Software Verification & Validation

Natthapong Jungteerapanich

Structural and Data Flow Testing

Acknowledgement

- Some slides in this lecture are adapted from

  - **Paul Ammann and Jeff Offutt's** slides for their textbook **"Introduction to Software Testing".** Cambridge University Press, 2008.

  - **Lee Copeland's** "**A Practitioner's Guide to Software Test Design**". Artech House, 2004.

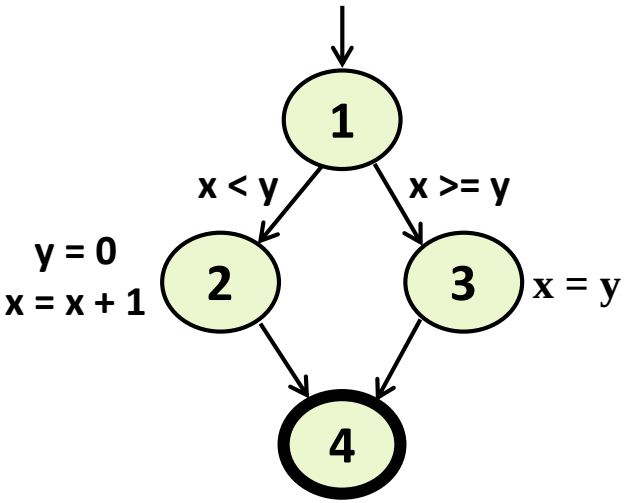
Control Flow Graphs

- A CFG models all executions of a method by describing control structures
- Nodes : Statements or sequences of statements (basic blocks)
- Edges : Transfers of control
- Basic Block : A sequence of statements such that if the first statement is executed, all statements will be (no branches)

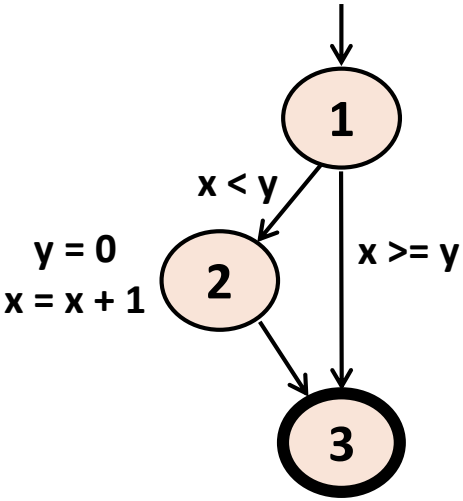
- CFGs are sometimes annotated with extra information
  - branch predicates
  - defs
  - uses
- Rules for translating statements into graphs …

CFG : The if Statement

```
if (x < y)
{
  y = 0;
  x = x + 1;
}
else
{
  x = y;
}
```

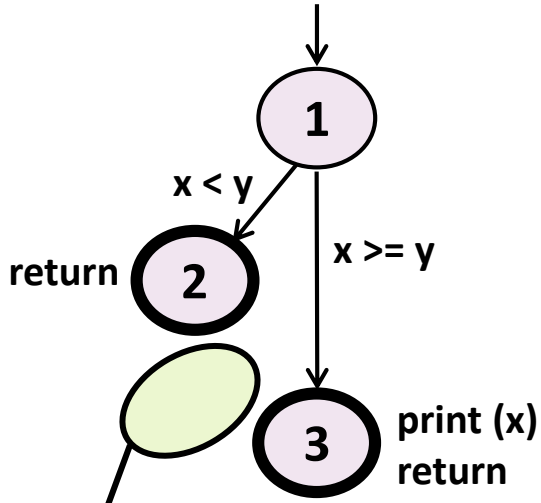



```
if (x < y)
{
  y = 0;
  x = x + 1;
}
```



CFG : The if-Return Statement

```
if (x < y)
{
  return;
}
print (x);
return;
```



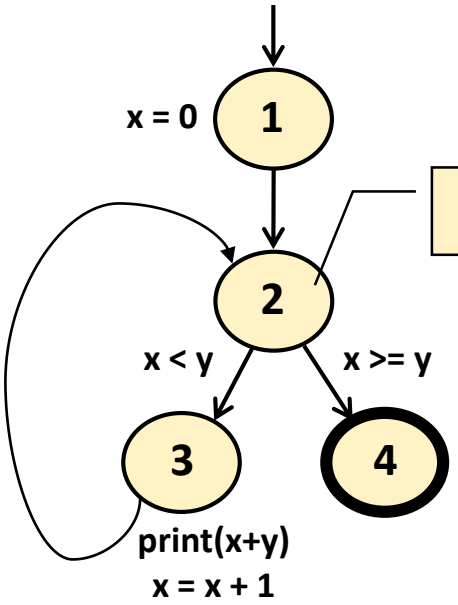
No edge from node 2 to 3.
The return nodes must be distinct.

Loops

- Loops require "*extra*" nodes to be added
- Nodes that do not represent statements or basic blocks

CFG : while and for Loops

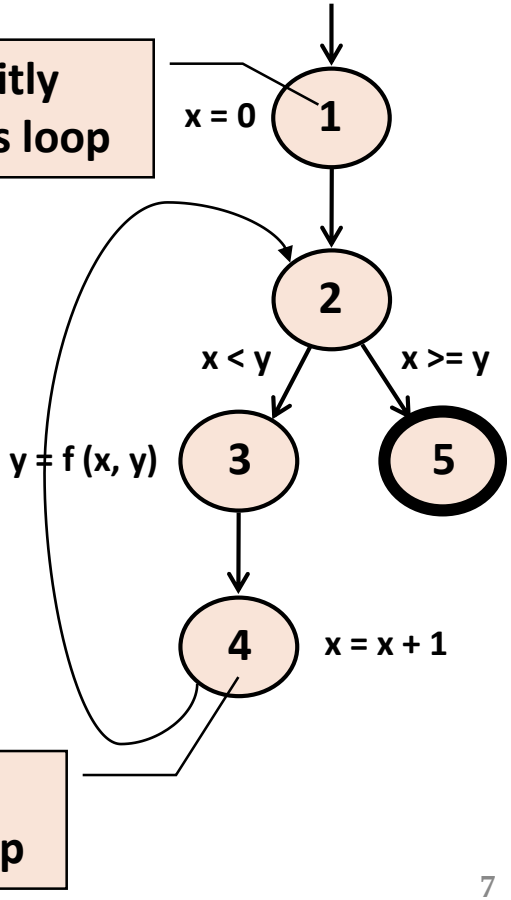
```
x = 0;
while (x < y)
{
  print(x+y);
  x = x + 1;
}
```



dummy node

implicitly initializes loop

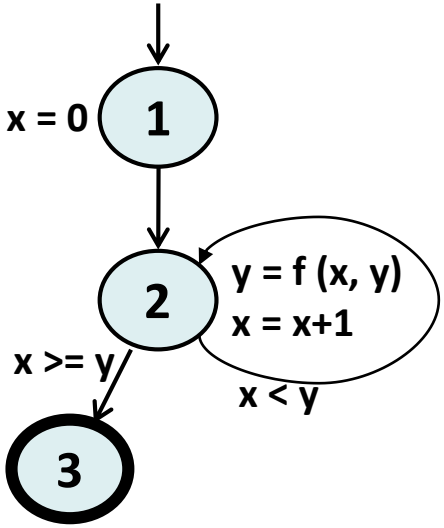
```
for (x = 0; x < y; x++)
{
  y = f (x, y);
}
```



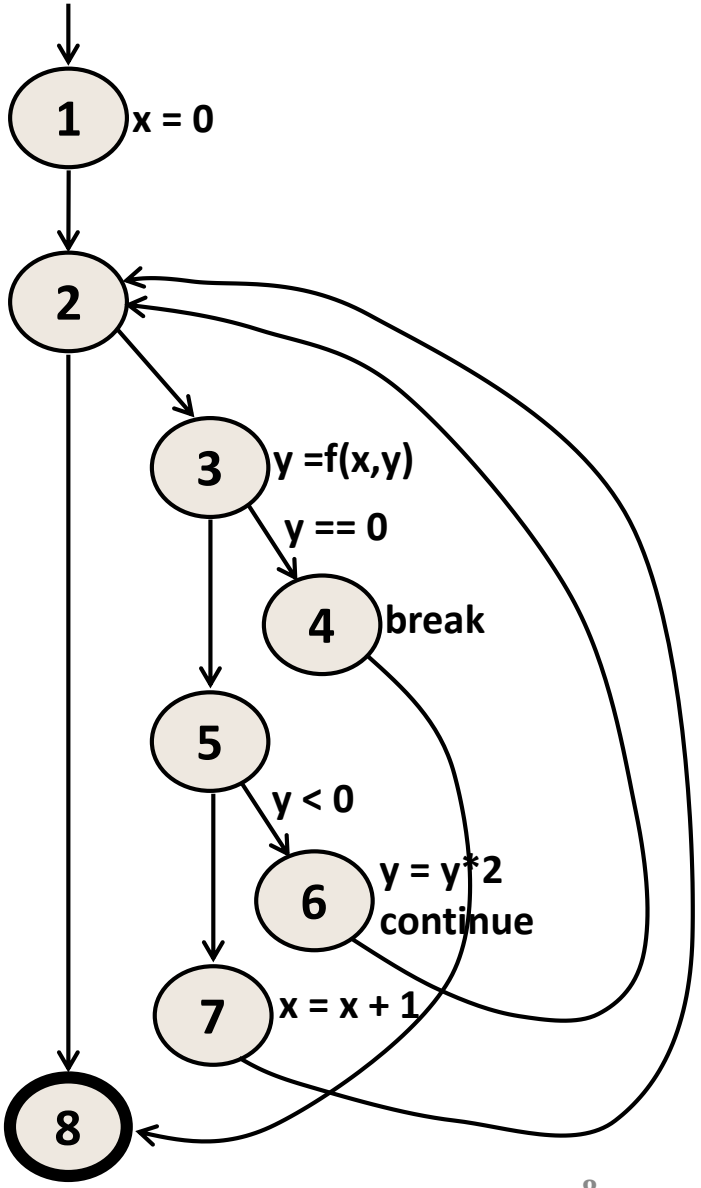
implicitly increments loop

CFG : do Loop, break and continue

```
x = 0;
do
{
   y = f (x, y);
   x = x + 1;
} while (x < y);
println (y)
```

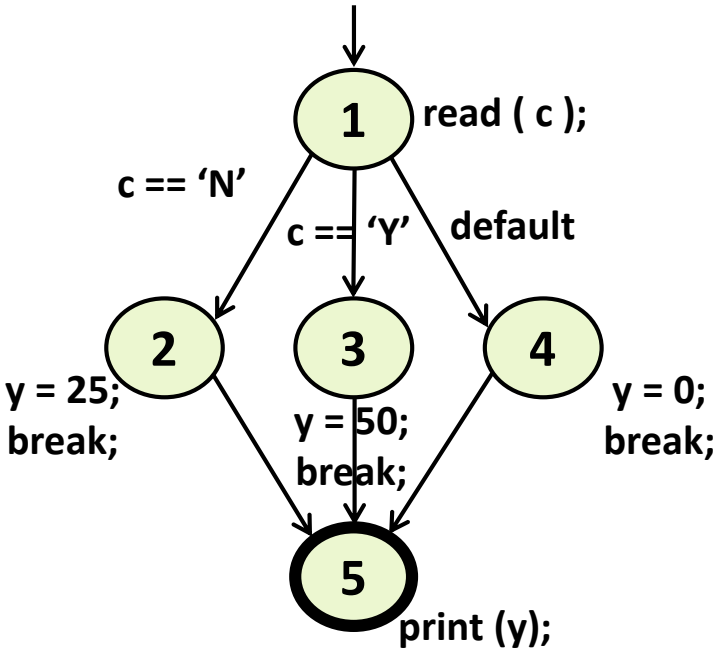



```
x = 0;
while (x < y)
{
   y = f (x, y);
   if (y == 0)
   {
      break;
   } else if y < 0)
   {
      y = y*2;
      continue;
   }
   x = x + 1;
}
print (y);
```



CFG : The case (switch) Structure

```
read ( c) ;
switch ( c )
{
  case ‘N’:
    y = 25;
    break;
  case ‘Y’:
    y = 50;
    break;
  default:
    y = 0;
    break;
}
print (y);
```

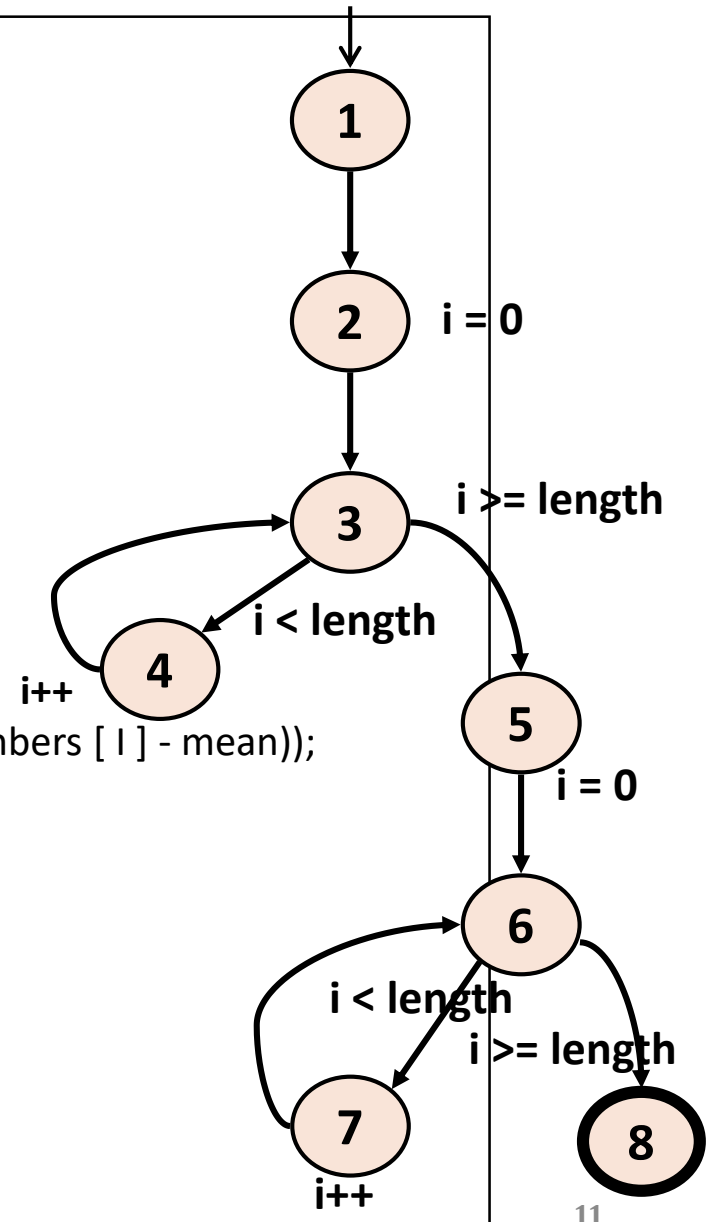


Example Control Flow – Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }
    med   = numbers [ length / 2];
    mean = sum / (double) length;

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum  + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt ( var );

    System.out.println ("length:                " + length);
    System.out.println ("mean:                  " + mean);
    System.out.println ("median:                " + med);
    System.out.println ("variance:               " + var);
    System.out.println ("standard deviation: " + sd);
}
```

Control Flow Graph for Stats

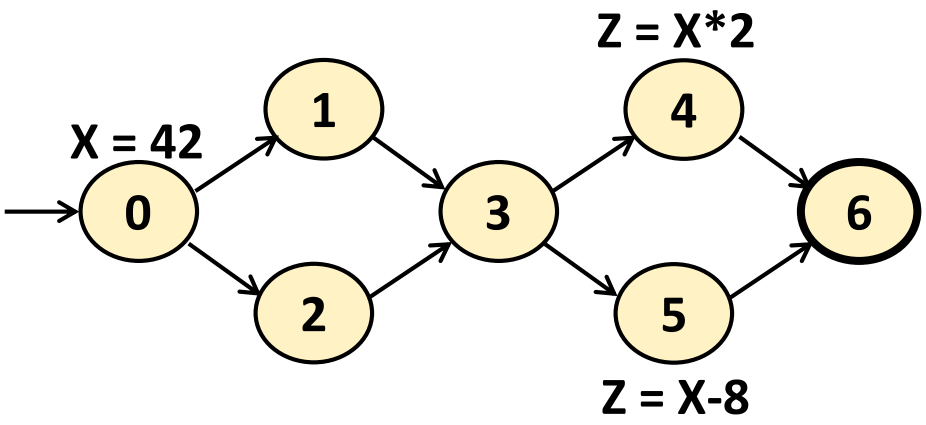
```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }
    med   = numbers [ length / 2];
    mean = sum / (double) length;

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum  + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt ( var );

    System.out.println ("length:                " + length);
    System.out.println ("mean:                  " + mean);
    System.out.println ("median:                " + med);
    System.out.println ("variance:              " + var);
    System.out.println ("standard deviation: " + sd);
}
```



Data Flow Criteria

Goal: Try to ensure that values are computed and used correctly

- Definition (Def) : A location where a value for a variable is stored into memory
- Use : A location where a variable's value is accessed
- Def(n) : The set of variables that are defined by node n
- Use(n) : The set of variables that are used by node n



Defs: Def (0) = {X}

Def (4) = {Z}

Def (5) = {Z}

Uses: Use (4) = {X}

Use (5) = {X}

Data Flow Coverage for Source Code

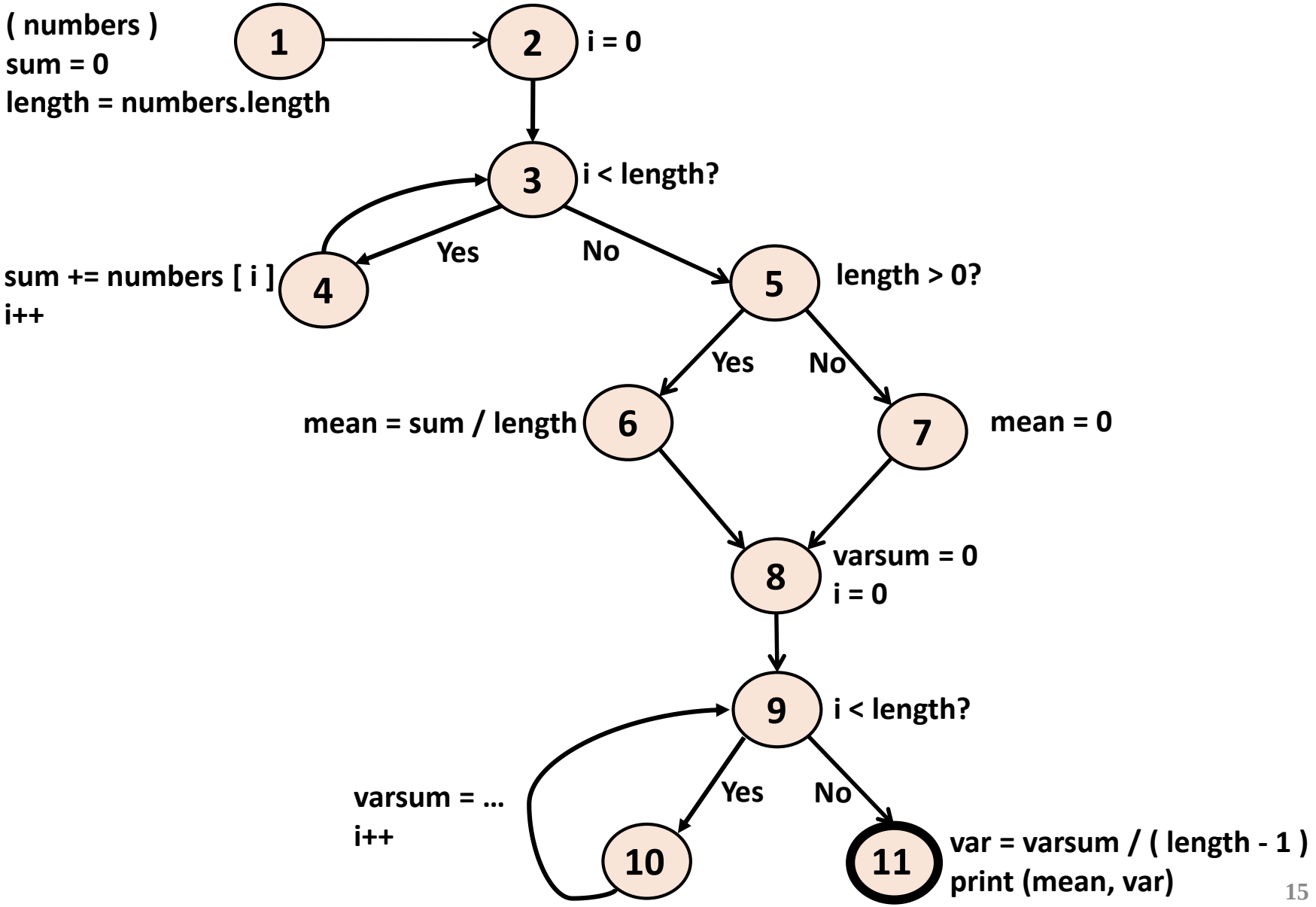
- Def : a location where a value is stored into memory
  - x appears on the left side of an assignment (x = 44;)
  - x is an actual parameter in a call and the method changes its value
  - x is a formal parameter of a method (implicit def when method starts)
  - x is an input to a program

- Use : a location where variable's value is accessed
  - x appears on the right side of an assignment
  - x appears in a conditional test
  - x is an actual parameter to a method
  - x is an output of the program
  - x is an output of a method in a return statement

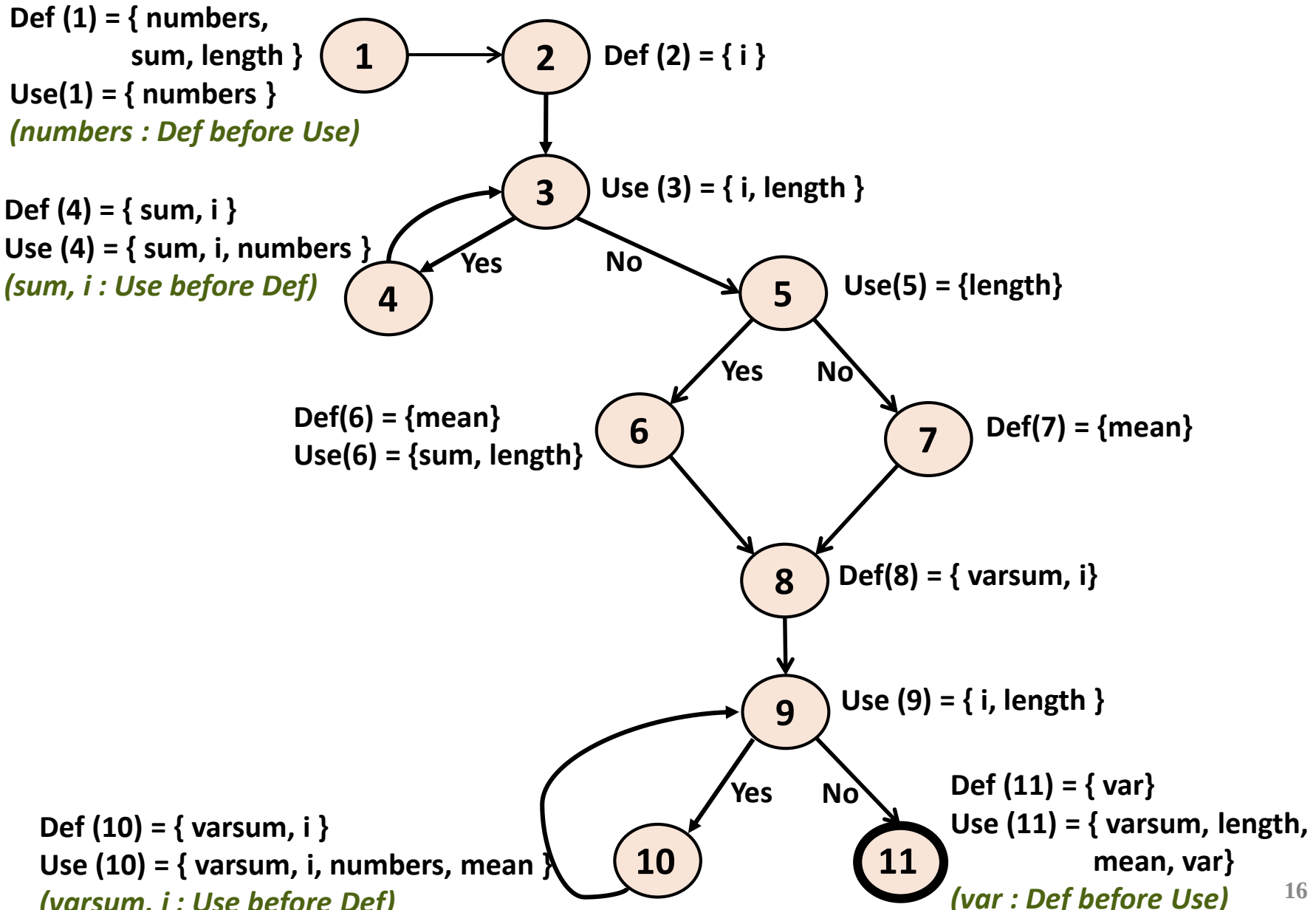
Example Data Flow – **Stats**

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double var, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }

    if (length > 0)
       mean = sum / length;
    else
       mean = 0;

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum  + ((numbers [ i ] - mean) * (numbers [ i ] - mean));
    }
    var = varsum / ( length - 1 );

    System.out.println("mean: " + mean);
    System.out.println("variance: " + var);
}
```

Control Flow Graph for **Stats**



Control Flow Graph for **Stats** – with Defs and Uses



Defs and Uses Tables for **Stats**

Node	Def	Use	Note
1	numbers, sum, length	numbers	numbers: Def before Use
2	i		
3		i, length	
4	sum, i	sum, i, numbers	sum, i : Use before Def
5		length	
6	mean	sum, length	
7	mean		
8	varsum, i		
9		i, length	
10	varsum, i	varsum, i, numbers, mean	varsum, i : Use before Def
11	var	varsum, length, mean, var	var : Def before Use

DU Paths

- **Def-clear** : A path from m to n is ***def-clear*** with respect to variable *v* if *v* is not defined at any node in the path, except possibly at the first node.

- **DU-path**: A **DU-path** with respect to variable *v* is a simple path from a node where v is defined to a node where v is used *such that v is def clear on that path.*

- **Reach** : If there is a DU-path from *m* to *n* with respect to *v*, we say that the def of *v* at *m* **reaches** the use of *v* at *n.*

Touring DU-Paths

- Three criteria:
  - Use every def
  - Get to every use
  - Follow all du-paths

Data Flow Test Criteria

- First, we make sure every def reaches a use

All-Defs Coverage Criterion: For each variable *v* and each Def node *n* of *v*, if there is a DU-Path with respect to *v* from *n*, then at least one of such DU-Path must be toured by some test path.

Given a test suite T, the degree of All-Defs coverage of T can be calculated form:

$$C_{\text{All-Defs}} = \frac{\text{Number of (v,n) pairs where v is defined at n and some DU-Path w.r.t. v from n is toured by a test path in T}}{\text{Number of (v,n) pairs where v is defined at n and there is some DU-Path w.r.t. v from n}} \times 100\%$$

Data Flow Test Criteria

- Then we make sure that every def reaches all possible uses

All-Uses Coverage Criterion: For each variable *v*, each Def node *n* of *v*, and each Use node *m* of *v*, if there is a DU-Path with respect to *v* from *n to m*, then at least one of such DU-Path must be toured by some test path.

- All-Use Coverage is also called **All-DU-Pairs Coverage**.

Given a test suite T, the degree of All-Uses coverage of T can be calculated form:

$$C_{\text{All-Uses}} = \frac{\text{Number of (v,n,m) pairs where v is defined at n and used at m and some DU-Path w.r.t. v from n to m is toured by a test path in T}}{\text{Number of (v,n,m) pairs where v is defined at n and used at m and there is some DU-Path w.r.t. v from n to m}} \times 100\%$$

Data Flow Test Criteria

- Finally, we cover all the paths between defs and uses

All-DU-Paths Coverage Criterion: For each variable *v*, each Def node *n* of *v,* and each Use node *m* of *v*, every DU-Path with respect to *v* from *n* to *m* must be toured by some test path.

Given a test suite T, the degree of All-DU-Paths coverage of T can be calculated form:

$$C_{\text{All-DU-Paths}} = \frac{\text{Number of DU-Paths (w.r.t. any variable) toured by a test path in T}}{\text{Number of DU-Paths (w.r.t. any variable)}} \times 100\%$$

DU Paths for Stats

- To find out what DU Paths that the test paths must tour, we first find all the DU paths for each variable.
- Let start with ***numbers:***

Variable	Def	Use	DU Path
numbers	1	1	[1]
	1	4	[1, 2, 3, 4]
	1	10	[1, 2, 3, 5, 6, 8, 9, 10] [1, 2, 3, 5, 7, 8, 9, 10]

- ***numbers*** is defined at node 1, but is used at three nodes: 1, 4, 10.
- Notice that at node 1, **numbers** is ***Def*** before ***Use***.
- To satisfy All-Def Coverage, one of the DU paths above must be toured by some test path.

DU Paths for Stats

Variable	Def	Use	DU Path
numbers	1	1	[1]
	1	4	[1, 2, 3, 4]
	1	10	[1, 2, 3, 5, 6, 8, 9, 10] [1, 2, 3, 5, 7, 8, 9, 10]

- To satisfy All-Use Coverage, one DU path for each Use node above must be toured, i.e.
  - both [1] and [1, 2, 3, 4] must be toured by some test paths, and
  - Either [1, 2, 3, 5, 6, 8, 9, 10] or [1, 2, 3, 5, 7, 8, 9, 10] must be toured by some test paths.

- To satisfy All-DU-Paths Coverage, each DU path above must be toured by some test path.

DU Paths for Stats

- We then have to find the DU paths for other variables.
- Consider ***varsum****:*

Variable	Def	Use	DU Path
varsum	8	10	[8, 9, 10]
	8	11	[8, 9, 11]
	10	10	[10, 9, 10]
	10	11	[10, 9, 11]

- ***varsum*** is defined at nodes 8 and 10, and is used at nodes 10 and 11.

- Observe that, although **varsum** is Def at 10 and Use at 10, the path [10] is not a DU Path, because **varsum** is Use before Def at node 10.

DU Paths for Stats

Variable	Def	Use	DU Path
varsum	8	10	[8, 9, 10]
	8	11	[8, 9, 11]
	10	10	[10, 9, 10]
	10	11	[10, 9, 11]

- To satisfy All-Def Coverage, one of the DU paths for each Def node above must be toured by some test path, i.e.
  - Either [8, 9, 10] or [8, 9, 11] must be toured by some test path, **and**
  - Either [10, 9, 10] and [10, 9, 11] must be toured by some test path.

- To satisfy All-Use Coverage, one DU path for each Def-Use pair of nodes must be toured. In this case, each path above must be toured by some test path.

- To satisfy All-DU-Paths Coverage, each DU-Path above must be toured by some test path.

DU Paths for Stats

The following is the complete table of DU-Paths for each variable.

Variable	Def	Use	DU Path
numbers	1	1	[1]
	1	4	[1, 2, 3, 4]
	1	10	[1, 2, 3, 5, 6, 8, 9, 10] [1, 2, 3, 5, 7, 8, 9, 10]
varsum	8	10	[8, 9, 10]
	8	11	[8, 9, 11]
	10	10	[10, 9, 10]
	10	11	[10, 9, 11]
sum	1	4	[1, 2, 3, 4]
	1	6	[1, 2, 3, 5, 6]
	4	4	[4, 3, 4]
	4	6	[4, 3, 5, 6]

DU Paths for Stats

Variable	Def	Use	DU Path
length	1	3	[1, 2, 3]
	1	5	[1, 2, 3, 5]
	1	6	[1, 2, 3, 5, 6]
	1	9	[1, 2, 3, 5, 6, 8, 9] [1, 2, 3, 5, 7, 8, 9]
	1	11	[1, 2, 3, 5, 6, 8, 9, 11] [1, 2, 3, 5, 7, 8, 9, 11]
mean	6	10	[6, 8, 9, 10]
	6	11	[6, 8, 9, 11]
	7	10	[7, 8, 9, 10]
	7	11	[7, 8, 9, 11]
var	11	11	[11]

DU Paths for Stats

Variable	Def	Use	DU Path
i	2	3	[2, 3]
	2	4	[2, 3, 4]
	2	9	-
	2	10	-
	4	3	[4, 3]
	4	4	[4, 3, 4]
	4	9	-
	4	10	-
	8	3	-
	8	4	-
	8	9	[8, 9]
	8	10	[8, 9, 10]
	10	3	-
	10	4	-
	10	9	[10, 9]
	10	10	[10, 9, 10]

Test Cases and Test Paths

- The following set of test paths satisfies the **All-Defs coverage**:
  - [1, 2, 3, 5, 7, 8, 9, 11]
  - [1, 2, 3, 4, 3, 5, 6, 8, 9, 10, 9, 11]

- The test set that covers these two paths must contain the following test inputs:
  - **numbers** = the empty array
  - **numbers** = an array of length one

Test Cases and Test Paths

- The following set of test paths satisfies the **All-Use coverage**:
  - [1, 2, 3, 5, 6, 8, 9, 11]
  - [1, 2, 3, 5, 7, 8, 9, 11]
  - [1, 2, 3, 5, 7, 8, 9, 10, 9, 11]
  - [1, 2, 3, 4, 3, 4, 3, 5, 6, 8, 9, 10, 9, 10, 9, 11]

- Is there a test set that satisfies All-Use coverage?

Test Cases and Test Paths

- The following set of test paths satisfies the **All-DU-Paths coverage**:
  - [1, 2, 3, 5, 6, 8, 9, 11]
  - [1, 2, 3, 5, 7, 8, 9, 11]
  - [1, 2, 3, 5, 6, 8, 9, 10, 9, 11]
  - [1, 2, 3, 5, 7, 8, 9, 10, 9, 10, 9, 11]
  - [1, 2, 3, 4, 3, 4, 3, 5, 6, 8, 9, 11]

- Is there a test set that satisfies All-DU-Paths coverage?