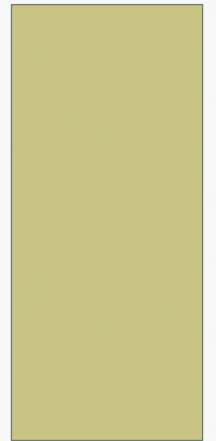
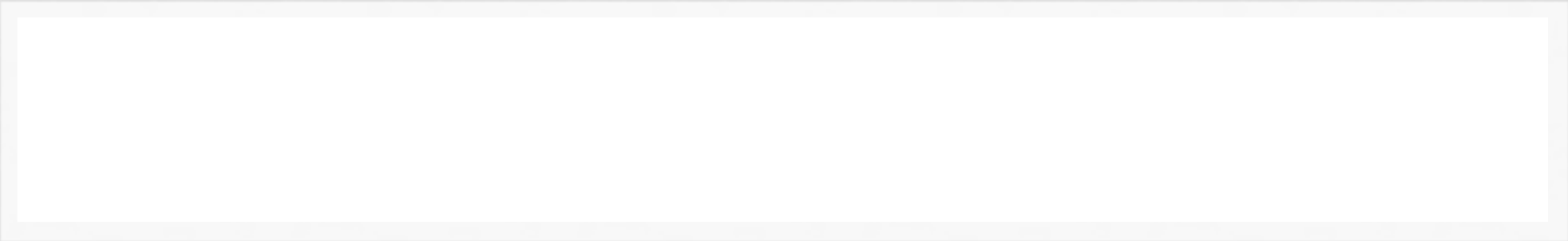


MODULE 2

DR. SANDIP MAL



- 
- **Flow graphs and Path testing**
 - **Transaction Flow Testing**

FLOW GRAPHS AND PATH TESTING

- Understand the concept of path testing.
- Identify the components of a control flow diagram and compare the same with a flowchart.
- Represent the control flow graph in the form of a Linked List notation.
- Understand the path testing and selection criteria and their limitations.
- Classify the predicates and variables as dependant/independent and correlated/uncorrelated.
- Understand the path sensitizing method and classify whether the path is achievable or not.
- Identify the problem due to co-incidental correctness and choose a path instrumentation method to overcome the problem.

PATH TESTING

- Path Testing is the name given to a family of test techniques based on judiciously selecting a set of test paths through the program.
- If the set of paths are properly chosen then we have achieved some measure of test thoroughness. For example, pick enough paths to assure that every source statement has been executed at least once.
- Path testing techniques are the oldest of all structural test techniques.
- Path testing is most applicable to new software for unit testing. It is a structural technique.
- It requires complete knowledge of the program's structure.
- It is most often used by programmers to unit test their own code.

THE BUG ASSUMPTION

- The bug assumption for the path testing strategies is that something has gone wrong with the software that makes it take a different path than intended.
- As an example "GOTO X" where "GOTO Y" had been intended.
- Structured programming languages prevent many of the bugs targeted by path testing: as a consequence the effectiveness for path testing for these languages is reduced and for old code in COBOL, ALP, FORTRAN and Basic, the path testing is indispensable.

CONTROL FLOW GRAPHS

- The control flow graph is a graphical representation of a program's control structure. It uses the elements named process blocks, decisions, and junctions.
- The flow graph is similar to the earlier flowchart, with which it is not to be confused.
- **Flow Graph Elements:** A flow graph contains four different types of elements.
 - **(1) Process Block**
 - **(2) Decisions**
 - **(3) Junctions**
 - **(4) Case Statements.**

PROCESS BLOCK

- A process block is a sequence of program statements uninterrupted by either decisions or junctions.
- It is a sequence of statements such that if any one of statement of the block is executed, then all statement thereof are executed.
- Formally, a process block is a piece of straight line code of one statement or hundreds of statements.
- A process has one entry and one exit. It can consists of a single statement or instruction, a sequence of statements or instructions, a single entry/exit subroutine, a macro or function call, or a sequence of these.

DECISIONS

- A decision is a program point at which the control flow can diverge.
- Machine language conditional branch and conditional skip instructions are examples of decisions.
- Most of the decisions are two-way but some are three way branches in control flow.

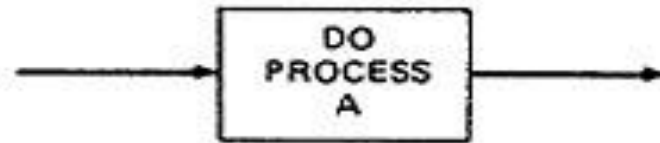
CASE STATEMENTS

- A case statement is a multi-way branch or decisions.
- Examples of case statement are a jump table in assembly language, and the PASCAL case statement.
- From the point of view of test design, there are no differences between Decisions and Case Statements

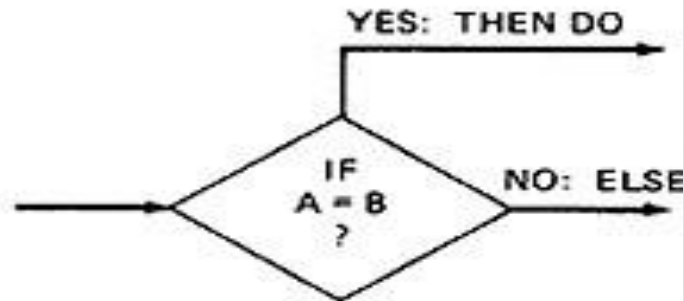
JUNCTIONS

- A junction is a point in the program where the control flow can merge.
- Examples of junctions are: the target of a jump or skip instruction in ALP, a label that is a target of GOTO.

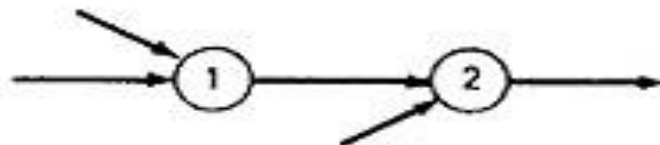
Processes



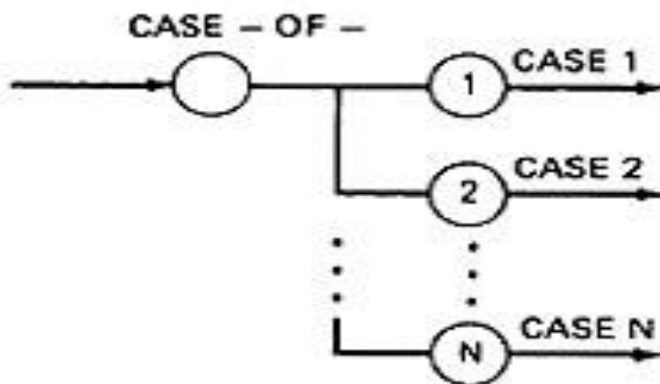
Decisions



Junctions



Case Statement



CONTROL FLOW GRAPHS VS FLOWCHARTS

- A program's flow chart resembles a control flow graph.
- In flow graphs, we don't show the details of what is in a process block.
- In flow charts every part of the process block is drawn.
- The flowchart focuses on process steps, where as the flow graph focuses on control flow of the program.
- The act of drawing a control flow graph is a useful tool that can help us clarify the control flow and data flow issues.

NOTATIONAL EVOLUTION

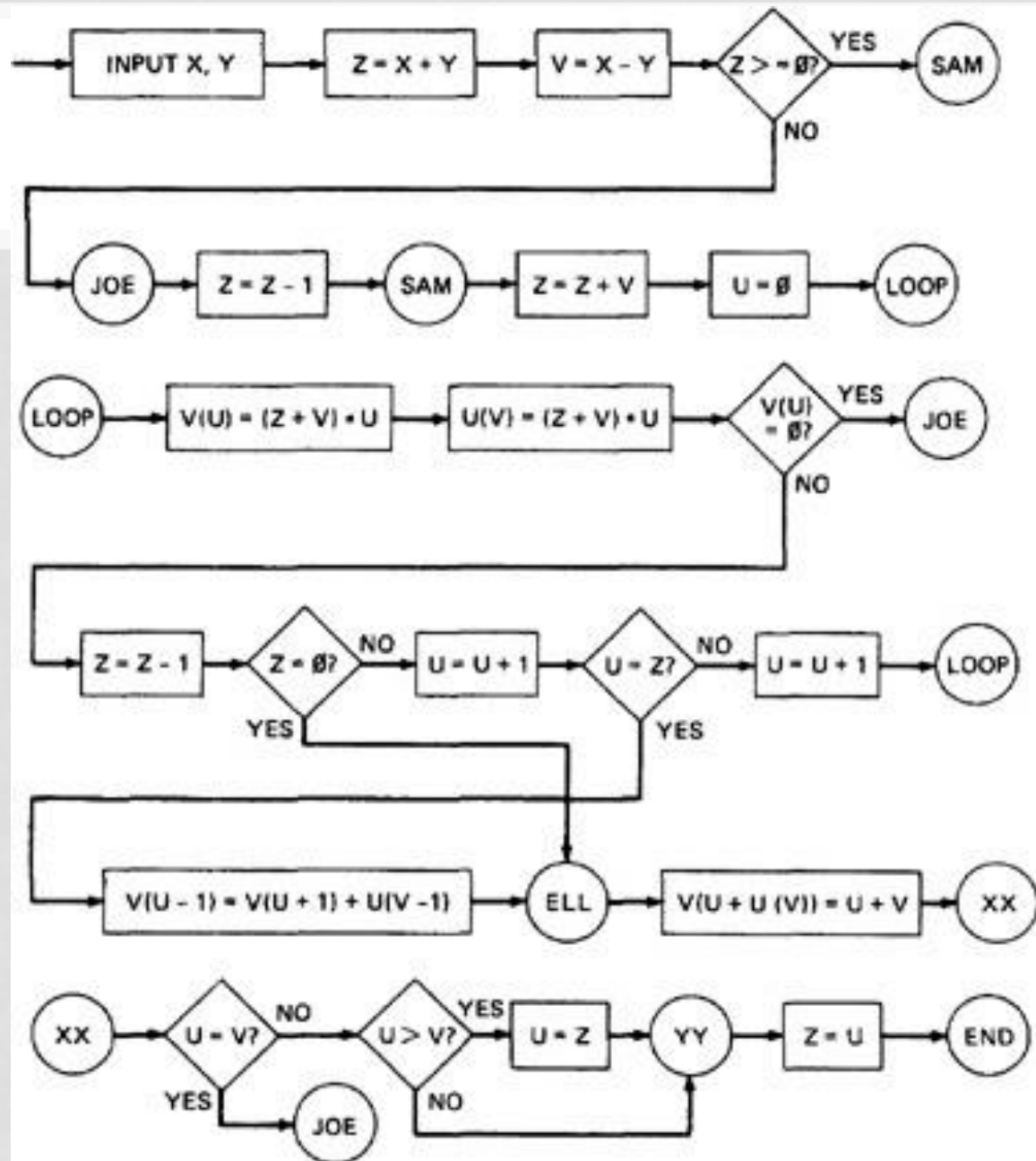
- The control flow graph is simplified representation of the program's structure.
- The notation changes made in creation of control flow graphs:
 - The process boxes weren't really needed. There is an implied process on every line joining junctions and decisions.
 - We don't need to know the specifics of the decisions, just the fact that there is a branch.
 - The specific target label names aren't important-just the fact that they exist. So we can replace them by simple numbers.
 - To understand this, we will go through an example written in a FORTRAN like programming language called **Programming Design Language (PDL)**. The program's corresponding flowchart and flow graph were also provided below for better understanding.
 - The first step in translating the program to a flowchart is shown in Figure, where we have the typical one-for-one classical flowchart. Note that complexity has increased, clarity has decreased, and that we had to add auxiliary labels (LOOP, XX, and YY), which have no actual program counterpart. In next Figure we merged the process steps and replaced them with the single process box. We now have a control flow graph. But this representation is still too busy. We simplify the notation further to achieve next Figure, where for the first time we can really see what the control flow looks like.

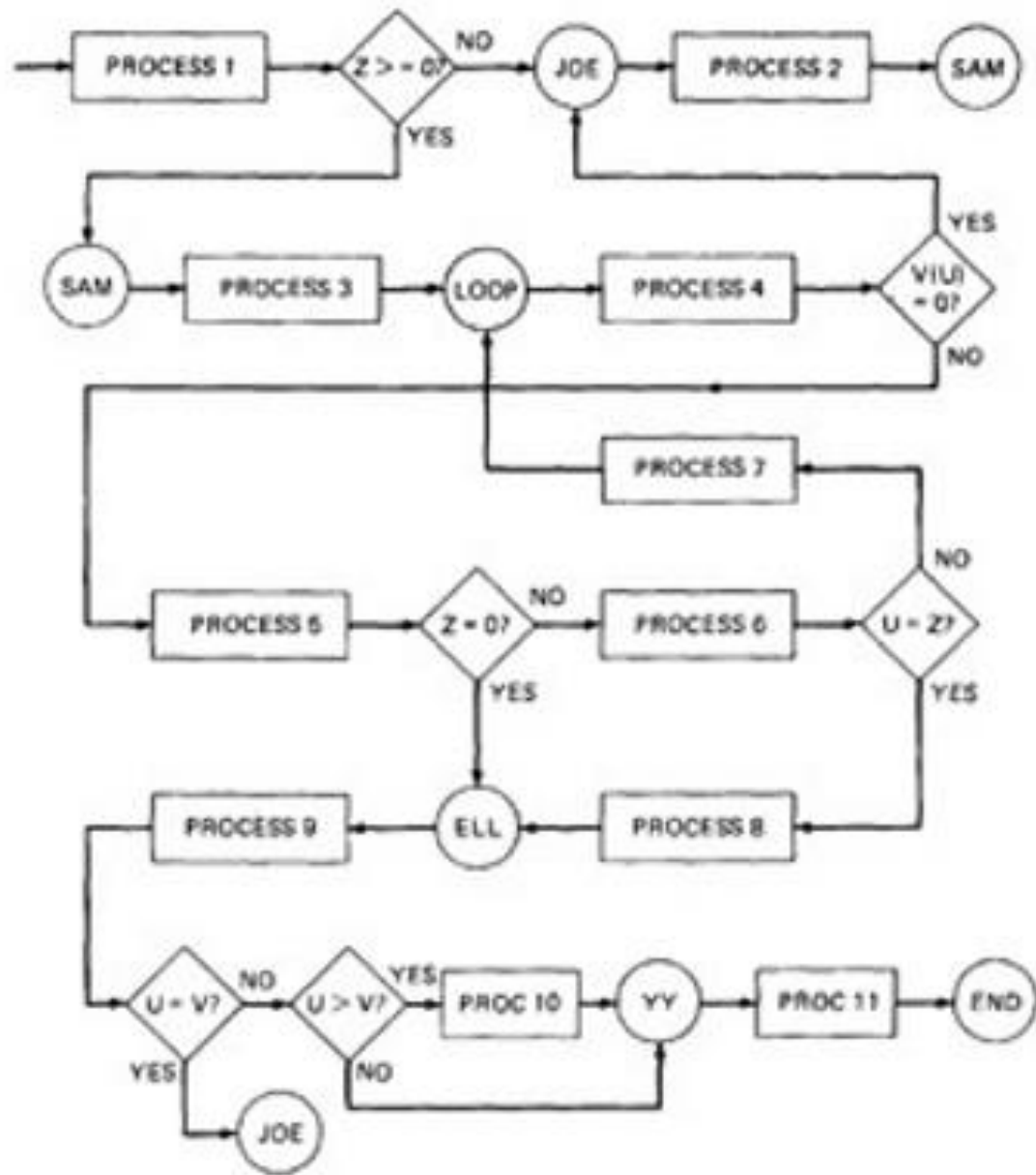
CODE* (PDL)

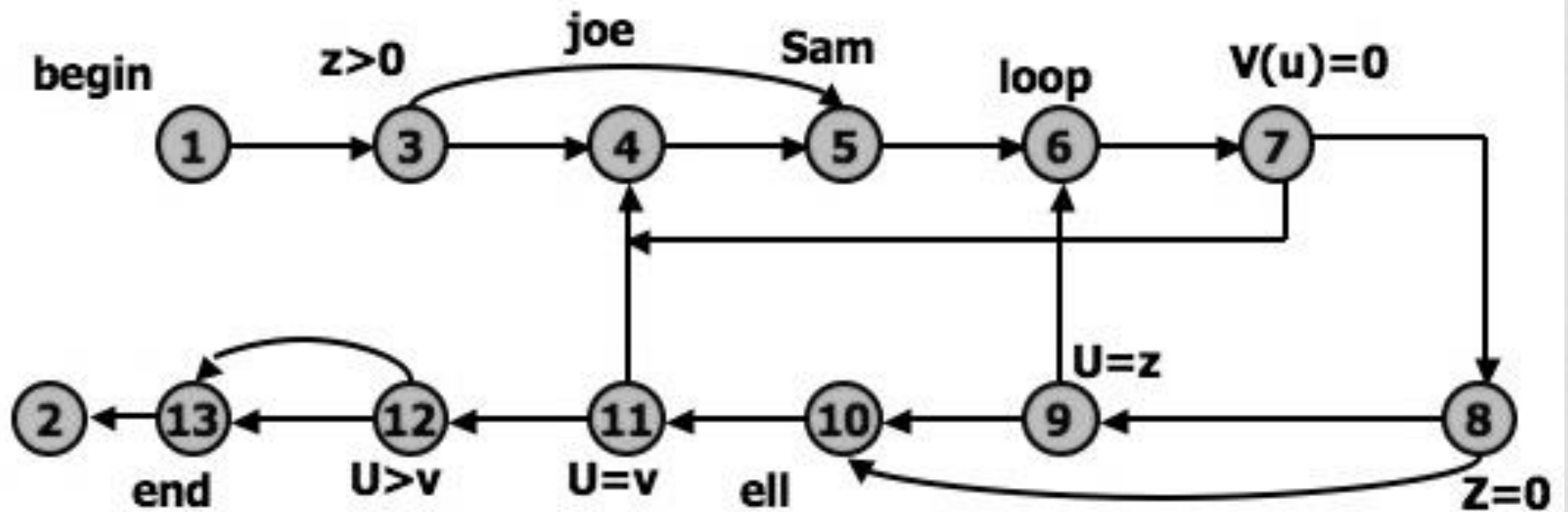
```
INPUT X, Y
Z := X + Y
V := X - Y
IF Z >= 0 GOTO SAM
JOE: Z := Z - 1
SAM: Z := Z + V
FOR U = 0 TO Z
  V(U), U(V) := (Z + V)*U
  IF V(U) = 0 GOTO JOE
  Z := Z - 1
  IF Z = 0 GOTO ELL
  U := U + 1
NEXT U

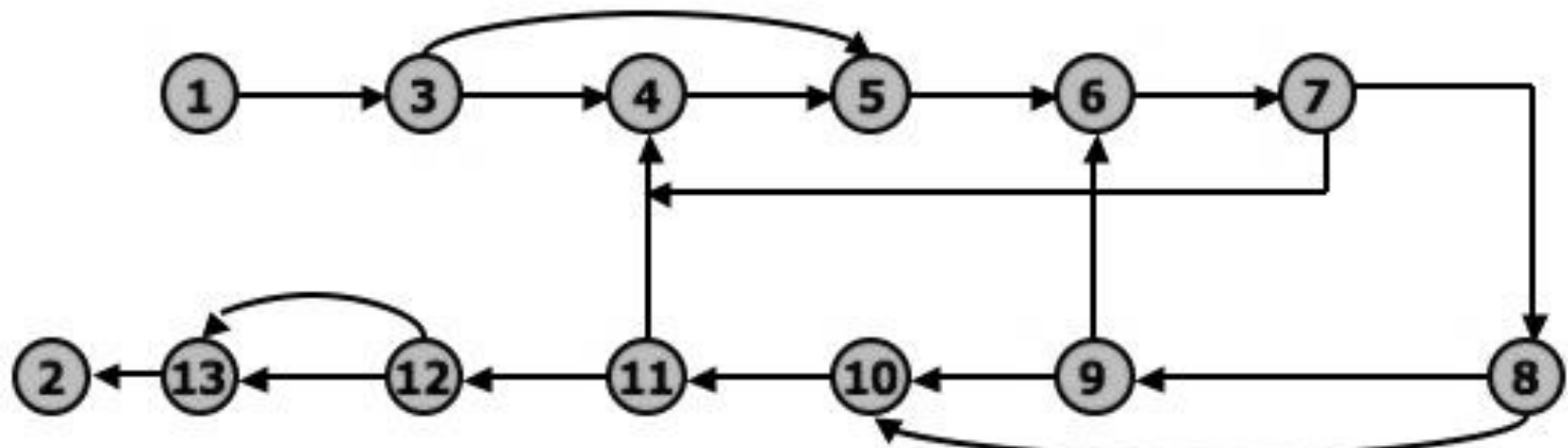
V(U-1) := V(U+1) + U(V-1)
ELL: V(U+U(V)) := U + V
IF U = V GOTO JOE
IF U > V THEN U := Z
Z := U
END
```

* A contrived horror









LINKED LIST REPRESENTATION

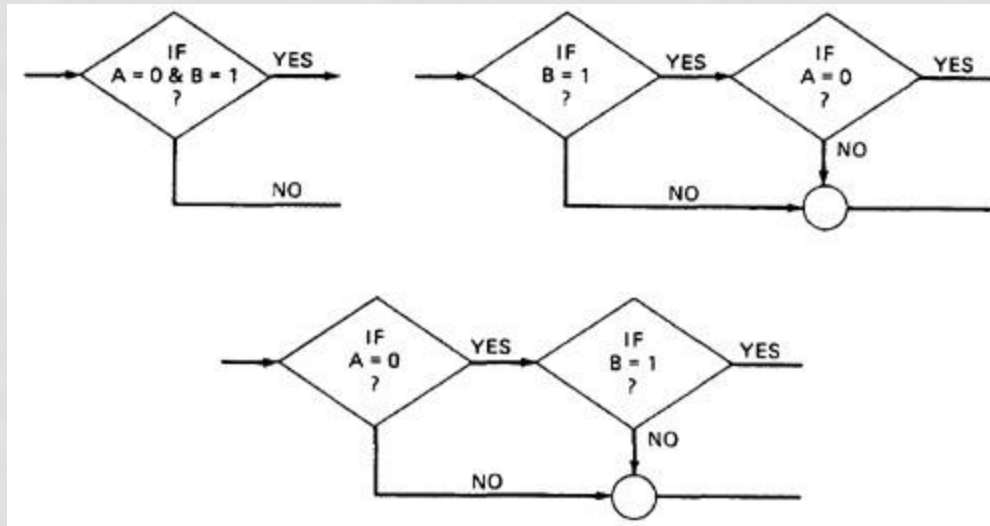
- Although graphical representations of flow graphs are revealing, the details of the control flow inside a program they are often inconvenient.
- In linked list representation, each node has a name and there is an entry on the list for each link in the flow graph. only the information pertinent to the control flow is shown.

LINKED LIST REPRESENTATION OF FLOW GRAPH

1 (BEGIN)	: 3	
2 (END)	:	Exit, no outlink
3 ($Z > \emptyset$?)	: 4 (FALSE)	
	: 5 (TRUE)	
4 (JOE)	: 5	
5 (SAM)	: 6	
6 (LOOP)	: 7	
7 ($V(U) = \emptyset$?)	: 4 (TRUE)	
	: 8 (FALSE)	
8 ($Z = \emptyset$?)	: 9 (FALSE)	
	: 10 (TRUE)	
9 ($U = Z$?)	: 6 (FALSE) = LOOP	
	: 10 (TRUE) = ELL	
10 (ELL)	: 11	
11 ($U = V$?)	: 4 (TRUE) = JOE	
	: 12 (FALSE)	
12 ($U > V$?)	: 13 (TRUE)	
	: 13 (FALSE)	
13	: 2 (END)	

FLOWGRAPH - PROGRAM CORRESPONDENCE

- A flow graph is a pictorial representation of a program and not the program itself, just as a topographic map.
- You cant always associate the parts of a program in a unique way with flow graph parts because many program structures, such as if-then-else constructs, consists of a combination of decisions, junctions, and processes.
- The translation from a flow graph element to a statement and vice versa is not always unique.
- An improper translation from flow graph to code during coding can lead to bugs, and improper translation during the test design lead to missing test cases and causes undiscovered bugs.



Alternative Flow graphs for same logic (Statement "IF (A=0) AND (B=1) THEN ...")

FLOWGRAPH AND FLOWCHART GENERATION

- Flowcharts can be
 - Handwritten by the programmer.
 - Automatically produced by a flowcharting program based on a mechanical analysis of the source code.
 - Semi automatically produced by a flow charting program based in part on structural analysis of the source code and in part on directions given by the programmer.
- There are relatively few control flow graph generators.

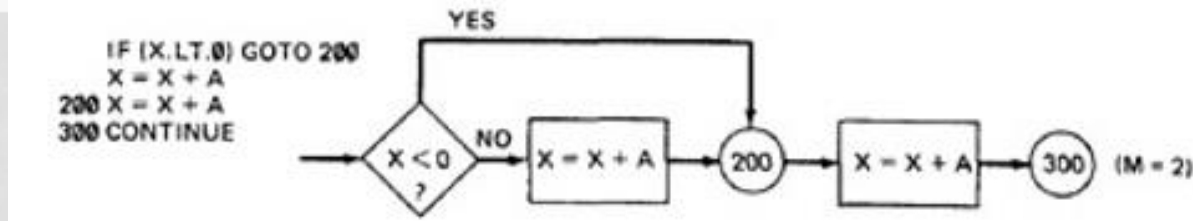
PATH TESTING - PATHS, NODES AND LINKS

- **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.

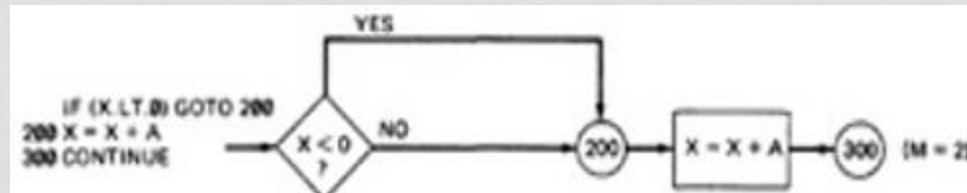
FUNDAMENTAL PATH SELECTION CRITERIA

- 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.
- Defining complete testing:
 - Exercise every path from entry to exit
 - Exercise every statement or instruction at least once
 - Exercise every branch and case statement, in each direction at least once
- If prescription 1 is followed then 2 and 3 are automatically followed. But it is impractical for most routines. It can be done for the routines that have no loops, in which it is equivalent to 2 and 3 prescriptions.

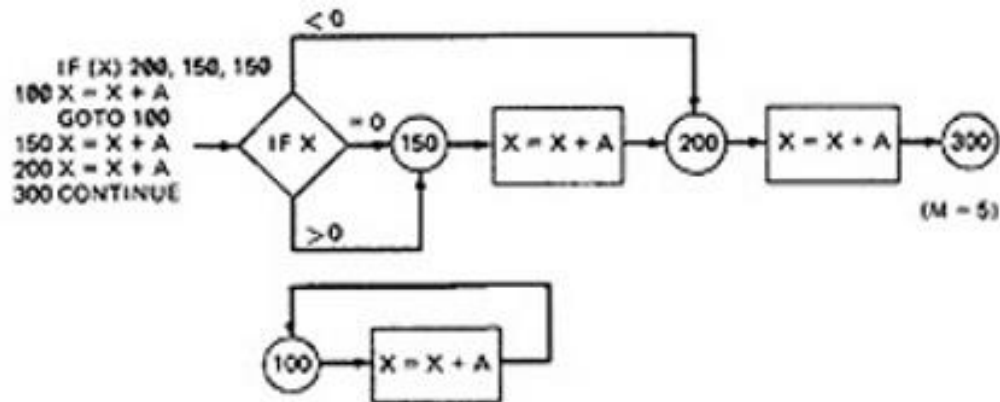
EXAMPLE



For X negative, the output is $X + A$, while for X greater than or equal to zero, the output is $X + 2A$. Following prescription 2 and executing every statement, but not every branch, would not reveal the bug in the following incorrect version:



A negative value produces the correct answer. Every statement can be executed, but if the test cases do not force each branch to be taken, the bug can remain hidden. The next example uses a test based on executing each branch but does not force the execution of all statements:



The hidden loop around label 100 is not revealed by tests based on prescription 3 alone because no test forces the execution of statement 100 and the following GOTO statement. Furthermore, label 100 is not flagged by the compiler as an unreferenced label and the subsequent GOTO does not refer to an undefined label.



Thank You