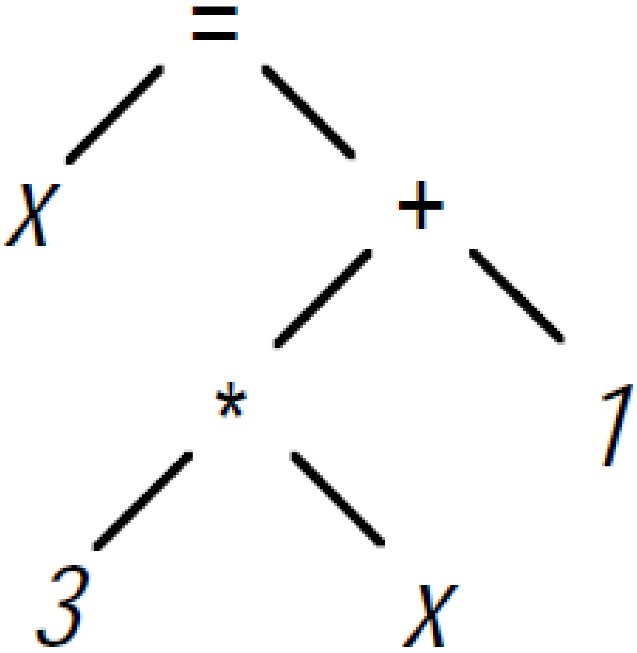
Abstract Syntax Tree (AST)

What is AST?

- An abstract syntax tree (AST) differs from a *concrete syntax tree* (or *parse tree*) in that it does not reflect the parsed productions but rather the logical structure of the compiled program.

Example

- An AST for the statement x = 3 * x + 1 could look like:



General idea…

- Every nonterminal symbol has an output attribute which returns the AST of this nonterminal.

- The AST of a production's left-hand-side nonterminal is built from the ASTs obtained from the right-hand-side nonterminals.

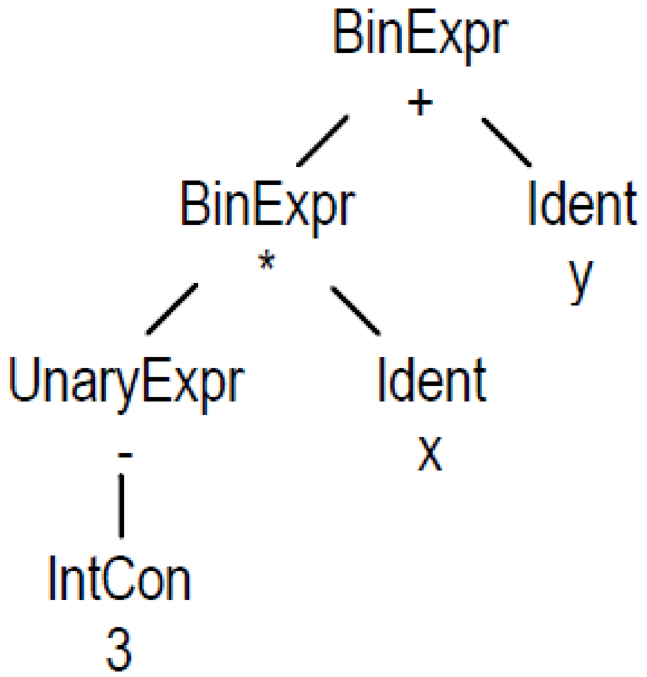
AST for Expressions

- When designing the AST for a language we first have to think about the kinds of nodes that we need and how to create and link those nodes.

- Each node type is implemented as a class that has fields for the children of this node as well as a constructor for creating the node and linking it with its children.

Example

- The expression -3 * x + y is translated to the following AST:



Classes:

BinExpr:- Expression

Ident:- Identifier

UnaryExpr:- Unary operator

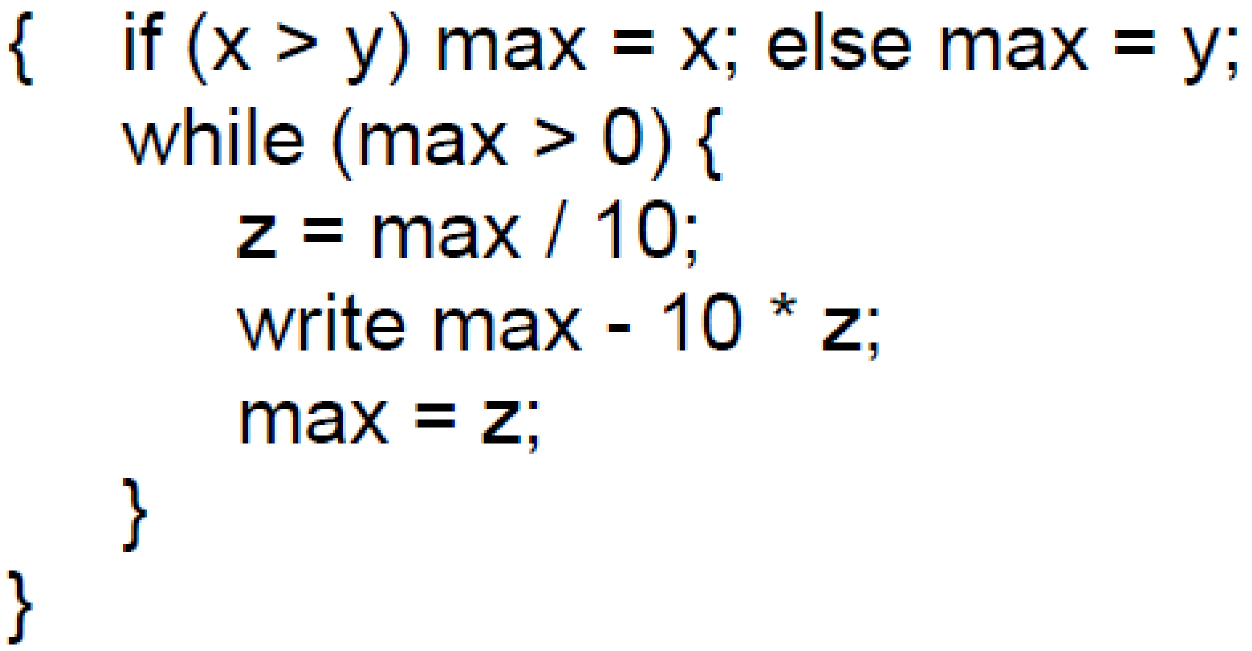
IntCon:- Integer Constant

AST for statements

- Kinds of statements we have in our language and how we want to represent them as abstract syntax trees.
- We may have assignments, procedure calls, if statements (with and without else part), while statements, read statements, write statements and blocks.

Example

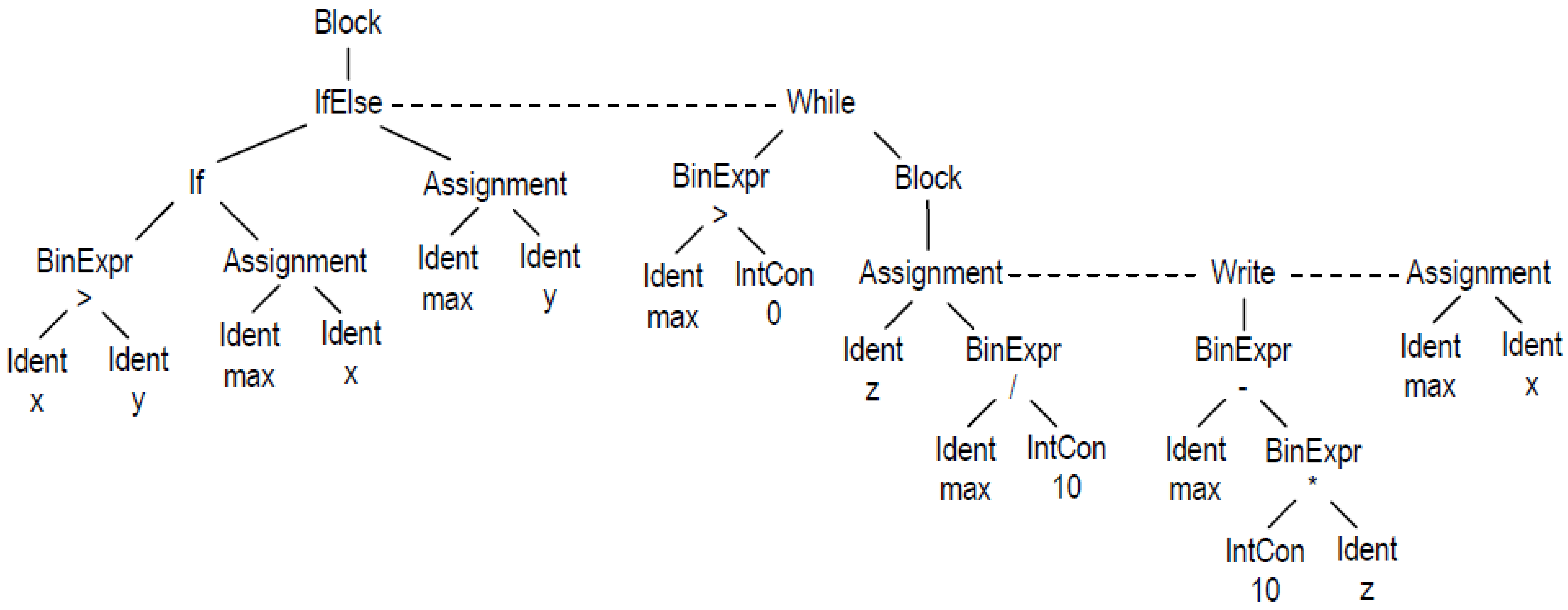
- The set of statements is given below:



```
{    if (x > y) max = x; else max = y;
     while (max > 0) {
          z = max / 10;
          write max - 10 * z;
          max = z;
     }
}
```

Example

- AST for the previous set of statements:

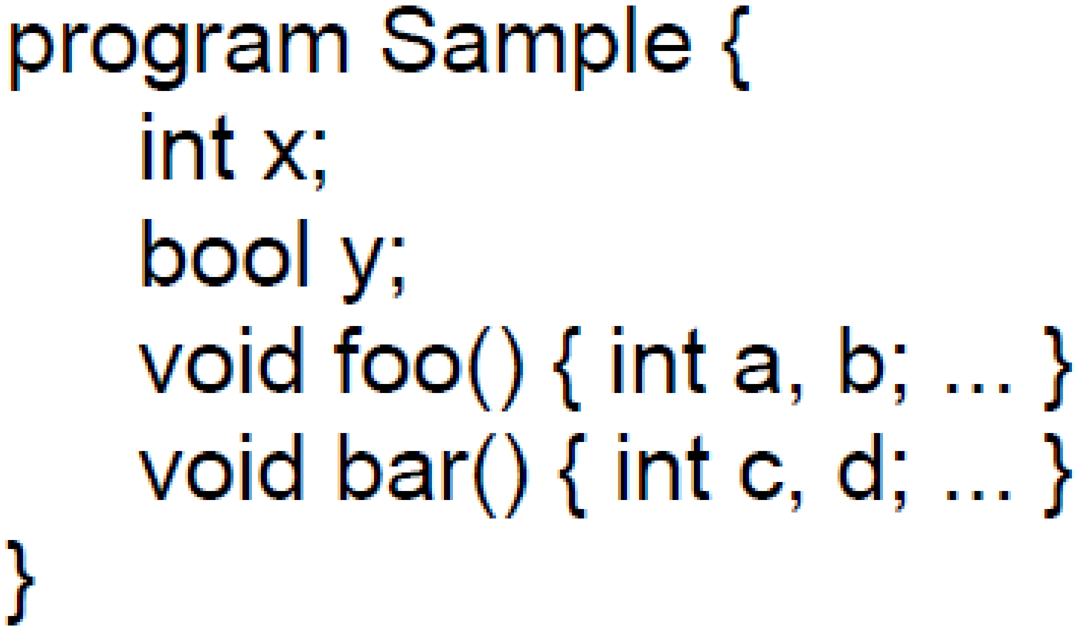


AST for Declarations and Procedures

- Declarations introduce names and associate them with properties such as a type or an address.
- Every declaration belongs to the program unit in which it appears, i.e., a procedure contains the declarations of its local variables and a program contains the declarations of the global variables and procedures.
- All declarations together form the *symbol table* of the compiled program.

Example

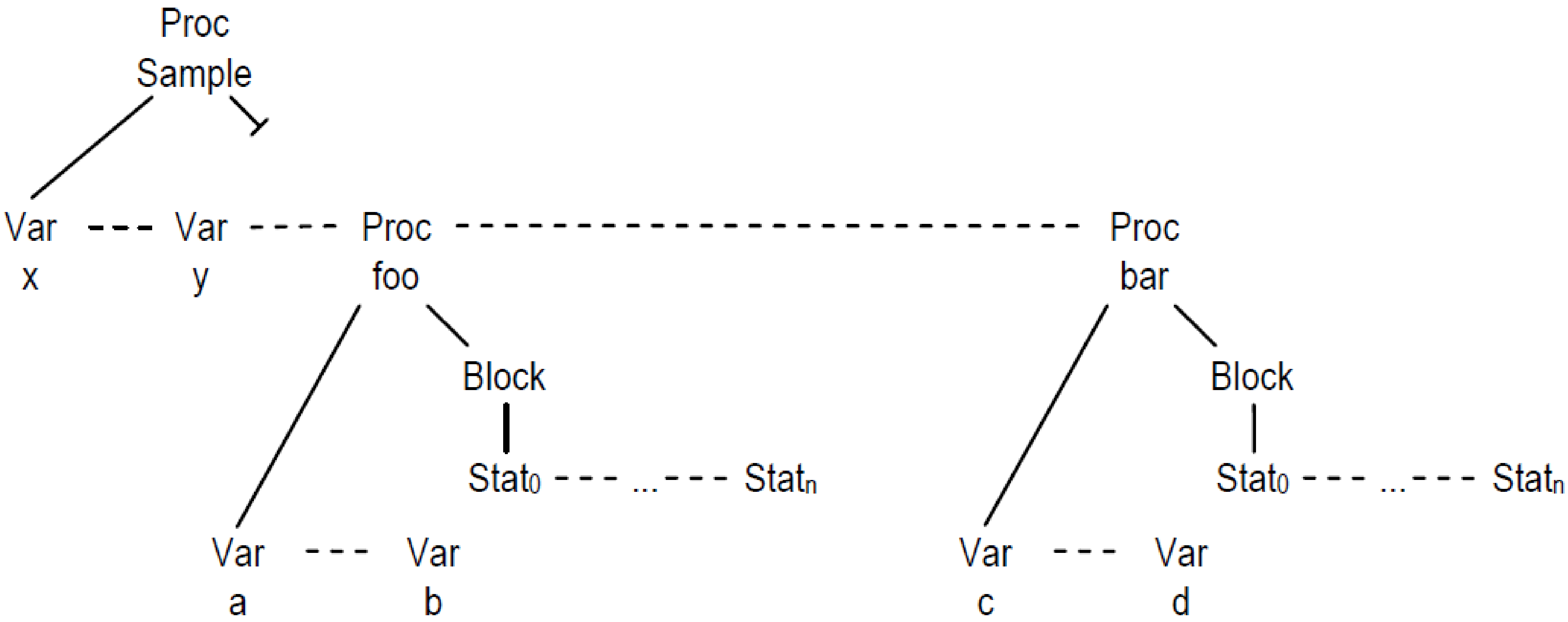
- The procedure is given below with its declarations.



```
program Sample {
    int x;
    bool y;
    void foo() { int a, b; ... }
    void bar() { int c, d; ... }
}
```

Example

- The AST for the previous procedure example



Exercise

Create AST for the following:

- 1 + 3 / x + 5 * (x + y) ^ 2
- d + (a – b * c)
- (((((1+2)*3)/(-a))-b)*((c+d)+(e*4)))

Exercise

-

```
x := a + b;
y := a * b;
while (y > a)
{
        a:= a + 1;
        x := a + b;
}
```