Data Flow Coverage for Source

- 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

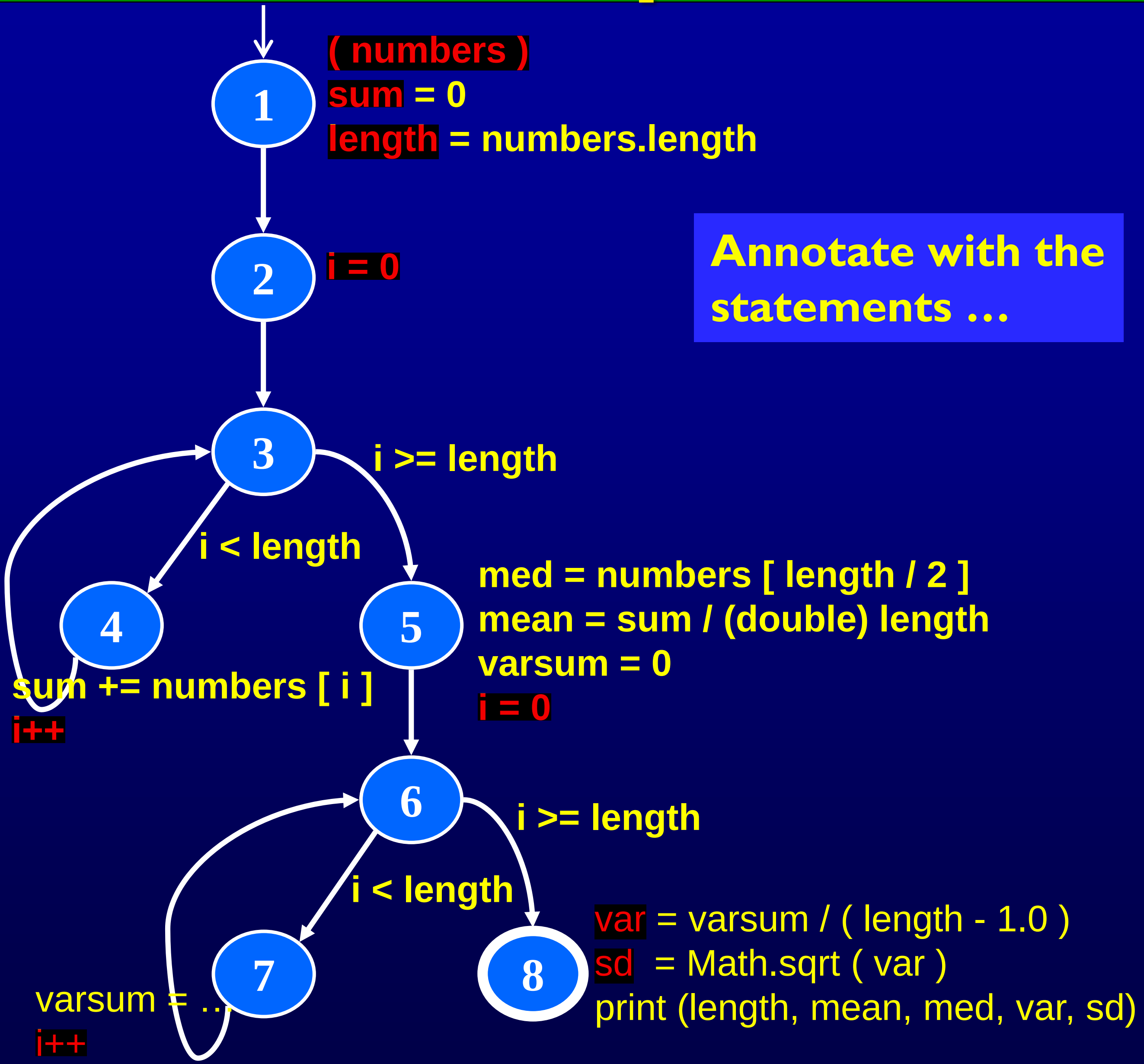
    یه پارامتری را به متدی
    پاس بدیم
  - x is an output of the program
  - x is an output of a method in a return statement
- If a def and a use appear on the same node, then it is only a DU-pair if the def occurs after the use and the node is in a loop

Example Data Flow – Stats

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

    sum = 0.0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }
    med   = numbers [ length / 2 ];
    mean = sum / (double) length;

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

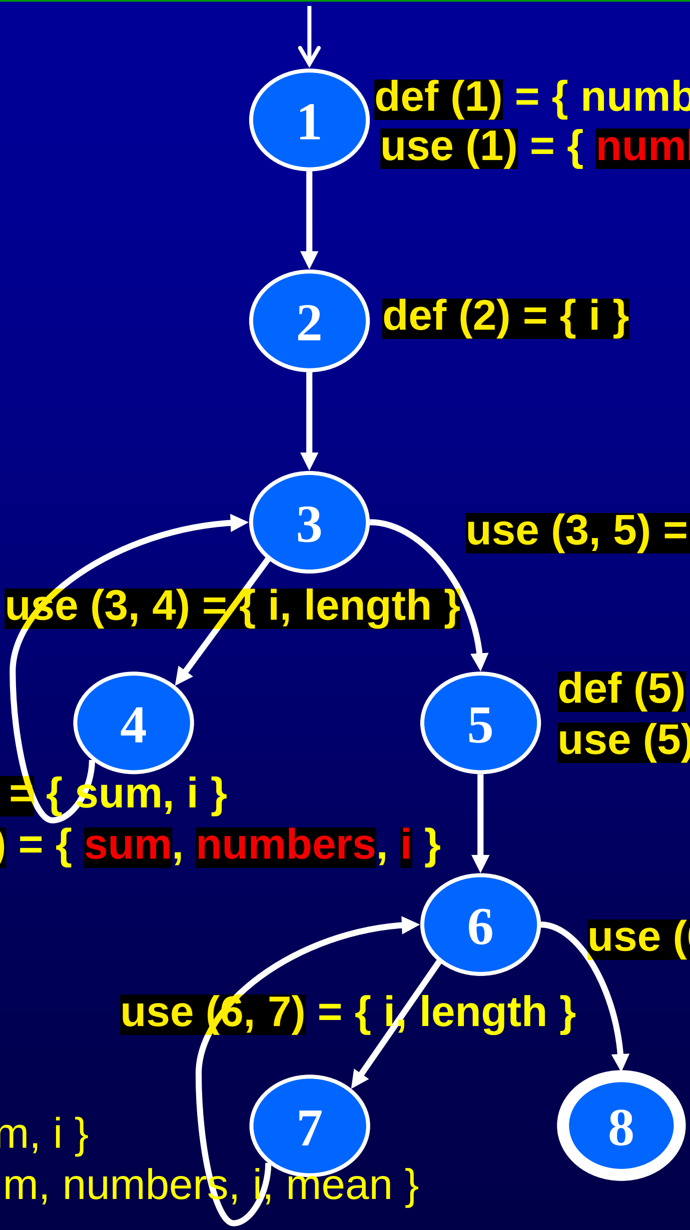
def

Control Flow Graph for Stats



Annotate with the statements …

CFG for Stats – With Defs & Uses



def (1) = { numbers, sum, length }
use (1) = { numbers}

def (2) = { i }

use (3, 5) = { i, length }

use (3, 4) = { i, length }

def (5) = { med, mean, varsum, i }
use (5) = { numbers, length, sum }

def (4) = { sum, i }
use (4) = { sum, numbers, i }

use (6, 8) = { i, length }

use (6, 7) = { i, length }

def (8) = { var, sd }
use (8) = { varsum, length, mean,
med, var, sd }

def (7) = { varsum, i }
use (7) = { varsum, numbers, i, mean }

Turn the annotations into def and use sets …

Defs and Uses Tables for Stats

Node	Def	Use
1	{ numbers, sum, length }	{ numbers }
2	{ i }	
3		
4	{ sum, i }	{ numbers, i, sum }
5	{ med, mean, varsum, i }	{ numbers, length, sum }
6		
7	{ varsum, i }	{ varsum, numbers, i, mean }
8	{ var, sd }	{ varsum, length, var, mean, med, var, sd }

Edge	Use
(1, 2)	
(2, 3)	
(3, 4)	{ i, length }
(4, 3)	
(3, 5)	{ i, length }
(5, 6)	
(6, 7)	{ i, length }
(7, 6)	
(6, 8)	{ i, length }

DU Pairs for Stats

variable	DU Pairs
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(2, 4) (2, (3,4)) (2, (3,5)) ~~(2, 7) (2, (6,7)) (2, (6,8))~~ (4, 4) (4, (3,4)) (4, (3,5)) ~~(4, 7) (4, (6,7)) (4, (6,8))~~ (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))

defs come before uses, do not count as DU pairs

defs after use in loop, these are valid DU pairs

No def-clear path … different scope for i

A simple subpath that is def-clear with respect to v from a def of v to a use of v

DU Paths for Stats

variable	DU Pairs	DU Paths
numbers	(1, 4) (1, 5) (1, 7)	[1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7]
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))	[1, 2, 3, 5] [1, 2, 3, 5, 6, 8] [1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7] [1, 2, 3, 5, 6, 8]
med	(5, 8)	[5, 6, 8]
var	(8, 8)	*No path needed*
sd	(8, 8)	*No path needed*
sum	(1, 4) (1, 5) (4, 4) (4, 5)	[1, 2, 3, 4] [1, 2, 3, 5] [4, 3, 4] [4, 3, 5]

variable	DU Pairs	DU Paths
mean	(5, 7) (5, 8)	[5, 6, 7] [5, 6, 8]
varsum	(5, 7) (5, 8) (7, 7) (7, 8)	[5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 8]
i	(2, 4) (2, (3,4)) (2, (3,5)) (4, 4) (4, (3,4)) (4, (3,5)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))	[2, 3, 4] [2, 3, 4] [2, 3, 5] [4, 3, 4] [4, 3, 4] [4, 3, 5] [5, 6, 7] [5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 7] [7, 6, 8]

DU Paths for Stats—No Duplicates

There are 38 DU paths for Stats, but only 12 unique

✦ [1, 2, 3, 4]

★ [1, 2, 3, 5]

✦ [1, 2, 3, 5, 6, 7]

★ [1, 2, 3, 5, 6, 8]

✦ [2, 3, 4]

★ [2, 3, 5]

[4, 3, 4] ✹

[4, 3, 5] ✦

[5, 6, 7] ✦

[5, 6, 8] ★

[7, 6, 7] ✹

[7, 6, 8] ✦

★ 4 expect a loop not to be "entered"

✦ 6 require at least one iteration of a loop

✹ 2 require at least two iterations of a loop

Test Cases and Test Paths

Test Case : numbers = (44) ; length = 1

Test Path : [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]

Additional DU Paths covered (no sidetrips)

[1, 2, 3, 4] [2, 3, 4] [4, 3, 5] [5, 6, 7] [7, 6, 8]

The five stars ✦ that require at least one iteration of a loop

Test Case : numbers = (2, 10, 15) ; length = 3

Test Path : [1, 2, 3, 4, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 7, 6, 8]

DU Paths covered (no sidetrips)

[4, 3, 4] [7, 6, 7]

The two stars ✹ that require at least two iterations of a loop

Other DU paths ★ require arrays with length 0 to skip loops

But the method fails with index out of bounds exception…

```
med = numbers [length / 2];
```

A fault was found

Summary

- Applying the graph test criteria to control flow graphs is relatively straightforward
  - Most of the developmental research work was done with CFGs
- A few subtle decisions must be made to translate control structures into the graph
- Some tools will assign each statement to a unique node
  - These slides and the book uses basic blocks