

**Last time.** We discussed benefits of exploratory testing and a sketch of how to do it. We also started to do a case study of WaterlooWorks.

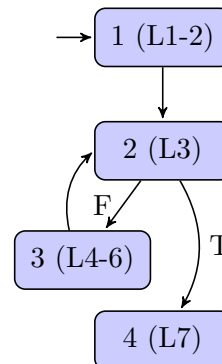
## Source Code Coverage Criteria

We alluded to statement coverage and branch coverage. It is possible to evaluate these criteria directly on the source code, but better to use a sensible intermediate representation. The fundamental graph for source code is the *Control-Flow Graph* (CFG), which originates from compilers.

- CFG nodes: a node represents zero or more statements;
- CFG edges: an edge  $(s_1, s_2)$  indicates that  $s_1$  may be followed by  $s_2$  in an execution.

**Basic Blocks.** We can simplify a CFG by grouping together statements which always execute together (in sequential programs):

```
1      x = 5
2      z = 2
3 q0:  if (z < 17) goto q1
4      z = z + 1
5      print (x)
6      goto q0
7 q1:  nop
```



We use the following definition:

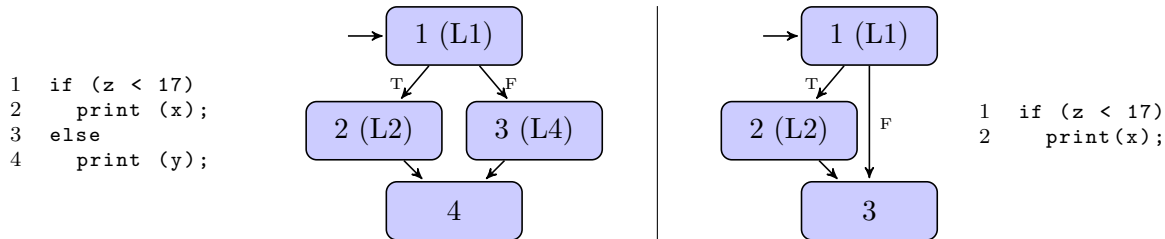
**Definition 1** *A basic block has one entry point and one exit point.*

Note that a basic block may have multiple successors. However, there may not be any jumps into the middle of a basic block (which is why statement 10 has its own basic block.)

## Some Examples

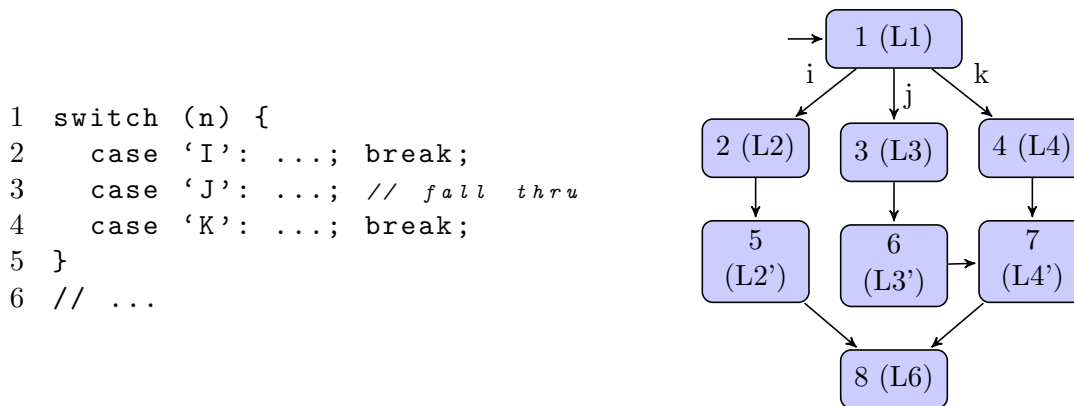
We'll now see how to construct control-flow graph fragments for various program constructs.

**if statements:** One can put the conditions (and hence uses) on the control-flow edges, rather than in the if node. I prefer putting the condition in the node.

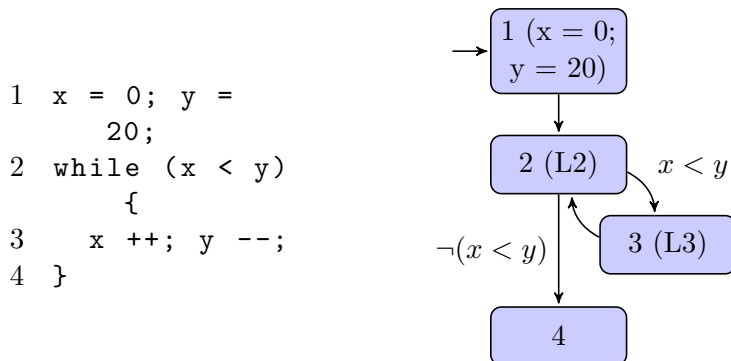


Short-circuit if evaluation is more complicated; I recommend working it out yourself.

**case / switch statements:**



**while statements:**



Note that arbitrarily complicated structures may occur inside the loop body.

**for statements:**

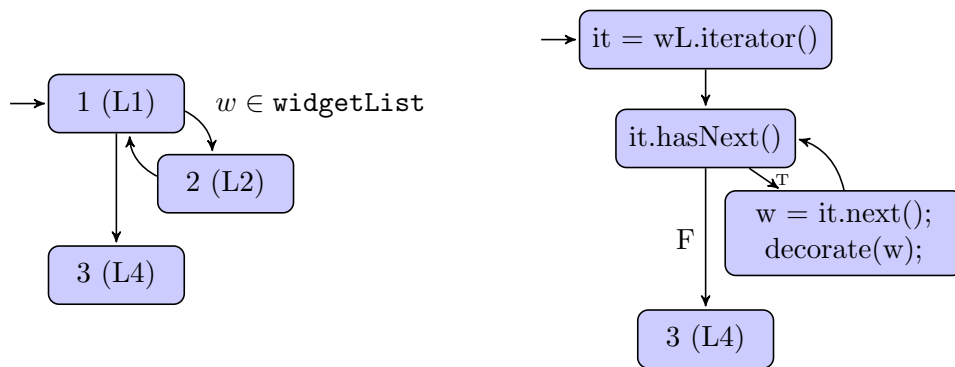
```
1 for (int i = 0; i < 57; i++) {  
2     if (i % 3 == 0) {  
3         print (i);  
4     }  
5 }
```

(an exercise for the reader!)

This example uses Java's enhanced for loops, which iterates over all of the elements in the `widgetList`:

```
1 for (Widget w : widgetList) {  
2     decorate(w);  
3 }  
4 // ...
```

I will accept the simplified CFG or the more useful one on the right:

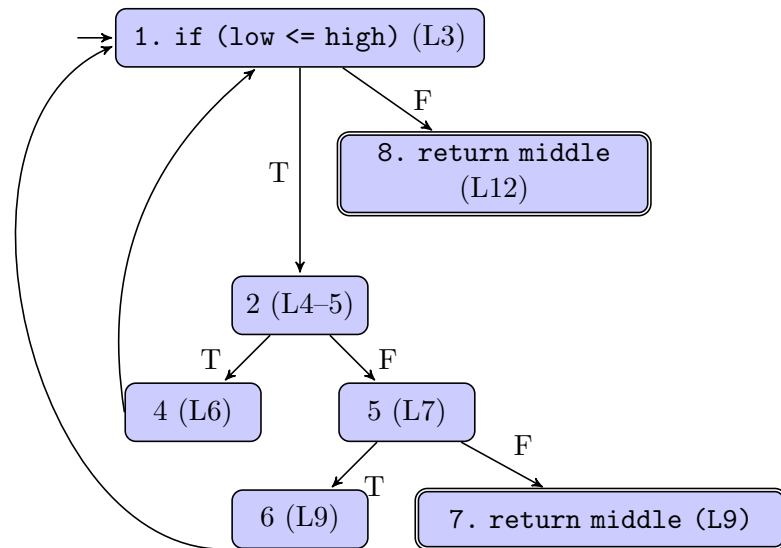


**Larger example.** You can draw a 7-node CFG for this program:

```

1  /** Binary search for target in sorted subarray a[low..high] */
2  int binary_search(int[] a, int low, int high, int target) {
3      while (low <= high) {
4          int middle = low + (high-low)/2;
5          if (target < a[middle])
6              high = middle - 1;
7          else if (target > a[middle])
8              low = middle + 1;
9          else
10             return middle;
11     }
12     return -1; /* not found in a[low..high] */
13 }

```



Here are more exercise programs that you can draw CFGs for.

```

1  /* effects: if x==null, throw NullPointerException
2     otherwise, return number of elements in x that are odd, positive or both. */
3  int oddOrPos(int[] x) {
4      int count = 0;
5      for (int i = 0; i < x.length; i++) {
6          if (x[i]%2 == 1 || x[i] > 0) {
7              count++;
8          }
9      }
10     return count;
11 }
12
13 // example test case: input: x=[-3, -2, 0, 1, 4]; output: 3

```

Finally, we have a really poorly-designed API (I'd give it a D at most, maybe an F) because it's impossible to succinctly describe what it does. **Do not design functions with interfaces like this.** But we can still draw a CFG, no matter how bad the code is.

```
1  /** Returns the mean of the first maxSize numbers in the array,  
2      if they are between min and max. Otherwise, skip the numbers. */  
3  double computeMean(int[] value, int maxSize, int min, int max) {  
4      int i, ti, tv, sum;  
5  
6      i = 0; ti = 0; tv = 0; sum = 0;  
7      while (ti < maxSize) {  
8          ti++;  
9          if (value[i] >= min && value[i] <= max) {  
10             tv++;  
11             sum += value[i];  
12         }  
13         i++;  
14     }  
15     if (tv > 0)  
16         return (double)sum/tv;  
17     else  
18         throw new IllegalArgumentException();  
19 }
```