FACULTY OF ENGINEERING AND TECHNOLOGY

PARUL INSTITUTE OF TECHNOLOGY

BACHELOR OF TECHNOLOGY

Software Testing and Quality Assurance

(203105396)

LAB MANUAL

7th SEMESTER

210305124032

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

<u>CERTIFICATE</u>

*This is to certify that **Mr. Gaurav Paliwal** with Enrollment no.**210305124032** has successfully completed his/her laboratory experiments in the **Software Testing and Quality Assurance** from the department of PIT-CSE (AI).During the academic year.2024-25.*

Date of Submission: ....   Staff In charge: ....

Head Of Department: ....

Index

Sr. No.	Experiment Title	Page No.	Performance Date	Submission Date	Marks	Sign
1.	Design test cases using Boundary value analysis.					
2.	Design test cases using Equivalence class partitioning.					
3.	Design independent paths by calculating cyclomatic complexity using date problem.					
4.	Design test cases using Decision table.					
5.	Design independent paths by takingDD path using date problem.					
6.	Understand the Automation Testing Approach (Theory Concept).					
7.	Using Selenium IDE, write a test suite containing minimum 4 test cases.					
8.	Install Selenium server and demonstrate it using a script in Java/ PHP.					
9.	Write and test a program to login a specific web page.					
10.	Write and test a program to provide total number of objects present/ available on the page.					

11.	Write and test a program to update 10 student records into table into Excel file.					

Practical: 1

AIM:- Design test cases using Boundary value analysis.

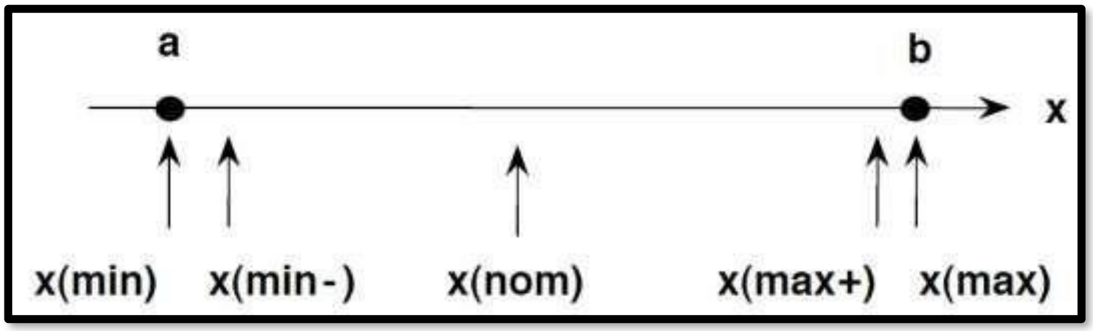
THEORY:-

What is Boundary Testing?

Boundary testing is the process of testing between extreme ends or boundaries betweenpartitions of the input values.

- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, JustInside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in normal boundary value testing is to select input variable values attheir:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum



-
- In Boundary Testing, Equivalence Class Partitioning plays a good role Boundary Testing comes after the Equivalence Class Partitioning.

User AGE:

Example on Boundary Value Analysis Test Case Design

Technique:Assume, we have to test a field which accepts

User AGE

Minimum boundary value is 18

Maximum boundary value is 60

Valid Inputs: 18,19,59,60

Invalid Inputs: 17 and 61

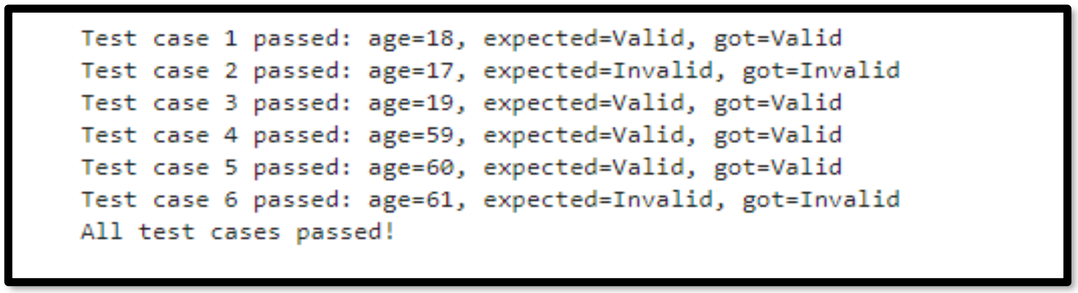
Test cases	Enter the value	Valid	Invalid	Pass/Fail
1	18	YES	NO	PASS
2	17(18-1)	NO	YES	FAIL
3	19(18+1)	YES	NO	PASS
4	59(60-1)	YES	NO	PASS
5	60	YES	NO	PASS
6	61(60+1)	NO	YES	FAIL

CODE:-

```
def
   is_valid_ag
   e(age):if 18
   <= age <=
    60:
        return
    "Valid"
    else:
        return
"Invalid"#
Define test
cases
test_cases =
[
    (18, "Valid"),  # Minimum boundary  (17,
    "Invalid"), # Just below minimum
    boundary(19, "Valid"), # Just above
    minimum boundary (59, "Valid"), # Just
    below maximum boundary (60, "Valid"),
    # Maximum boundary
    (61, "Invalid")  # Just above maximum boundary
]
# Run test cases
for i, (age, expected) in
    enumerate(test_cases):result =
    is_valid_age(age)
    assert result == expected, f"Test case {i+1} failed: age={age},
expected={expected},got={result}"
    print(f"Test case {i+1} passed: age={age}, expected={expected},
got={result}")print("All test cases passed!")
```

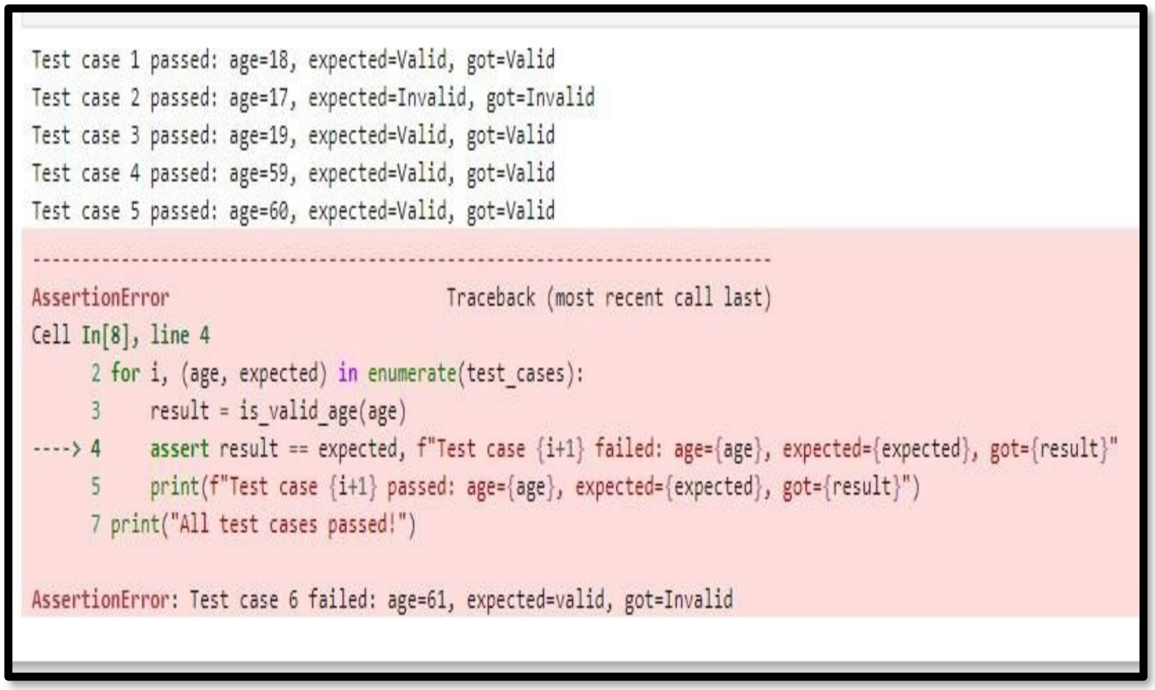
OUTPUT:-

1) For all test cases are passed.



```
Test case 1 passed: age=18, expected=Valid, got=Valid
Test case 2 passed: age=17, expected=Invalid, got=Invalid
Test case 3 passed: age=19, expected=Valid, got=Valid
Test case 4 passed: age=59, expected=Valid, got=Valid
Test case 5 passed: age=60, expected=Valid, got=Valid
Test case 6 passed: age=61, expected=Invalid, got=Invalid
All test cases passed!
```

2) For , one test case is failed.



```
Test case 1 passed: age=18, expected=Valid, got=Valid
Test case 2 passed: age=17, expected=Invalid, got=Invalid
Test case 3 passed: age=19, expected=Valid, got=Valid
Test case 4 passed: age=59, expected=Valid, got=Valid
Test case 5 passed: age=60, expected=Valid, got=Valid

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[8], line 4
      2 for i, (age, expected) in enumerate(test_cases):
      3     result = is_valid_age(age)
----> 4     assert result == expected, f"Test case {i+1} failed: age={age}, expected={expected}, got={result}"
      5     print(f"Test case {i+1} passed: age={age}, expected={expected}, got={result}")
      7 print("All test cases passed!")

AssertionError: Test case 6 failed: age=61, expected=valid, got=Invalid
```

Practical: 2

Aim: Testcase Design Using Equivalence class Partitioning:

Equivalence Class Partitioning (ECP) is a testing technique that divides input data into partitions of equivalent data from which test cases can be derived. The idea is to reduce the number of test cases to a manageable level while still ensuring adequate coverage. Each partition represents a set of values that are expected to be treated similarly by the software.

For the is_valid_age function, the valid age range is 18 to 60, inclusive. We can divide the input into several equivalence classes:

1. **Valid age range (18 to 60)**
2. **Below the valid range (< 18)**
3. **Above the valid range (> 60)**

Equivalence Classes

1. Valid ages: 18 to 60
2. Invalid ages (below the range): any age < 18
3. Invalid ages (above the range): any age > 60

Test Cases

From these equivalence classes, we can derive the following test cases:

Valid Equivalence Class

- Test case 1: An age within the valid range (middle value)
  - Input: 30
  - Expected Output: "Valid"

Invalid Equivalence Class (below the range)

- Test case 2: An age just below the valid range (boundary value)
  - Input: 17
  - Expected Output: "Invalid"

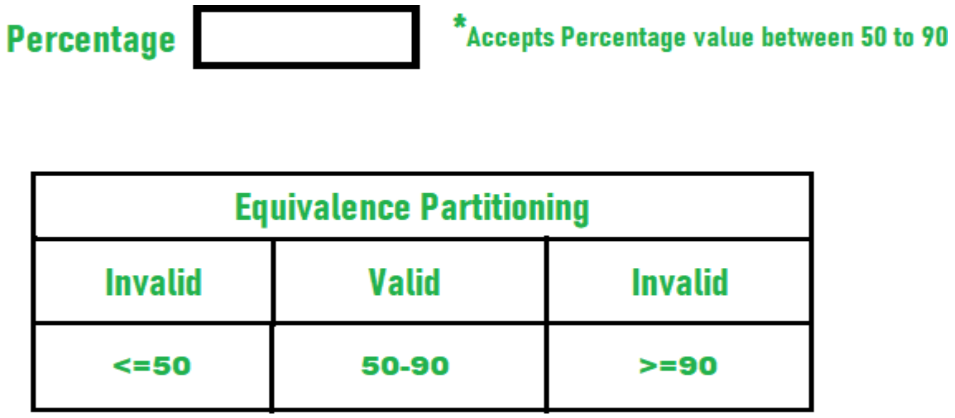
- Test case 3: An age well below the valid range
    - Input: 10
    - Expected Output: "Invalid"

Invalid Equivalence Class (above the range)

- Test case 4: An age just above the valid range (boundary value)
    - Input: 61
    - Expected Output: "Invalid"
- Test case 5: An age well above the valid range
    - Input: 70
    - Expected Output: "Invalid"

Summary of Test Cases

1. **Age 30** (within the valid range): Expected "Valid"
2. **Age 17** (just below the valid range): Expected "Invalid"
3. **Age 10** (well below the valid range): Expected "Invalid"
4. **Age 61** (just above the valid range): Expected "Invalid"
5. **Age 70** (well above the valid range): Expected "Invalid"
6. **Example-1:**
   Let us consider an example of any college admission process. There is a college that gives admissions to students based upon their percentage.
7. 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.



Equivalence Partitioning		
Invalid	Valid	Invalid
<=50	50-90	>=90

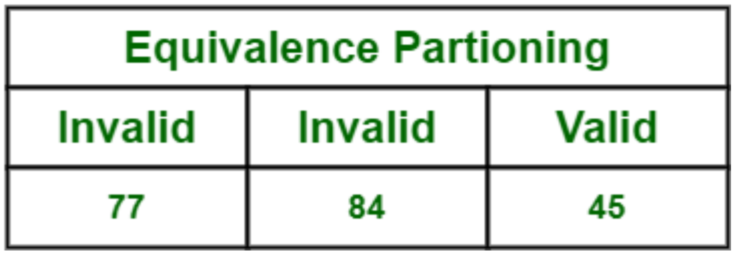
8.

9. **Example 2:**
   Let us consider an example of an online shopping site. In this site, each of products has a specific product ID and product name. We can search for product either by using name of product or by product ID. Here, we consider search field that accepts only valid product ID or product name.

10. Let us consider a set of products with product IDs and users wants to search for Mobiles. Below is a table of some products with their product Id.

Product	**Product ID**
Mobiles	45
Laptops	54
Pen Drives	67
Keyboard	76
Headphones	34

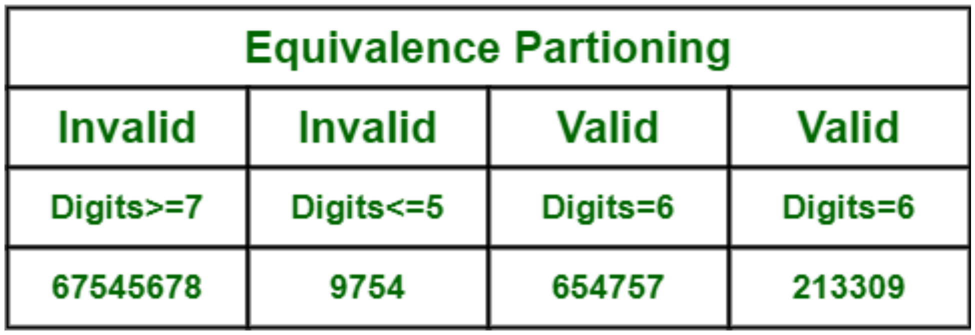
11. If the product ID entered by user is invalid then application will redirect customer or user to error page. If product ID entered by user is valid i.e. 45 for mobile, then equivalence partitioning method will show a valid product ID.

12. 

Equivalence Partioning		
Invalid	Invalid	Valid
77	84	45

13. **Example-3 :**
    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.

14. 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.



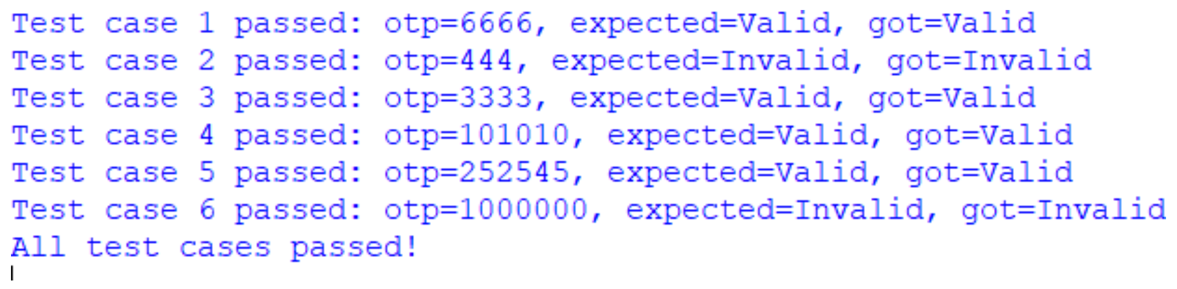
Equivalence Partioning			
Invalid	Invalid	Valid	Valid
Digits>=7	Digits<=5	Digits=6	Digits=6
67545678	9754	654757	213309

15.

Code:

```
def is_valid_otp(otp):
    if 1000 <= otp <= 999999:
        return "Valid"
    else:
        return "Invalid"

test_cases =
    [ (6666,
    "Valid"),
    (444, "Invalid"),
    (3333, "Valid"),
    (101010, "Valid"),
    (252545, "Valid"),
    (1000000, "Invalid")
]

for i, (otp, expected) in enumerate(test_cases):
    result = is_valid_otp(otp)
    assert result == expected, f"Test case {i+1} failed: otp={otp}, expected={expected},
got={result}"
    print(f"Test case {i+1} passed: otp={otp}, expected={expected}, got={result}")

print("All test cases passed!")
```

Output:



```
Test case 1 passed: otp=6666, expected=Valid, got=Valid
Test case 2 passed: otp=444, expected=Invalid, got=Invalid
Test case 3 passed: otp=3333, expected=Valid, got=Valid
Test case 4 passed: otp=101010, expected=Valid, got=Valid
Test case 5 passed: otp=252545, expected=Valid, got=Valid
Test case 6 passed: otp=1000000, expected=Invalid, got=Invalid
All test cases passed!
```

Practical: 3

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

Cyclomatic complexity is a software metric used to measure the complexity of a program. It is calculated by using the control flow graph of the program, where nodes represent the commands/statements of the program, and edges represent the control flow between the nodes.

Here's how to calculate cyclomatic complexity:

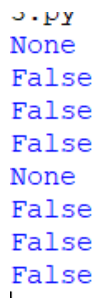
1. Identify the control flow graph of the program.
2. Count the number of edges (E) in the graph.
3. Count the number of nodes (N) in the graph.
4. Count the number of connected components (P) (usually P is 1 if the program is a single connected graph).

The cyclomatic complexity (V(G)) is then calculated using the formula: V(G)=E−N+2PV(G) = E - N + 2PV(G)=E−N+2P

Date Problem Example

Let's consider a simple problem where we need to determine if a given date is valid. The logic might involve checking the validity of day, month, and year. Here's a simple pseudo-code representation:

```
def is_leap_year(y):

    if (y % 100 == 0 and y % 400 == 0) or y % 4 == 0:

        print(True)

    else:

        print(False)


def is_valid_date(day, month, year):

    if year < 1:

        return False

    if month < 1 or month > 12:
        return False
    if day < 1:
        return False
    if month in {1, 3, 5, 7, 8, 10, 12}:
        if day > 31:
            return False
    elif month in {4, 6, 9, 11}:
        if day > 30:
            return False
    elif month == 2:
        if is_leap_year(year):
            if day > 29:
                return False
        else:
            if day > 28:
                return False
    return True


print(is_valid_date(26, 6, 2024))
print(is_valid_date(29, 2, 2021))
print(is_valid_date(31, 4, 2021))
print(is_valid_date(31, 12, 2021))
print(is_valid_date(0, 1, 2021))
print(is_valid_date(15, 13, 2021))
print(is_valid_date(15, 10, -1))
```



```
5.py
None
False
False
False
None
False
False
False
```

Control Flow Graph

1. Nodes:
   - Start
   - Check if year is valid
   - Check if month is valid
   - Check if day is valid
   - Check month days for 31-day months
   - Check month days for 30-day months
   - Check leap year condition for February
   - Final validity check

2. Edges:
   - Start -> Check if year is valid
   - Check if year is valid -> Check if month is valid (if true)
   - Check if year is valid -> End (if false)
   - Check if month is valid -> Check if day is valid (if true)
   - Check if month is valid -> End (if false)
   - Check if day is valid -> Check month days (31-day months) (if true)
   - Check if day is valid -> End (if false)
   - Check month days (31-day months) -> End (if false)
   - Check month days (31-day months) -> Check month days (30-day months) (if true)
   - Check month days (30-day months) -> End (if false)
   - Check month days (30-day months) -> Check February (if true)
   - Check February -> End (if false)
   - Check February -> Check leap year condition (if true)
   - Check leap year condition -> End (if false)
   - Check leap year condition -> Final validity check (if true)
   - Final validity check -> End
3. Connected Components: 1 (assuming it's a single function/module)

   Calculation of Cyclomatic Complexity

   - E=16E = 16E=16

- N=10N = 10N=10
- P=1P = 1P=1

Using the formula: V(G)=E−N+2PV(G) = E - N + 2PV(G)=E−N+2P V(G)=16−10+2×1V(G) = 16 - 10 + 2 \times 1V(G)=16−10+2×1 V(G)=16−10+2V(G) = 16 - 10 + 2V(G)=16−10+2 V(G)=8V(G) = 8V(G)=8

Thus, the cyclomatic complexity of the given date validation function is 8.

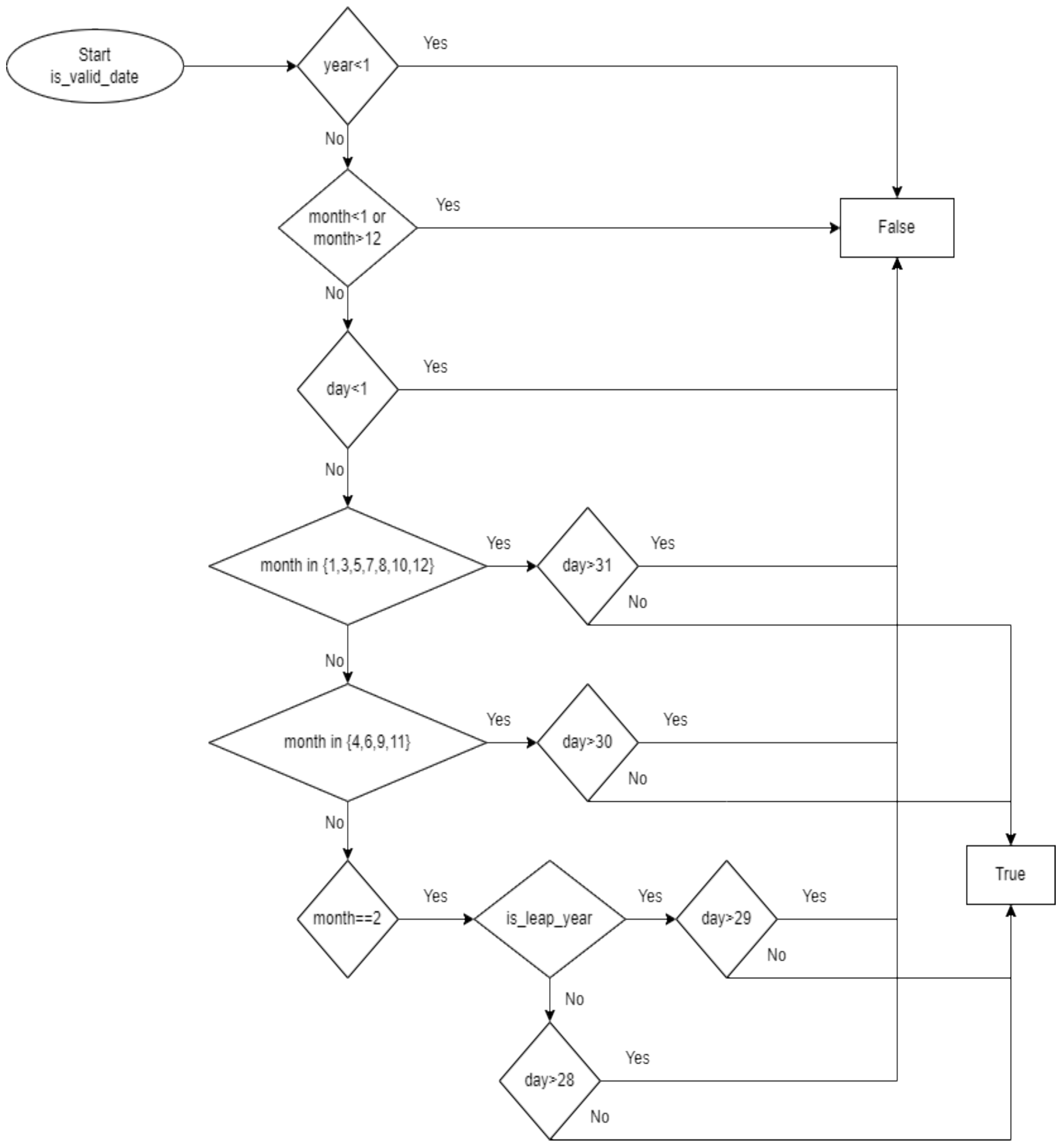
Independent Paths

To ensure complete testing, we need to identify the independent paths in the control flow graph. Independent paths are those that traverse new edges or nodes:

1. Start -> Check year -> Check month -> Check day -> Check 31-day month -> End (Valid 31-day date)
2. Start -> Check year -> Check month -> Check Day -> Check 30-day month -> End (Valid 30-day date)
3. Start -> Check year -> Check month -> Check Day -> Check February -> Leap year -> End (Valid 29-day February date)
4. Start -> Check year -> Check month -> Check Day -> Check February -> Not leap year -> End (Valid 28-day February date)
5. Start -> Check year -> Check month -> End (Invalid month)
6. Start -> Check year -> End (Invalid year)
7. Start -> Check year -> Check month -> Check Day -> End (Invalid day for month with 31 days)
8. Start -> Check year -> Check month -> Check Day -> Check February -> Leap year -> End (Invalid day for February in leap year)
9. Start -> Check year -> Check month -> Check Day -> Check February -> Not leap year -> End (Invalid day for February in non-leap year)

These paths ensure that all logical branches and conditions in the program are tested.

Graph:


Start
is_valid_date
year<1
Yes
No
month<1 or month>12
Yes
No
False
day<1
Yes
No
month in {1,3,5,7,8,10,12}
Yes
day>31
Yes
No
No
month in {4,6,9,11}
Yes
day>30
Yes
No
No
month==2
Yes
is_leap_year
Yes
day>29
Yes
No
No
True
day>28
Yes
No

Practical: 4

AIM: Design test cases using Decision table

Decision tables are a systematic way of identifying and documenting the different conditions and their corresponding actions in a decision-making process. They are particularly useful in testing because they help ensure that all possible combinations of conditions are considered. Here's a step-by-step guide on how to design test cases using a decision table:

Step 1: Identify the Conditions and Actions

First, identify all the conditions (inputs) and actions (outputs) relevant to the decision-making process.

Example Scenario: Loan Approval System

- **Conditions**:
  1. Applicant's credit score (Good, Bad)
  2. Applicant's income level (High, Low)
  3. Collateral available (Yes, No)
- **Actions**:
  4. Approve loan
  5. Deny loan

Step 2: List All Possible Combinations of Conditions

Create a table listing all possible combinations of conditions. For three conditions with two possible values each, there are 23=82^3 = 823=8 combinations.

Condition 1: Credit Score	**Condition 2: Income Level**	**Condition 3: Collateral**	**Action**
Good	High	Yes	Appro ve loan

Condition 1: Credit Score	**Condition 2: Income Level**	**Condition 3: Collateral**	**Action**
Good	High	No	Approve loan
Good	Low	Yes	Approve loan
Good	Low	No	Deny loan
Bad	High	Yes	Approve loan
Bad	High	No	Deny loan
Bad	Low	Yes	Deny loan
Bad	Low	No	Deny loan

Step 3: Assign Actions for Each Combination

Based on business rules, determine the appropriate action for each combination of conditions.

Step 4: Convert the Decision Table into Test Cases

Each row in the decision table represents a unique test case. Translate these rows into test cases.

Test Case 1:

- **Credit Score**: Good
- **Income Level**: High
- **Collateral**: Yes
- **Expected Result**: Approve loan

Test Case 2:

- **Credit Score**: Good
- **Income Level**: High
- **Collateral**: No
- **Expected Result**: Approve loan

Test Case 3:

- **Credit Score**: Good
- **Income Level**: Low
- **Collateral**: Yes
- **Expected Result**: Approve loan

Test Case 4:

- **Credit Score**: Good
- **Income Level**: Low
- **Collateral**: No
- **Expected Result**: Deny loan

Test Case 5:

- **Credit Score**: Bad
- **Income Level**: High
- **Collateral**: Yes
- **Expected Result**: Approve loan

Test Case 6:

- **Credit Score**: Bad
- **Income Level**: High
- **Collateral**: No[
- **Expected Result**: Deny loan

Test Case 7:

- **Credit Score**: Bad
- **Income Level**: Low
- **Collateral**: Yes
- **Expected Result**: Deny loan

Test Case 8:

- **Credit Score**: Bad
- **Income Level**: Low

- **Collateral**: No
- **Expected Result**: Deny loan

Step 5: Implement and Execute Test Cases

Use the above test cases to test the system. Ensure each condition combination is tested and the expected outcome is verified against the actual outcome.

Example Decision Table Template

Here's a blank template for a decision table you can use for your own scenarios:

Condition 1	Condition 2	Condition 3	...	Condition N	Action
Value A1	Value B1	Value C1	...	Value N1	Action 1
Value A1	Value B1	Value C2	...	Value N2	Action 2
Value A1	Value B2	Value C1	...	Value N3	Action 1
Value A2	Value B1	Value C1	...	Value N4	Action 3
...	...	...	...	...	...

By following these steps, you can systematically design comprehensive test cases to ensure that all scenarios are tested and your system behaves as expected under all conditions.

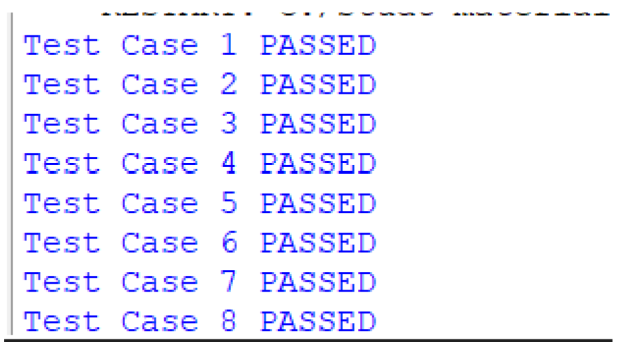
Code:

Input:

```
decision_table = [
    {"credit_score": "Good", "income_level": "High", "collateral": "Yes", "expected_result": "Approve loan"},
    {"credit_score": "Good", "income_level": "High", "collateral": "No", "expected_result": "Approve loan"},
    {"credit_score": "Good", "income_level": "Low", "collateral": "Yes", "expected_result": "Approve loan"},
    {"credit_score": "Good", "income_level": "Low", "collateral": "No", "expected_result": "Deny loan"},
{"credit_score": "Bad", "income_level": "High", "collateral": "Yes", "expected_result": "Approve
loan"},

{"credit_score": "Bad", "income_level": "High", "collateral": "No", "expected_result": "Deny
loan"},
  {"credit_score": "Bad", "income_level": "Low", "collateral": "Yes", "expected_result": "Deny
loan"},
  {"credit_score": "Bad", "income_level": "Low", "collateral": "No", "expected_result": "Deny
loan"},
]


def loan_approval_system(credit_score, income_level, collateral):
  if credit_score == "Good" and (income_level == "High" or collateral == "Yes"):
    return "Approve loan"
  elif credit_score == "Bad" and income_level == "High" and collateral == "Yes":
    return "Approve loan"
  else:
    return "Deny loan"


def run_test_cases(decision_table):
  for i, test_case in enumerate(decision_table):
    credit_score = test_case["credit_score"]
    income_level = test_case["income_level"]
    collateral = test_case["collateral"]
    expected_result = test_case["expected_result"]


    actual_result = loan_approval_system(credit_score, income_level, collateral)
```

```
        if actual_result == expected_result:
            print(f"Test Case {i+1} PASSED")
        else:
            rint(f"Test Case {i+1} FAILED: expected
{expected_result} but got {actual_result}")


run_test_cases(decision_table)
```

Output:



```
Test Case 1 PASSED
Test Case 2 PASSED
Test Case 3 PASSED
Test Case 4 PASSED
Test Case 5 PASSED
Test Case 6 PASSED
Test Case 7 PASSED
Test Case 8 PASSED
```

Practical: 5

Aim: Design independent paths by taking DD path using date problem.

THEORY: -

A Decision Table is a tabular representation of inputs versus rules/cases/test conditions. It is a very effective tool used for both complex software testing and requirements management. Decision table helps to check all possible combinations of conditions for testing and testers can also identify missed conditions easily. The conditions are indicated as True(T) and False(F) values.

Decision table testing is a software testing technique used to test system behaviour for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behaviour (Output) are captured in a tabular form. That is why it is also called as a Cause-Effect table where Cause and effects are captured for better test coverage.

Let's learn with an example.

Example 1: Date Problem example:

Condition	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Conditions					
Input format valid?	Yes	Yes	Yes	Yes	No
Day valid?	Yes	Yes	No	-	-
Month Valid?	Yes	Yes	-	No	-
Leap Year?	Yes	No	-	-	-
Actions					
Print "Valid date!"	Yes	Yes	No	No	No
Print "Invalid day"	No	No	Yes	No	No
Print "Invalid month"	No	No	No	Yes	No

Print "Invalid format"	No	No	No	No	Yes
Print "Leap year"	Yes	No	No	No	No
Print "Not a leap year"	No	Yes	No	No	No

Calculating Cyclomatic Complexity

Cyclomatic Complexity=E−N+2P

- E is the number of edges in the graph.
- N is the number of nodes in the graph.
- P is the number of connected components (for a single program, this is usually 1).

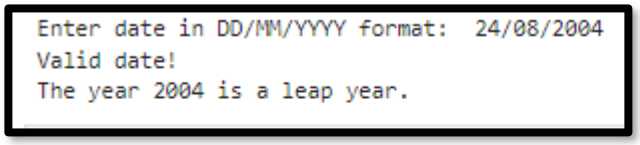
Cyclomatic Complexity=E−N+2P
Cyclomatic Complexity=10−11+2×1
Cyclomatic Complexity=10−11+2
Cyclomatic Complexity=1
#program to find the entered date is leap year or not

CODE:-

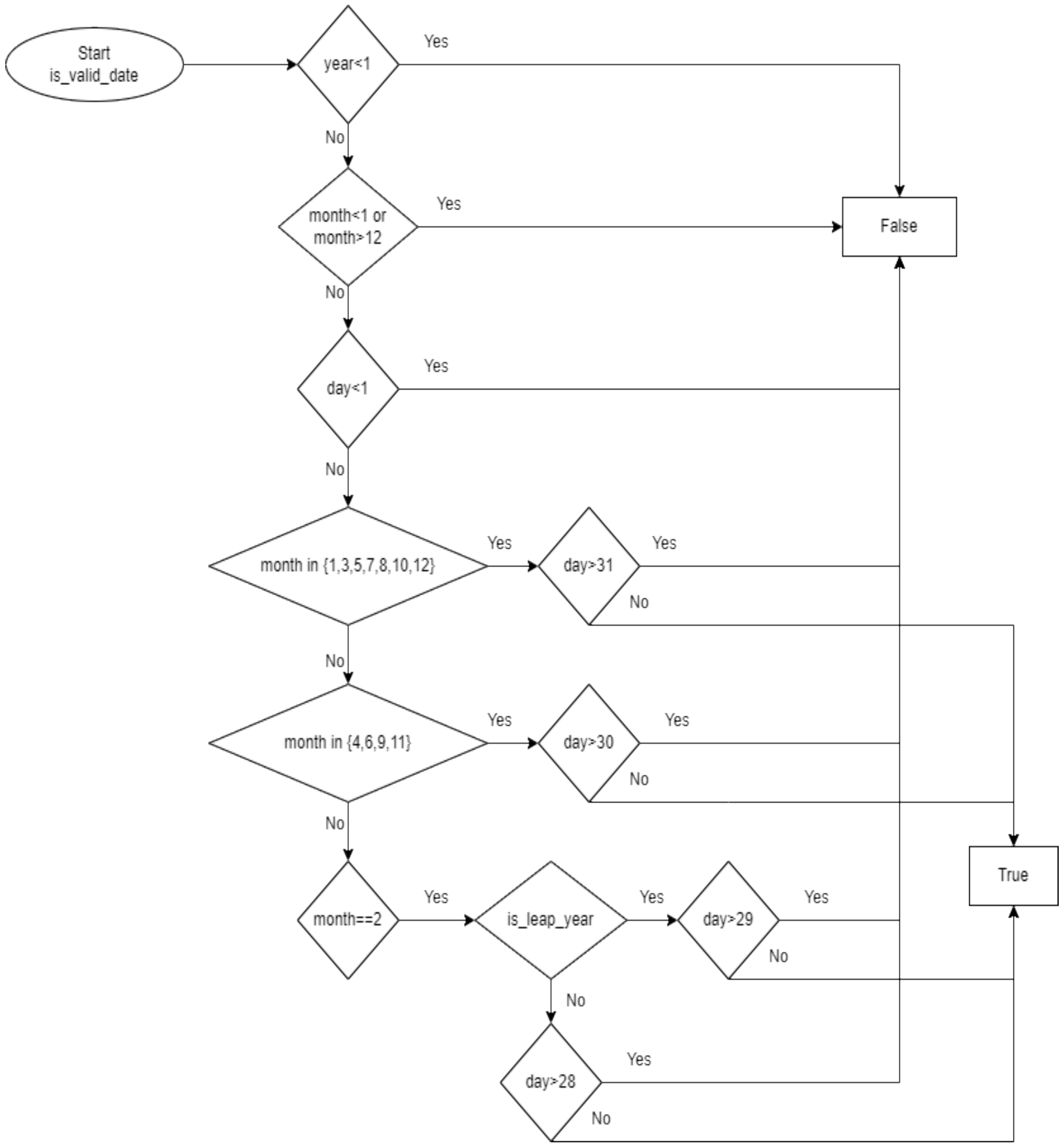
```
def validate_day(day, month, year):#

    Check for months with 31 daysif

    month in [1, 3, 5, 7, 8, 10, 12]:

        return 1 <= day <= 31

    # Check for months with 30 days

    elif month in [4, 6, 9, 11]:

        return 1 <= day <= 30#

    Check for February elif

    month == 2:
        if is_leap_year(year): return

           1 <= day <= 29

        else:
        return 1 <= day <= 28

     else:

        return False

     def

  validate_month(month):

  return 1 <= month <= 12def

  is_leap_year(year):

     return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)def

  validate_date(date_str):

     try:

        day, month, year = map(int, date_str.split("/"))

        if not validate_day(day, month, year):

           print(f"Invalid day for {month}/{year}!")

           return False

        if not validate_month(month):

           print("Invalid month! Please enter a month between 1 and 12.")

           return False

        return True except

     ValueError:

        print("Invalid input format! Please enter the date in DD/MM/YYYY format.")

        return False

  date_str = input("Enter date in DD/MM/YYYY format: ")
```

```
if validate_date(date_str):
  print("Valid date!")
  day, month, year = map(int, date_str.split("/"))
  if is_leap_year(year):
    print(f"The year {year} is a leap year.")
  else:
    print(f"The year {year} is not a leap year.")
else:
  print("Invalid date!")
```

OUTPUT:-



```
Enter date in DD/MM/YYYY format:  24/08/2004
Valid date!
The year 2004 is a leap year.
```


Start
is_valid_date
year<1
Yes
No
month<1 or month>12
Yes
No
False
day<1
Yes
No
month in {1,3,5,7,8,10,12}
Yes
day>31
Yes
No
No
month in {4,6,9,11}
Yes
day>30
Yes
No
No
True
month==2
Yes
is_leap_year
Yes
day>29
Yes
No
No
day>28
Yes
No

Practical: 6

Aim: Understand the Automation Testing Approach (Theory Concept).

Automation Testing: Automated Testing means using special software for tasks that people usually do when checking and testing a software product. Nowadays, many software projects use automation testing from start to end, especially in agile and DevOps methods.

Types of automation testing:

Unit testing: Unit testing is a phase in software testing to test the smallest piece of code known as a unit that can be logically isolated from the code. It is carried out during the development of the application.

Integration testing: Integration testing is a phase in software testing in which individual software components are combined and tested as a group. It is carried out to check the compatibility of the component with the specified functional requirements.

Smoke testing: Smoke testing is a type of software testing that determines whether the built software is stable or not. It is the preliminary check of the software before its release in the market.

Performance testing: Performance testing is a type of software testing that is carried out to determine how the system performs in terms of stability and responsiveness under a particular load.

Regression testing: Regression testing is a type of software testing that confirms that previously developed software still works fine after the change and that the change has not adversely affected existing features.

Security testing: Security testing is a type of software testing that uncovers the risks, and vulnerabilities in the security mechanism of the software application. It helps an organization to identify the loopholes in the security mechanism and take corrective measures to rectify the security gaps.

Acceptance testing: Acceptance testing is the last phase of software testing that is performed after the system testing. It helps to determine to what degree the application meets end users' approval.

PI testing: API testing is a type of software testing that validates the Application Programming Interface (API) and checks the functionality, security, and reliability of the programming interface.

UI Testing: UI testing is a type of software testing that helps testers ensure that all the fields, buttons, and other items on the screen function as desired.

Advantages of Automation Testing

Simplifies Test Case Execution: Automation testing can be left virtually unattended and thus it allows monitoring of the results at the end of the process. Thus, simplifying the overall test execution and increasing the efficiency of the application.

Improves Reliability of Tests: Automation testing ensures that there is equal focus on all the areas of the testing, thus ensuring the best quality end product.

Increases amount of test coverage: Using automation testing, more test cases can be created and executed for the application under test. Thus, resulting in higher test coverage and the detection of more bugs. This allows for the testing of more complex applications and more features can be tested.

minimizing Human Interaction: In automation testing, everything is automated from test case creation to execution thus there are no changes for human error due to neglect. This reduces the necessity for fixing glitches in the post-release phase.

Saves Time and Money: The initial investment for automation testing is on the higher side but it is cost-efficient and time-efficient in the long run. This is due to the reduction in the amount of time required for test case creation and execution which contributes to the high quality of work.

Earlier detection of defects: Automation testing documents the defects, thus making it easier for the development team to fix the defect and give a faster output. The earlier the defect is identified, the easier and more cost-efficient it is to fix the defects.

Need of automation testing:

In 1994, an aircraft completing its routine flight crashed just before landing. This was due to a bug or defect in their software. The testers didn’t even care about the final testing, so this accident happened. In such high-priority cases, accurate automation testing becomes critical to replace a few manual tests. Automation testing tools like Browser Stack can help teams thoroughly test and speed up the execution time of test suites. Below are some of the reasons for using automation testing:

Quality Assurance: Manual testing is a tedious task that can be boring and at the same time error-prone. Thus, using automation testing improves the quality of the software under test as more test coverage can be achieved.

Error or Bug-free Software: Automation testing is more efficient for detecting bugs in comparison to manual testing.

No Human Intervention: Manual testing requires huge manpower in comparison to automation testing which requires no human intervention and the test cases can be executed unattended.

Increased test coverage: Automation testing ensures more test coverage in comparison to manual testing where it is not possible to achieve 100% test coverage.

Testing can be done frequently: Automation testing means that the testing can be done frequently thus improving the overall quality of the software under test.

Test tool: can be very useful. Some tools are essential. For example- An incident tracking system. Generally, tools are meant to improve the efficiency and accuracy of testing. Hence, it is very important that we carefully select tools and properly implement them as well. Many times, it has been observed that a good tool also does not get properly implemented resulting in inefficiencies. Ongoing maintenance of scripts and data is also very important to reap the benefits of test tools.

Most of the time, automation is equated with test execution automation. While it is true for the most part, the other aspects of testing are also automated.

Practical 7

Aim: Using Selenium IDE, write a test suite containing minimum 4 test cases.

Selenium is a popular open-source framework for automating web browsers. It is used to test web applications to ensure they work as expected. Selenium provides a suite of tools that cater to different testing needs. The main components of Selenium are:

1. **Selenium WebDriver**: This is the most commonly used tool. It provides a programming interface to create and execute test cases. WebDriver interacts directly with the web browser and mimics the actions of a real user.
2. **Selenium IDE**: This is an integrated development environment for Selenium scripts. It is a browser extension that allows users to record their interactions with a website and play them back. It's useful for creating quick bug reproduction scripts, exploratory testing, and creating scripts to aid in automation-aided exploratory testing.
3. **Selenium Grid**: This is used for running test scripts on multiple machines and browsers simultaneously. It is useful for cross-browser and cross-platform testing.

Steps to Do Testing in Selenium Using Selenium IDE

Install Selenium IDE:

- Go to the browser's extension store (e.g., Chrome Web Store or Firefox Add-ons).
- Search for "Selenium IDE" and add it to your browser.

Create a New Project:

- Open Selenium IDE from your browser's extensions.
- Click on "Create a new project" and provide a name for your project.

Record a Test:

- Click on the "Record a new test in a new project" option.
- Enter the base URL of the application you want to test.
- Click on the "Start recording" button.
- Perform the actions you want to test on your web application. Selenium IDE will record these actions.

Save the Test:

- Once you have performed the actions, click on the "Stop recording" button.
- Save the test with a relevant name.

Edit the Test:

- You can edit the recorded test to add assertions, loops, and other commands.
- Selenium IDE provides a user-friendly interface to modify the test steps.

Run the Test:

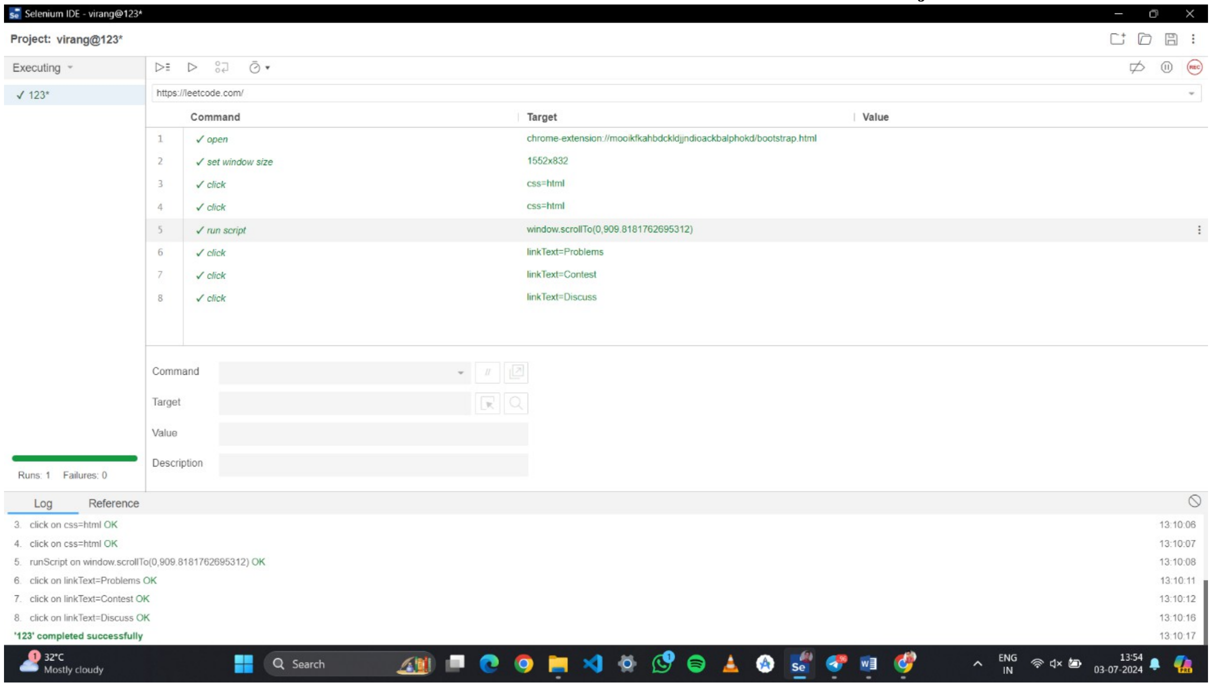
- Select the test you want to run.
- Click on the "Run current test" button.
- Observe the execution of the test in the browser.

View Test Results:

- After the test execution, Selenium IDE will display the test results.
- Review the results to ensure that all the test steps have passed.

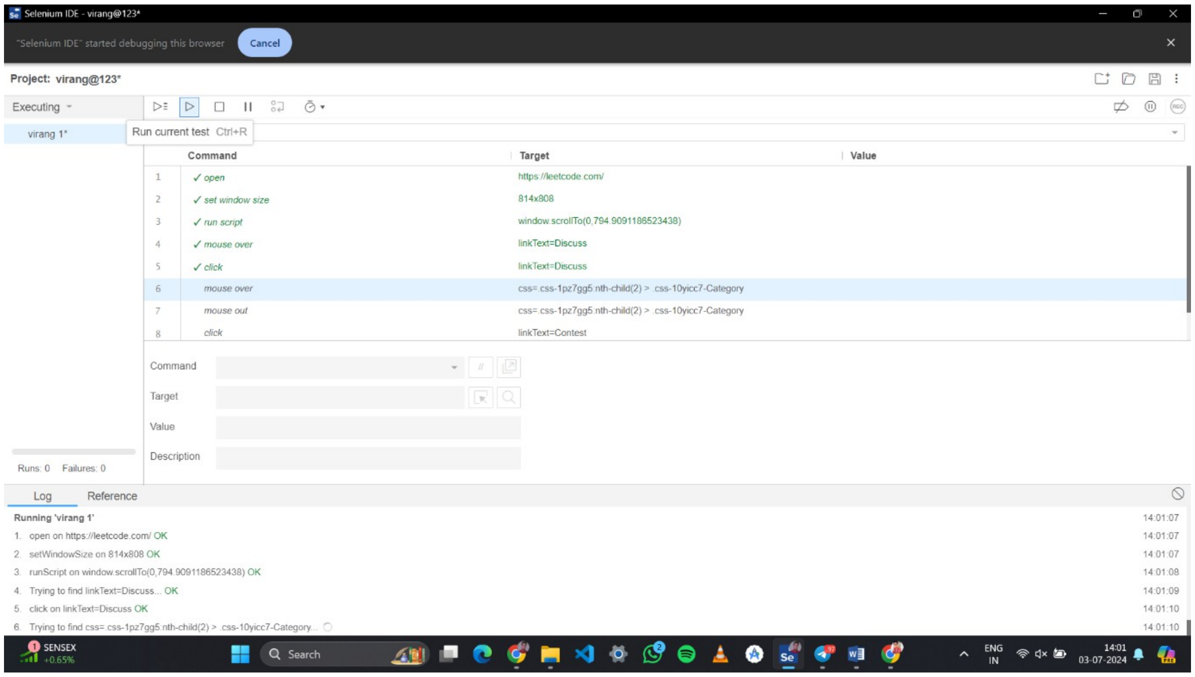
Example:

Website: https://leetcode.com/



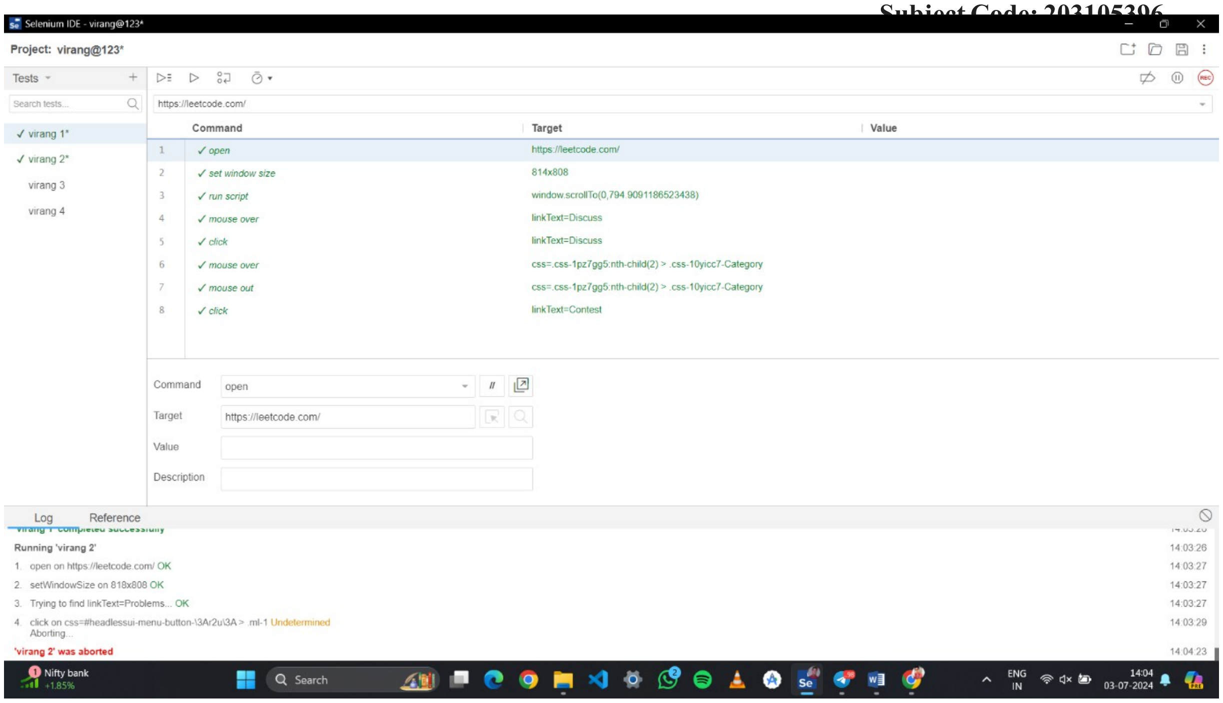
	Command	Target	Value
1	✓ open	chrome-extension://mooikfkahbdckldjjndioackbalphokd/bootstrap.html	
2	✓ set window size	1552x832	
3	✓ click	css=html	
4	✓ click	css=html	
5	✓ run script	window.scrollTo(0,909.8181762695312)	
6	✓ click	linkText=Problems	
7	✓ click	linkText=Contest	
8	✓ click	linkText=Discuss	

```
3.  click on css=html OK                                          13:10:06
4.  click on css=html OK                                          13:10:07
5.  runScript on window.scrollTo(0,909.8181762695312) OK          13:10:08
6.  click on linkText=Problems OK                                 13:10:11
7.  click on linkText=Contest OK                                  13:10:12
8.  click on linkText=Discuss OK                                  13:10:16
'123' completed successfully                                      13:10:17
```



	Command	Target	Value
1	✓ open	https://leetcode.com/	
2	✓ set window size	814x808	
3	✓ run script	window.scrollTo(0,794.9091186523438)	
4	✓ mouse over	linkText=Discuss	
5	✓ click	linkText=Discuss	
6	mouse over	css=.css-1pz7gg5:nth-child(2) > .css-10yicc7-Category	
7	mouse out	css=.css-1pz7gg5:nth-child(2) > .css-10yicc7-Category	
8	click	linkText=Contest	

```
Running 'virang 1'                                                          14:01:07
1.  open on https://leetcode.com/ OK                                        14:01:07
2.  setWindowSize on 814x808 OK                                             14:01:07
3.  runScript on window.scrollTo(0,794.9091186523438) OK                    14:01:08
4.  Trying to find linkText=Discuss... OK                                   14:01:09
5.  click on linkText=Discuss OK                                            14:01:10
6.  Trying to find css=.css-1pz7gg5:nth-child(2) > .css-10yicc7-Category... 14:01:10
```



Project: virang@123*

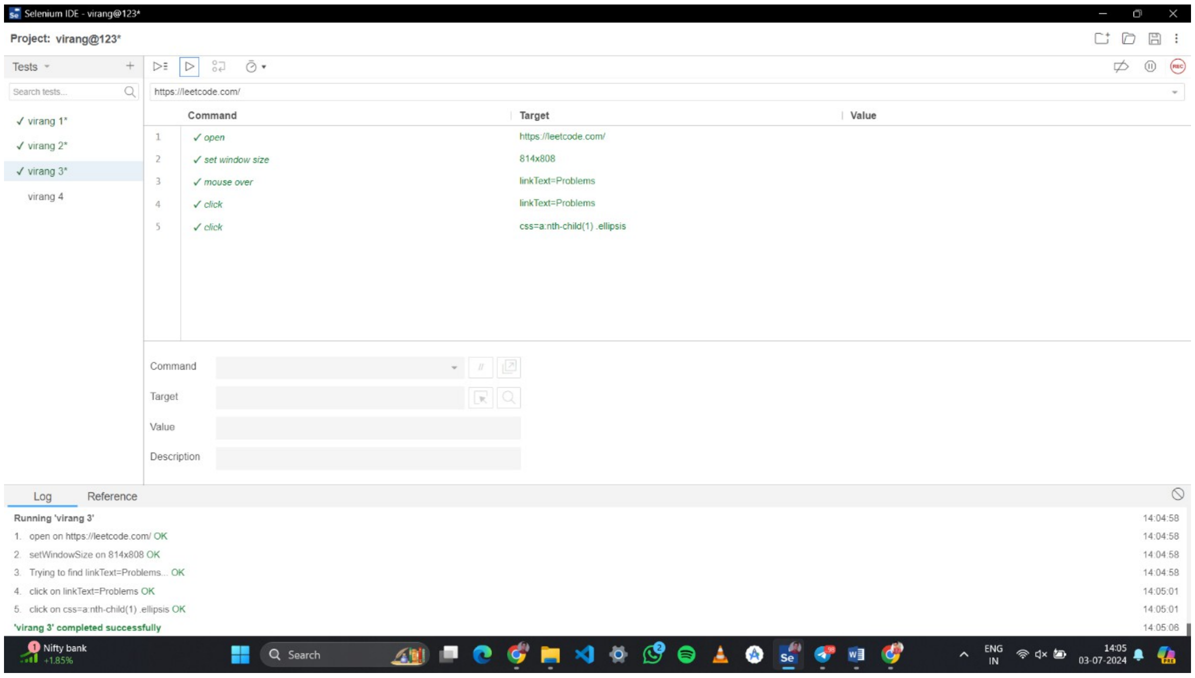
Tests: virang 1*, virang 2*, virang 3, virang 4

https://leetcode.com/

	Command	Target	Value
1	✓ open	https://leetcode.com/	
2	✓ set window size	814x808	
3	✓ run script	window.scrollTo(0,794.9091186523438)	
4	✓ mouse over	linkText=Discuss	
5	✓ click	linkText=Discuss	
6	✓ mouse over	css=.css-1pz7gg5:nth-child(2) > .css-10yicc7-Category	
7	✓ mouse out	css=.css-1pz7gg5:nth-child(2) > .css-10yicc7-Category	
8	✓ click	linkText=Contest	

Command: open
Target: https://leetcode.com/
Value:
Description:

```
'virang 1' completed successfully                                14:03:26
Running 'virang 2'                                               14:03:26
1. open on https://leetcode.com/ OK                              14:03:27
2. setWindowSize on 818x808 OK                                   14:03:27
3. Trying to find linkText=Problems... OK                        14:03:27
4. click on css=#headlessui-menu-button-\3Ar2u\3A > .ml-1 Undetermined   14:03:29
   Aborting...
'virang 2' was aborted                                           14:04:23
```



Project: virang@123*

Tests: virang 1*, virang 2*, virang 3*, virang 4

https://leetcode.com/

	Command	Target	Value
1	✓ open	https://leetcode.com/	
2	✓ set window size	814x808	
3	✓ mouse over	linkText=Problems	
4	✓ click	linkText=Problems	
5	✓ click	css=a:nth-child(1) .ellipsis	

```
Running 'virang 3'                                               14:04:58
1. open on https://leetcode.com/ OK                              14:04:58
2. setWindowSize on 814x808 OK                                   14:04:58
3. Trying to find linkText=Problems... OK                        14:04:58
4. click on linkText=Problems OK                                 14:05:01
5. click on css=a:nth-child(1) .ellipsis OK                      14:05:01
'virang 3' completed successfully                                14:05:06
```

Practical 8

AIM: Install Selenium server and demonstrate it using a script in Java/PHP.

We need to download Selenium RC server / client driver and configure that to Eclipse

1. Download Selenium server: http://seleniumhq.org/download/

2. Download Selenium Client driver for Java (from Selenium ClientDrivers section)

3. Create "Selenium" folder in C: drive and copy the Selenium-server.jaras well as unzip the Selenium Client driver(C:Selenium)

Downloading and unzipping the files into a folder is done. We need to configure the appropriate Selenium Client driver Jar file to the Eclipse.

1. Go to Eclipse –> Click File –> New –> Project (from various options need to select just"project")

2. In Select Wizard –> Click Java –> "Java Project" (demonstrated in the below figure)

3. Give the project name (e.g.SugarCRMTests)

4. Click Finish – Click Yes

5. Now we are done with creation of project and need to configurethe Selenium Client driver to this Project

6. Right Click "SugarCRMTests"project

7. Click "Java Build Path"

8. Click Libraries tab

9. Click "Add External JARs"button

10. Select "Selenium Client Drivers" unzipped in C:Selenium folder(Selenium Server JAR file should not be added.

11. Click OK

12. Referenced libraries –> contains both the Selenium Client driver jar files as shown in the below picture.

1. Install any IDE (Eclipse, IntelliJ, IDEA), create a project.
2. Download the selenium RC file from Selenium website
3. Extract the selenium-remote-control-1.0.3.zip file and add the selenium- remote-control-1.0.3\selenium-java-client-driver-1.0.1\selenium-java- clientdriver.jar and selenium-remote-control-1.0.3\selenium-server- 1.0.3\selenium-server.jar file to the build path.
4. Create a new package and a class in the newly created package.

```
import  com.thoughtworks.selenium.SeleneseTestCase;
public class SeleniumTest1 extends SeleneseTestCase {
   public void setUp() throws Exception { setUp("http://www.hexbytes.com/",
      "*firefox");}
   public void testNew() throws Exception
      { selenium.open("/"); selenium.type("searchbox",
      "selenium rc");
      selenium.click("css=input[type='submit'][value='Searc h']");
      selenium.waitForPageToLoad("30000"); assertTrue(selenium.isTextPresent("selenium
      rc"));
   } }
```

5. Start the selenium server.

Click on the Run As button and run the script as JUnit Test. If you don’t find the option Install the JUnit plug-in and run the script.

6. Install any IDE (Eclipse, IntelliJ, IDEA), create a project.
7. Download the selenium RC file from Selenium website
8. Extract the selenium-remote-control-1.0.3.zip file and add the selenium- remote-control-1.0.3\selenium-java-client-driver-1.0.1\selenium-java- clientdriver.jar and selenium-remote-control-1.0.3\selenium-server- 1.0.3\selenium-server.jar file to the build path.
9. Create a new package and a class in the newly created package.
10. Copy the below code:

```
  import  com.thoughtworks.selenium.SeleneseTestCase;
   public class SeleniumTest1 extends SeleneseTestCase
     { public  void    setUp()    throws    Exception    {
        setUp("http://www.hexbytes.com/", "*firefox");
   }
   public void testNew() throws Exception
      { selenium.open("/"); selenium.type("searchbox",
      "selenium rc");
```

```
        selenium.click("css=input[type='submit'][value='Searc h']");
        selenium.waitForPageToLoad("30000"); assertTrue(selenium.isTextPresent("selenium
        rc"));
    }
}
```

11. Start the selenium server.

Click on the Run As button and run the script as JUnit Test. If you don’t find the option Install the JUnit plug-in and run the script.

Practical 9

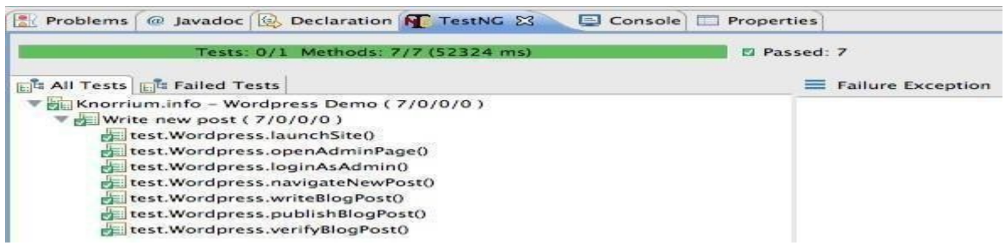
AIM: Write and test a program to login a specific web page. THEORY:
TestNG Framework: Inspired by JUnit and NUnit with additional functionalities.

Features:

- Annotations.
- Support for multi-threaded tests.
- Flexible test configurations.
- Data-driven testing via `@DataProvider`.
- Parameterized tests and powerful execution models.
- Compatible with Eclipse, IDEA, Maven, and other tools.
- Supports all test types (unit, functional, end-to-end, integration).
- Embedded BeanShell for more flexibility.
- Uses JDK functions for logging and runtime (no external dependencies).

Installing TestNG in Eclipse:
- Go to Help > Software Updates > Find and Install.
- For Eclipse 3.4+: use http://beust.com/eclipse.
- For Eclipse 3.3 and below: use http://beust.com/eclipse1.
- Follow the installation steps guided by Eclipse.

Launching Tests in Eclipse:
- You can launch tests via command line, Eclipse plugin, or programmatically.
- Once TestNG is installed, right-click on the XML file, select **Run as TestNG Suite** to execute tests.



```
Tests: 0/1  Methods: 7/7 (52324 ms)    Passed: 7
Knorrium.info - Wordpress Demo ( 7/0/0/0 )
  Write new post ( 7/0/0/0 )
    test.Wordpress.launchSite()
    test.Wordpress.openAdminPage()
    test.Wordpress.loginAsAdmin()
    test.Wordpress.navigateNewPost()
    test.Wordpress.writeBlogPost()
    test.Wordpress.publishBlogPost()
    test.Wordpress.verifyBlogPost()
```

Selenium Tests with Microsoft Excel:
- **Parameterization:** Recommended to handle large test data by external sources like Excel.
- **Jxl.jar:** Open-source Java API for reading and writing Excel spreadsheets.
  - **Operations Supported:**
    1. Read/write data and formulas.
    2. Generate spreadsheets.
    3. Formatting of fonts, numbers, and dates.
    4. Cell coloring.

Integration: Add `jxl.jar` to Eclipse's Java Build Path to access API methods.

CODE:

```
import com.thoughtworks.selenium.*; import org.junit.After; import
org.junit.Before; import org.junit.Test;
import java.util.regex.Pattern;

public class exp5 extends SeleneseTestCase
{ @Before
public void setUp() throws Exception {
selenium = new DefaultSelenium("localhost", 4444, "*chrome",
"http://demo.opensourcecms.com/");
selenium.start();
}
@Test
public void testExp5() throws Exception { selenium.open("/wordpress/wp-login.php");
selenium.type("id=user_login", "admin"); selenium.type("id=user_pass", "demo123");
selenium.click("id=wp-submit"); selenium.waitForPageToLoad("30000");
}
@After
 public void tearDown() throws Exception { selenium.stop(
```

Practical 10

AIM: Write and test a program to provide total numbers of objects present on the page.

```
package testscripts;
import com.thoughtworks.selenium.*;
import org.openqa.selenium.server.*;
import org.testng.annotations.*; public
class exp8 {
   public Selenium selenium;
   public SeleniumServer seleniumserver;
   @BeforeClass
   public void setUp() throws Exception {

      RemoteControlConfiguration rc = new RemoteControlConfiguration(); seleniumserver =
      new SeleniumServer(rc);
      selenium = new DefaultSelenium("localhost", 4444, "*firefox", "http://");
      seleniumserver.start();
      selenium.start();
   }
   @Test
   public void testDefaultTNG() throws Exception { selenium.open("http://
      www.google.co.in/"); selenium.windowMaximize();
      String lc[] = selenium.getAllLinks();
      System.out.println("TOTAL NO OF LINKS=" + lc.length);

      String bc[] = selenium.getAllButtons(); System.out.println("TOTAL
      NO OF BUTTONS=" + bc.length); String fc[] =
      selenium.getAllFields();
      System.out.println("TOTAL NO OF INPUT FIELDS=" + fc.length);
   }
   @AfterClass
   public void tearDown() throws Exception
         { selenium.stop();

}}
```

OUTPUT: TOTAL NO OF LINKS = 41 TOTAL NO OF BUTTONS = 1 TOTAL NO OF INPUT FIELDS=3

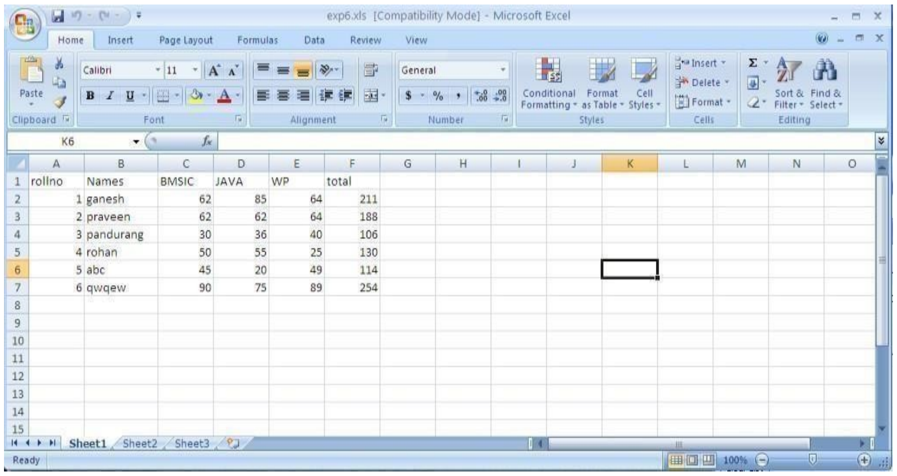
Practical 11

AIM: Write and test a program to update 10 student records into table into Excel file. CODE:

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import jxl.Sheet;
import jxl.Workbook;
import jxl.write.Label;
import  jxl.write.WritableSheet;
importjxl.write.WritableWorok;
import  org.testng.annotations.*;
public class newj {
  @BeforeClass
  public void setUp() throws Exception {} @Test
  public void testImportexport1() throws Exception
    { FileInputStream fi = new FileInputStream("D:\\exp6.xls");
    Workbook w = Workbook.getWorkbook(fi);
    Sheet s = w.getSheet(0);
    String a[][] = new String[s.getRows()][s.getColumns()];
    FileOutputStream fo = new FileOutputStream("D://exp6Result.xls");
    WritableWorkbook wwb = Workbook.createWorkbook(fo);
    WritableSheet ws = wwb.createSheet("result1", 0);
    for (int i = 0; i < s.getRows(); i++)
      for (int j = 0; j < s.getColumns(); j++) { a[i]
        [j] = s.getCell(j, i).getContents(); Label
        l2 = new Label(j, i, a[i][j]);
        ws.addCell(l2);
```

```
            Label l1 = new Label(6, 0, "Result");
            ws.addCell(l1);
         }
      for (int i = 1; i < s.getRows(); i++) {
         for (int j = 2; j < s.getColumns(); j++)
            { a[i][j] = s.getCell(j, i).getContents();
            intx = Integer.parseInt(a[i][j]);
            if (x > 35) {
               Label l1 = new Label(6, i, "pass");
               ws.addCell(l1);
            }
            Else
            {
               Label l1 = new Label(6, i, "fail");
               ws.addCell(l1);
               break;
              }}}
        wwb.write();
        wwb.close();
 }}
```

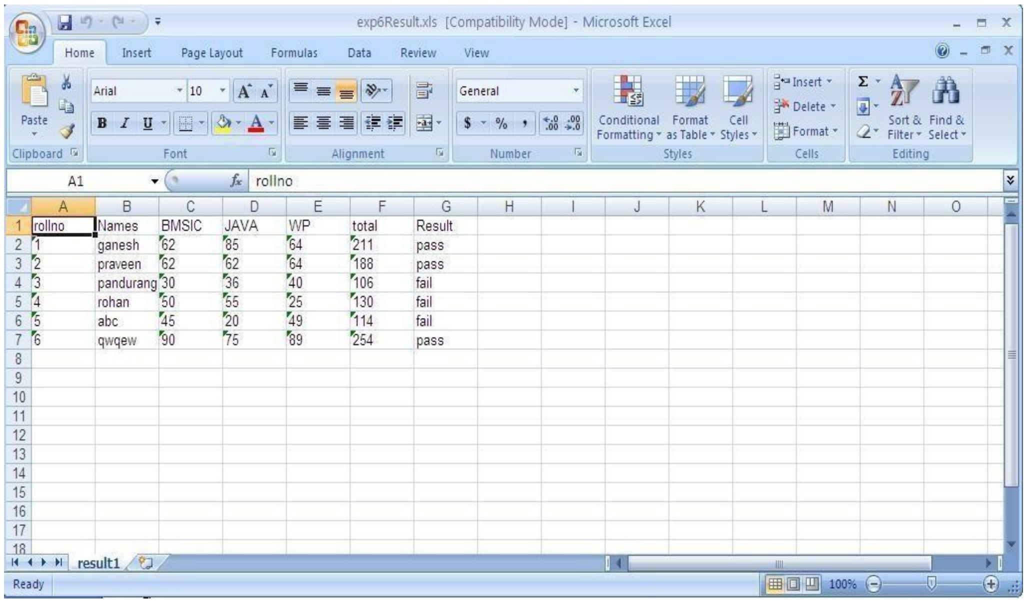
INPUT:



exp6.xls [Compatibility Mode] - Microsoft Excel

rollno	Names	BMSIC	JAVA	WP	total
1	ganesh	62	85	64	211
2	praveen	62	62	64	188
3	pandurang	30	36	40	106
4	rohan	50	55	25	130
5	abc	45	20	49	114
6	qwqew	90	75	89	254

OUTPUT:



exp6Result.xls [Compatibility Mode] - Microsoft Excel

rollno	Names	BMSIC	JAVA	WP	total	Result
1	ganesh	62	85	64	211	pass
2	praveen	62	62	64	188	pass
3	pandurang	30	36	40	106	fail
4	rohan	50	55	25	130	fail
5	abc	45	20	49	114	fail
6	qwqew	90	75	89	254	pass