STMT Assignment-I

B Ravikumar
2209IA05C7
III CSE-A3

1 a) Explain about the importance of test design in testing process.

b) Explain about the importance of testing to improve the quality.

a) importance of test design in testing process:

- Test themselves are needed to be designed as like how the code built be designed and tested.
- Test cases must be attempted without prior analysis of the program or requirements or structure.
- If the test design doesn't follow the above scenario then it will result into a disorganized series of ad-lib (unplanned) cases that are not documented either before or after the tests are executed.
- Because ad-lib test are not formally designed. they cannot be precisely repeated and no one is sure whether they was a bug or not.
- ad-lib test are useful during debugging, where their primary purpose is to help locate the bug, but ad-lib test done in support of debugging, no matter how exhausting, are not substituted for designed tests.
- The design phase of programming should be explicitly specified.
- Instead of "design, code, desk check, test and debug", the programming process should described as "define, test design, code, test code, Program inspection, test Inspection, test debugging, test execution, program debugging, testing".

b) Importance of Testing to improve Quality.

Testing is process for ensuring the development of high-quality software products by identifying defects early and improving their satisfact. it importance include

1. defect detection: Testing ensures the bugs, glitches, and performance issue are identified and resolved before release.

2. Enhanced Reliability: A well-tested product performs as expected under various conditions.

3. Cost Efficiency: Detecting and fixing issues early in the development cycle reduce the cost of post-release maintenance and support.

4. Customer Satisfaction: Quality assurance leads to fewer complaints, better user Experience, and higher Customer Satisfaction.

5. Security Assurance: Testing helps identify vulnerabilities that could compromise sensitive information or lead to security breaches.

7a) a) write the control flow graph for matrix multiplication

b) Explain about loop taking time

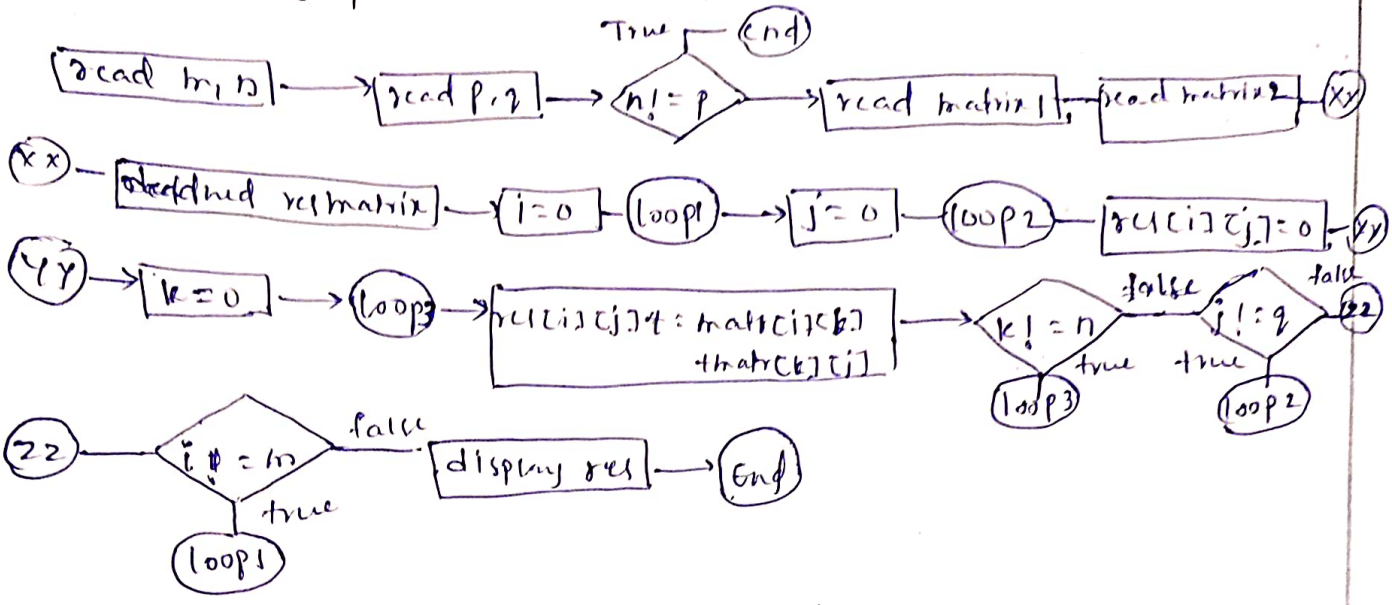
c) Explain about multi entry/multi exit routines.

a) Control flow graph for matrix multiplication

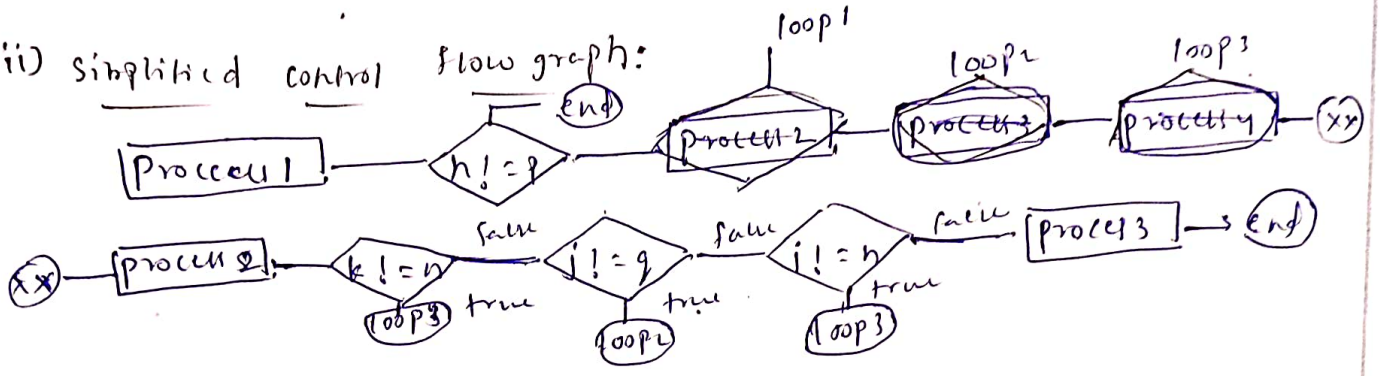
algorithm:

```
read m, n;
read p, q
if (n != p) goto END
   read mat1[m][n];
   read mat2[p][q];
   define ru[m][q];
    for i = 0 to m
        for j = 0 to q
            ru[i][j] = 0;
            for k = 0 to n
                ru[i][j] = ru[i][j] + mat1[i][k] * mat2[k][j];
            end for
        end for
display ru_s → END: exit
```

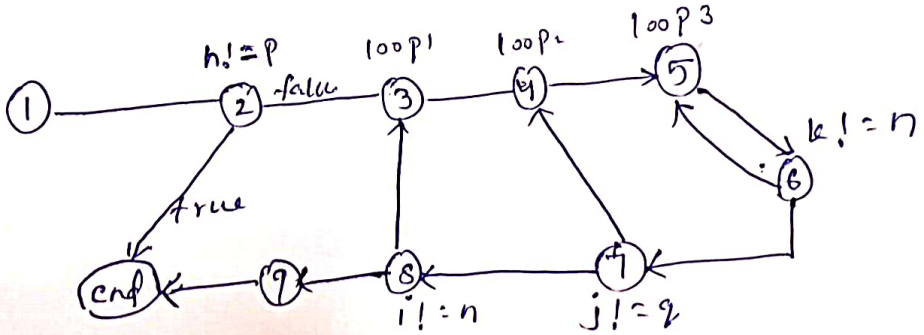
i) Control flow graph for matrix multiplication



ii) Simplified control flow graph:



iii) CFG - Simplified control flow graph:



(b) Loop Testing:

Loop testing is a type of software testing focused on validating the behaviour and performance of loops with in a program. It is a part of white-box testing techniques used to alter loops in different configurations and ensure that they work as intended.

Types of loops in loop testing:

1. Simple Loop: Loops that executes a single block of code with a fixed or variable number of iterations

2. Nested Loops: Loops within loops, where the inner loop executes completely for each iteration of the outer loop.

3. Concated Loops:
Sequential loops that do not depend on each other but execute one after another.

4 Unstructured Loops:
Loops with complex logic or conditions, seen in poorly designed code.

Loop Testing Techniques:

1. Zero iteration: Test the condition when loop not executed at all

2. One iteration: Ensure the loop runs exactly once without any issues.

3. Multiple iterations: Validate the loop's behaviours with different number of iterations.

4. Boundary Testing: Check conditions at the boundaries of the loop to catch potential off-by-one errors.

2c) Multi-entry / multi-exit routine:

i) Multi-entry Routine:

A multi-entry routine allows that code to be entered at multiple points during execution, rather than just the beginning of a single entry point.

Characteristics:

→ provides flexibility for entering a function at different stages

→ often found in low-level or older programming paradigms.

→ can make debugging and maintenance is challenging

3. Multi-exit Routine:

A multi-exit routine is a routine that can terminate and return control to the calling function from multiple points in its code. This is more common and seen in structured programming languages.

Characteristics:

→ Used for handling special conditions or early exits efficiently

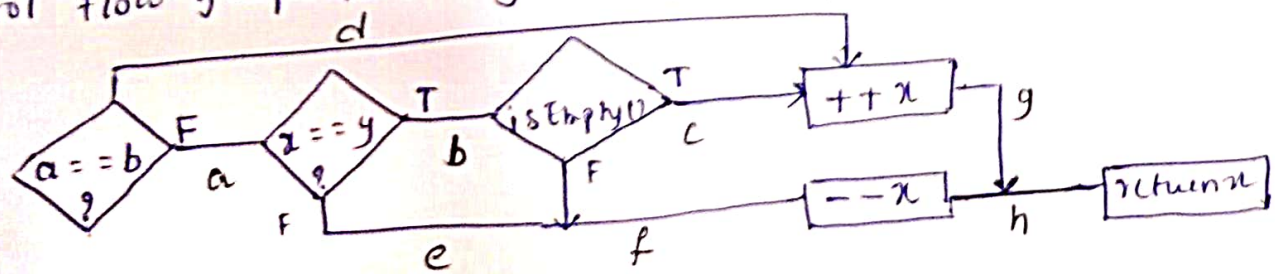
→ Common in modern programming for error handling.

3a)

```
int foo (int x)
{
    if((a == b) || (x == y && isEmpty()))
    {
        ++x;
    }
    else
    {
        --x;
    }
    return x;
}
```

What test case will give 100% branch coverage? Does this achieve 100% multiple conditional coverage? which coverage criteria should be used here to maximize coverage?

Ans: Control flow graph for the given code snippet



Path	Predicates			Segment paths.							
	pred1	pred2	pred3	a	b	c	d	e	f	g	h
dg	true	-	-				✓			✓	
abcgh	false	True	True	✓	✓	✓				✓	✓
abfh	false	True	false	✓	✓				✓		✓
aefh	false	false	-	✓				✓	✓		✓

* The following test cases according to the conditional flow graphs will give the 100% branch coverage for give code snippen

dg (a==b) true — true — true — true

abcgh (a==b) false (x==y) && (isEmpty()) true

abfh (a==b) false (x==y) && (isEmpty()) false — false

aefh → (a==b) false (x==y) && ⇒ false
true
false

* It does not 100% multi-condition coverage. Because of the logical or (||). It will evaluate (a==b) condition only when it is true.

* Path coverage criteria is used for the to maximize the coverage.