# Practical 1

## Analyse the Medical management system and prepare functional and non- functional requirements.

**Introduction**

The project medical Management system includes registration of medicine, storing their details into the system, and also computerized billing in the pharmacy, and labs. The software has the facility to give a unique id for every client and stores the details of every client and the staff automatically. It includes a search facility to know the current status of each medicine. Users can search the availability of a medicine and the details of a medicine using the id. The medicine Management System can be entered using a username and password. It is accessible either by an administrator or staff. Only they can add data into the database. The data can be retrieved easily. The interface is very user-friendly.

## Functional Requirement

1. **client Management**
   * **Registration**: Capture client demographics, contact information, and medicine details.
   * **Medicine Scheduling**: Allow client to purchase, reorder, and cancel order with medicine.
   * **order History**: Maintain a comprehensive medical history, including past treatments and allergies.

## Staff Management

* + **Employee Records**: Store information on admin, staff, and client, including roles, schedules, and credentials.
  + **Scheduling**: Manage staff shifts and availability.
  + **Performance Tracking**: Monitor staff performance through evaluations and feedback.

## User Interface

* + **User-Friendly Design**: Provide an intuitive interface for both client and staff.
  + **Mobile Access**: Allow access to the system through mobile devices for staff and patients.

**Attendance**

**Client Help Desk**

**Electronic Medical Record**

## Non-Functional Requirements

### Performance

The system should handle up to 500 concurrent users without performance degradation.

Response time for all user actions should be under 2 seconds.

### Security

Data encryption for sensitive client information.

Regular security audits to identify and mitigate vulnerabilities.

### Usability

User-friendly interface with intuitive navigation. Accessibility features to assist users with disabilities. **Scalability**

The system must be scalable to accommodate increasing numbers of users and data.

Support for future modules (e.g., telemedicine) without major redesign.

# Practical 2

## Analyse the medical managment and prepare User story and Acceptance criteria.

**User stories**

### User story-1

As a customer, I want to be able to order with a medicine of my choice, so that I can receive timely medical consultation.

### User story-2

As a customer, I want to be to receive reminders about my upcoming order via SMS or email, so that I don’t miss them.

### User story-3

As a customer, I want to be able to view my lab test results online, so that I don’t have to visit the lab to collect the report.

**Acceptance criteria:**

|  |  |
| --- | --- |
| **Functionality** | client management |
| **Story** | As a client,  I want to register into an App,  So that I can do appointment and do check reports. |
| **Scenario 1.1** | client can register based on the correct e-mail and password. |

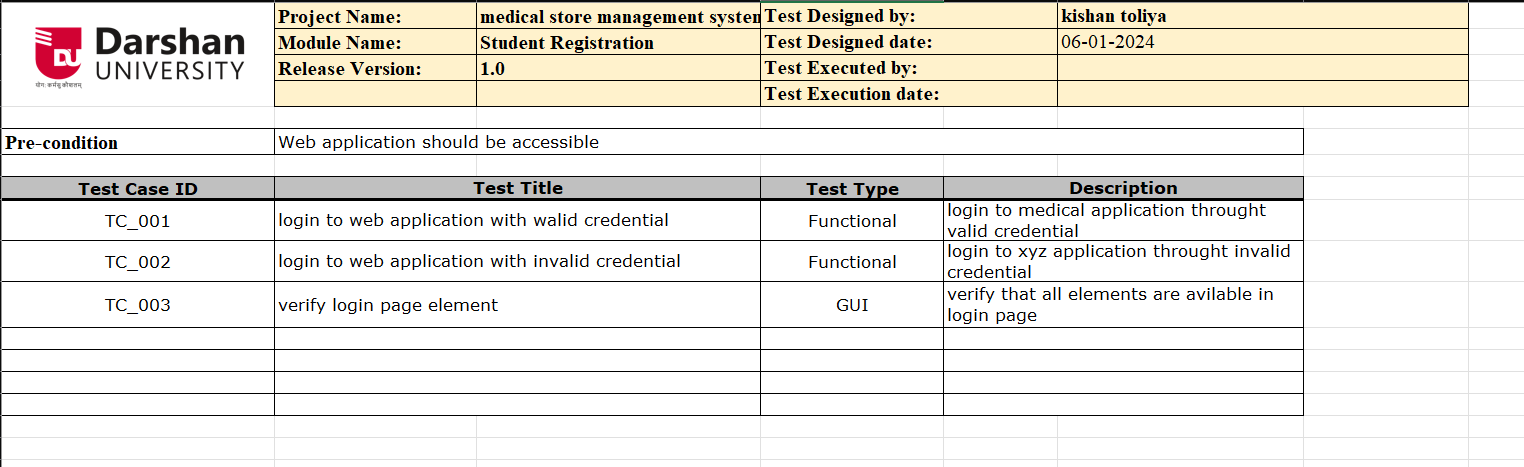
|  |  |
| --- | --- |
| **Acceptance criteria** | Given: client navigate to the registration page.  When: User can enter correct e-mail and password and then press on the login button.  Then: user navigate to the app. Display order & scheduling Display Reporting & Analytics  Display Prescription |

|  |  |
| --- | --- |
| **Functionality** | Staff management |
| **Story** | As a staff,  I want to regularly see my attendance,  So I can manage my attendance efficiently . |
| **Scenario 1.2** | The medical is preparing for an influx of client due to a local health crisis, requiring precise management of staff schedules and  performance monitoring. |
| **Acceptance criteria** | **Given:** Navigated to upload attendance  **When: staff open a software and check it**  **Then:** staff improved a attendance and available maximum in hospital. |

# Practical 3

## Write a Test case and test scenario.

Test case list



Tc-2

A screenshot of a computer

Description automatically generated

**Tc-3**

A screenshot of a computer

Description automatically generated

**Bug report**

A screenshot of a computer

Description automatically generated

# Practical 4

**Implement Equivalence Partitions**

Equivalence partitioning is based on the principle that if one test case in a specific category detects a defect, it is likely that other test cases within the same category will also reveal the same defect. It involves dividing the input space of a software system into classes or partitions, where each partition is expected to exhibit similar behaviour.

* 1. Let's consider a simple login form that accepts a username and password. The requirements for the login form are as follows:
     + The username field must be alphanumeric and have a length between 5 and 10 characters.
     + The password field must be alphanumeric and have a length between 8 and 12 characters.

Based on these requirements, create equivalence partitions for each input field.

## Solution

### For username:

#### Valid Class:

* The name of the username must be more than 5 character and less than
* 10 character and contain alphanumeric such as numbers and alphabets.

#### Invalid Class:

* The username not contain space and symbols .
* Username not to long (more than 10 characters).
* Username not to short (less than 5 characters).

### For password:

#### Valid Class:

* Password must be more than 8 and less than 12 character and
* contain alphanumeric password ,with special symbol

#### Invalid Class:

* Not all characters are small character
* Not all characters are capital character
* Not allow without special character
* Not allow too short (less than 7) and too long (less than 12)
  1. ​

Let us consider an example of any college admission process. There is a college that gives admissions to students based upon their percentage. Consider percentage field that will accept percentage only between 50 to 90 %, more and even less than not be accepted, and application will redirect user to an error page.

* + - If percentage entered by user is less than 50 %or more than 90 %, that equivalence partitioning method will show an invalid percentage.
    - If percentage entered is between 50 to 90 %, then equivalence partitioning method will show valid percentage

### Solution .

#### Valid Class:

The entered percentage must be more than 50% and less than 90%

* + - * **Invalid Class 1:** Percentage not be equal to or less than 50%
      * **Invalid Class 2:** Percentage not be equal to or more than 90%
  1. Let us consider an example of software application. There is function of software application that accepts only particular number of digits, not even greater or less than that particular number. Consider an OTP number that contains only 6-digit number, greater and even less than six digits will not be accepted, and the application will redirect customer or user to error page.
     + If password entered by user is less or more than six characters, that equivalence partitioning method will show an invalid OTP.
     + If password entered is exactly six characters, then equivalence partitioning method will show valid OTP.

### Solution .

**Valid Class:** OTP must be contain exact 6 digits

* + - * **Invalid Class 1:** Not allow characters or symbol
      * **Invalid Class 2:** Nor allow more or less than 6 characters
  1. Testing an input box for a mobile number accepting ten digits (i.e. length of input value has to be ten).

### Solution .

#### Valid Class:

Mobile number must be exact 10 digits

* + - **Invalid Class 1:** Not allow more than or less than 10 digits
    - **Invalid Class 2 :** Not allow special character or alphabets
  1. Consider this scenario: For a bank savings account, a 4% interest rate is given if the account balance is between $01 and $200, 7% rate of interest is given if the account balance is in the range of $201 to 900, and 9% interest rate is considered for account balance is $901 & above. For the above scenario, we can identify three valid equivalence classes and one invalid.

Test cases for the above (interest rate) scenario:

#### Test cases for the above (interest rate) scenario:

* + - **Invalid data class**: balance less than 1.
    - **Valid data class 1**: balance between 1 to 200.
    - **Valid data class 2**: balance between 201 to 900.
    - **Valid data class 3:** balance greater than 901.

# Practical 5

## Implement Boundary-Value Analysis

* 1. Let's consider a system that accepts Age 18 – 56. The requirements for the system are as follows:
     + Minimum boundary value is 18
     + Maximum boundary value is 56
     + Valid Inputs: 18,19,55,56 and Invalid Inputs: 17 and 57

**Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of the age values.**

### Solution:

|  |  |  |
| --- | --- | --- |
| BOUNDARY VALUE TEST CASE | | |
| **INVALID TEST CASE**  (Min Value – 1) | **VALID TEST CASES**  (Min, +Min, Max, -Max) | **INVALID TEST CASE**  (Max Value + 1) |
| 17 | 18, 19, 55, 56 | 57 |

* Test case 1: Enter the value 17 (18-1) = Invalid
* Test case 2: Enter the value 18 = Valid
* Test case 3: Enter the value 19 (18+1) = Valid
* Test case 4: Enter the value 55 (56-1) = Valid
* Test case 5: Enter the value 56 = Valid
* Test case 6: Enter the value 57 (56+1) = Invalid
  1. Let's consider a system that have to test a text field (Name) which accepts the length between 6-12 characters.
     + Minimum boundary value is 6
     + Maximum boundary value is 12
     + Valid text length is 6, 7, 11, 12
     + Invalid text length is 5, 13

### Solution:

|  |  |  |
| --- | --- | --- |
| BOUNDARY VALUE TEST CASE | | |
| **INVALID TEST CASE**  (Min Value – 1) | **VALID TEST CASES**  (Min, +Min, Max, -Max) | **INVALID TEST CASE**  (Max Value + 1) |
| 5 | 6,7,11,12 | 13 |

* Test case 1: Enter the value 5(6-1) – Invalid
* Test case 2: Enter the value 6 – valid
* Test case 3: Enter the value 7 – valid
* Test case 4: Enter the value 11 – valid
* Test case 5: Enter the value 12 – valid
* Test case 6: Enter the value 13(12+1) – Invalid
  1. Let's consider a system that calculates the shipping cost based on the weight of a package. The requirements for the system are as follows:
     + Weight must be a positive numeric value.
     + The shipping cost is determined based on weight ranges: 0-10 kg, 11-20 kg, and 21-30 kg.
     + The shipping cost for each weight range is $10, $15, and $20, respectively.

### Solution

|  |  |  |
| --- | --- | --- |
| BOUNDARY VALUE TEST CASE | | |
| **INVALID TEST CASE**  (Min Value – 1) | **VALID TEST CASES**  (Min, +Min, Max, -Max) | **INVALID TEST CASE**  (Max Value + 1) |
| -1 | 0,1,9,10 | 11 |
| 10 | 11,12,19,20 | 21 |
| 20 | 21,22,29,30 | 31 |

Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of the weight ranges.

**Test Cases: For Shipping cost $10:**

* enter value -1 = (0-1) = invalid testcase
* enter value 0 = valid testcase
* enter value 1 = (0+1) = valid testcase
* enter value 9 = (10-1) = valid testcase
* enter value 10 = valid testcase
* enter value 11 = (10+1) = invalid testcase

**Test Cases: For Shipping cost $20**

* enter value 10 = (11-1) = invalid testcase
* enter value 11 = valid testcase
* enter value 12 = (11+1) = valid testcase
* enter value 19 = (20-1) = valid testcase
* enter value 20 = valid testcase
* enter value 21 = (20+1) = invalid testcase

**Test Cases: For Shipping cost $30**

* enter value 20 = (21-1) = invalid testcase
* enter value 21 = valid testcase
* enter value 22 = (21+1) = valid testcase
* enter value 29 = (30-1) = valid testcase
* enter value 30 = valid testcase
* enter value 31 = (30+1) = invalid testcase

5.2 Let us assume there is a test case that takes the car speed from 40 km/hr- 80 km/hr, and if the speed is lower than 40 and higher than 80, then our software should send the notification.

* If the values are entered below 40 and above 80 then it will be considered invalid case.
* If the value entered is between 40-80 then it will be a valid case.

**Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of car speed.**

### Solution

|  |  |  |
| --- | --- | --- |
| BOUNDARY VALUE TEST CASE | | |
| **INVALID TEST CASE**  (Min Value – 1) | **VALID TEST CASES**  (Min, +Min, Max, -Max) | **INVALID TEST CASE**  (Max Value + 1) |

|  |  |  |
| --- | --- | --- |
| 39 | 40,41,79,80 | 81 |

#### Test Cases:

* enter car speed 39 = (40-1) = invalid testcase
  + enter car speed 40 = valid testcase
  + enter car speed 41 = (40+1) = valid testcase
  + enter car speed 79 = (80-1) = valid testcase
  + enter car speed 80 = valid testcase
  + enter car speed 81 = (80+1) = invalid testcase

5.3 Consider the Order Pizza Text Box's behaviour. Pizza values ranging from 1 to 10 are deemed legitimate. The message "success" is shown. While values 11 to 99 are deemed invalid for ordering, an error notice stating "Only 10 Pizza may be ordered" will show.

* Any number submitted in the Order Pizza form that is larger than 10 (let's say 11) is deemed invalid.
* Any number that is less than 1 and is 0 or below is deemed invalid.
* The numbers 1 to 10 are accepted as acceptable.
* Any three-digit number, such as -100, is unusable.

**Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of pizza quantity.**

### Solution

|  |  |  |
| --- | --- | --- |
| BOUNDARY VALUE TEST CASE | | |
| **INVALID TEST CASE**  (Min Value – 1) | **VALID TEST CASES**  (Min, +Min, Max, -Max) | **INVALID TEST CASE**  (Max Value + 1) |
| 0 | 1,2,9,10 | 11 |

**Test Cases:**

* enter value 0 = (1-1) = invalid testcase
* enter value 1 = valid testcase
* enter value 2 = (1+1) = valid testcase
* enter value 9 = (10-1) = valid testcase
* enter value 10 = valid testcase
* enter value 11 = (10+1) = invalid testcase

# Practical 6

## Understanding Decision Table Testing

* 1. **Based on** Let's consider a simple login screen of any website. So create a decision table for a login functionality.
     + If the user provides the correct username and password the user will be redirected to the homepage.
     + If any of the input is wrong, an error message will be displayed.

**these requirements, create a decision table to systematically test all possible combinations.**

## Solution

Decision table for login functionality

|  |  |  |  |
| --- | --- | --- | --- |
| **Conditions** | **User name** | **Password** | **Output/Outcome** |
| Case 1 | False | False | Error Message |
| Case 2 | True | False | Error Message |
| Case 3 | False | True | Error Message |
| Case 4 | True | True | Navigate to home page |

**Test Case 1** – Username and password both are wrong. The user is shown an error message.

**Test Case 2** – Username was correct, but the password was wrong. The user is shown an error message.

**Test Case 3** – Username was wrong, but the password was correct. The user is shown an error message.

**Test Case 4** – Username and password both were correct, and the user navigated to the home Screen

* 1. Let’s consider a dialogue box that will ask the user to upload a photo with certain conditions. The requirements for the photo upload are as follows:
     + You can upload only ‘.jpg’ format image
     + file size less than 32kb
     + resolution 137\*177.
     + If any of the conditions fails the system will throw a corresponding error message stating the issue and if all conditions are met photo will be updated successfully

**Based on these requirements, create a decision table to systematically test all possible combinations.**

### Solution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Condition** | **.jpg**  **format** | **Size <**  **32kb** | **Resolution137\*177** | **output** |
| Case 1 | T | T | T | Move to next page |
| Case 2 | T | T | F | Invalid resolution |
| Case 3 | T | F | T | Invalid photo size |
| Case 4 | F | T | T | Invalid photo format |
| Case 5 | T | F | F | Invalid size and resolution |
| Case 6 | F | T | F | Invalid size and format |
| Case 7 | F | F | T | Invalid format and size |
| Case 8 | F | F | F | Invalid data entered |

**Test case 1:** Photo .jpg format , size is less 32 kb and resolution 137\*177 all condition are true then move to next page.

**Test case 2:** Resolution is more than 137\*177 then image cannot be upload and give an error message .

**Test case 3:** Size is more than 32kb then image cannot be upload and give an error message.

**Test case 4:** Photo is not in .jpg format then image cannot be upload and give an error message.

**Test case 5:** Both size and resolution are invalid then image cannot be upload and give an error message.

**Test case 6:** Both format and resolution are invalid then image cannot be upload and give an error message.

**Test case 7:** Both size and format are invalid then image cannot be upload and give an error message.

**Test case 8:** All input data are invalid then image cannot be upload and give an error message.

* 1. Let’s consider a piece of software determining whether a student is eligible for a scholarship based on their grades and extracurricular activities. The requirements are as follows:
     + If percentage is greater than 90% and extracurricular activities are greater than 2 than only scholarship will be given.
     + If both the condition are true than and only than scholarship will be given, otherwise not given.

**Based on these requirements, create a decision table to systematically test all possible combinations.**

### Solution

|  |  |  |  |
| --- | --- | --- | --- |
| **Condition** | **Per>90** | **Extra activity>2** | **Output** |
| Case 1 | T | T | Scholarship granted |
| Case 2 | F | T | Percentage less than 90 |

|  |  |  |  |
| --- | --- | --- | --- |
| Case 3 | T | F | Extra activity less than 2 |
| Case 4 | F | F | Both Percentage and activity  less |

**Test case 1:** Percentage is greater than 90% and extracurricular activities are greater than 2 , then student get scholarship.

**Test case 2:** Percentage is not greater than 90% then student cant get scholarship.

**Test case 3:** Extra activity is not more than 2 then student can’t get scholarship.

**Test case 4:** Both Percentage and Extracurricular are unsufficient then student can’t get scholarship.

* 1. Let's consider a simple e-commerce system that calculates discounts based on the customer type and the total order amount. The requirements for the discount calculation are as follows:

1. If the customer is a regular customer:
   * If the total order amount is less than $100, no discount is applied.
   * If the total order amount is between $100 and $500 (inclusive), a 10% discount is applied.
   * If the total order amount is greater than $500, a 15% discount is applied.
2. If the customer is a premium customer:
   * If the total order amount is less than $100, a 5% discount is applied.
   * If the total order amount is between $100 and $500 (inclusive), a 15% discount is applied.
   * If the total order amount is greater than $500, a 20% discount is applied.

**Based on these requirements, create a decision table to systematically test all possible combinations**

### Solution

Decision table for item discount

|  |  |  |  |
| --- | --- | --- | --- |
| **Condition** | **Customer type** | **Order amount** | **Discount (%)** |
| Case 1: regular, <$100 | Regular | < $100 | 0 |
| Case 2: regular, $100 - $500 | Regular | $100 - $500 | 10 |
| Case 3: regular, >$500 | Regular | >$500 | 15 |
| Case 4: premium, <$100 | Premium | < $100 | 5 |
| Case 5: premium, $100 - $500 | Premium | $100 - $500 | 15 |
| Case 6: premium, >$500 | Premium | >$500 | 20 |

#### Test case 1:

The customer is a regular member and the total amount is less than $100, the outcome will be a 0% discount on the total amount in the shopping cart.

#### Test case 2:

The customer is a regular member and the total amount is between $100 - $500, the outcome will be a 10% discount on the total amount in the shopping cart.

#### Test case 3:

The customer is a regular member and the total amount is greater than $500, the outcome will be a 15% discount on the total amount in the shopping cart.

#### Test case 4:

The customer is a premium member and the total amount is less than $100, the outcome will be a 5% discount on the total amount in the shopping cart.

#### Test case 5:

The customer is a premium member and the total amount is between $100 - $500, the outcome will be a 15% discount on the total amount in the shopping cart.

#### Test case 6:

The customer is a premium member and the total amount is greater than $500, the outcome will be a 20% discount on the total amount in the shopping cart.

## OR

Decision table for item discount

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TestcaseID** | **Cust\_type**  **Regular** | **Cust\_type**  **Premium** | **<$100** | **$100 -**  **$500** | **>$500** | **Discount**  **%** |
| Case 1 | T |  | T |  |  | 0 |
| Case 2 | T |  |  | T |  | 10 |
| Case 3 | T |  |  |  | T | 15 |
| Case 4 |  | T | T |  |  | 5 |
| Case 5 |  | T |  | T |  | 15 |
| Case 6 |  | T |  |  | T | 20 |

# Practical 7

## Draw a control flow diagram and apply cyclomatic complexity for the given codes. Be sure about following points.

* Guarantees that all independent execution path is exercised at least once.
* Guarantees that both the true and false side of all logical decisions are exercised.
* Executes the loop at the boundary values and within the boundaries.
* Identify numbers of independence path require for testing.

## Exercise 1:

void main(){

int i,j,k; readln (i,j,k);

if( (i < j) || ( i > k) ){

writeln(“then part”); if (j < k)

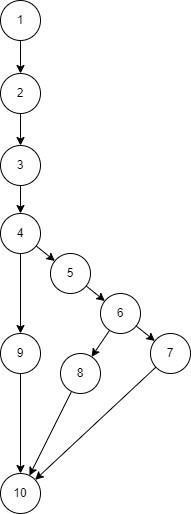
writeln (“j less then k”);

else writeln (“j not less than k”);} else writeln(“else Part”);

}

**Solution**

### Control flow graph:



**Cyclomatic complexity:**

N = No. of nodes = 10 E = No. of edges = 11

V(G) = E - N + 2 = 11 - 10 + 2 = 3

P=No. of predicate node= 3 V(G) = P+1=3+1=4

## Independent paths:

**Path 1** : 1 - 2 - 3 - 4 - 5 - 6 - 7 - 10

**Path 2** : 1 - 2 - 3 - 4 - 9 - 10

**Path 3** : 1 - 2 - 3 - 4 - 5 - 6 - 8 - 10

## Exercise 2:

i = 0;

n=4; //N-Number of nodes present in the graph while (i<n-1) do

j = i + 1;

while (j<n) do

if A[i]<A[j] then

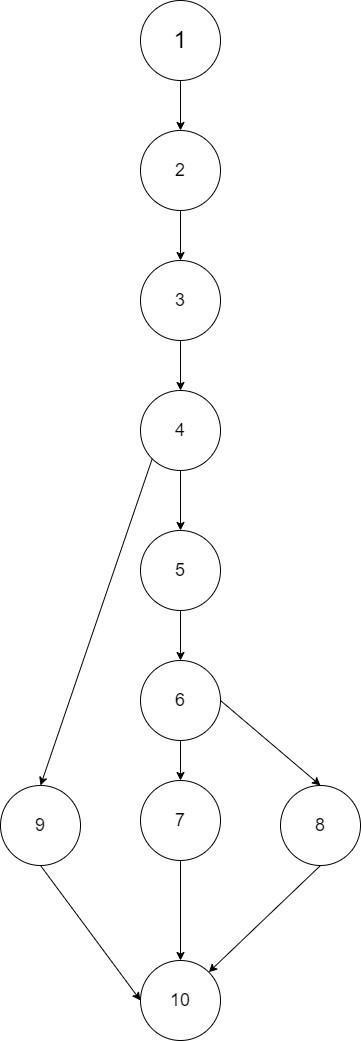
swap(A[i], A[j]);

end do; i=i+1;

end do;

**Solution**

### Control flow graph:



**Cyclomatic complexity:**

N = No. of nodes = 10 E = No. of edges = 12

V(G) = E - N + 2 = 12 - 10 + 2 = 4

P=No. of predicate node= 3 V(G) = P+1=3+1=4

### Independent paths:

**Path 1:** 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 3 - 10

**Path 2:** 1 - 2 - 3 - 4 - 5 - 6 - 5 - 8 - 9 - 3 - 10

**Path 3:** 1 - 2 - 3 - 4 - 5 - 8 - 9 - 3 - 10

**Path 4:** 1 - 2 - 3 - 10

public Hashtable countAlphabet(String aString)

{

Hashtable table = new Hashtable(); If (aString.length > 4000)

return table;

StringBuffer buffer = new StringBuffer(aString); While (buffer.length() > 0){

String firstChar = buffer.substring(0, 1); Integer count = (Integer)table.get(firstChar); if (count == null){

count = new Integer(1);

}

else{

count = new Integer(count.intValue() + 1);

}

table.put(firstChar, count); buffer.delete(0, 1);

}

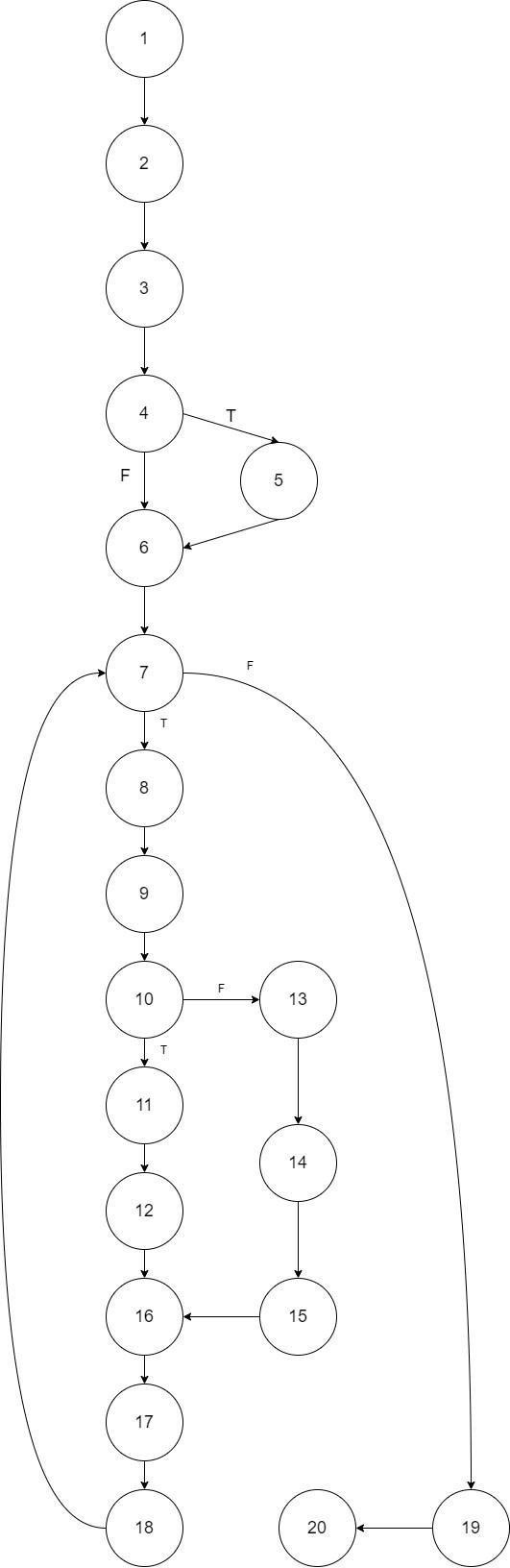
Return table;

}

## Exercise 3:

**Solution**

### Control flow graph:



**Cyclomatic complexity:**

N = No. of nodes = 20 E = No. of edges = 23

V(G) = E - N + 2 = 23 - 20 + 2 = 5

P=No. of predicate node= 3 V(G) = P+1=3+1=4

### Independent paths:

**Path 1:** 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 16 - 17 - 18

**Path 2:** 1 - 2 - 3 - 4 - 6 - 7 - 19 - 20

**Path 3:** 1 - 2 - 3 - 4 - 6 - 7 - 8 - 9 - 10 - 13 - 14 - 15 - 16 - 17 - 18

**Path 4:** 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 13 - 14 - 15 - 16 - 17 - 18

public class CyclomaticComplexityDemo { public static void main(String[] args) {

// TODO Auto-generated method stub int var1 = 10;

int var2 = 9; int var3 = 8; int var4 = 7;

if (var1 == 10){

if(var2 > var3){

var2 = var3;

}

else{

if (var3 > var4){

var3 = var4;

}

else{

var4 = var1;

}

}

}

else{ var1=var4;

}

System.out.println("Printing value for var1, var2, var3, and var4"+var1+" "+var2+" "+var3+" "+var4);

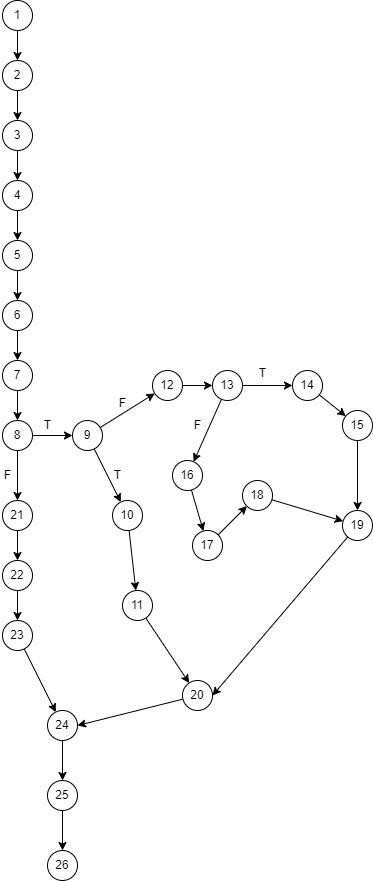
}

}

## Exercise 4:

**Solution**

### Control flow graph:



**Cyclomatic complexity:**

N = No. of nodes = 26 E = No. of edges = 29

V(G) = E - N + 2 = 29-25+ 2 = 5

P=No. of predicate node= 3 V(G) = P+1=3+1=4

### Independent paths:

**Path 1:** 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 20 - 24 - 25 - 26

**Path 2:** 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 12 - 13 - 14 - 15 - 19 - 20 - 24 - 25 - 26

**Path 3:** 1 - 2 – 3 - 4 - 5 - 6 - 7 - 8 - 9 - 12 - 13 - 16 - 17 - 18 - 19 - 20 - 24 - 25 - 26

**Path 4:** 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 21 - 22 - 23 - 24 - 25 - 26

# Practical 8

## To study and perform sample testing using Selenium Testing Tools. Answer following questions:

1. What is selenium? use of selenium?

### Answer:

* + Selenium is an open-source, automated testing tool used to test web applications across various browsers.
  + Selenium can only test web applications, unfortunately, so desktop and mobile apps can't be tested.
  + However, other tools like Appium and HP's QTP can be used to test software and mobile applications.
  + Selenium webdrive is a programming interface made up of different language building that can be used to create and execute test cases across all major programming language browser and operating system.

1. How it works?

### Answer:

* + Write a Test Script: You create a script (a set of instructions) in a programming language (like Python or Java) that tells Selenium what to do on a website.
  + Launch the Browser: When you run your script, Selenium opens a web browser (like Chrome or Firefox).
  + Perform Actions: The script instructs the browser to perform actions like:
  + Go to a specific website.
  + Click on buttons.
  + Enter text in forms.
  + Check Results: After performing actions, the script can check if everything worked as expected. For example, it might check if a success message appears on the page.
  + Close the Browser: Once the test is done, Selenium can close the browser automatically.

1. Advantages and disadvantages of selenium.

Answer:

Advantages of Selenium

* + Open Source: Free to use.
  + Multi-Browser Support: Works with all major browsers (Chrome, Firefox, etc.).
  + Multiple Language Support: Scripts can be written in Java, Python, C#, etc.
  + Integration Capabilities: Easily integrates with testing frameworks and CI/CD tools.
  + Cross-Platform: Runs on various operating systems (Windows, macOS, Linux).
  + Large Community Support: Extensive resources and documentation available.
  + Record and Playback: Tools like Selenium IDE help quickly create tests.

Disadvantages of Selenium

* + Steep Learning Curve: Challenging for beginners.
  + Limited to Web Applications: Not designed for mobile apps.
  + Dynamic Elements: Can be difficult to test dynamic content.
  + No Built-in Reporting: Requires integration with other tools for reports.
  + Maintenance Overhead: Tests may break with application changes.
  + Requires a Browser: Slower execution compared to headless testing.
  + Synchronization Issues: Managing timing can complicate tests.

# Practical 9

## To study and perform Unit testing using JUnit. Answer following questions:

1. What is Junit? Explain use of Junit.

### Answer:

JUnit is a popular open-source testing framework for Java. It provides annotations, assertions, and test runners to help developers write and run repeatable tests for their code. JUnit is an essential part of Test-Driven Development (TDD) and is widely used for unit testing in Java applications

* + **Unit tests**: Test individual pieces of code, such as a line, class, or method.
  + **Functional tests:** Test if a module or system can perform its intended functions.
  + **Integration tests**: Test how all the parts of a system work together.
  + **System tests:** Test entire systems, such as web servers.
  + **Regression tests**: Check if recent code changes negatively affect previously written code.
  + **Performance tests**: Benchmark software components to ensure they run fast enough under high load.

1. Explain all methods of assert class of JUnit.

### Answer:

The Assert class in JUnit is a fundamental component of unit testing in Java. It provides various methods to validate the expected results of a test case. Here are the different methods of the Assert class in JUnit:

#### assertEquals()

* + assertEquals(expected, actual): Verifies that the expected value is equal to the actual value.
  + assertEquals(expected, actual, message): Verifies that the expected value is equal to the actual value, and provides a custom error message if the assertion fails.

#### assertNotEquals()

* + assertNotEquals(unexpected, actual): Verifies that the unexpected value is not equal to the actual value.
  + assertNotEquals(unexpected, actual, message): Verifies that the unexpected value is not equal to the actual value, and provides a custom error message if the assertion fails

#### assertSame()

* + assertSame(expected, actual): Verifies that the expected object is the same as the actual object (i.e., they are the same instance).
  + assertSame(expected, actual, message): Verifies that the expected object is the same as the actual object, and provides a custom error message if the assertion fails.

#### assertNotSame()

* + assertNotSame(unexpected, actual): Verifies that the unexpected object is not the same as the actual object (i.e., they are not the same instance).
  + assertNotSame(unexpected, actual, message): Verifies that the unexpected object is not the same as the actual object, and provides a custom error message if the assertion fails.

#### assertTrue()

* + assertTrue(condition): Verifies that the given condition is true.
  + assertTrue(condition, message): Verifies that the given condition is true, and provides a custom error message if the assertion fails.

#### assertFalse()

* + assertFalse(condition): Verifies that the given condition is false.
  + assertFalse(condition, message): Verifies that the given condition is false, and provides a custom error message if the assertion fails.

#### assertNull()

* + assertNull(object): Verifies that the given object is null.
  + assertNull(object, message): Verifies that the given object is null, and provides a custom error message if the assertion fails.

#### assertNotNull()

* + assertNotNull(object): Verifies that the given object is not null.
  + assertNotNull(object, message): Verifies that the given object is not null, and provides a custom error message if the assertion fails.

#### assertArrayEquals()

* assertArrayEquals(expected, actual): Verifies that the expected array is equal to the actual array.
* assertArrayEquals(expected, actual, message): Verifies that the expected array is equal to the actual array, and provides a custom error message if the assertion fails.

1. Consider a simple Java class called **Calculator** that performs basic arithmetic operations. To perform unit testing using JUnit, you can create a test class and write test methods to verify the behaviour of the **addition() method**.

**Solution**

**//class calculator** import java.util.Scanner; public class calci {

public static void main(String[] args) { char operator;

int number1, number2;

calci c=new calci(); // create an object of Scanner class Scanner input = new Scanner(System.in);

System.out.println("Choose an operator: +, -, \*, or /"); // ask users to enter operator operator = input.next().charAt(0);

// ask users to enter numbers System.out.println("Enter first number"); number1 = input.nextInt(); System.out.println("Enter second number"); number2 = input.nextInt();

int x; if(operator=='+')

x=c.addition(number1,number2); else if(operator=='-') x=c.subtraction(number1,number2); else if(operator=='\*') x=c.multiplication(number1,number2); else

x=c.division(number1,number2); System.out.println("answer is:"+x);

}

int addition(int n1,int n2)

{

return n1+n2;

}

int subtraction(int n1,int n2)

{

return n1-n2;

}

int multiplication(int n1,int n2)

{

return n1\*n2;

}

int division(int n1,int n2)

{

return n1/n2;

}

}

#### //class test

import static org.junit.Assert.assertEquals; import org.junit.Before;

import org.junit.Test;

public class test { calci c;

@Before

public void setUp() { c=new calci();

}

@Test

public void testAdd\_Positive() {

int result = c.addition(5, 10); assertEquals(15, result);

}

@Test

public void testAdd\_PositiveNegative() { int result = c.addition(10, -5); assertEquals(4, result);

}

// Test case for addition of two negative numbers @Test

public void testAdd\_Negative() { int result = c.addition(-3, -7); assertEquals(-10, result);

}

// Test case for addition involving zero @Test

public void testAdd\_ZeroPositive() { int result = c.addition(0, 10); assertEquals(10, result);

}

/\*Test case for addition of large numbers \*/ @Test

public void testAdd\_LargeNumbers() {

int result = c.addition(10000000, 2000000); assertEquals(12000000, result);

}

}

1. Consider a Java class called **InterestCalculator** that calculates the interest amount based on a principal amount and an interest rate. To perform unit testing for the **InterestCalculator** class using JUnit, you can create a test class and write test methods to verify the behaviour of the **calculateInterest** method.

### Solution

#### //class Interest calculator

import java.util.Scanner;

public class simple\_interest {

public static void main(String[] args) { Scanner in = new Scanner(System.in);

System.out.println("Enter the principal (amount), time, and rate::\n"); float p = in.nextFloat(); // value of the principal

float t = in.nextFloat(); // value of the time float r = in.nextFloat(); // value of the rate CalculateSimpleInterest(p, t, r);

}

// This method will calculate the simple interest

public static void CalculateSimpleInterest(float x, float y, float z) { float SI; // Value of the simple interest

SI = (x \* y \* z) / 100; System.out.println("\nSimple Interest = " + SI);

}

}

#### //class testinterest

import static org.junit.Assert.assertEquals; import org.junit.Before;

import org.junit.Test;

public class testinterest {

@Before

public void setUp() {

}

// Test case for multiplication of all positive @Test

public void testall\_Positive() {

double result=simple\_interest.CalculateSimpleInterest(50000.00, 4.0,8.0); assertEquals(16000, result,0.001);

}

// Test case for multiplication of two positive one negetive @Test

public void testtwo\_Positive() {

double result=simple\_interest.CalculateSimpleInterest(50000.00, -4.0,8.0); assertEquals(16000, result,0.001);

}

// Test case for multiplication of two negetive one positive @Test

public void testtwo\_Negative() {

double result=simple\_interest.CalculateSimpleInterest(-50000.00, -4.0,8.0); assertEquals(16000, result,0.001);

}

// Test case for multiplication involving zero @Test

public void test\_ZeroPositive() {

double result=simple\_interest.CalculateSimpleInterest(0, 4.0,8.0); assertEquals(16000, result,0.001

}

# Practical 10

## To study and perform validation testing.

**Validation testing** is a type of software testing that ensures the product meets the user's needs and requirements. It confirms that the software behaves as expected in a real-world scenario and answers the question, *"Are we building the right product?"*

Validation testing is performed during or at the end of the development process to verify the system's functionality from the user's perspective. It typically involves methods like functional testing, system testing, and acceptance testing.

## Key Aspects of Validation Testing:

1. **User-Centered**: Focuses on whether the product fulfills the user’s requirements.
2. **Real-World Scenarios**: Simulates actual use cases to ensure functionality.
3. **High-Level Testing**: Typically comes after verification testing, which checks whether the product is built correctly.

## Example of Validation Testing

**Scenario:** A team develops a mobile banking app that allows users to check balances, transfer funds, and pay bills.

* + **Test Case:** Validate the fund transfer functionality.

#### Steps:

* + 1. Open the app and log in with valid credentials.
    2. Navigate to the "Transfer Funds" section.
    3. Select the "From Account" and "To Account."
    4. Enter an amount to transfer and submit.
  + **Expected Result:** The app should successfully transfer the entered amount and display a confirmation message.
  + **Validation Check:** Ensure the transfer is reflected in the account balances correctly and that the user receives an email confirmation. The transfer process must meet user expectations (like speed and ease of use) and business rules.

## Popular Tools for validation testing

* + 1. **Selenium**: For web application validation.
    2. **Katalon Studio**: Supports web, API, and mobile validation.
    3. **Appium**: Used for validating mobile applications.
    4. **Postman (for API Validation):** used for API validation testing
    5. **Cypress:** For web application validation.

# Practical 11

## To study and perform performance based testing.

**Performance testing** is a type of software testing that evaluates how a system performs under various conditions. The primary focus is to check the speed, responsiveness, scalability, and stability of an application under a specific workload.

Performance testing ensures that the system behaves as expected in terms of:

* + **Response time**: How fast the system responds to a user’s request.
  + **Throughput**: The number of transactions processed in a given time period.
  + **Scalability**: The system’s ability to handle increased load.
  + **Stability**: How the system performs under stress for a prolonged period.

## Types of Performance Testing

1. **Load Testing**: Measures system performance under expected user loads.
2. **Stress Testing**: Puts the system under extreme conditions to see how it behaves under stress or when it exceeds its limits.
3. **Spike Testing**: Tests how the system reacts to sudden increases in user load.
4. **Endurance Testing**: Checks how the system behaves when subjected to high loads over extended periods (also called soak testing).
5. **Scalability Testing**: Evaluates the system’s ability to scale up or down based on user demand.

## Example of Performance Testing

**Scenario**: A retail website experiences high traffic during flash sales. The development team wants to ensure that the website performs well under such conditions.

* + **Test Case**: Load testing for the website's checkout process.

#### Steps:

* + 1. Simulate 100, 500, 1,000, and 5,000 concurrent users accessing the checkout page.
    2. Each simulated user adds items to their cart and attempts to make a purchase.
    3. Monitor system metrics such as response time, CPU usage, memory consumption, and transaction success rate.

#### Expected Outcome:

* + 1. The website should handle up to 1,000 concurrent users without any noticeable performance degradation.
    2. For 5,000 concurrent users, response time might slightly increase, but the system should still function without crashing.

#### Metrics Monitored:

* **Response Time**: Should be under 2 seconds for up to 1,000 users.
* **Throughput**: The system should process at least 50 transactions per second.
* **CPU/Memory Usage**: Should not exceed 80% utilization.

## Popular Tools for Performance Testing

1. **Apache JMeter**: Open-source tool for load and performance testing of web applications.
2. **LoadRunner**: Commercial tool by Micro Focus, widely used for large-scale performance testing.
3. **Gatling**: Open-source tool designed for web applications, focusing on scalability testing.
4. **Neoload**: Tool for continuous performance testing integrated into DevOps environments.

# Practical 12

## To study and perform regression based testing.

**Regression testing** is a type of software testing that ensures recent code changes haven't negatively impacted the existing functionality of the application. The goal is to verify that previously developed and tested features still work as expected after modifications like bug fixes, feature enhancements, or system upgrades.

## Key Objectives of Regression Testing:

1. **Detect Unintended Side Effects**: Ensure that changes in one part of the system do not break other parts.
2. **Maintain Stability**: Keep the system stable by validating that old features function as intended after new updates.
3. **Revalidate Existing Functionality**: Confirm that previously working features continue to operate correctly after code changes.

## When is Regression Testing Performed?

* + After bug fixes.
  + After enhancements or new feature additions.
  + Following performance or security patches.
  + During integration of new modules or systems.

## Types of Regression Testing

1. **Corrective Regression**: Testing when no changes are made to the existing system, and existing test cases can be reused.
2. **Retest-all Regression**: Retesting all the tests in the application to ensure no functionality is broken.
3. **Selective Regression**: Testing specific parts of the application that might have been affected by the changes.
4. **Progressive Regression**: Used when new test cases are developed during the modification process.

## Example of Regression Testing

**Scenario**: A team adds a "Wishlist" feature to an e-commerce website that previously allowed only adding items to a cart and purchasing them.

* + **Test Case**: Ensure the "Add to Cart" and "Purchase" functionalities still work after introducing the "Wishlist" feature.

#### Steps:

* + 1. Add an item to the cart and proceed with the checkout process.
    2. Ensure the payment gateway redirects properly and the purchase completes without errors.
    3. Add the same item to the wishlist and verify that it appears correctly.

#### Expected Outcome:

* The existing "Add to Cart" and "Purchase" functionalities should work as before.
* The new "Wishlist" feature should not interfere with the cart and purchase processes.
  + **Regression Check**: The core e-commerce features like adding to the cart, payment, and order confirmation should remain unaffected.

## Tools for Regression Testing

1. **Selenium**: An open-source tool used for automating web applications for regression testing.
2. **TestComplete**: A commercial tool for automating desktop, web, and mobile applications.
3. **QTP/UFT**: Unified Functional Testing by Micro Focus for automating functional and regression tests.
4. **Rational Functional Tester (RFT)**: IBM's tool for functional and regression testing.
5. **Katalon Studio**: An easy-to-use tool for automating regression tests across web, mobile, and desktop platforms.