Software Quality Testing

Presented by: Ritesh Jain

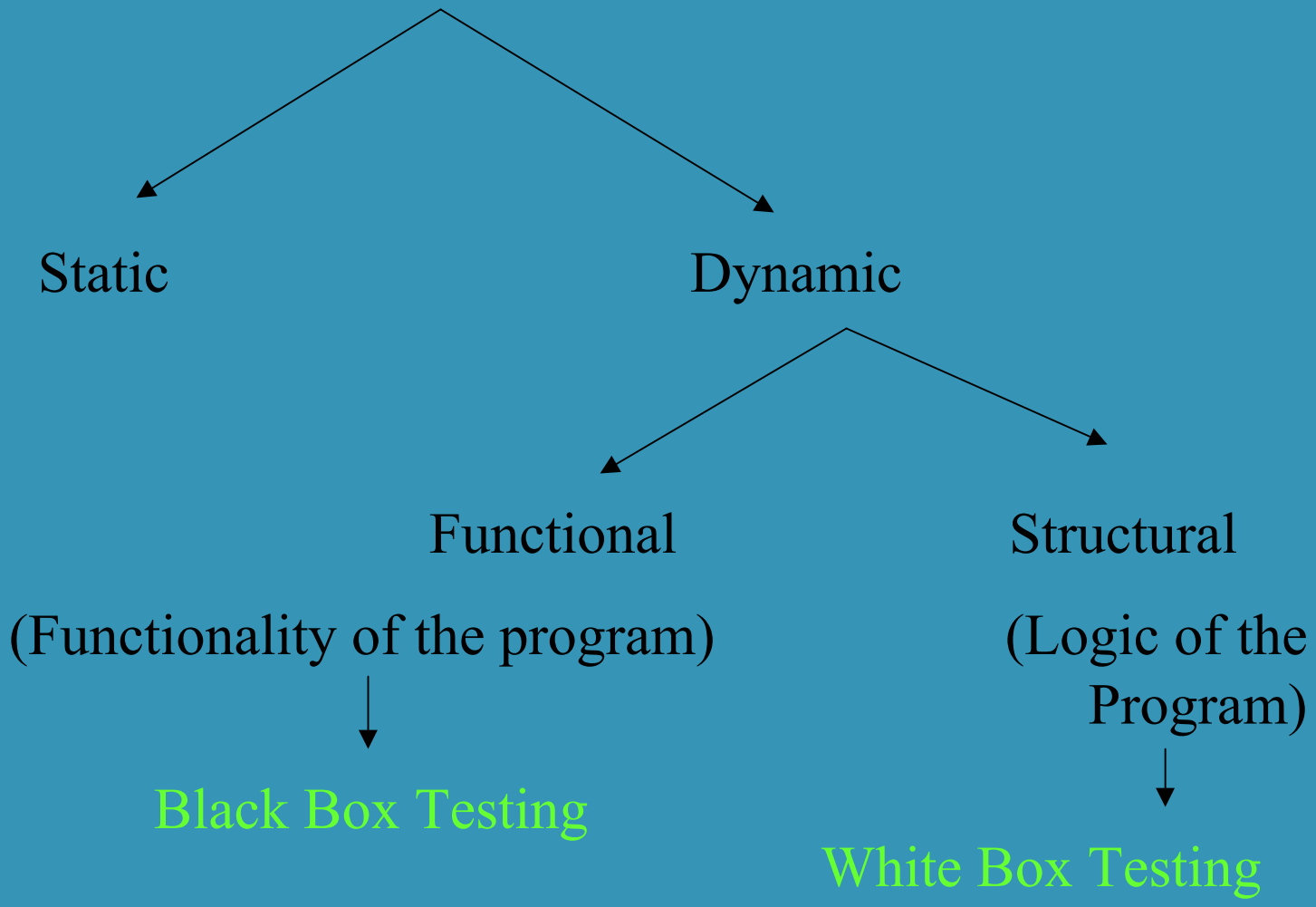
Date: 16-Jun-05

Strategies & Methodologies of testing

- White Box.
- Black Box.

Testing

- Testing



Black Box Testing Methods.

1.Functional Testing
2.Load Testing
3.Stress Testing
4.Ad-hoc Testing
5.Exploratory Testing
6.Usability Testing
7.Smoke Testing
8.Recovery Testing
9.Volume Testing.
10 Regression Testing.
11 User Acceptance Testing
12 Alpha Testing
13 Beta Testing

White Box Testing

Focus: thoroughness (coverage) every statement in the component is executed at least once.

Four types of White Box testing techniques

1.Statement Testing

2.Loop Testing

3.Path Testing

4.Branch Testing.

Techniques of white box testing

- Memory –access & memory-management error detection.
- Memory leaks
- Uninitialized memory reads
- Array-bounds errors
- Memory allocation errors
- Garbage-collection issues (JAVA)
- Internationalization errors

  embedded string literals

  Use of unsafe functions

  Pointer arithmetic

  Time , Date, Number, Currency functions.

Performance profiling

Fault Injection

Static Analysis.

I Statement Testing (Algebraic Testing): Test single statements (choice of operators in polynomials)

II Loop Testing

1Simple Loop

2Nested Loop

3Concatenated Loop

4Unstructured Loop

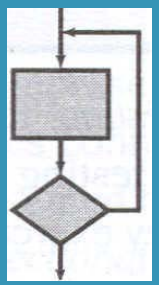
III Path Testing

IV Branch Testing.

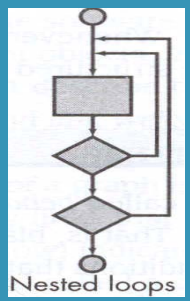
V Mutation Testing

Types of Loop & their Testing.

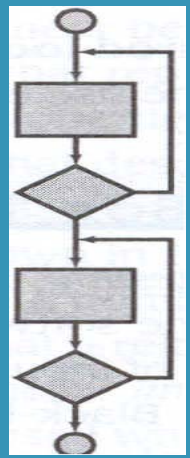
Simple Loop



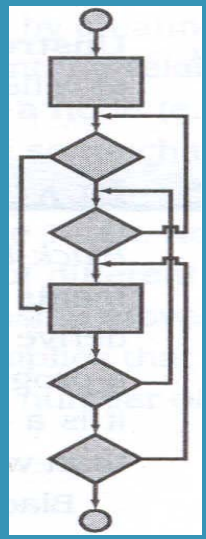
Nested Loop



Concatenated Loop



Unstructured Loop



Testing Simple Loop

- The following sets of tests can be applied to simple loops, where ‘n’ is the maximum number of allowable passes through the loop.
- 1. Skip the loop entirely.
- 2. Only one pass through the loop.
- 3. Two passes through the loop.
- 4. ‘m’ passes through the loop where m<n.
- 5. n-1, n, n+1 passes through the loop

Testing Nested loop

- If we extend the test approach from simple loops to nested loops, the number of possible tests would grow geometrically as the level of nesting increases.
- 1. Start at the innermost loop. Set all other loops to minimum values.
- 2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter values. Add other tests for out-of-range or exclude values.
- 3. Work outward, conducting tests for the next loop, but keep all other outer loops at minimum values and other nested loops to “typical” values.
- 4. Continue until all loops have been tested.

Testing Concatenated loop

- Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other. However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent.

Testing Unstructured loop

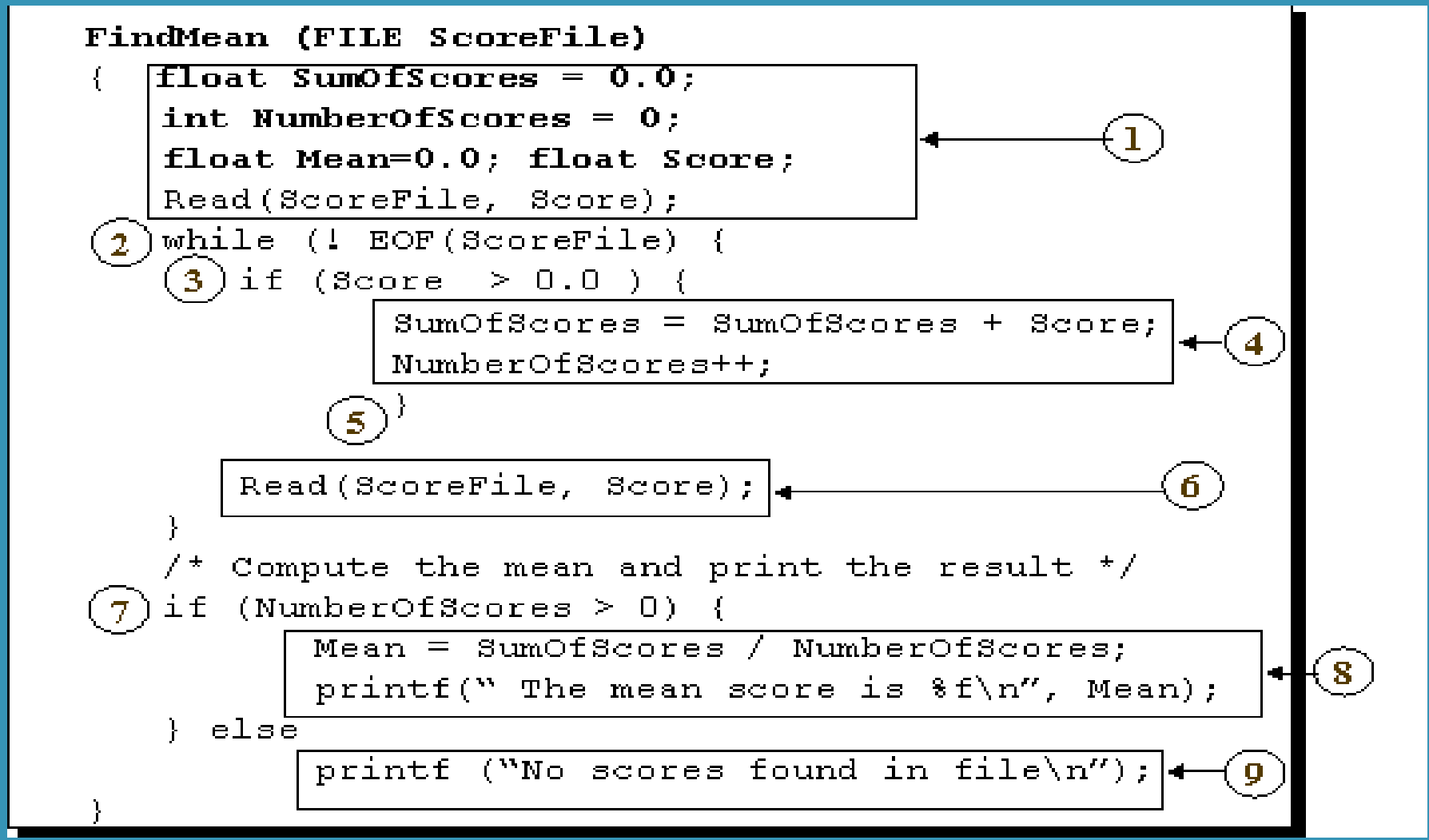
- Whenever possible, this class of loops should be redesigned to reflect the use of the structured programming constructs.

Path Testing

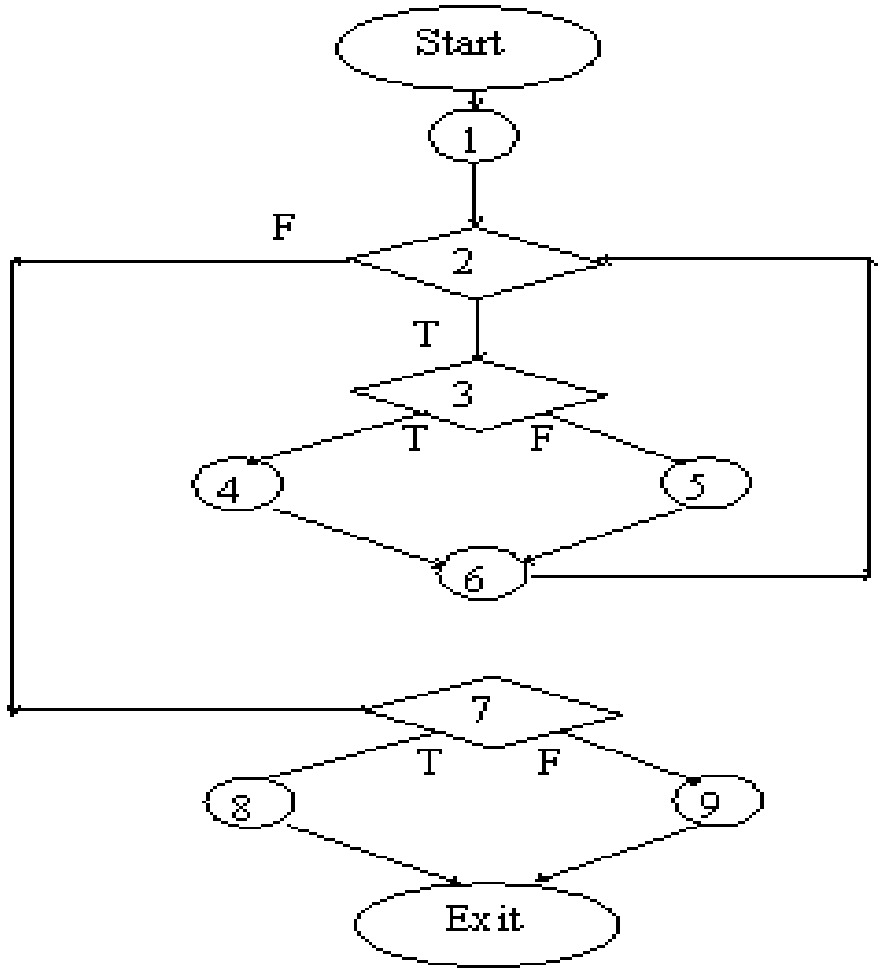
- Make sure all the path should be executed at least once.

```
FindMean(float Mean, FILE ScoreFile)
{ SumOfScores = 0.0; NumberOfScores = 0; Mean = 0;
 Read(ScoreFile, Score);  /*Read in and sum the scores*/
 while (! EOF(ScoreFile) {
                if ( Score > 0.0 ) {
                 SumOfScores = SumOfScores + Score;
                NumberOfScores++;
               }
 Read(ScoreFile, Score);
 }
 /* Compute the mean and print the result */
 if (NumberOfScores > 0 ) {
              Mean = SumOfScores/NumberOfScores;
              printf("The mean score is %f \n",  Mean);
 } else
              printf("No scores found in file\n");
}
```

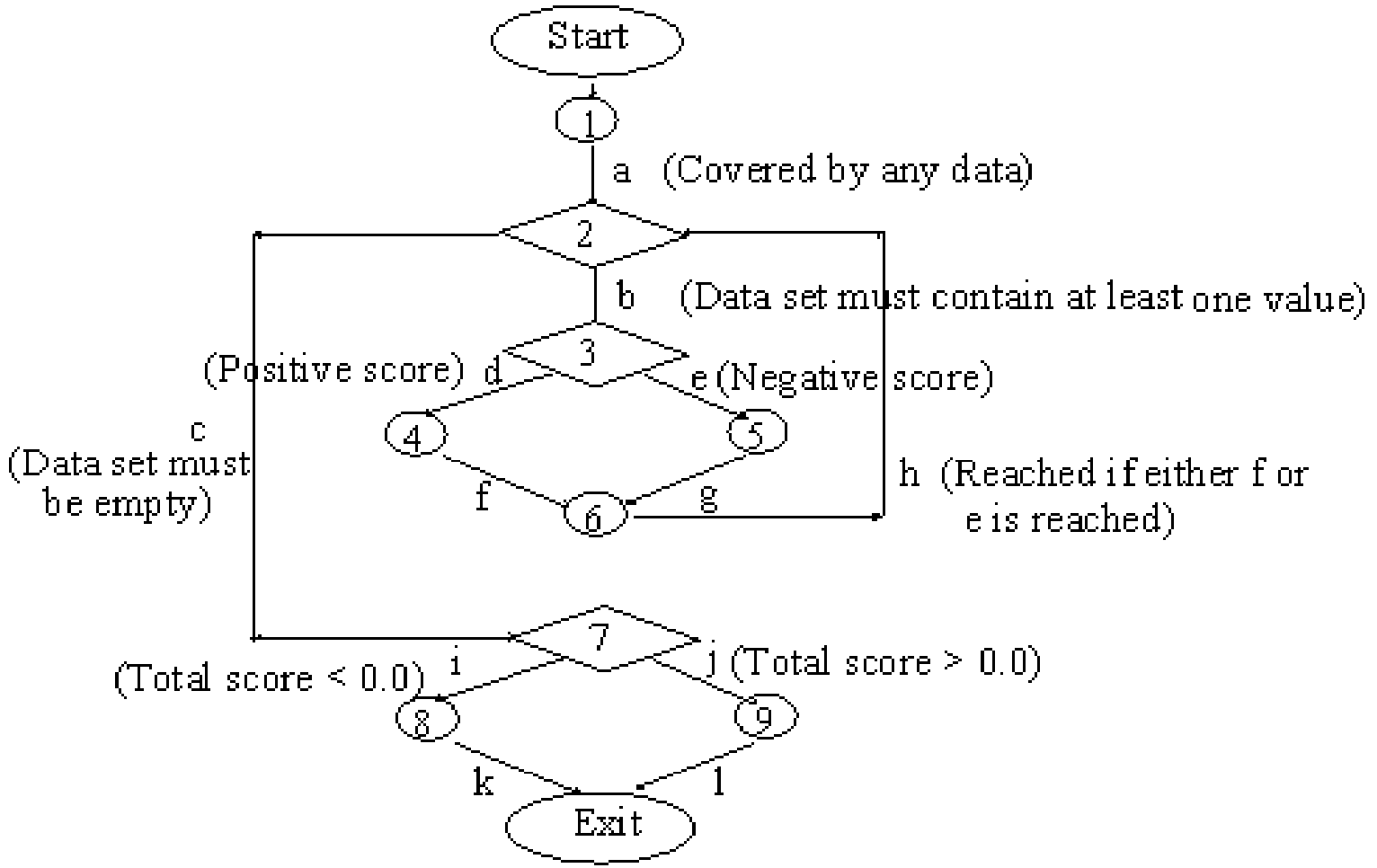
Path Testing



Constructing the logical flow diagram



Finding the test cases

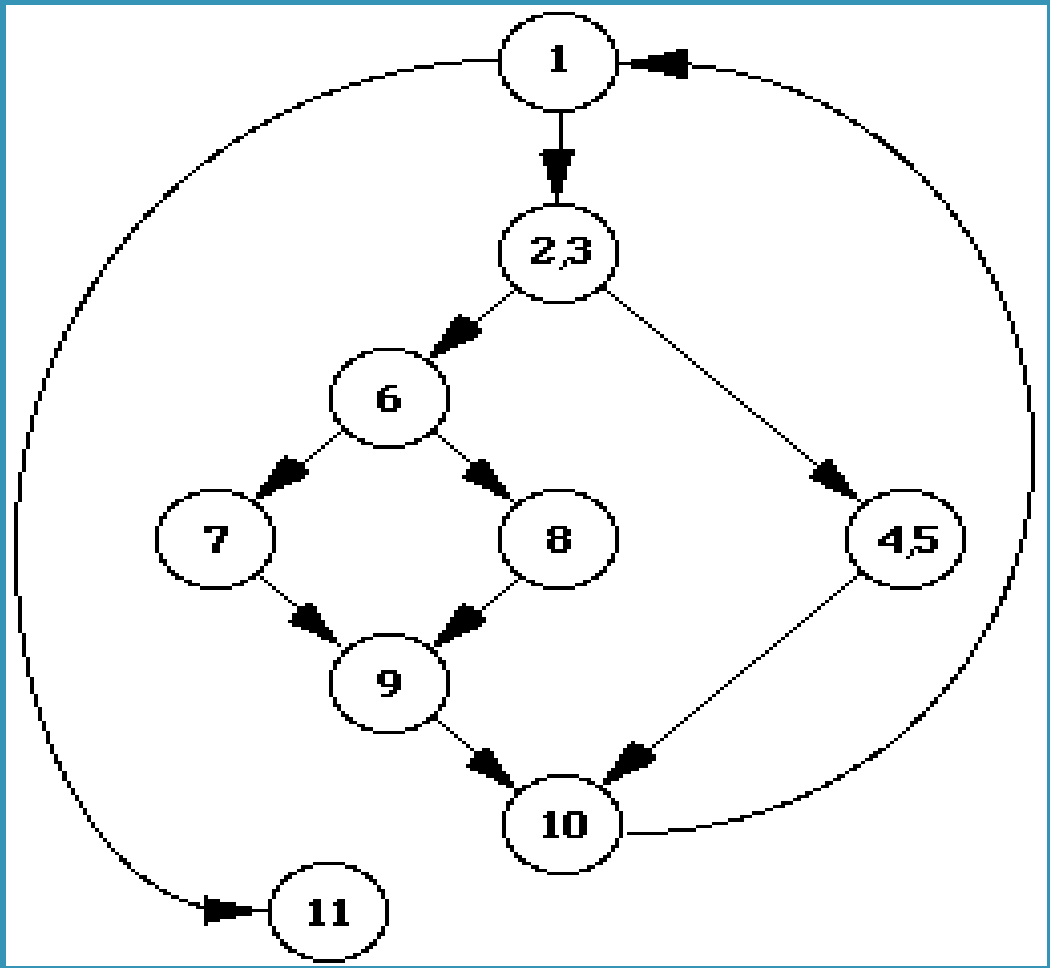


Cyclomactic Complexity

- Cyclomatic complexity is a measure of software complexity first introduced by Thomas McCabe in 1976 and measures the number of linearly-independent paths through a program module.
- How to measure the Cyclomactic Complexity.
- V(G)= No of Regions
- V(G)=E-N+2
- V(G)=P+1

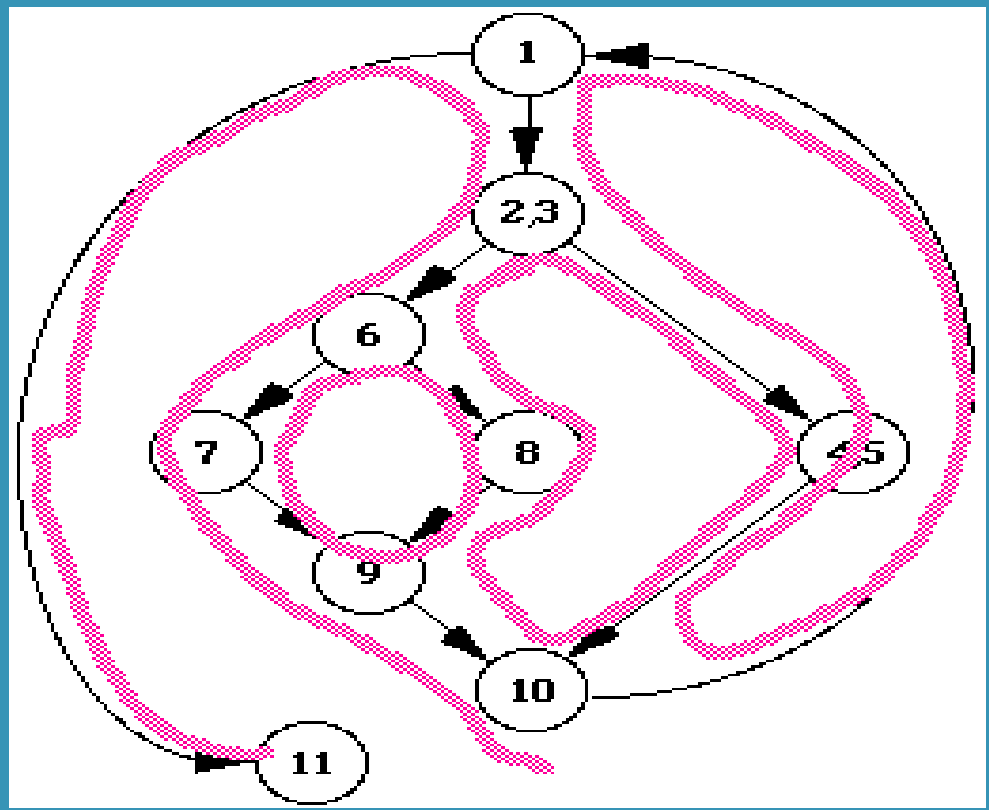
Where V(G) Represents the cyclomactic complexity,E represents the no of edges, N represents the no of node, P represents the predicate node.

Ex.



Solutions

No of Regions is 4.



On the basis of Edges & Nodes

- V(G)=E-N+2
- No of Edges =11
- No of Nodes=9
- V(G)=11-9+2
- V(G)=4

On the basis of Predicate nodes

- V(G)=P+1
- Predicate Node = 3(1,2.3,6)
- V(G)=3+1
- V(G)=4.

Devise a test plan

- A program reads 3 integer values. The 3 values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.

- Write test cases that would adequately test this program.

Test Cases

- 1. Valid scalene (5, 3, 4) => scalene
- 2. Valid isosceles (3, 3, 4) => isosceles
- 3. Valid equilateral (3, 3, 3,) => equilateral
- 4. First permutation of 2 sides (50, 50, 25) => isosceles
- 5. Second perm of 2 sides (25, 50, 50) => isosceles
- 6. Third perm of 2 sides (50, 25, 50) => isosceles
- 7. One side zero (1000, 1000, 0) => invalid

Test Cases

- 8. One side has negative length (3, 3, -4) => invalid
- 9. first perm of two equal sides (5, 5, 10) => invalid
- 10. Second perm of 2 equal sides (10, 5, 5) => invalid
- 11. Third perm of 2 equal sides (5, 10, 5) => invalid
- 12. Three sides >0, sum of 2 smallest < largest (8,2,5) => invalid
- 13. Perm 2 of line lengths in test 12 (2, 5, 8) => invalid
- 14. Perm 3 of line lengths in test 12 (2, 8, 5) => invalid

Test Cases

- 15. Perm 4 of line lengths in test 12 (8, 5, 2) => inv
- 16. Perm 5 of line lengths in test 12 (5, 8, 2) => inv
- 17. Perm 6 of line lengths in test 12 (5, 2, 8) => inv
- 18. All sides zero (0, 0, 0) => inv
- 19. Non-integer input, side a (@, 4, 5) => inv
- 20. Non-integer input, side b (3, $, 5) => inv
- 21. Non-integer input, side c (3, 4, %) => inv

Test Cases

- 22. Missing input a (, 4, 5) => invalid
- 23. Missing input b (3, , 5) => invalid
- 24. Missing input c (3, 4,) => invalid
- 25. Three sides > 0, one side equals the sum of the other two (12, 5, 7) => inv
- 26. Perm 2 of line lengths in test 25 (12, 7, 5) => inv
- 27. Perm 3 of line lengths in test 25 (7, 5, 12) => inv
- 28. Perm 4 of line lengths in test 25 (7, 12, 5) => inv
- 29. Perm 5 of line lengths in test 25 (5, 12, 7) => inv
- 30. Perm 6 of line lengths in test 25 (5, 7, 12) => inv

Test Cases

- 31. Three sides at max values (32767, 32767, 323767) => inv
- 32. Two sides at max values (32767, 32767, 1) => inv
- 33. One side at max values (32767, 1, 1) => inv

White Box Testing Tools

- Purify by Rational Software Corporation
  - Run-time error and memory-leak detection
  - C/C++, FORTRAN and Java
  - UNIX, NT, and W2K
- Insure++ by ParaSoft Corporation
  - Run-time error and memory-leak detection
  - Add-on module TCA measures code coverage.
  - C/C++
  - UNIX, NT, and W2K
- Quantify by Rational Software Corporation
  - Performance profiling
  - C/C++, Java, and VB
  - UNIX, NT, and W2K
- Expeditor by OneRealm Inc.
  - Source code scanner for I18N errors
  - C/C++, Java, and HTML
  - UNIX, NT, and W2K

Queries.