Software Testing, Quality Assurance and Maintenance Winter 2017

Lecture 6 — January 16, 2017

Patrick Lam *version 2*

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.

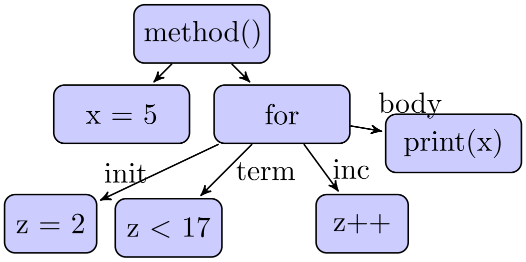
Example. Consider the following code.

```
x = 5;
for (z = 2; z < 17; z++)
  print(x);
```

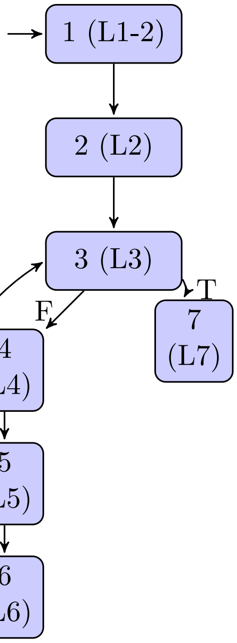
Recall the steps in compilation:

- lexing: input = stream of characters, output = stream of tokens (if, while, strings)
- parsing: input = stream of tokens, output = concrete syntax tree
- construction of Abstract Syntax Tree (AST): cleans up the concrete syntax tree
- conversion to Control Flow Graph: input = AST, output = CFG
- optimizations: input = CFG, output = CFG
- convert to bytecode/machine code: input = CFG, output = bytecode/machine code

The Abstract Syntax Tree corresponding to the example code might look like this:



From ASTs to CFGs. We can convert the Abstract Syntax Tree into the following Control Flow Graph (and we'll see how to do so in Lecture 7).



From CFG to low-level code. And we can convert the CFG into the following low-level code:

```
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
```