Overview

- Just getting the syntax (structure) of the language right doesn’t mean you are generating code that works correctly. (run-time errors or unexpected results)

- You need to understand not just how to write a program, but what each part of the program actually does.

- Example:
  • Here is a C program that declares an int with value 4, and increment it and assignment it to itself. (show funnyIncrement.c)

```c
#include <stdio.h>

void foo () {
    int i = 4;
    i = ++i + i++ + ++i;
    printf("foo: %d\n", i);
}

int main () {
    foo();
    return 0;
}
```

• What are the possible outputs of the program? [17 on my mac, 18 on gloin]

• The confusion here is when to execute the pre-increment operators.

• If we draw a simplified concrete syntax tree, it will be a tree like this

```
  =
 /   \
i    +
/     \
+ pre++
/   \
pre++ post++ i
  i   i
```

- my mac: execute let to right when the node is reached during tree traversal
- gloin: execute all pre- and post-increment before traversal
The tree can handle operator precedence. However, it does not guarantee the order or timing of the execution of the sub-trees.

A left-to-right pass to generate the machine code can produce different results than a right-to-left pass.

C language specification does not define the behavior of the program where there is more than one pre- or post-increment operator applied to a single variable within a single expression.

The behaviors end up being compiler specific. The semantic meaning of the program is undefined.

Example:

- Let int x = 3; x += x++ + ++x; This is legal in C and Java, so the syntax is correct.
- What about the semantics? After the two statements have been executed, what is the value of x? And does it matter if we consider it part of a C program or part of a Java program?
  - Yes, it matters.
  - If it is a C program, the value of x depends on the compiler.
  - If it is a Java program, the value of x is precisely defined, which is 11. x = 3 + 3 + 5.
    - Increment and decrement operators have a higher precedence than the other mathematical operators. They are applied from left to right in the order in which they occur unless one of them has been given a higher precedence through the parentheses.

Example:

- if (x++ == x++) {System.out.println("Yes");}
  - This condition is always false in Java.
- if (++x == x++) {System.out.println("Yes");}
  - This condition is always true in Java.