

# **SOFTWARE TESTING AND QUALITY ASSURANCE**

**ICT3075**

**Course Lecturer: Dr. KIMBI  
Xaveria**

# STRUCTURAL TESTING TECHNIQUES

- In structural testing, the software is viewed as a white box and test cases are determined from the implementation of the software.
- Structural testing techniques include:
  - (1) Control Flow Testing
  - (2) Data Flow Testing.

# CONTROL FLOW TESTING

- Control flow testing uses the control structure of a program to develop the test cases for the program.
- The test cases are developed to sufficiently cover the whole control structure of the program.
- The control structure of a program can be represented by the control flow graph of the program.

# White Box Testing/Structural Testing

- Testing based on program source code
- Extent to which (source) code is executed, i.e. Covered
- Different kinds of coverage: (i.e the extent to which the code has been covered)
  - Path Coverage Criterion
  - Statement Coverage Criterion
  - Condition/Branch/Decision Coverage Criterion
  - Predicate Coverage Criterion

# CONTROL FLOW GRAPH (CFG)

- The adequacy of the test cases is measured with a metric called coverage (**Coverage** is a measure of the completeness of the set of test cases) (i.e the extent to which the code has been covered)
- A **test coverage criterion** measures the extent to which a set of test cases covers a program.
- Control Flow Graph is a graphical representation of a program Unit. I.e **A CFG is a graphical representation of a program control structure.**
- A Control Flow Graph for a program consists of a node for each statement in the program and edges representing the flow of control between statements.
- The control flow graph  **$G = (N, E)$**  of a program consists of a set of **Nodes N** and a set of **Edge E**.
- Each node represents a set of program statements.
- There is a unique entry node and a unique exit node.
- There is an edge from node  $n1$  to node  $n2$  If the control may flow from the last statement in  $n1$  to the first statement in  $n2$
- .

- Some of the limitations of control flow testing are:
- 
- (1) Control flow testing cannot catch all initialization mistakes.
- (2) Requirement Specification mistakes are not caught by control flow testing.
- (3) **Therefore,** It's unlikely to find missing paths and features if the **program and the model on which the tests are based are done by the same person.**
-

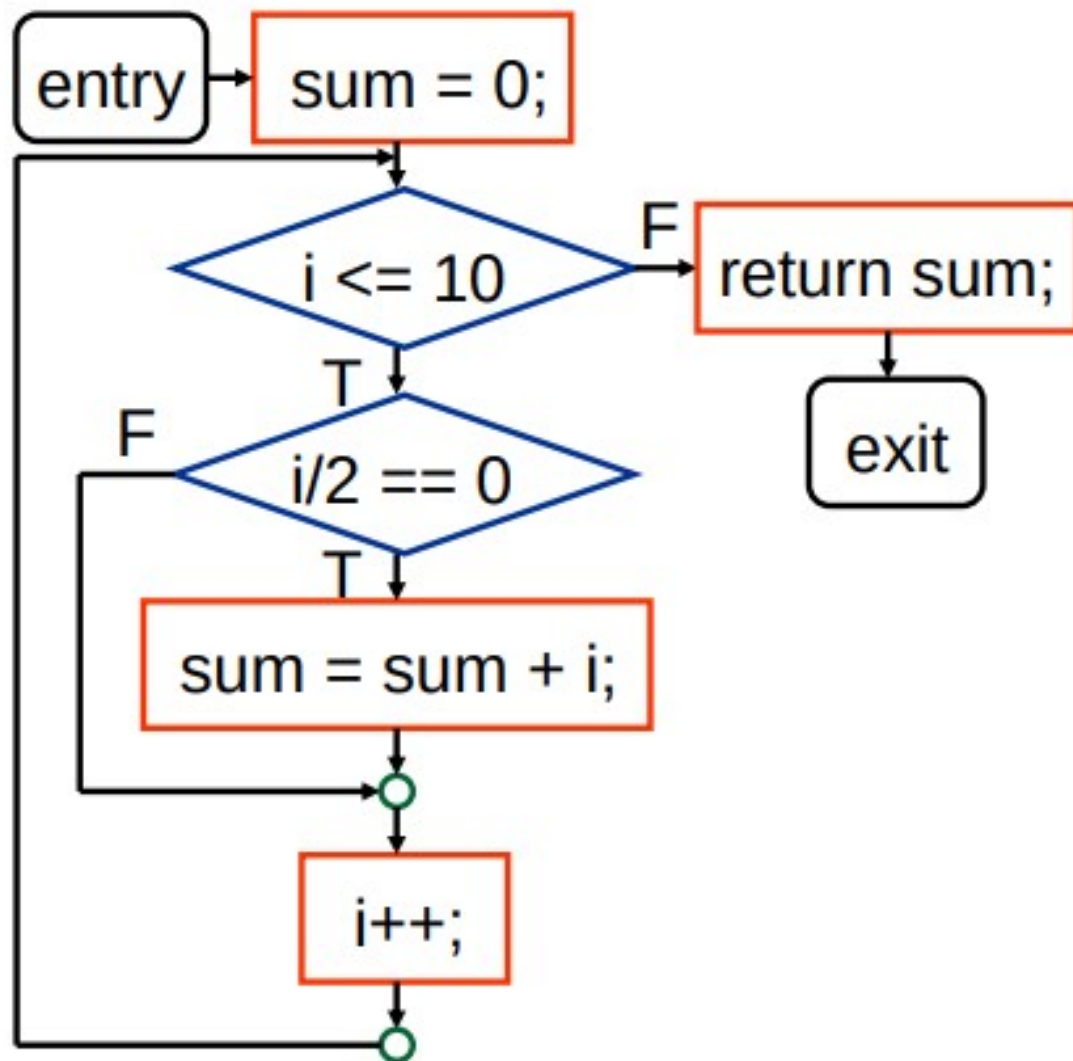
# Path Testing

- Path testing is a **Structural Testing method** that involves using the source code of a program to attempt to find every possible executable path.
- The idea is that are able to test each **individual path from source code** is as many way as possible in order to maximize the coverage of each test case.
- Therefore, we use **knowledge of the source code** to define the test cases and to examine outputs.
- Test cases derived to exercise the basis set are guaranteed to execute **every statement** in the program **at least one time** during testing

# Control Flow Graph: An Example

```
int evensum(int i)
{
    int sum = 0;

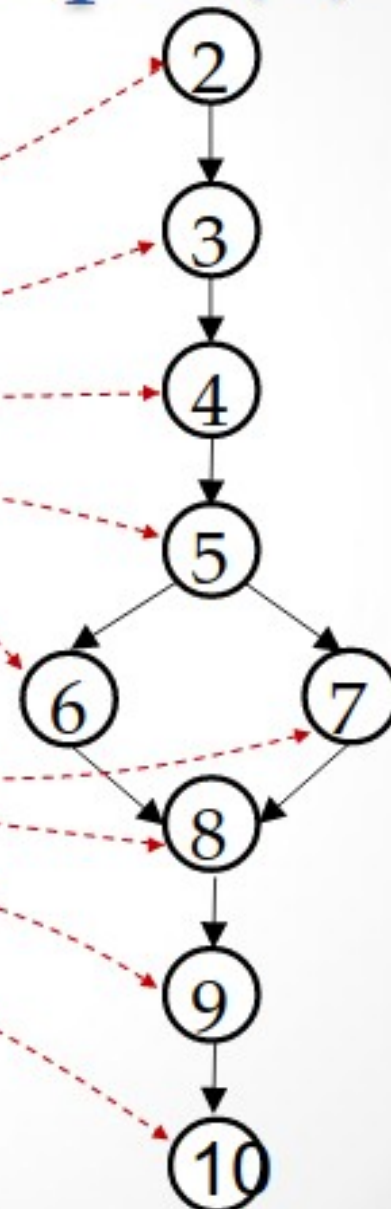
    while (i <= 10) {
        if (i/2 == 0)
            sum = sum + i;
        i++;
    }
    return sum;
}
```





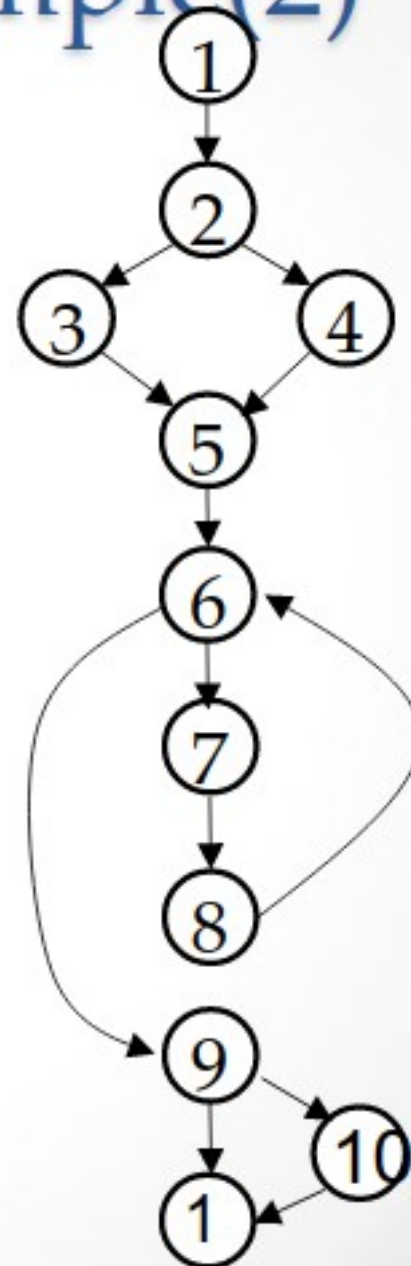
# Flow Graph Example(1)

1. Program 'Simple Subtraction'
2. Input(x,y)
3. Output (x)
4. Output(y)
5. If  $x > y$  then Do
6.  $x - y = z$
7. Else  $y - x = z$
8. EndIf
9. Output(z)
10. Output "End Program"



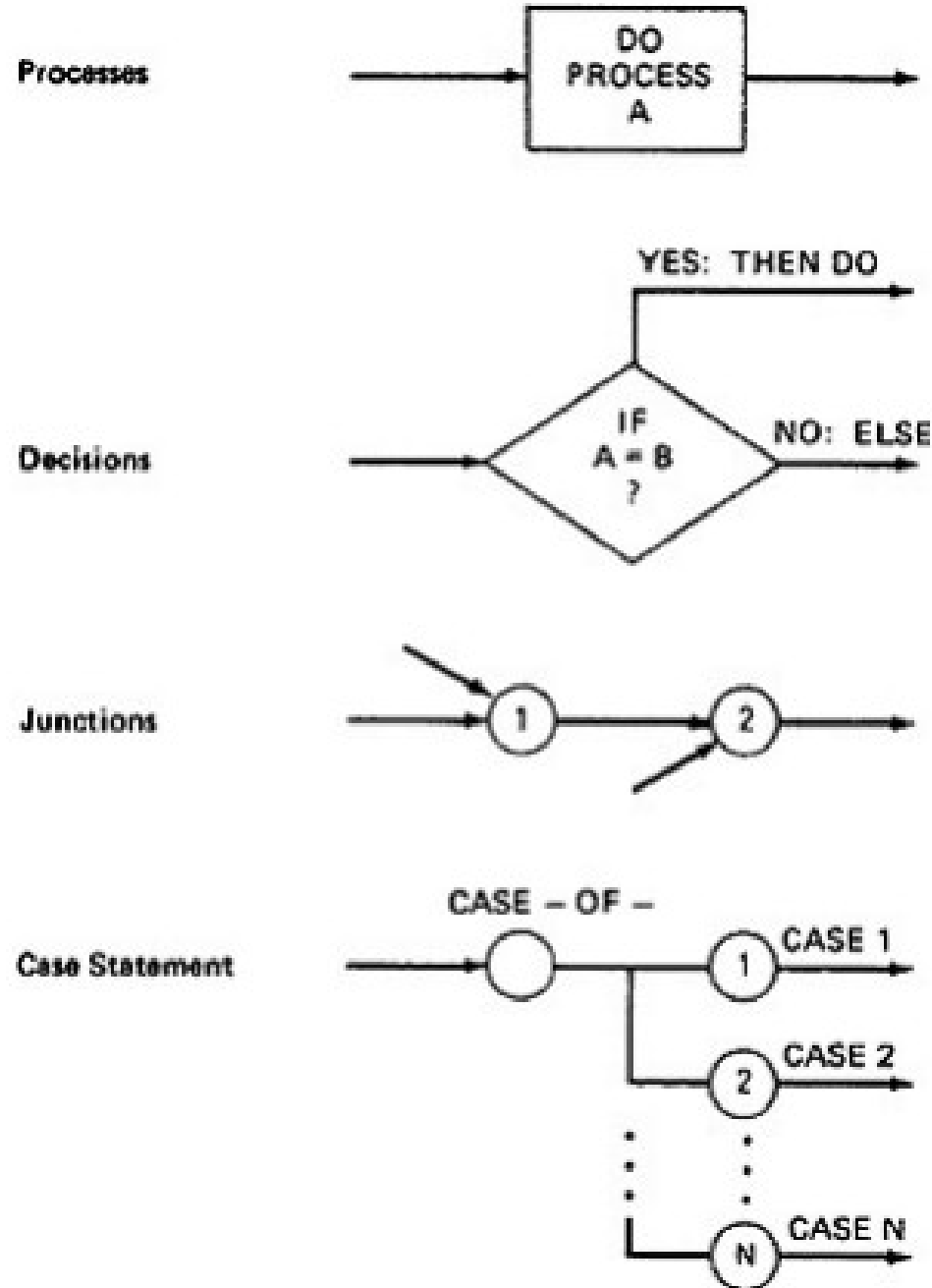
# Flow Graph Example(2)

```
1. scanf("%d %d",&x, &y);  
2. If (y<0)  
3.   pow = -y;  
   else  
4. pow = y;  
5. Z = 1.0  
6. While (pow != 0) {  
7.   z = z*x;  
8.   pow = pow -1; }  
9. If (y <0)  
10. z = 1.0 /z;  
11. printf("%f",z);
```



A flow graph contains four different types of elements (Nodes).

- Process Block
- Decisions
- Junctions
- Case Statements

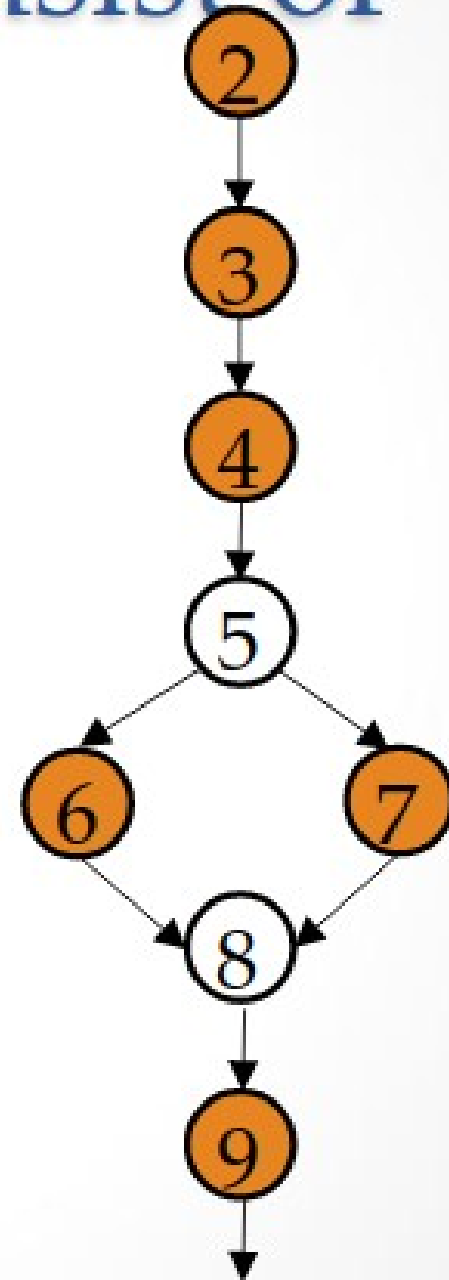


- A **process block (Process Link)** is a sequence of program statements uninterrupted by either decisions or junctions.
- A **decision node** is a program point at which the control flow can diverge. I.e A decision node contains a conditional statement that creates 2 or more control branches (e.g. if or switch statements).
- A **junction node** is a point in the program where the control flow can merge
- A **case node** is a multi-way branch or decisions.

# Flow Graphs consist of

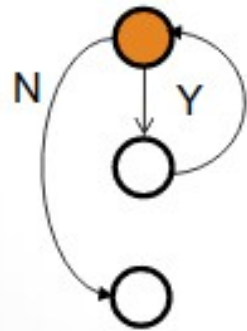
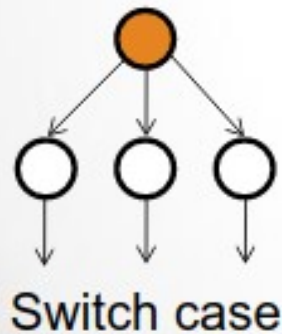
- A **process block** is a sequence of program statements uninterrupted by either decisions or junctions.
- (i.e. straight-line code)

Sequence

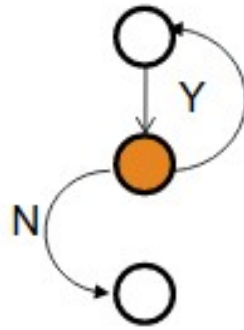


# Flow Graphs consist of

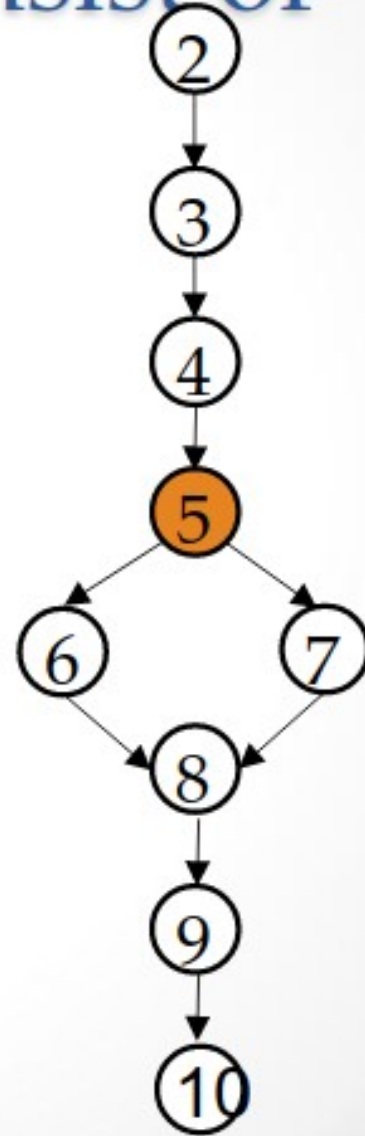
- A **decision** is a program point at which the control can diverge.
- (e.g., if and case statements)



While Loop



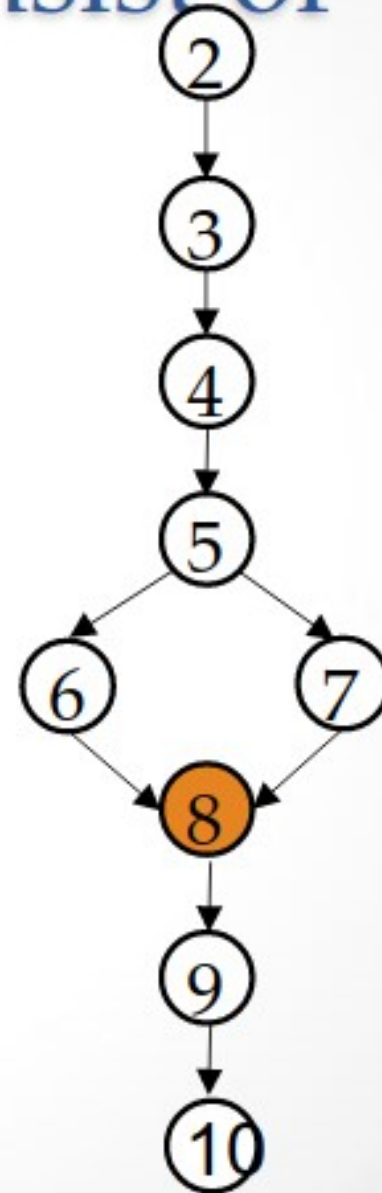
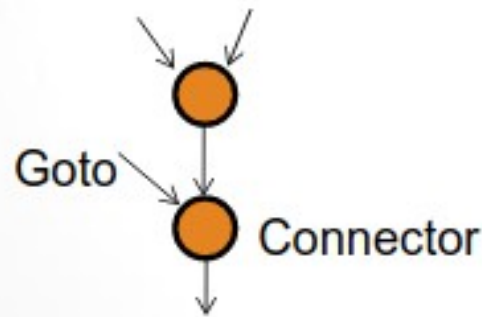
Until Loop





# Flow Graphs consist of

- A **junction** is a program point where the control flow can merge.
- (e.g., end if , end loop, goto label)



# What is Path?

- A path through a program is a sequence of statements that starts at an entry, junctions, or decision and ends.
- A path may go through several junctions, processes, or decisions, on or more times.
- Paths consist of segments that has smallest segment is a link between 2 nodes.

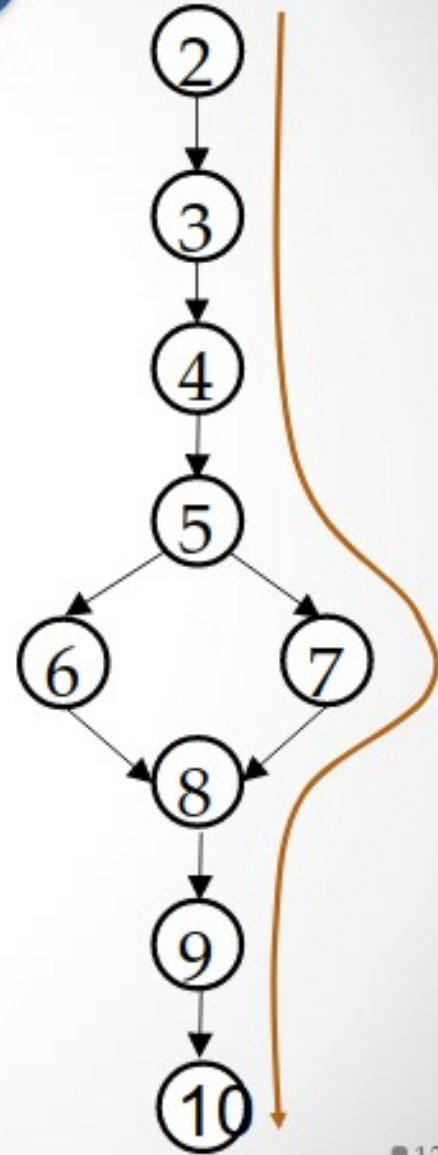


# What is a Path?

- A path through a program is a sequence of instructions or statements that starts at an entry, junction, or decision and ends at another, or possibly the same junction, decision, or exit.
- A path may go through several junctions, processes, or decisions, one or more times.
- **Paths consists of segments.**
- **The segment is a link - a single process that lies between two nodes.**
- A path segment is succession of consecutive links that belongs to some path.
- The length of path measured by the number of links in it and not by the number of the instructions or statements executed along that path.
- **The name of a path is the name of the nodes along the path.**

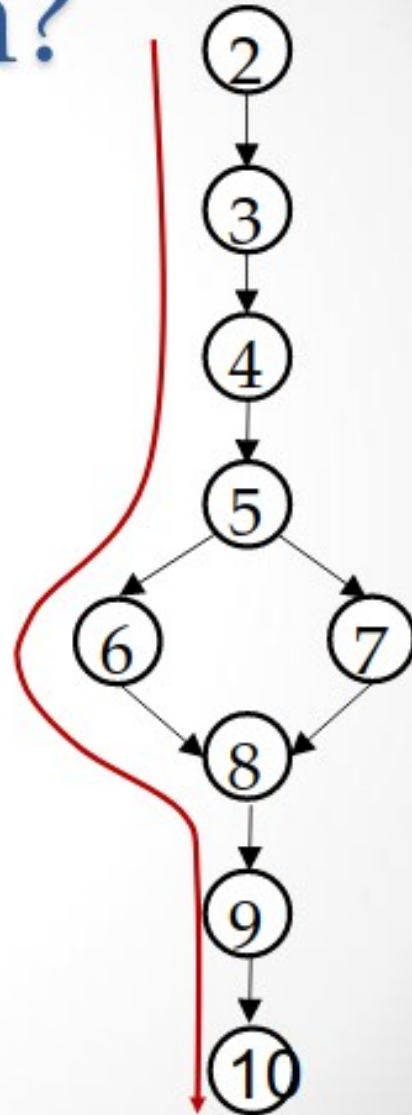
# What is Path?

- For example:  
Path2 = 2-3-4-5-7-8-9-10



# What is Path?

- For example:  
Path1 = 2-3-4-5-6-8-9-10



## **FUNDAMENTAL PATH SELECTION CRITERIA:**

- There are many paths between the entry and exit of a typical routine.
- Every decision doubles the number of potential paths. And every loop multiplies the number of potential paths by the number of different iteration values possible for the loop.

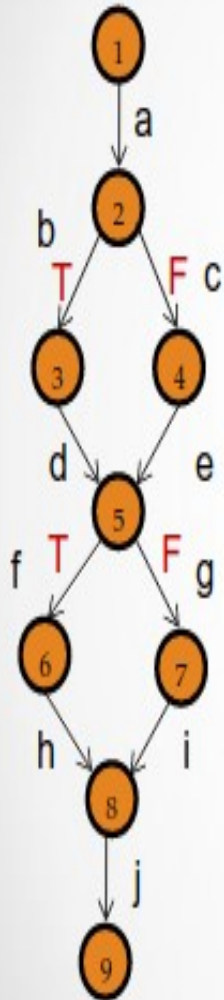
### **In an attempt to define complete testing:**

- 0. Exercise every path from entry to exit**
- 1. Exercise every statement or instruction at least once**
- 2. Exercise every branch and case statement, in each direction at least once**

## **PATH TESTING STRATEGIES**

- Any testing strategy based on paths must at least both exercise every instruction and take branches in all directions.
- A set of tests that does this is not complete in an absolute sense, but it is complete in the sense that anything less must leave something untested.
- **So we will explore three different testing criteria or strategies out of a potentially infinite family of strategies.**
  - (1) Statement Testing/Statement Coverage**
  - (2) Branch Testing / Branch Coverage**
  - (3) Path Testing/ Path Coverage**

# Path Testing Strategies



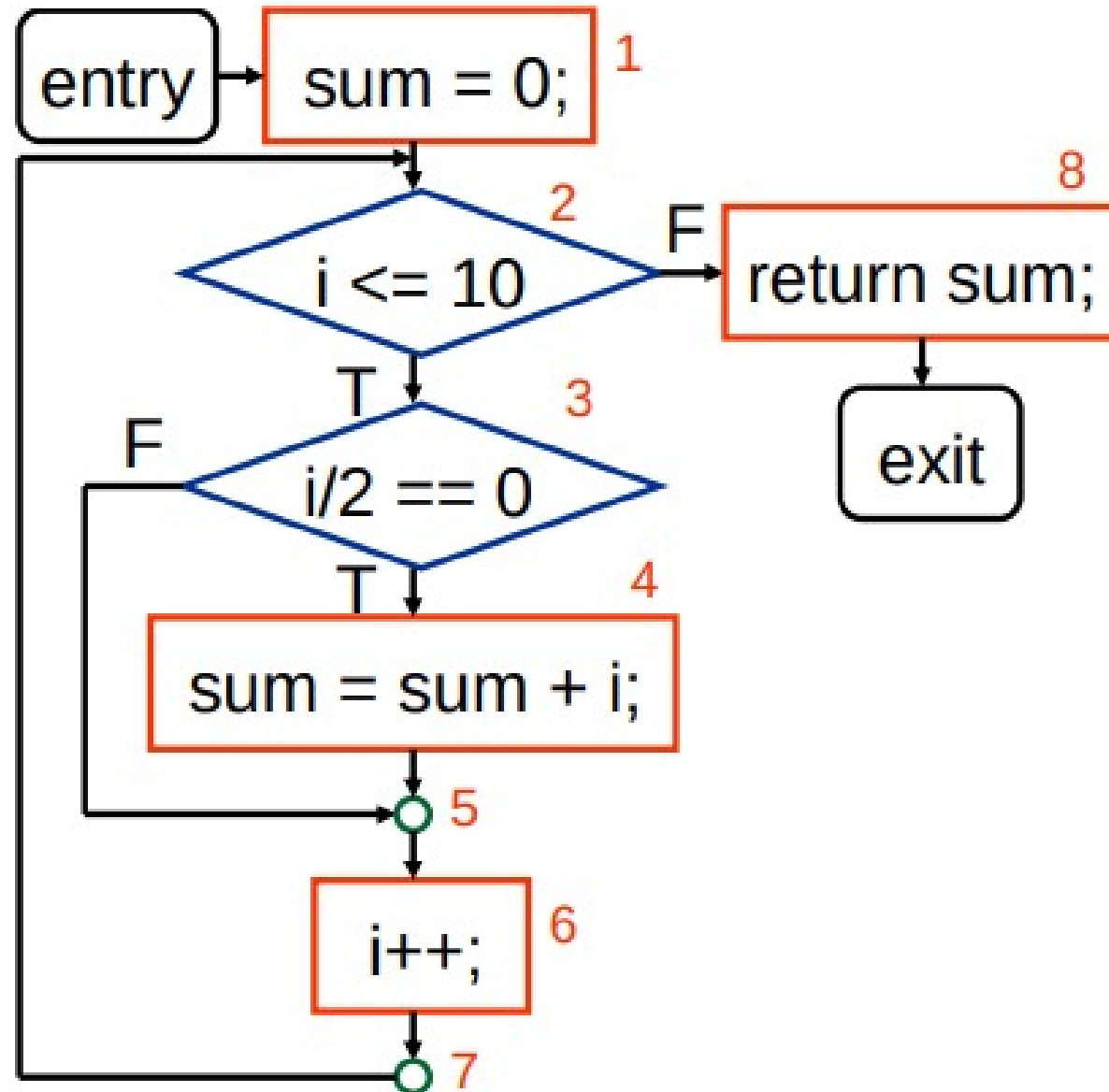
- Statement Testing
  - 100% statement / node coverage
- Branch Testing
  - 100% branch/ link coverage
- Path Testing
  - 100% path coverage

- Statement testing or statement coverage executes all statements in the program at least once under some test. If we do enough tests to achieve this, we are said to have achieved 100% statement coverage.
- An alternate equivalent characterization is to say that we have achieved 100% **node coverage**.
- **Let us denote this by C1.**
- This is the weakest criterion in the family: testing less than this for new software is unconscionable (unprincipled or can not be accepted) and should be criminalized (not acceptable).

# Statement Coverage

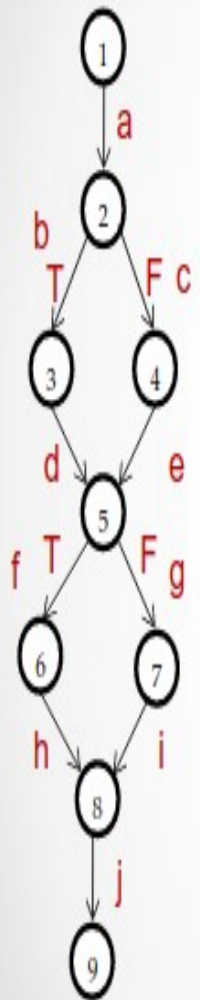
- Every **statement** in the program has been executed at least once.

1 → 2 → 3 → 4 →  
5 → 6 → 7 → 2 → 8





# Path Testing Strategies

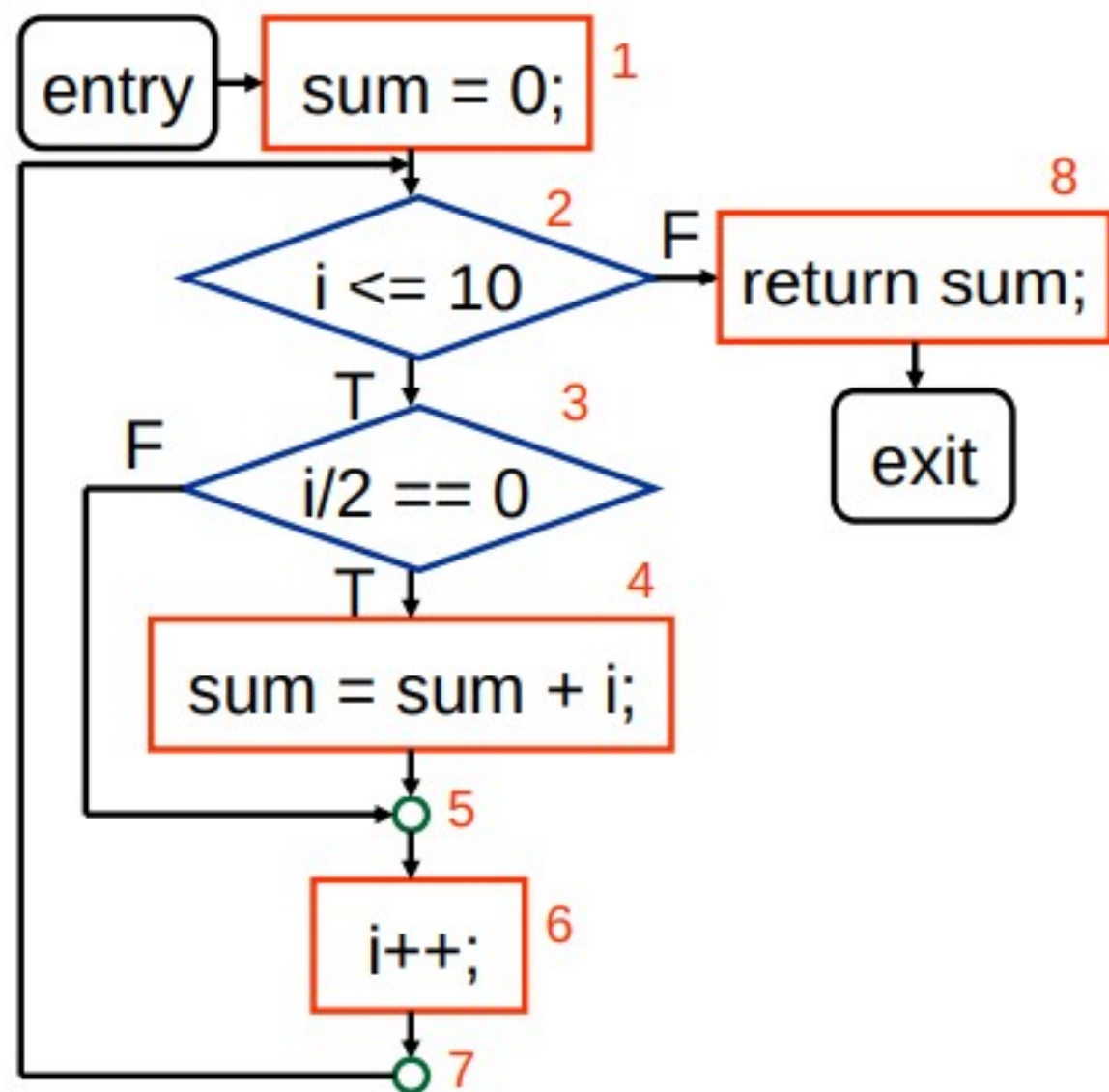


- Statement Testing
  - 100% statement / node coverage
- Branch Testing
  - 100% branch/ link coverage
- Path Testing
  - 100% path coverage

- Branch testing/ branch coverage/condition testing, executes enough tests to assure that every branch alternative has been exercised at least once under some test I.e branch testing executes must every statement and every condition at once under some test.
- If we do enough tests to achieve this prescription, then we have achieved 100% branch coverage.
- An alternative characterization is to say that we have achieved 100% **link coverage**.
- For structured software, branch testing and therefore branch coverage strictly includes statement coverage.
- **Let us denote branch coverage by C2.**



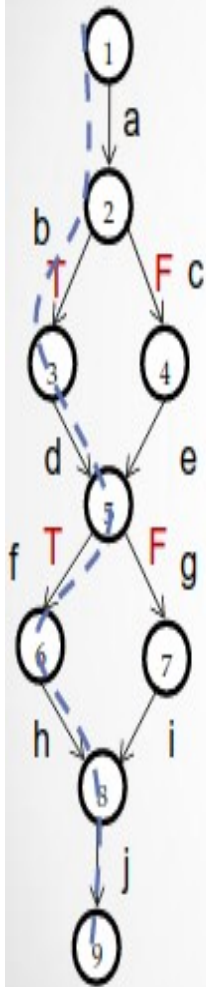
- Every **statement** in the program has been executed at least once, and every **decision** in the program has taken all possible outcomes at least once.



1 → 2 → 3 → 5 → 6 → 7 → 2 →  
3 → 4 → 5 → 6 → 7 → 2 → 8

# Path Testing Strategies

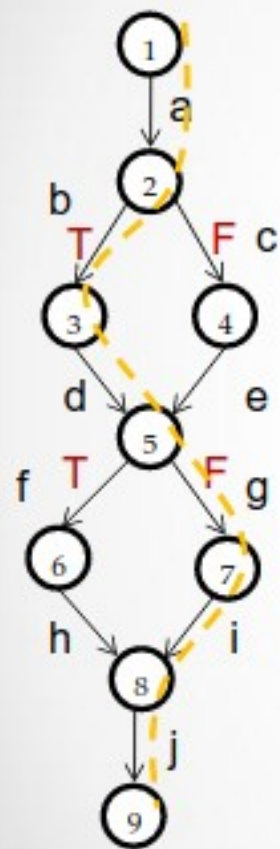
- Statement Testing
  - 100% statement / node coverage
- Branch Testing
  - 100% branch / link coverage
- Path Testing
  - 100% path coverage



Statement Testing < Branch Testing < Path Testing

- Path testing or path coverage execute all possible control flow paths through the program must be executed at least once: typically, this is restricted to all possible entry/exit paths through the program.
- If we achieve this prescription, we are said to have achieved 100% path coverage which is practically impossible due to the inherent problem of **Path Explosion**.
- **This is the strongest criterion in the path testing strategy family: it is generally impossible to achieve.**

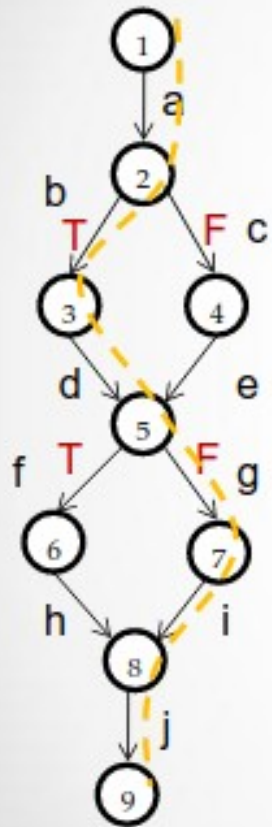
# Path Testing Strategies



- Statement Testing
  - 100% statement / node coverage
- Branch Testing
  - 100% branch/ link coverage
- Path Testing
  - 100% path coverage

Statement Testing < Branch Testing < Path Testing

# Path Testing Strategies

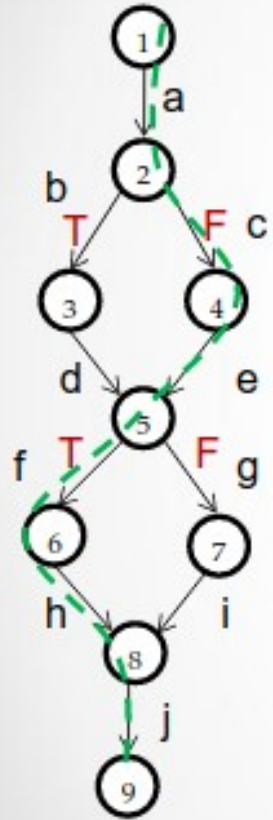


- Statement Testing
  - 100% statement / node coverage
- Branch Testing
  - 100% branch/ link coverage
- Path Testing
  - 100% path coverage

Statement Testing < Branch Testing < Path Testing



# Path Testing Strategies



- Statement Testing
  - 100% statement / node coverage
- Branch Testing
  - 100% branch/ link coverage
- Path Testing
  - 100% path coverage

Statement Testing < Branch Testing < Path Testing

## Some Commonsense and Strategies:

### The question "why not use a judicious sampling of paths?"

- Branch and statement coverage are accepted today as the minimum mandatory testing requirement.

The question "why not use a judicious sampling of paths?, what is wrong with leaving some code, untested?" is ineffectual in the view of common sense and experience since:

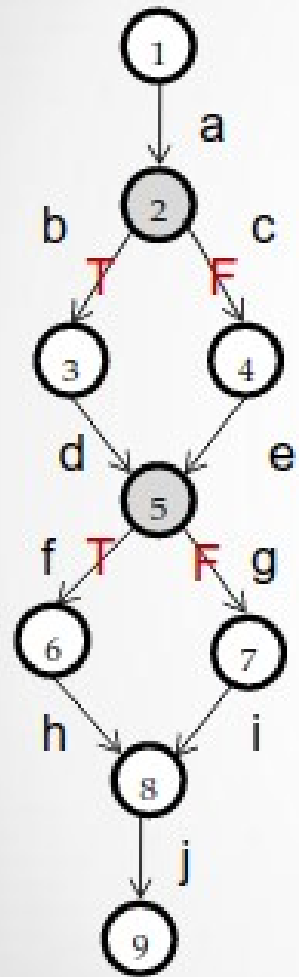
(1.) Not testing a piece of a code leaves a residue of bugs in the program in proportion to the

size of the untested code and the probability of bugs.

(2.) The high probability paths are always thoroughly tested if only to demonstrate that the system works properly.

- **Now Which paths are to be tested? =====>>C1+C2.**
- **You must pick enough paths to achieve C1+C2.** The question of what is the fewest number of such paths is interesting to the designer of test tools that help automate the path testing, but it is not crucial to the pragmatic (practical) design of tests.
- **It is better to make many simple paths than a few complicated paths.**

# A Coverage Table



Paths	Decisions		Process-Link									
	2	5	a	b	c	d	e	f	g	h	i	j
1-2-3-5-6-8-9 (a-b-d-f-h-j)	T	T	✓	✓		✓		✓		✓		✓
1-2-4-5-7-8-9 (a-c-e-g-i-j)	F	F	✓		✓		✓		✓		✓	✓
1-2-3-5-7-8-9 (a-b-d-g-i-j)	T	F	✓	✓		✓			✓		✓	✓
1-2-4-5-6-8-9 (a-c-e-f-h-j)	F	T	✓		✓		✓	✓		✓		✓

# Example Flow Graph

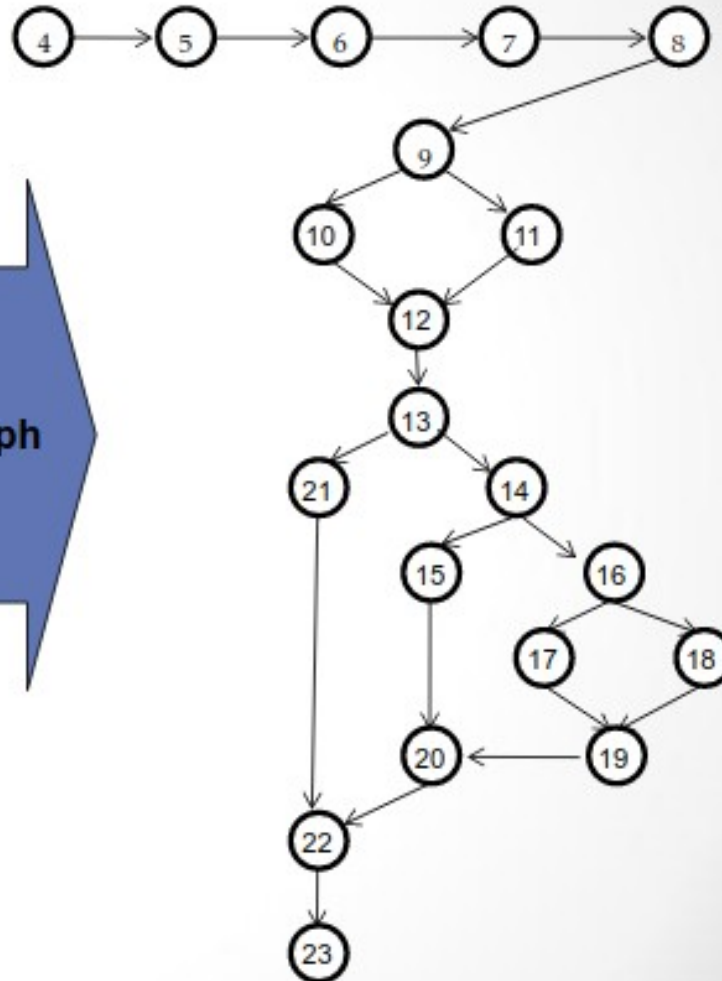
1. Program Triangle
2. Dim a, b, c As Integer
3. Dim IsTriangle As Boolean

4. Output ( "enter a,b, and c integers")
5. Input (a,b,c)
6. Output ("side 1 is", a)
7. Output ("side 2 is", b)
8. Output ("side 3 is", c)

9. If (a<b+c) AND (b<a+c) And (c<b+a)
10. then IsTriangle = True
11. else IsTriangle = False
12. endif

13. If IsTriangle
14. then if (a=b) AND (b=c)
15. then Output ("equilateral")
16. else if (a != b) AND (a != c) AND (b != c)
17. then Output ( "Scalene")
18. else Output ("Isosceles")
19. endif
20. endif
21. else Output ("not a triangle")
22. endif
23. end Triangle2

Flow Graph





# Test Case for Path Coverage

- ① 4-5-6-7-8-9-10-12-13-21-22-23
- ② 4-5-6-7-8-9-11-12-13-14-15-20-22-23
- ③ 4-5-6-7-8-9-11-12-13-14-16-17-19-20-22-23
- ④ 4-5-6-7-8-9-11-12-13-14-16-18-19-20-22-23

Path	Decision				Test case			Expected Results
	9	13	14	16	a	b	c	
①	T	F			100	100	200	Not A triangle
②	F	T	T		100	100	100	Equilateral
③	F	T	F	T	100	50	60	Scalene
④	F	T	T	F	100	100	50	Isosceles

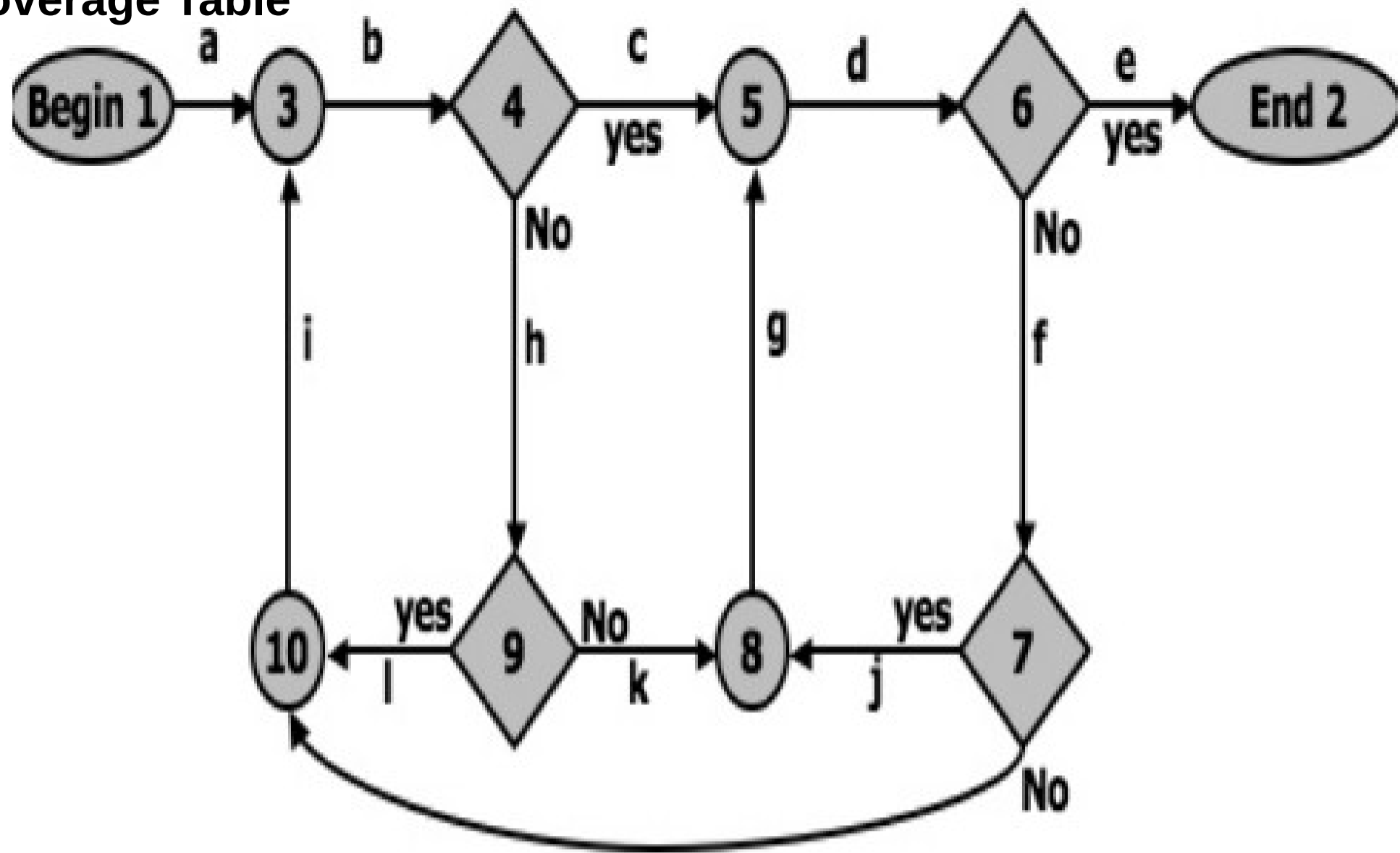
# EXERCISE ONE (20 Marks)

(1) Assuming you are to develop a software for the **Management of Student Results** in University of Yaounde I with the following grading scheme:

Grades	Range
$80 \leq x \leq 100$	A <sup>+</sup>
$75 \leq x < 80$	A
$70 \leq x < 75$	B <sup>+</sup>
$65 \leq x < 70$	B
$60 \leq x < 65$	B <sup>-</sup>
$55 \leq x < 60$	C <sup>+</sup>
$50 \leq x < 55$	C
$45 \leq x < 50$	C <sup>-</sup>
$40 \leq x < 45$	D <sup>+</sup>
$35 \leq x < 40$	D
$0 \leq x < 35$	E

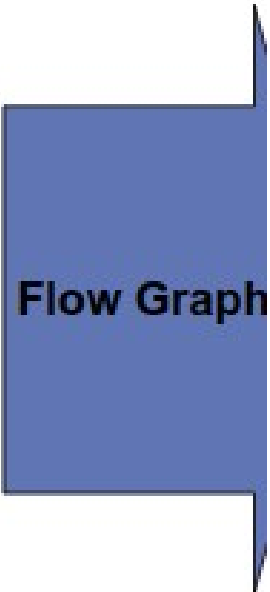
- (a) Write an algorithm for the above Software Project and hence state the characteristics of Algorithms?
- (b) Define Test Cases and hence explain the criteria for selecting Test Cases
- (c) Present the Test Case Document of the chosen Project
- (d) Using **Java NetBeans**, write down the function for computing the grades of N students and hence depict the Control Flow Graph (CFG) of the System. (both in terminal and GUI)
- (e) With respect to (d) above, explain in details:
  - Disruptive Testing and Exhaustive Testing
  - Data Flow Testing (All-Defs, All-Uses, All-DU-Paths, All-P-Uses, Some-C-Uses, All-C-Uses, Some-P-Uses, All-P-Uses, All-C-Uses)
  - Path Analysis – Cyclomatic Complexity Measure and Path Selection
- (f) Identify five paths in (d) and hence Produce the Coverage Table, stating explicitly valid and invalid input.
- (g) Recommend three Path testing Strategies for the system giving your justification and the significant of each strategy.

Exercise Two: Identify five paths in the diagram CFG and hence produce the Coverage Table



**Exercise Three: Identify five paths in the diagram CFG, hence produce the Coverage Table bearing in mind a triangle is made up of 180 degree and produce the corresponding program in Java NetBeans.**

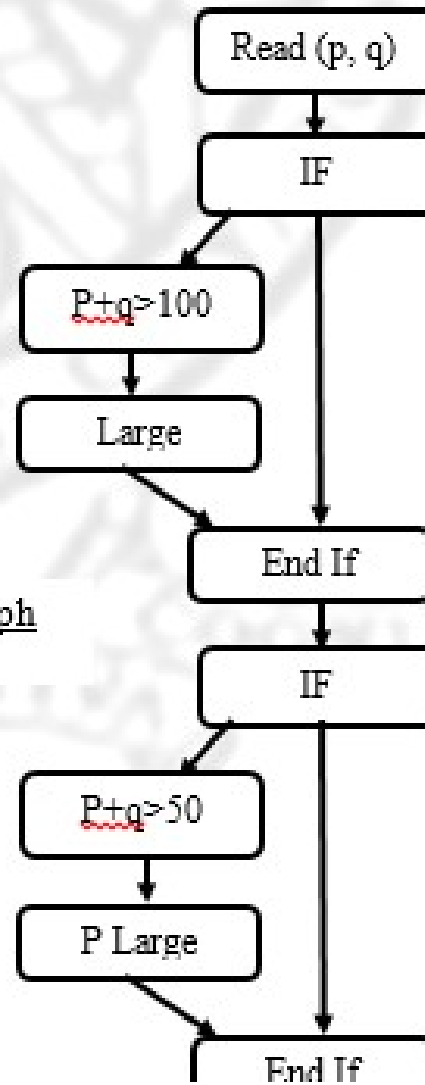
1. Program Triangle
2. Dim a, b, c As Integer
3. Dim IsTriangle As Boolean
4. Output ( "enter a,b, and c integers")
5. Input (a,b,c)
6. Output ("side 1 is", a)
7. Output ("side 2 is", b)
8. Output ("side 3 is", c)
9. If (a<b+c) AND (b<a+c) And (c<b+a)
10. then IsTriangle = True
11. else IsTriangle = False
12. endif
13. If IsTriangle
14. then if (a=b) AND (b=c)
15. then Output ("equilateral")
16. else if (a != b) AND (a != b) AND (b != c)
17. then Output ( "Scalene")
18. else Output ("Isosceles")
19. endif
20. endif



Flow Graph

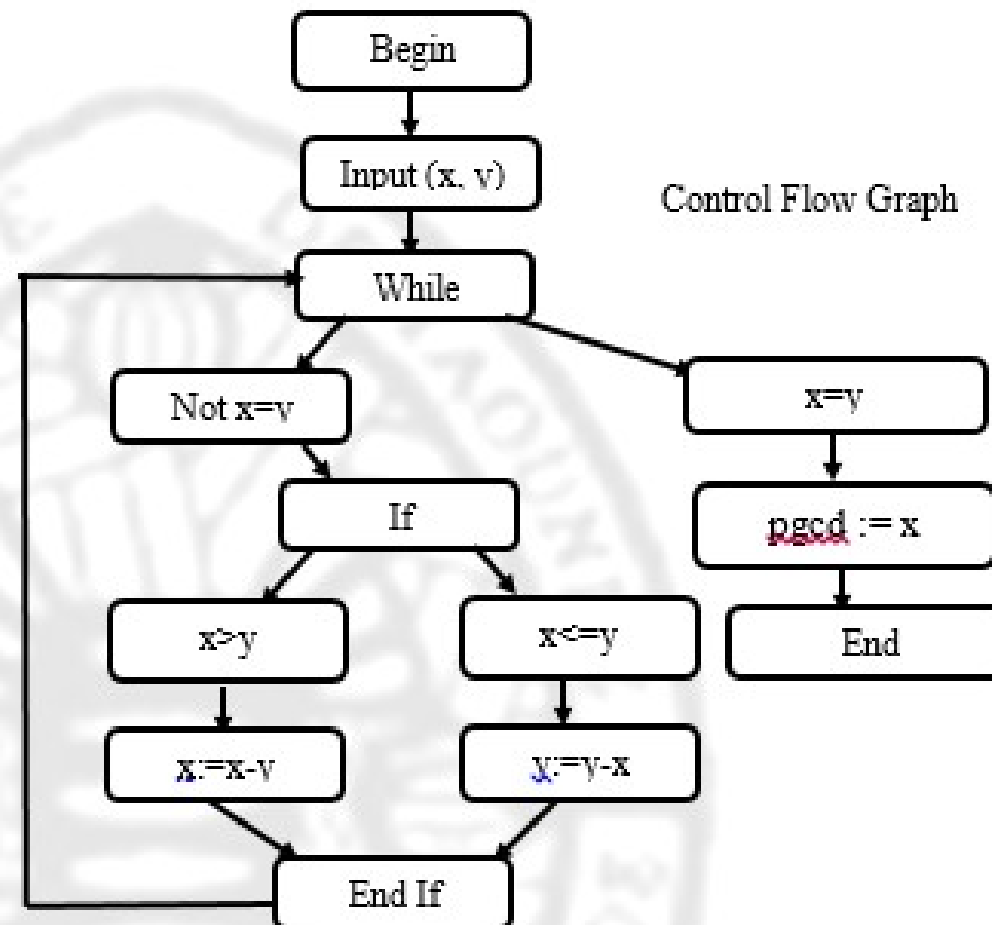
```
Read p
Read q
IF p+q > 100
THEN
Print "Large"
ENDIF
IF p > 50 THEN
Print "p Large"
ENDIF
```

Control Flow Graph



Using the pseudo Code below, produce the corresponding Control Flow Graphs

① Begin  
Read(x); read(y);  
While (not (x = y) ) loop  
  If x > y Then  
    x := x - y ;  
  Else  
    y := y - x ;  
  End if ;  
End loop ;  
Print pgcd := x ;  
End;



# DATA FLOW TESTING

- Team Work
-



# Data Flow Testing

```
If ( Condition 1 ) {  
    x = a;  
}  
Else {  
    x = b;  
}
```

```
If ( Condition 2 ) {  
    y = x + 1;  
}  
Else {  
    y = x - 1;  
}
```

What test cases do we need?

Definitions: 1)  $x = a$ ; 2)  $x = b$ ;

Uses: 1)  $y = x + 1$ ; 2)  $y = x - 1$ ;

1.  $(x = a, y = x + 1)$

2.  $(x = b, y = x + 1)$

3.  $(x = a, y = x - 1)$

4.  $(x = b, y = x - 1)$