

# Assignment Expression

- Currently, you can only use assignment at the statement level, such as  $a = b$ ;
- It would be nice to have assignment available at the expression level too. For example consider  $b = c$  as an expression just like  $b + c$  is an expression with  $=$  as the binary operator instead of  $+$ .
- Then we could use statements like  $a = b = c$ ; (notice the similarity to  $a = b + c$ )

# Implementation

The production

stmt: asgn

already exists in the grammar.

Add the production

expr: asgn

When you do, your compiler will syntactically recognize statements like  $a = b = c$ ; but when you run programs containing assignment expressions, you will get a stack underflow.

# Stack Underflow

- The stack underflow problem occurs because the current assignment production removes all values from the stack when assignment is complete
- This is fine for the assignment statement, because values will never be used again
- It is not O.K. for the assignment expression. For  $a = b = c$ , the expression sets the value of  $c$  to  $b$ , and we want the value of  $b$  to be available to set to  $a$  at the statement level. We need to retain the value of  $b$  on the stack after the expression has been executed.

# Solution

- Instead of just having one asgn, you could have two

stmt: asgn1

expr: asgn2

- The semantic action for the production

asgn1: ID ASSIGN expr

is unchanged

- The semantic action for the production

asgn2: ID ASSIGN expr

Is the same as the semantic action for asgn1 except it contains a push instruction on \$1, i.e.

push(Inst(\$1));