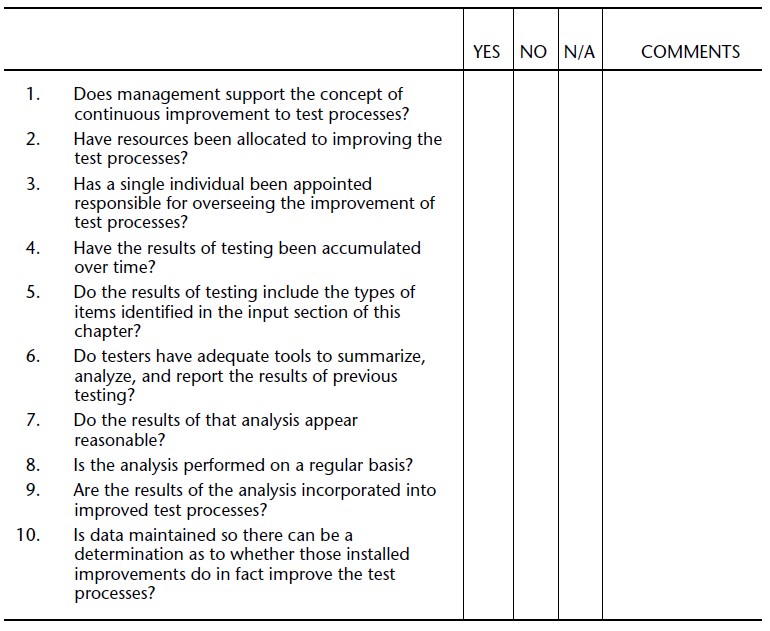
1. **Software testing should reduce software development risk.** Risk is present in all software development projects, and testing is a control that reduces those risks.
2. **Testing should be performed effectively.** Testing should be performed in a manner in which the maximum benefits are achieved from the software testing efforts.
3. **Testing should uncover defects.** Ideally, at the conclusion of testing there should be no defects in the software.
4. **Testing should be performed using business logic.** Money should not be spent on testing unless it can be spent economically to reduce business risk. In other words, it does not make business sense to spend more money on testing than the losses that might occur from the business risk.
5. **Testing should occur throughout the development life cycle.** Testing is not a phase, but rather a process. It begins when development begins and ends when the software is no longer being used.
6. **Testing should test both structure and function.** Testing should test the functional requirements to ensure they are correct, and test the adequacy of the software structure to process those functional requirements in an effective and efficient manner.

**Answer 3:**

The company will recruit people with the following qualities:

1. A **clear communicator**. A defect report is no good if we can't understand it.
2. **Patient.** Sometimes it takes a lot of back-and-forth to get to the root of a problem. And programmers have egos, they'll often try to push issues back to the tester.
3. **Passionate.** The best developers are the ones that really care about development and maybe even get a little excited about it sometimes. Testing isn't that much different.
4. **Creative.** Really exercising a system requires one to try non-intuitive ways of accomplishing tasks, to go *outside* the workflow that the program expects of them and do things that normal users wouldn't do. Task-oriented people who receive a set of instructions and do the exact same thing every time are *no good* for this job.
5. **Analytical**. Just finding a defect isn't enough - a tester has to be able to figure out how to reproduce it. If a report comes in as "intermittent" then there's about a 10% chance it'll get solved. Most developers won't even look at a case without a reasonably concise sequence of repro steps. Good testers have to be able to retrace their steps and narrow down the field of possibilities so as to come up with the simplest possible sequence of actions that trigger a bug.
6. **Not a programmer.** Programmers never want to do any *actual work* testing. They'll spend all their time trying to write automated tests and not do what really matters, which is to make sure the damn thing actually *works the way it is supposed to*. Although there are exceptions to every rule, most programmers simply find testing boring and will do the absolute minimum amount required.

**Answer 4:**



**Answer 5:**

Write about any 5 automation tools:

* 1. Selenium
  2. AutoIT
  3. Bugzilla
  4. QTP
  5. WAPT
  6. Win runner
  7. VTest

**Practical No 3.**

**Title:** Black Box Testing – Equivalence Partitioning and Boundary value Analysis

**Problem Statement:**

The program reads an arbitrary number of temperatures (as integer numbers) within the range -60°C … +60°C and prints their mean value. Design test cases for testing the program with the black-box strategy.

* Decision tables;
* State transition testing.

**Solution:**

The program accepts integers between -60 and +60 so we apply Equivalence partitioning and Boundary Value Analysis:

Equivalence partitions:

|  |  |  |
| --- | --- | --- |
| <-60 | –60 to +60 | >60 |
| Invalid Partition | Valid Partition | Invalid Partition |

Boundary Value Analysis:

|  |  |  |
| --- | --- | --- |
| -61 | -60 | -59 |
| Invalid | Valid Boundary | Valid |
| 59 | 60 | 61 |
| Valid | Valid Boundary | Invalid |

**Test Cases:**

|  |  |  |
| --- | --- | --- |
| Test Case ID | Input Value | Remark |
| 1 | -61 | Invalid Boundary |
| 2 | -100 | Invalid Partition |
| 3 | -60 | Valid Boundary Value |
| 4 | 0 | Valid Partition |
| 5 | +61 | Invalid Boundary |
| 6 | 150 | Invalid Partition |

**Conclusion:**

As the program specifications are given. With the given specifications, the above test cases are generated and seem to work well.

===================================================================

**Practical No 4.**

**Title:** Black Box Testing – Equivalence Partitioning and Boundary value Analysis

**Problem Statement:**

When getting a person’s weight and height as input, the program prints the person’s body weight index. The weight is given in kilograms (as a real number, for instance:

82.0) and the height in meters (as a real number, for instance: 1.86). The body weight index equals weight divided by height squared: weight / (height \* height). Design test cases for testing the program with the black-box strategy.

Program:

#include <stdio.h> #include <conio.h> void main() { float ht, wt, bmi; clrscr(); ht = 0; wt = 0; while (ht<0.3 || ht>2.4) {

printf("Enter valid height in meters"); scanf("%f", &ht);

}

while (wt<2.0 || wt>350) {

printf("\nEnter valid weight in kilograms"); scanf("%f", &wt);

}

bmi=wt/(ht\*ht); printf("\nBMI=%f kg/sq.m",bmi); getch();

}

Equivalence partitions (ht):

|  |  |  |
| --- | --- | --- |
| <0.3 | 0.3 to 2.4 | >2.4 |
| Invalid Partition | Valid Partition | Invalid Partition |
| Equivalence partitions (wt): |  |  |
| <2.0 | 2.0 to 350.0 | >350.0 |
| Invalid Partition | Valid Partition | Invalid Partition |

Boundary Value Analysis (ht):

|  |  |  |
| --- | --- | --- |
| 0.29 | 0.3 | 0.31 |
| Invalid | Valid Boundary | Valid |
| 2.39 | 2.40 | 2.41 |
| Valid | Valid Boundary | Invalid |

Boundary Value Analysis (wt):

|  |  |  |
| --- | --- | --- |
| 1.99 | 2.0 | 2.01 |
| Invalid | Valid Boundary | Valid |
| 349.99 | 350 | 350.01 |
| Valid | Valid Boundary | Invalid |

**Test Cases:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **ht (Input)** | **wt (Input)** |  | **Remark** | **Expected Output** |
| 1 | 0.29 | \* | (1) |  | Enter valid ht |
| 2 | \* | 1.99 | (2) |  | Enter valid wt |
| 3 | 0.30 | 2.00 | (3) |  | 22.22 kg/sq.m |
| 4 | 2.39 | 349.99 | (4) |  |  |
| 5 | 2.40 | 350 | (5) |  |  |
| 6 | 2.41 | \* | (6) |  | Enter valid ht |
| 7 | \* | 350.01 | (7) |  | Enter valid wt |
| 8 | 1.86 | 86.00 | (8) |  |  |

**Conclusion:**

The program was tested with the above test cases and worked fine.

# Practical No 5

**Title:** Black Box Testing: Decision table and Cause Effect Graphing

**Problem Statement:**

An insurance agency has the following norms fixed to provide premium for its policy holders:

1. If age<=25 and no claim has been made, premium increase will be $50, else $25.
2. If age <=25 and number of claims made is one, premium increase will be $100, else $50.
3. If age <=25 and number of claims made is 2-4, premium increase will be $400, else $200.
4. If one or more claims are made, send warning letter. If the number of claims made is 5 or more, cancel policy.

Draw the decision table and cause effect graph for Insurance renewal.

**Solution:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Conditions** | **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7** | **R8** | **R9** | **R10** | **R11** | **R12** | **R13** | **R14** | **R15** | **R16** |
| Age<=25 | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T |
| No Claim | T | T | T | T | T | T | T | T | F | F | F | F | F | F | F | F |
| 1 Claim | T | T | T | T | F | F | F | F | T | T | T | T | F | F | F | F |
| 2-4 Claims | T | T | F | F | T | T | F | F | T | T | F | F | T | T | F | F |
| **>=5 Claims** | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F |
| **Causes** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Pr Inc $25 | X | X | X | X | X | X | X |  | X | X | X |  | X |  |  | X |
| Pr Inc $50 | X | X | X | X | X | X | X | T | X | X | X |  | X |  |  | X |
| Pr Inc $100 | X | X | X | X | X | X | X |  | X | X | X | T | X |  |  | X |
| Pr Inc $200 | X | X | X | X | X | X | X |  | X | X | X |  | X |  |  | X |
| Pr Inc $400 | X | X | X | X | X | X | X |  | X | X | X |  | X | T |  | X |
| Send Warning | X | X | X | X | X | X | X |  | X | X | X | T | X | T |  | X |
| Cancel Policy | X | X | X | X | X | X | X |  | X | X | X |  | X |  | T | X |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Conditions** | **R17** | **R18** | **R19** | **R20** | **R21** | **R22** | **R23** | **R24** | **R25** | **R26** | **R27** | **R28** | **R29** | **R30** | **R31** | **R32** |
| Age<=25 | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| No Claim | T | T | T | T | T | T | T | T | F | F | F | F | F | F | F | F |
| 1 Claim | T | T | T | T | F | F | F | F | T | T | T | T | F | F | F | F |
| 2-4 Claims | T | T | F | F | T | T | F | F | T | T | F | F | T | T | F | F |
| **>=5 Claims** | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F |
| **Causes** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Pr Inc $25 | X | X | X | X | X | X | X | T | X | X | X |  | X |  |  | X |
| Pr Inc $50 | X | X | X | X | X | X | X |  | X | X | X | T | X |  |  | X |
| Pr Inc $100 | X | X | X | X | X | X | X |  | X | X | X |  | X |  |  | X |
| Pr Inc $200 | X | X | X | X | X | X | X |  | X | X | X |  | X | T |  | X |
| Pr Inc $400 | X | X | X | X | X | X | X |  | X | X | X |  | X |  |  | X |
| Send Warning | X | X | X | X | X | X | X |  | X | X | X | T | X | T |  | X |
| Cancel Policy | X | X | X | X | X | X | X |  | X | X | X |  | X |  | T | X |

**Condensed decision table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of Claims | Insured Age | Premium Increase $ | Send Warning | Cancel |
| 0 | 25 | 50 | No | No |
| 0 | >25 | 25 | No | No |
| 1 | 25 | 100 | Yes | No |
| 1 | >25 | 50 | Yes | No |
| 2-4 | 25 | 400 | Yes | No |
| 2-4 | >25 | 200 | Yes | No |
| 5 or more | \* | 0 | No | Yes |

**Cause Effect Graph:**

**Myer Cause & Effect Graphing is done through the following steps:**

**Step – 1:** For a module, identify the input conditions (causes) and actions (effect).

**Step – 2:** Develop a cause-effect graph.

**Step – 3:** Transform cause-effect graph into a decision table.

**Step – 4:** Convert decision table rules to test cases. Each column of the decision table represents a test case.

Cause Effect

Age >

25

Age <=25

Pr. In. $25

Pr. In. $50

Pr. In. $100

Pr. In. $200

No Claim

Claim

1

2

-

4

claims

>=5

claims

Pr. In. $400

Send

Warning

Cancel Policy

ʌ

ʌ

ʌ

ʌ

ʌ

ʌ

**Conclusion:**

The decision table and cause effect graph for the given situation are drawn.

==========================================================================

**Practical No 6.**

**Title:** Branch – Decision – Condition Coverage

**Problem Statement:**

For the following liability procedure, design test cases using branch, condition, decision and multiple decision coverage.

Procedure Liability(Age,Gender, Married,Premium)

Begin

Premium:=500;

If(Age<25) and (Gender=Male) and (not Married) Then

Premium=Premium+1500;

Else (if (Married or Gender=Female)) Then

Premium=Premium-200; if (Age>45) and (Age<65) Then

Premium=Premium-100;

End;

**Solution:**

**Branch or decision coverage:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decision Coverage | Age | Gender | Married | Test Case |
| IF-1 | <25 | Male | False | (1) 23, M F |
| IF-1 | <25 | Female | False | (2) 23, F F |
| IF-2 | \* | Female | \* | (2) |
| IF-2 | >=25 | Male | False | (3) 50 M F |
| IF-3 | <=45 | Female | \* | (2) |
| IF-3 | >45, <65 | \* | \* | (3) |

**Condition Coverage:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Condition Coverage | Age | Gender | Married | Test Case |
| IF-1 | <25 | Female | False | (1) 23, F F |
| IF-1 | >=25 | Male | True | (2) 30 M T |
| IF-2 | \* | Male | True | (2) |
| IF-3 | <=45 | \* | \* | (1) |
| IF-3 | >45 | \* | \* | (3) 70 F F |
| IF-3 | <65 | \* | \* | (2) |
| IF-3 | >=65 | \* | \* | (3) |

**Decision/Condition Coverage**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decision/Condition Coverage | Age | Gender | Married | Test Case |
| IF-1 (Decision) | <25 | Male | False | (1) 23 M F |
| IF-1 (Decision) | <25 | Female | False | (2) 23 F F |
| IF-1 (Condition) | <25 | Female | False | (2) |
| IF-1 (Condition) | >=25 | Male | True | (3) 70 M T |
| IF-2 (Decision) | \* | Female | \* | (2) |
| IF-2 (Decision) | >=25 | Male | False | (4) 50 M F |
| IF-2 (Condition) | \* | Male | True | (3) |
| IF-2 (Condition) | \* | Female | False | (2) |
| IF-3 (Decision) | <=45 | \* | \* | (2) |
| IF-3 (Decision) | >45, <65 | \* | \* | (4) |
| IF-3 (Condition) | <=45 | \* | \* | (2) |
| IF-3 (Condition) | >45 | \* | \* | (4) |
| IF-3 (Condition) | <65 | \* | \* | (4) |
| IF-3 (Condition) | >=65 | \* | \* | (3) |

**Multiple Condition Coverage:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decision/Condition Coverage | Age | Gender | Married | Test Case |
| IF-1 | <25 | Male | True | (1) |
| IF-1 | <25 | Male | False | (2) |
| IF-1 | <25 | Female | True | (3) |
| IF-1 | <25 | Female | False | (4) |
| IF-1 | >=25 | Male | True | (5) |
| IF-1 | >=25 | Male | False | (6) |
| IF-1 | >=25 | Female | True | (7) |
| IF-1 | >=25 | Female | False | (8) |
| IF-2 | \* | Male | True | (5) |
| IF-2 | \* | Male | False | (6) |
| IF-2 | \* | Female | True | (7) |
| IF-2 | \* | Female | False | (8) |
| IF-3 | <=45, >=65 | \* | \* | Impossible |
| IF-3 | <=45, <65 | \* | \* | (8) |
| IF-3 | >45,>65 | \* | \* | (6) |
| IF-3 | >45,<65 | \* | \* | (7) |

**Conclusion:**

All the types of test cases are designed for testing.

# Practical No 7

**Title:** State Transition Testing

**Problem Statement:**

Specifications:

The software responds to input requests to change the display mode for a time display device.

The display mode can be set to one of the four values:

Two corresponding to displaying either time or date.

The other two when altering either time or date.

Four possible input requests:

Change mode (CM)

Reset (R)

Time Set (TS)

Date Set (DS)

**State Transition Diagram:**

**Solution:**

State Transition Diagram:

Test cases are initially derived from the state transition diagram to exercise each of the possible transitions (using the abbreviated STD labels):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case | 1 | 2 | 3 | 4 | 5 | 6 |
| Start State | S1 | S1 | S3 | S2 | S2 | S4 |
| Input | CM | R | TS | CM | R | DS |
| Expected Output | D | AT | T | T | AD | D |
| Finish State | S2 | S3 | S1 | S1 | S4 | S2 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Start State | S1 | S1 | S1 | S3 | S3 | S2 | S2 | S2 | S4 | S4 |
| Input | CM | CM | R | TS | TS | CM | CM | R | DS | DS |
| Exp.Output | D | D | AT | T | T | T | T | AD | D | D |
| Next State | S2 | S2 | S3 | S1 | S1 | S1 | S1 | S4 | S2 | S2 |
| Input | CM | R | TS | CM | R | CM | R | DS | CM | R |
| Exp.Output | T | AD | T | D | AT | D | AT | D | T | AD |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Input 1 | Input 2 | etc. |
| Start State 1 | Entry A | Entry B | etc. |
| Start State 2 | Entry C | Entry D | etc. |
| etc. | etc. | etc. | etc. |

where Entry X = Finish State / Output for the given start state and input.

The state table for the *manage\_display\_changes* component is shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CM | R | TS | DS |
| S1 | S2/D | S3/AT | S1/N | S1/N |
| S2 | S1/T | S4/AD | S2/N | S2/N |
| S3 | S3/N | S3/N | S1/T | S3/N |
| S4 | S4/N | S4/N | S4/N | S2/D |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CM | R | TS | DS |
| S1 | S2/D (Test Case 1) | S3/AT (Test Case 2) | S1/N (Test Case 3) | S1/N (Test Case 4) |
| S2 | S1/T (Test Case 5) | S4/AD  (Test Case 6) | S2/N (Test Case 7) | S2/N (Test Case 8) |
| S3 | S3/N (Test Case 9) | S3/N (Test Case 10) | S1/T (Test Case 11) | S3/N (Test Case 12) |
| S4 | S4/N (Test Case 13) | S4/N (Test Case 14) | S4/N (Test Case 15) | S2/D (Test Case 16) |

which corresponds to:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case | 1 | 2 | 3 | 4 | 5 | .... | . . | .... | 12 | 13 | 14 | 15 | 16 |
| Start State | S1 | S1 | S1 | S1 | S2 | .... | . . | .... | S3 | S4 | S4 | S4 | S4 |
| Input | CM | R | TS | DS | CM | .... | . . | .... | DS | CM | R | TS | DS |
| Exp.Output | D | AT | N | N | T | .... | . . | .... | N | N | N | N | D |
| Finish State | S2 | S3 | S1 | S1 | S1 | .... | . . | .... | S3 | S4 | S4 | S4 | S2 |

**Conclusion:**

The state transition table is drawn and the test cases are developed.

# Practical No 8

**Title:** Data Flow Testing

**Problem Statement:**

1. void quad\_eqn (float A,B,C, Boolean Is\_Complex)
2. { float Discrim = B\*B-4\*A\*C
3. float R1,R2;
4. {
5. If Discrim<0.0
6. Is\_Complex=true;
7. Else
8. Is\_Complex=false
9. Endif;
10. If not Is\_Complex
11. R1=(-B+Sqrt(Discrim))/(2.0\*A);
12. R2=(-B-Sqrt(Discrim))/(2.0\*A);
13. Endif;
14. End quad\_eqn;}
15. }

**Solution:**

**Step 1: Number the lines:**

1. void quad\_eqn (float A,B,C, Boolean Is\_Complex)
2. { float Discrim = B\*B-4\*A\*C
3. float R1,R2;
4. {
5. If Discrim<0.0
6. Is\_Complex=true;
7. Else
8. Is\_Complex=false
9. Endif;
10. If not Is\_Complex
11. R1=(-B+Sqrt(Discrim))/(2.0\*A);
12. R2=(-B-Sqrt(Discrim))/(2.0\*A);
13. Endif;
14. End quad\_eqn;}
15. }

**Step 2: List the variables:**

* + A, B, C
  + DISCRIM
  + Is\_Complex
  + R1, R2

**Step 3: List occurrences and assign a category to each variable**

|  |  |  |  |
| --- | --- | --- | --- |
| **Line No** |  | **Category** |  |
|  | **Definition** | **c-use** | **p-use** |
| 0 | A,B,C |  |  |
| 1 | DISCRIM | A,B,C |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  | DISCRIM |
| 5 | Is\_Complex |  |  |
| 6 |  |  |  |
| 7 | Is\_Complex |  |  |
| 8 |  |  |  |
| 9 |  |  | Is\_Complex |
| 10 | R1 | A,B,DISCRIM |  |
| 11 | R2 | A,B,DISCRIM |  |
| 12 |  |  |  |
| 13 |  |  |  |
| 14 |  |  |  |

**Step 4: Identify du-pairs and their use (p- or c- )**

|  |  |  |
| --- | --- | --- |
| **Definition – use pair**  **Start line**  **end line** | **Variables** | |
| **c-use** | **p-use** |
| 0  1 | A,B,C |  |
| 0  10 | A,B |  |
| 0  11 | A,B |  |
| 1  4 |  | DISCRIM |
| 1  10 | DISCRIM |  |
| 1  11 | DISCRIM |  |
| 5  9 |  | Is\_Complex |
| 7  9 |  | Is\_Complex |
| 10  14 | R1 |  |
| 11  14 | R2 |  |

**Step 5: Define Test Cases:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **D cuse pairs** | **Variables** | **Subpaths** | **Test Case** | **Inputs** | | | **Is\_Co mplex** | **R1** | **R2** |
| A | B | C |
| 01 | A,B,C | 0-1 | 1 | 1 | 1 | 1 | T | . | . |
| 010 | A,B | 0-1-2-3-4-6-7-8-9-10 | 2 | 1 | 2 | 1 | F | -1 | -1 |
| 0-11 | A,B | 0-1-2-3-4-6-7-8-9-  10-11 | 3 | 1 | 2 | 1 | F | -1 | -1 |
| 1-10 | DISCRIM | 1-2-3-4-6-7-8-9-10 | 4 | 1 | 2 | 1 | F | -1 | -1 |
| 1-11 | DISCRIM | 1-2-3-4-6-7-8-9-10-  11 | 5 | 1 | 2 | 1 | F | -1 | -1 |
| 5-14 | Is\_Complex | 5-8-9-12-13-14 | 6 | 1 | 1 | 1 | T | . | . |
| 7-14 | Is\_Complex | 7-8-9-10-11-12-13-  14 | 7 | 1 | 2 | 1 | F | -1 | -1 |
| 10-14 | R1 | 10-11-12-13-14 | 8 | 1 | 2 | 1 | F | -1 | -1 |
| 11-14 | R2 | 11-12-13-14 | 9 | 1 | 2 | 1 | F | -1 | -1 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **d p-use pairs** | **Variable s** | **Subpaths** | **Test Case** | **I** | **nputs** | | **Is\_Co mplex** | **R1** | **R2** |
| A | B | C |
| 1-(4-5) | Discrim | 1-2-3-4-5 | 10 | 1 | 1 | 1 | T | . | . |
| 1-(4-6) | Discrim | 1-2-3-4-6 | 11 | 1 | 2 | 1 | F | -1 | -1 |
| 5-(9-12) | Is\_Comp  lex | 5-8-9-12 | 12 | 1 | 1 | 1 | T | . | . |
| 7-(9-10) | Is\_Comp  lex | 7-8-9-10 | 13 | 1 | 2 | 1 | F | -1 | -1 |

**Conclusion:**

13 test cases were designed for data flow testing.

**==========================================================================**

# Practical No 9

**Title:** Structured Testing – Loop Coverage, Call coverage and Path Coverage.

**Problem Statement:**

For the following program, draw the path coverage diagram, determine cyclomatic complexity write the basis paths to be tested and the test cases.

Euclid(int m, int n)

int r; if (n > m){ r = m; m = n; n = r;

}

r = m % n; while( r! = 0){

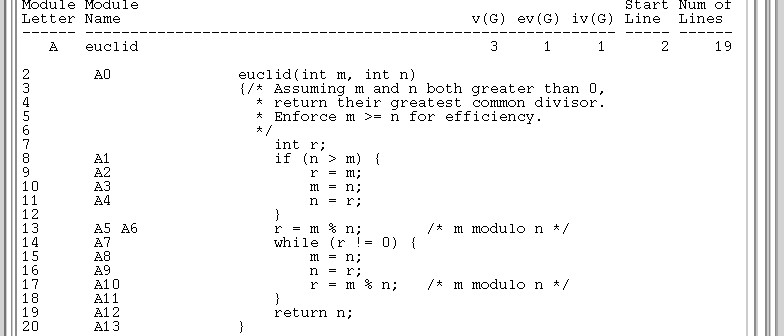
m = n; n = r; r = m % n;

}

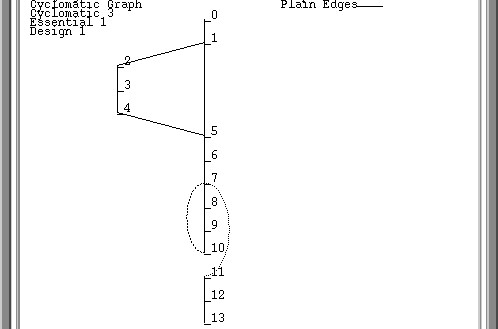
return n;

}

**Solution:**



Flow graph:



**Cyclomatic Complexity:**

V(G) = E – N + 2

= 15 – 14 + 2

= 3

**Basis Test Path:**

Basis Test Paths: 3 Paths

Test Path B1: 0 1 5 6 7 11 12 13

8( 1): n>m ==> FALSE

14( 7): r!=0 ==> FALSE

Test Path B2: 0 1 2 3 4 5 6 7 11 12 13

8( 1): n>m ==> TRUE

14( 7): r!=0 ==> FALSE

Test Path B3: 0 1 5 6 7 8 9 10 7 11 12 13

8( 1): n>m ==> FALSE

14( 7): r!=0 ==> TRUE

14( 7): r!=0 ==> FALSE

**Conclusion:**

The control flow graph is drawn and the cyclomatic complexity is determined to be 3. Three basis paths are to be tested.

# Practical No 10

**Title:** Test Automation using Selenium IDE

**Problem Statement:**

Automate web test using Selenium IDE and Firebug.

:

**Step 1:**

* Launch Firefox and Selenium IDE.
* Type the value for our Base URL: http://newtours.demoaut.com/.
* Toggle the Record button on (if it is not yet toggled on by default).

**Step 2:**

* In Firefox, navigate to http://newtours.demoaut.com/. Firefox should take you to the page similar to the one shown below.

**Step 3:**

* Right-click on any blank space within the page, like on the Mercury Tours logo on the upper left corner. This will bring up the Selenium IDE context menu. Note: Do not click on any hyperlinked objects or images • Select the “Show Available Commands” option. • Then, select “assertTitle exact:Welcome: Mercury Tours”. This is a command that makes sure that the page title is correct.

**Step 4:**

* In the “User Name” text box of Mercury Tours, type an invalid username, “invalidUN”.
* In the “Password” text box, type an invalid password, “invalidPW”.

**Step 5:**

**Step 6:**

* Toggle the record button off to stop recording. Your script should now look like the one shown below.

**Step 7:**

* Now that we are done with our test script, we shall save it in a test case. In the

File menu, select “Save Test Case”. Alternatively, you can simply press Ctrl+S.

**Step 8:**

* Choose your desired location, and then name the test case as “Invalid\_login”.
* Click the “Save” button.

**Step 9:**

* Notice that the file was saved as HTML.

**Step 10:**

* Go back to Selenium IDE and click the Playback button to execute the whole script. Selenium IDE should be able to replicate everything flawlessly.