Data Flow Testing cont…

Dr. Sangharatna Godboley

Assistant Professor

Department of CSE

NIT Warangal

Slides Credits: Dr. Durga Prasad Mohapatra, Professor, Department of CSE, NIT Rourkela

Static Analysis is not Enough

- All anomalies using static analysis cannot be determined and this is an unsolvable problem.

- For example, if the variable is used in an array as the index, we cannot determine its state by static analysis.

- Or it may be the case that the index is generated dynamically during execution, therefore we cannot guarantee what the state of the array element is referenced by that index.

Dynamic Data Flow Testing

- The test cases are designed in such a way that every definition of data variable to each of its use is traced to each of its definition. Various strategies are employed for the creation of test cases. All these strategies are defined below.

Dynamic Data Flow Testing cont…

- ***All-du Paths (ADUP)*** It states that every definition of every variable to every use of that definition should be exercised under some test. It is the strongest data flow testing strategies.

Dynamic Data Flow Testing cont…

- ***All-uses (AU)*** This states that for every use of the variable, there is a path from the definition of that variable (nearest to the use in backward direction) to the use.

Dynamic Data Flow Testing cont…

- ***All-p-uses/Some-c-uses (APU + C)*** This strategy states that for every variable and every definition of that variable, include at least one dc-path from the definition to every predicate use. If there are definitions of the variable with no p-use following it, then add computational use (c-use) test cases as required to cover every definition.

Dynamic Data Flow Testing cont…

- ***All-c-uses/Some-p-uses (ACU + P)*** This strategy states that for every variable and every definition of that variable, include at least one dc-path from the definition to every computational use. If there are definitions of the variable with no c-use following it, then add predicate use (c-use) test cases as required to cover every definition.

Dynamic Data Flow Testing cont…

- ***All-Predicate-uses (APU)*** It is derived from the APU + C strategy and states that for every variable, there is a path from every definition to every p-use of that definition. If there is a definition with no p-use following it, then it is dropped from contention.

Dynamic Data Flow Testing cont…

- ***All-Computational-Uses (ACU)*** It is derived from the strategy ACU + P strategy and states that for every variable, there is a path form every definition to every c-use of that definition. If there is a definition with no c-use following if, then it is dropped form contention.

Dynamic Data Flow Testing cont…

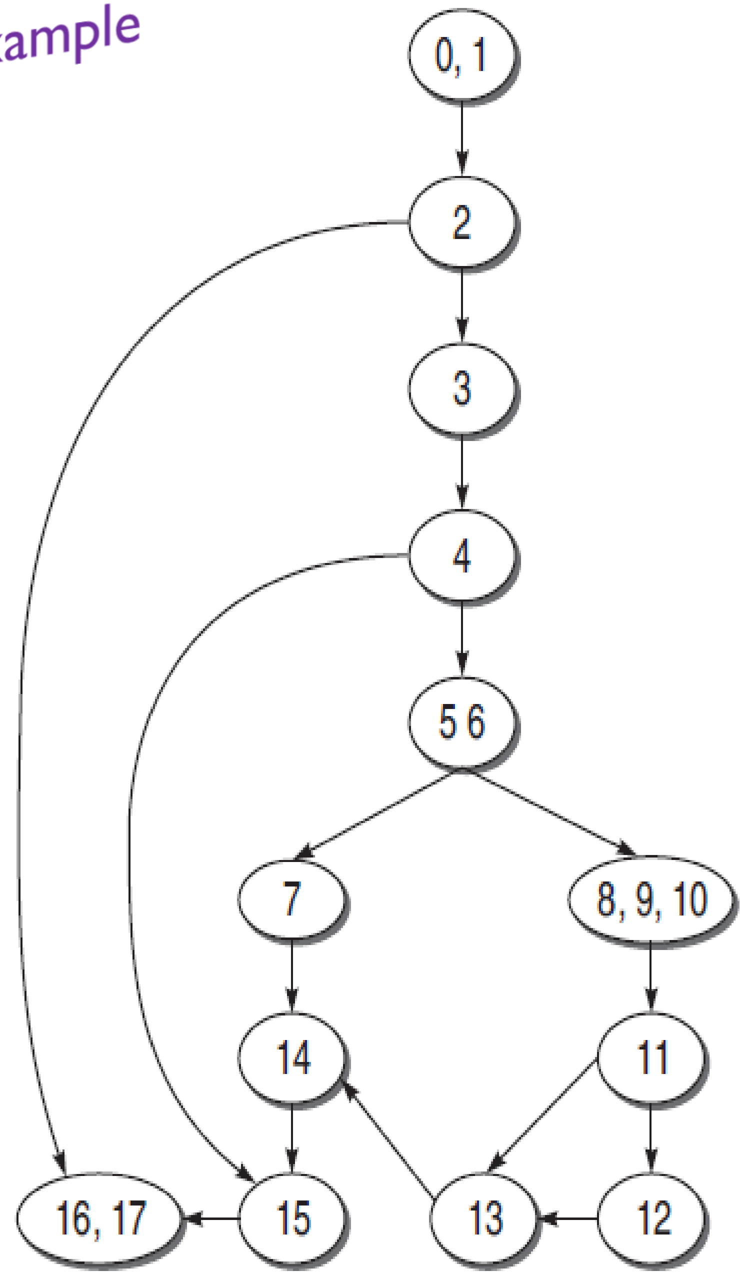
- ***All-Definition (AD)*** It states that every definition of every variable should be covered by at least one use of that variable, be that a computational use or a predicate use.

Example

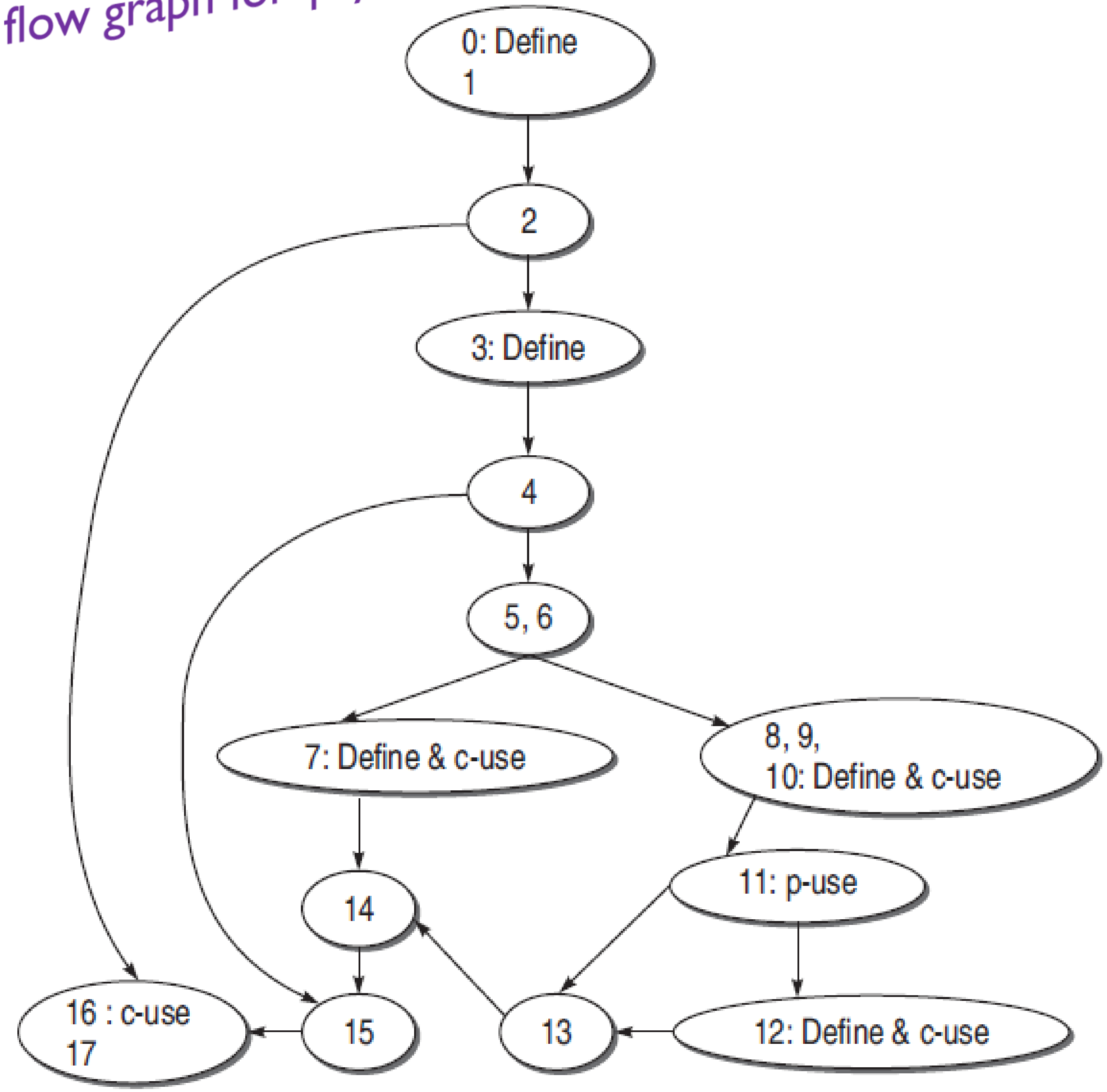
- Consider a program given below. Draw its control flow graph and data flow graph for each variable used in the program, and derive data flow testing paths with all the strategies discussed above.

```
        main()
         {
           int work;
0.          double payment =0;
1.          scanf(“%d”, work);
2.           if (work > 0) {
3.           payment = 40;
4.          if (work > 20)
5.           {
6.           if(work <= 30)
7.           payment = payment + (work – 25) * 0.5;
8.            else
9.             {
10.             payment = payment + 50 + (work –30) * 0.1;
11.            if (payment >= 3000)
12.            payment = payment * 0.9;
13.           }
14.            }
15.          }
16.            printf(“Final payment”, payment);
```

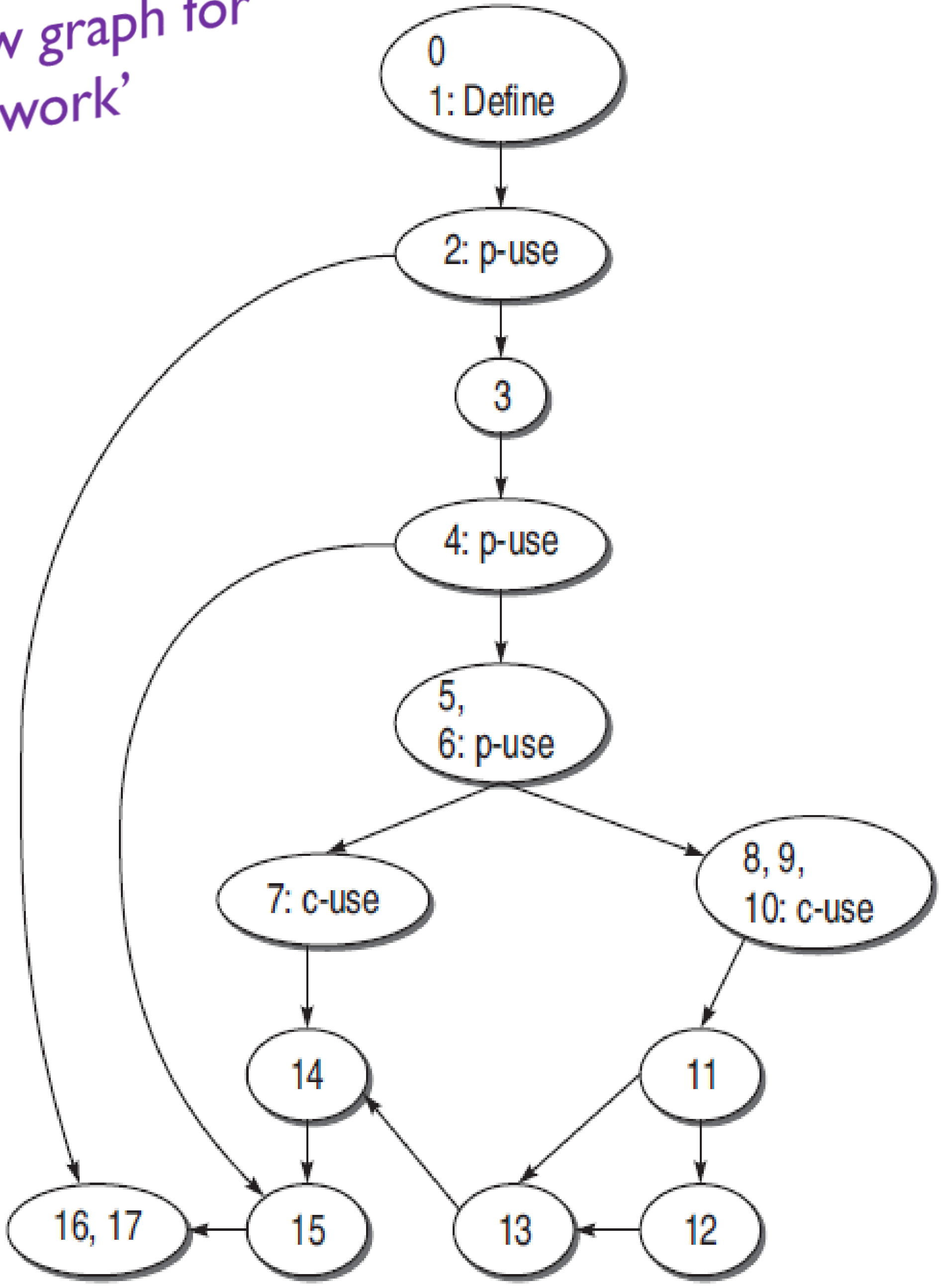
CFG for the Example



Data flow graph for 'payment'



Data flow graph for variable 'work'



- Prepare a list of all the definition nodes and usage nodes for all the variables in the program.

Variable	Defined At	Used At
Payment	0,3,7,10,12	7,10,11,12,16
Work	1	2,4,6,7,10

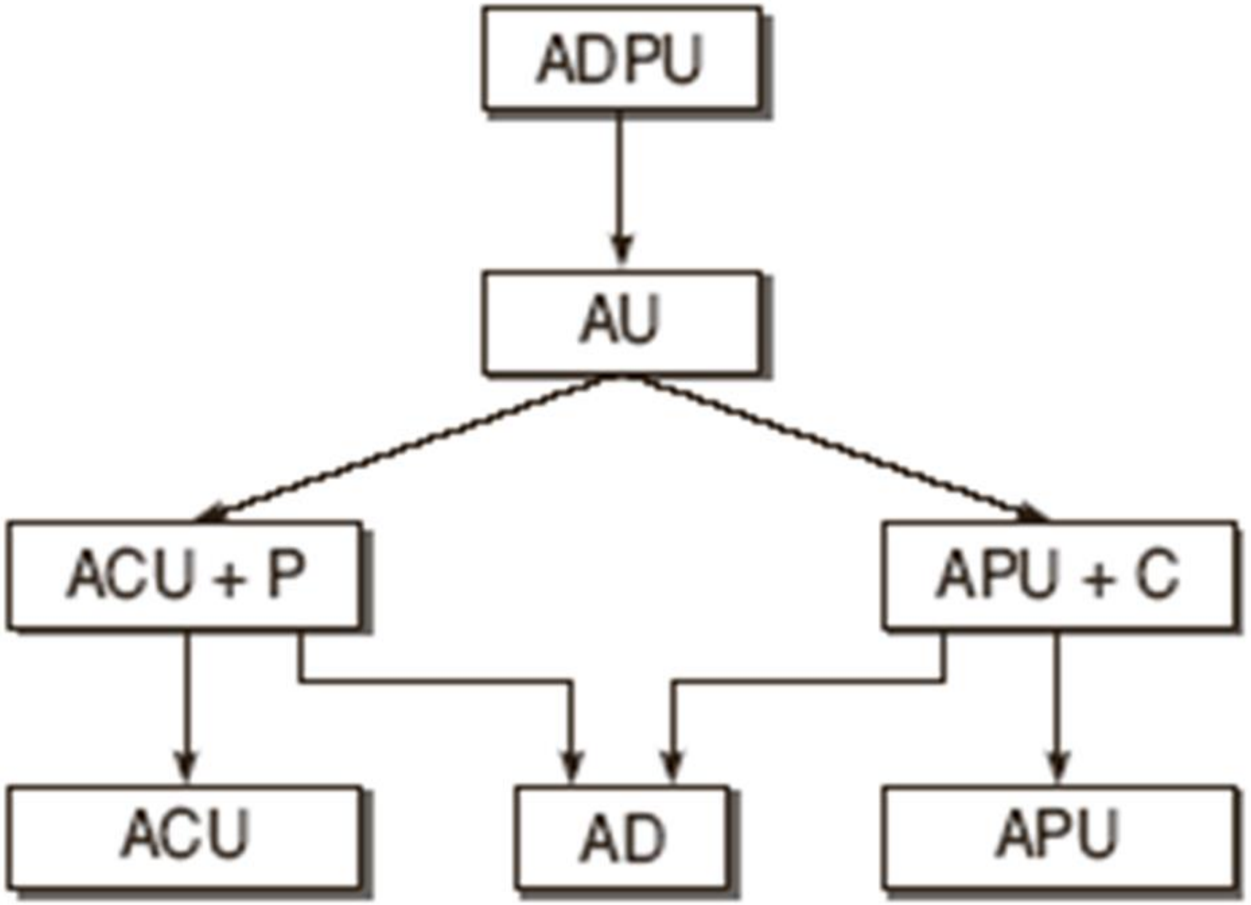
Strategy	Payment	Work
All Uses(AU)	3-4-5-6-7	1-2
	10-11	1-2-3-4
	10-11-12	1-2-3-4-5-6
	12-13-14-15-16	1-2-3-4-5-6-7
	3-4-5-6-8-9-10	1-2-3-4-5-6-8-9-10
All p-uses(APU)	0-1-2-3-4-5-6-8-9-10-11	1-2
		1-2-3-4
		1-2-3-4-5-6
All c-uses(ACU)	0-1-2-16	1-2-3-4-5-6-7
	3-4-5-6-7	1-2-3-4-5-6-8-9-10
	3-4-5-6-8-9-10	
	3-4-15-16	
	7-14-15-16	
	10-11-12	
	10-11-13-14-15-16	
	12-13-14-15-16	

Strategy	Payment	Work
All-p-uses / Some-c-uses (APU + C)	0-1-2-3-4-5-6-8-9-10-11	1-2
	10-11-12	1-2-3-4
	12-13-14-15-16	1-2-3-4-5-6
		1-2-3-4-5-6-8-9-10
All-c-uses / Some-p-uses (ACU + P)	0-1-2-16	1-2-3-4-5-6-7
	3-4-5-6-7	1-2-3-4-5-6-8-9-10
	3-4-5-6-8-9-10	1-2-3-4-5-6
	3-4-15-16	
	7-14-15-16	
	10-11-12	
	10-11-13-14-15-16	
	12-13-14-15-16	
	0-1-2-3-4-5-6-8-9-10-11	

Strategy	Payment	Work
All-du-paths (ADUP)	0-1-2-3-4-5-6-8-9-10-11	1-2
	0-1-2-16	1-2-3-4
	3-4-5-6-7	1-2-3-4-5-6
	3-4-5-6-8-9-10	1-2-3-4-5-6-7
	3-4-15-16	1-2-3-4-5-6-8-9-10
	7-14-15-16	
	10-11-12	
	10-11-13-14-15-16	
	12-13-14-15-16	
All Definitions (AD)	0-1-2-16	1-2
	3-4-5-6-7	
	7-14-15-16	
	10-11	
	12-13-14-15-16	

Ordering of data flow testing strategies

- While selecting a test case, we need to analyse the relative strengths of various data flow testing strategies.



Ordering of data flow testing strategies cont…

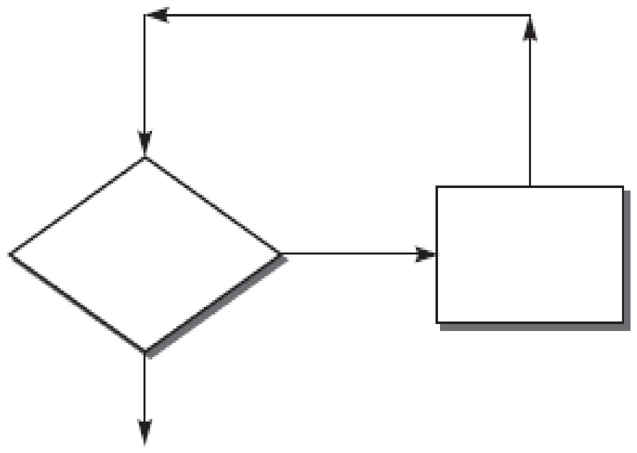
- The previous Figure depicts the relative strength of the data flow strategies.
- In this figure, the relative strength of testing strategies reduces along the direction of the arrow.
- It means that all-du-paths (ADUP) is the strongest criterion for selecting the test cases.

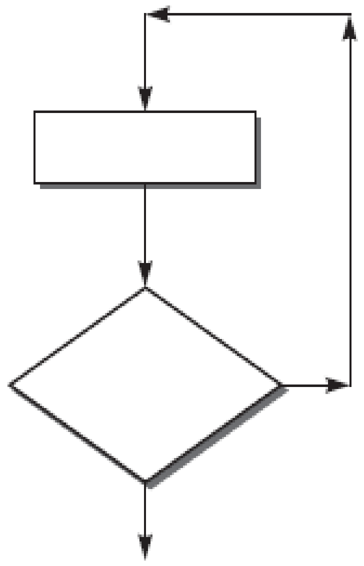
Loop testing

- Loop testing can be viewed as an extension to branch coverage.
- Loops are important in the software from the testing view point. If loops are not tested properly, bugs can go undetected.
- Loop testing can be done effectively while performing development testing (unit testing by the developer) on a module.
- Sufficient test cases should be designed to test every loop thoroughly.
- There are four different kinds of loops. How each kind of loop is tested, is discussed below.

Simple loops

- Simple loops mean, we have a single loop in the flow, as shown below.





Testing Simple Loops

- Check whether you can bypass the loop or not. If the test case for bypassing the loop is executed and, still you enter inside the loop, it means there is a bug.
- Check whether the loop control variable is negative.
- Write one test case that executes the statements inside the loop.

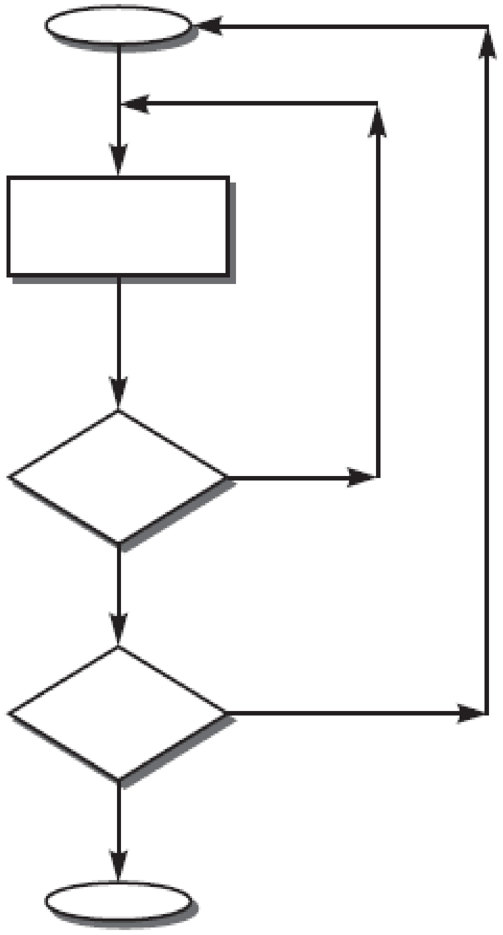
Testing Simple Loops cont …

- Write test cases for a typical number of iterations through the loop.
- Write test cases for checking the boundary values of the maximum and minimum number of iterations defined (say max and min) in the loop. It means we should test for min, min+1, min−1, max−1, max, and max+1 number of iterations through the loop.

Nested loops

- When two or more loops are embedded, it is called a nested loop, as shown in next slide. If we have nested loops in the program, it becomes difficult to test.

Nested loops



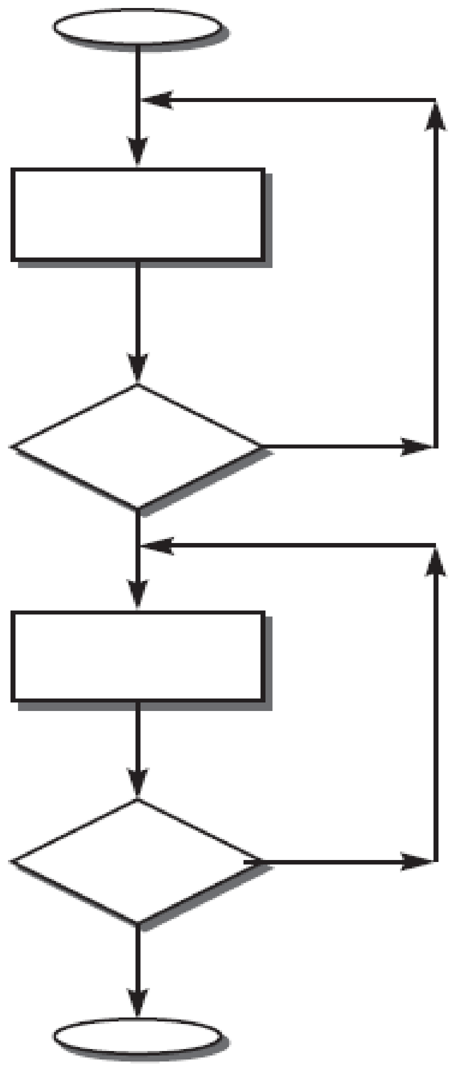
Testing Nested Loops

- If we adopt the approach of simple tests to test the nested loops, then the number of possible test cases grows geometrically.
- Thus, the strategy is to start with the innermost loops while holding outer loops to their minimum values. Continue this outward in this manner until all loops have been covered

Concatenated loops

- The loops in a program may be concatenated (shown in next slide).
- Two loops are concatenated if it is possible to reach one, after exiting the other, while still on a path from entry to exit.
- If the two loops are not on the same path, then they are not concatenated.
- The two loops on the same path may or may not be independent.
- If the loop control variable for one loop is used for another loop, then they are concatenated, but nested loops should be treated like nested only.

Concatenated loops



Testing Concatenated Loops

- The concatenated loop may be treated as a sequence of two or more numbers of simple loops.
- So, the strategy for testing of simple loops may be extended to testing of concatenated loops.

Unstructured loops

- This type of loops is really impractical to test.
- They must be redesigned or at least converted into simple or concatenated loops.

Thank you