Compilers and Interpreters 601.428/628, Lecture 11, October 11, 2021

Compiler project!

The language:

- Pascal-like
- Similar to Spring 2018 offering (slight differences)

Remaining assignments:

1. Semantic analysis and type checking
2. Basic code generation
3. Better code generation
4. Procedures

Language features:

- Statically typed (each variable and expression has a static type, type errors are detected at compile time)
- Types: INTEGER, CHAR, arrays, records
- Variables/assignments
- Usual arithmetic (addition, subtraction, multiplication, division)
- READ and WRITE for I/O
- IF, IF/ELSE statements
- WHILE loops
- REPEAT/UNTIL loops
- Top-level PROGRAM has a block of statements (essentially a main function)
- Eventually: procedures

Specification of lexical and grammatical structure coming soon.

Example program:

```
PROGRAM add;
  VAR a, b, sum: INTEGER;

BEGIN
  a := 3;
  b := 4;
  sum := a + b;
  WRITE sum;
END.
```

Another example program:

```
PROGRAM SumToN;
  VAR i, sum, n: INTEGER;

BEGIN
  -- read integer from user
  READ n;
  -- compute sum from 1 to the integer
  i := 1;
  sum := 0;
  WHILE i <= n DO
    sum := sum + i;
    i := i + 1;
  END;
  -- output the sum
  WRITE sum;
END.
```

Program using an array type:

```
PROGRAM arr;
  VAR months: ARRAY 12 OF INTEGER;

BEGIN
  months[0] := 31;
  months[1] := 28;
  months[2] := 31;
  months[3] := 30;

  WRITE months[3];
END.
```

Program using a record type:

```
PROGRAM person;
  TYPE Person = RECORD
    name: ARRAY 10 OF CHAR;
    age: INTEGER;
  END;
  VAR p: Person;

BEGIN
  READ p.name;
  READ p.age;

  IF p.age >= 18 THEN WRITE 1;
  ELSE WRITE 0; END;
END.
```

Representing types:

- "Primitive" types (INTEGER, CHAR)
- Array types
- Record types

Design sketch:

Semantic analysis and type checking: assume that the compiler front-end builds an AST from the source program. We need to check that:

- Names used in the program refer to something that is defined
- Expressions and assignments obey the type rules

Approach: traverse the AST to build symbol tables. Link AST nodes to the symbol table entry which specifies what the node refers to or what type the node's value belongs to.

A symbol table is a collection of symbol table entries.

Each symbol table entry specifies (minimally):

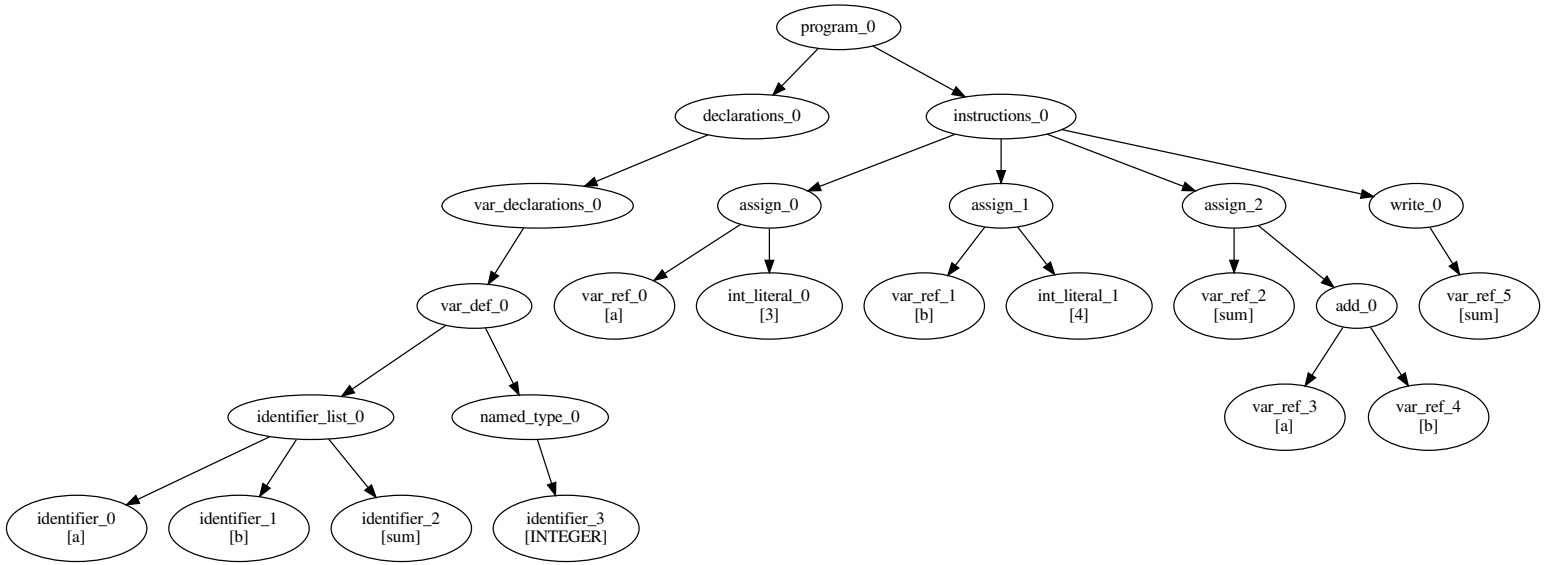
- A name
- A type
- What the entry refers to (variable, procedure, type)

A symbol table links to its "parent" symbol table, which represents the outer scope. The symbol table for the global (program-level) scope does not have a parent.

Design sketch:

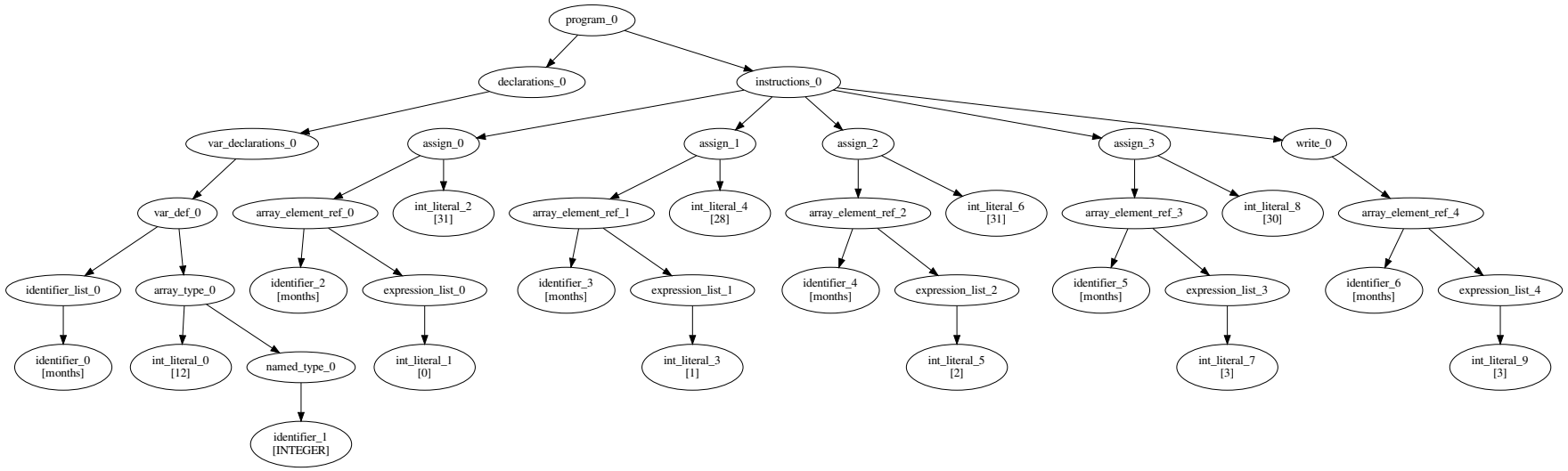
```
PROGRAM add;
  VAR a, b, sum: INTEGER;

BEGIN
  a := 3;
  b := 4;
  sum := a + b;
  WRITE sum;
END.
```



```
PROGRAM arr;
  VAR months: ARRAY 12 OF INTEGER;

BEGIN
  months[0] := 31;
  months[1] := 28;
  months[2] := 31;
  months[3] := 30;
  WRITE months[3];
END.
```



```
PROGRAM person;
  TYPE Person = RECORD
    name: ARRAY 10 OF CHAR;
    age: INTEGER;
  END;
  VAR p: Person;

BEGIN
  READ p.name;
  READ p.age;

  IF p.age >= 18 THEN WRITE 1;
  ELSE WRITE 0; END;
END.
```