HW4

50pts, no extra credit

This is a team assignment. Form teams and submit in Submitty!

Posted Friday, October 5, 2018
Due Tuesday, October 16, 2018

Problem 1 (10pts). What does the Pascal-like program print under static scoping? What does it print under dynamic scoping?

x : integer     -- global

procedure set_x(n : integer)
  x := n

procedure print_x
  write_integer(x)

procedure first
  set_x(1)
  print_x

procedure second
  x : integer
  set_x(2)
  print_x

set_x(0)
first()
print_x
second()
print_x
**Problem 2** (15pts). [From Scott] Consider the following pseudocode, assuming nested subroutines and static scoping:

```plaintext
procedure main
  g : integer

  procedure B(a : integer)
    x : integer

    procedure A(n : integer)
      g := n

    procedure R(m : integer)
      write_integer(x)
      x /:= 2 -- integer division
      if x > 1
        R(m + 1)
      else
        A(m)

    -- body of B
    x := a * a
    R(1)

  -- body of main
  B(3)
  write_integer(g)
```

a) (5pts) What does this program print?
b) (5pts) Show the frames on the stack when A has just been called. For each frame, show the static and dynamic links.
c) (5pts) Explain how A finds g.
**Problem 3** (10pts). [Modified from Aho, Sethi, Ullman] The following is the grammar for expressions formed by applying an arithmetic operator + to integer and real constants. When two integers are added, the resulting type is an integer, otherwise, it is real.

\[
E \rightarrow E + T \mid T \\
T \rightarrow \text{num} \cdot \text{num} \mid \text{num}
\]

Give an attribute grammar to determine the type of each subexpression.

**Problem 4** (15pts). [Modified from Aho, Sethi, Ullman] The grammar below generates Boolean expressions in prefix notation:

\[
B \rightarrow O B B \mid \text{not} B \mid \text{id} \\
O \rightarrow \text{and} \mid \text{or}
\]

Give an attribute grammar over the above CFG to translate Boolean expressions into properly parenthesized Boolean expressions in infix notation *without redundant parenthesis*. For example, since operators *and* and *or* associate to the left, and *and* has higher precedence than *or*, expression *and* and *a or b c d* can be rewritten in infix notation as *a and (b or c) and d*.

Note: You may recall Principles of Software HW8 where you wrote a visitor over a composite structure representing a Boolean expression. Recall that one of the visitors printed properly parenthesized expressions in infix notation. Parse trees and ASTs are important “clients” of the Composite and Visitor patterns, where visitors are essentially S-attributed grammars over an underlying grammar that constructs the composite structure.

This problem asks you to write down the attribute grammar you implemented back then. You may find omissions in your implementation even though you passed the HW8 tests.