Programming Languages
CSCI-GA.2110.001 Spring 2017

Homework 1
Answers

You should write the answers using Word, latex, etc., and upload them as a PDF document. No implementation is required. Since there are drawings, if you prefer, you can hand write the answers and scan them to a PDF.

1. Provide regular expressions for defining the syntax of the following. You can only use concatenation, |, $*$, parentheses, ϵ (the empty string), and expressions of the form [A-Z], [a-z0-9], etc., to create regular expressions. For example, you cannot use + or any kind of count variable.

   (a) Passwords consisting of letters and digits that contain at least one upper case letter and one digit. They can be of any length (obviously at least two characters).

   Answer:
   ([A-Za-z0-9]*[A-Z][A-Za-z0-9]*[0-9][A-Za-z0-9]*) |
   ([A-Za-z0-9]*[0-9][A-Za-z0-9]*[A-Z][A-Za-z0-9]*)

   (b) Floating point literals that specify an exponent, such as the following: 243.876E11 (representing 243.867×10^{11}).

   Answer:
   [0-9]*[0-9].[0-9][0-9]*E[0-9][0-9]*

   (c) Procedure names that: must start with a letter; may contain letters, digits, and _ (underscore); and must be no more than 7 characters.

   Answer:
   [a-zA-z]([a-zA-Z0-9_] | ϵ)([a-zA-Z0-9_] | ϵ)([a-zA-Z0-9_] | ϵ)([a-zA-Z0-9_] | ϵ)
   ([a-zA-Z0-9_] | ϵ)([a-zA-Z0-9_] | ϵ)

   Note: ϵ denotes the empty string.

2. (a) Provide a simple context-free grammar for the language in which the following program is written. You can assume that the syntax of names and numbers are already defined using regular expressions (i.e. you don't have to define the syntax for names and numbers).

```
program one;

  var x;

  function f(var x, var y)
    var z;
  begin
    z := x+y-1;
    z := z * 2.0;
    return z;
  end f;

  procedure g()
```

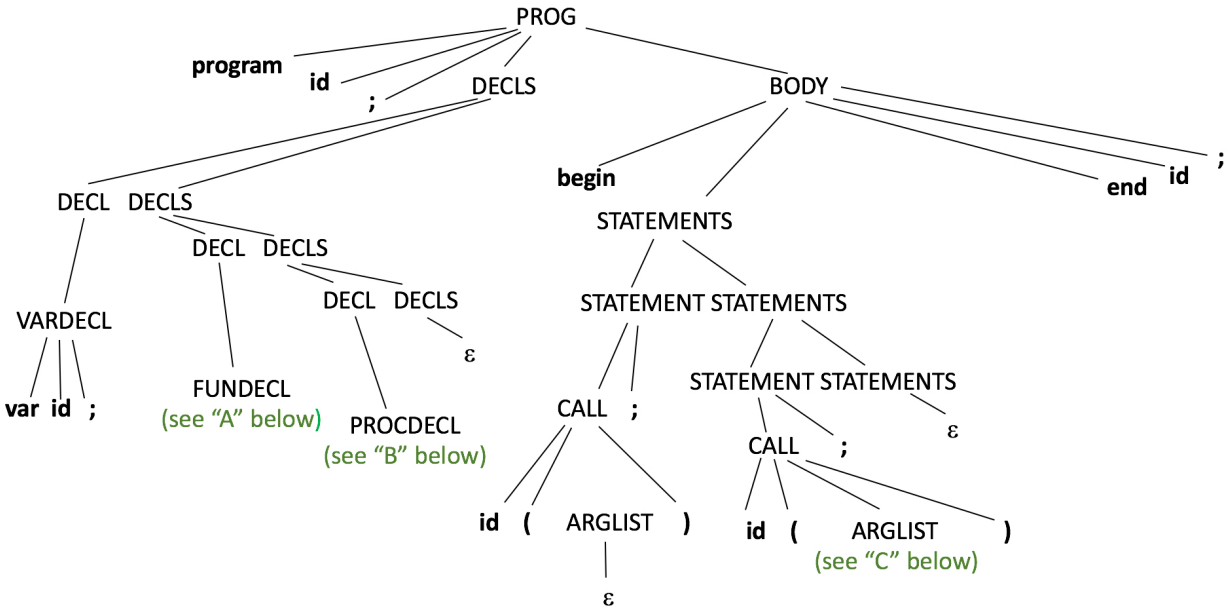
```
    var a;
  begin
    a := 3;
    x := a;
  end g;

begin
 g();
 print(f(x));
end one;
```

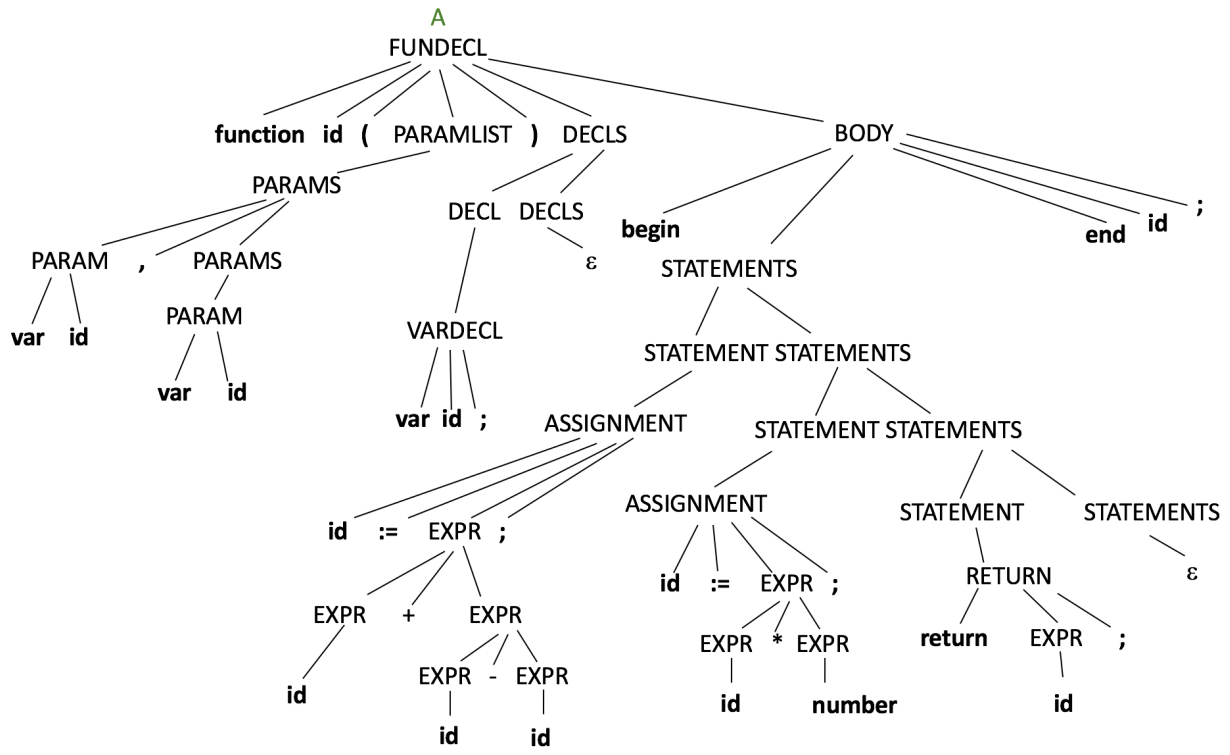
You only have to create grammar rules that are sufficient to parse the above program. **Here is one such grammar - though other ones are certainly possible.**

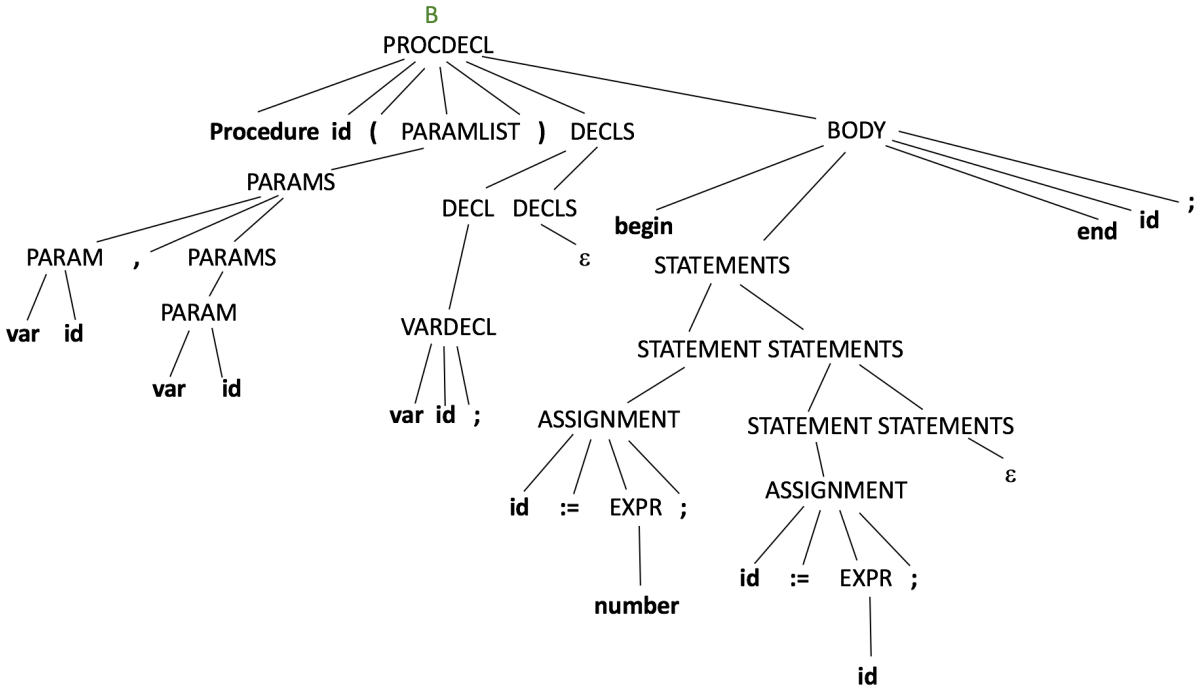
PROG → program id ; DECLS BODY
DECLS → DECL DECLS | ϵ
DECL → VARDECL | FUNDECL | PROCDECL
VARDECL → var id ;
FUNDECL → function id (PARAMLIST) DECLS BODY
PROCDECL → procedure id (PARAMLIST) DECLS BODY
PARAMLIST → PARAMS | ϵ
PARAMS → PARAM | PARAM , PARAMS
PARAM → var id
BODY → begin STATEMENTS end id ;
STATEMENTS → STATEMENT STATEMENTS | ϵ
STATEMENT → ASSIGNMENT | CALL ; | RETURN
ASSIGNMENT → id := EXPR ;
RETURN → return EXPR ;
EXPR → id | number | EXPR + EXPR | EXPR * EXPR | (EXPR) | CALL
CALL → id (ARGLIST)
ARGLIST → ARGS | ϵ
ARGS → EXPR | EXPR , ARGS

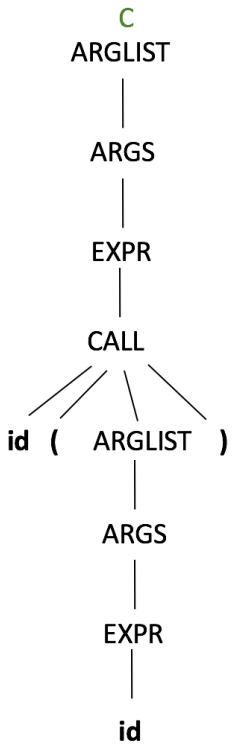
Note that the above grammar is ambiguous, which can be fixed by modifying the production for EXPR, as discussed in class (but not required for this assignment).

(b) Draw the parse tree for the above program.


PROG
program
id
;
DECLS
BODY
begin
end
id
;
DECL DECLS
STATEMENTS
VARDECL
DECL DECLS
DECL DECLS
STATEMENT STATEMENTS
var id ;
FUNDECL
(see "A" below)
DECL DECLS
ε
CALL ;
STATEMENT STATEMENTS
PROCDECL
(see "B" below)
ε
id (ARGLIST)
CALL ;
ε
id (ARGLIST)
(see "C" below)
ε


A
FUNDECL
function id (PARAMLIST) DECLS
BODY
PARAMS
DECL DECLS
begin
end
id
;
PARAM , PARAMS
ε
STATEMENTS
var id
PARAM
VARDECL
STATEMENT STATEMENTS
var id
var id ;
ASSIGNMENT
STATEMENT STATEMENTS
id := EXPR ;
ASSIGNMENT
STATEMENT
STATEMENTS
EXPR + EXPR
id := EXPR ;
RETURN
ε
id
EXPR - EXPR
EXPR * EXPR
return EXPR ;
id
id
id
number
id





3. (a) Define the terms *static scoping* and *dynamic scoping*.

In static scoping, the body of a function is evaluated in the environment of the function definition. In dynamic scoping, the body of a function is evaluated in the environment of the function call.

(b) Give a simple example, in any language you like (actual or imaginary), that would illustrate the difference between static and dynamic scoping. That is, write a short piece of code whose result would be different depending on whether static or dynamic scoping was used.

```
procedure A()
  x: integer = 5;

 procedure B()
 begin
  print(x); (* static scoping prints 5, dynamic prints 10 *)
 end;

 procedure C()
   x: integer = 10;
 begin
   B();
 end;

begin (*A*)
  C();
end;
```

(c) In a block structured, statically scoped language, what is the rule for resolving variable references (i.e. given the use of a variable, how does one find the declaration of that variable)?

Given the use of (reference to) a non-local variable in a function, the corresponding variable declaration is found (at compile time) by looking at the scope surrounding the definition of the function, then in the next outer scope, and so on. At run time, the variable is found by traversing the static chain for the predetermined number of hops, corresponding to the difference between the nesting levels of the variable's use and definition.

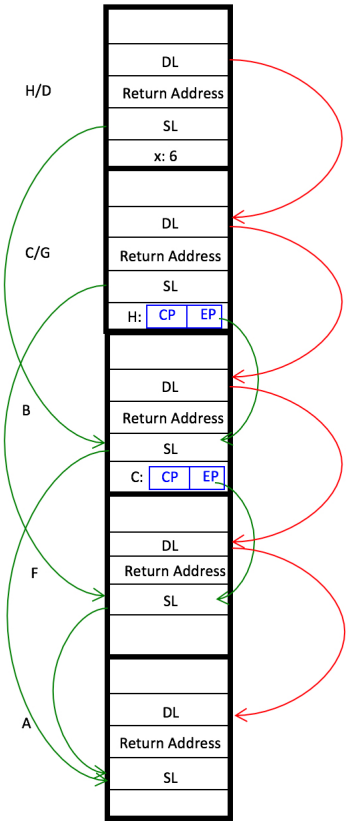
(d) In a block structured but dynamically scoped language, what would the rule for resolving variable references be?

Given the use of a non-local variable in a function, the corresponding variable declaration is found by looking at calling function to see if the variable is defined there. If not, the calling function of the calling function is examined to determine if the variable is defined there, and so on. That is, the the dynamic chain is traversed, and each encountered stack frame is checked, until the variable is found.

4. (a) Draw the state of the stack, including all relevant values (e.g. variables, return address, dynamic link, static link, closures), during the execution of procedure D.

```
procedure A;
  procedure B(procedure C)
    procedure  D(x:integer);
    begin
      writeln(x);
```

```
    end;
 begin
   C(D);
 end;
 procedure F;
   procedure G(procedure H);
    begin
       H(6);
    end;
  begin
      B(G);
  end;
begin
 F;
end;
```



(b) Explain what the two parts of a closure (as shown in your diagram) are for.

Within a closure, the code pointer (CP) points to the code for the function represented by the closure. The environment pointer (EP) contains the static link to be used when that function is called.

(c) If both the dynamic link and the return address in a stack frame point back to the calling procedure, what is the difference between them?

The dynamic link points to the stack from for the calling procedure, i.e. it

points to data stored in the stack section of the memory for the program. The return address points to the next instruction to execute in the calling procedure, i.e. it points to code in the code section of the memory for the program.

5. For each of these parameter passing mechanisms,

   (a) pass by value

   (b) pass by reference

   (c) pass by value-result

   (d) pass by name

   state what the following program (in some Pascal-like language) would print if that parameter passing mechanism was used:

```
program foo;
  var i,j: integer;
      a: array[1..5] of integer;

  procedure f(x,y:integer)
  begin
    x := x * 2;
    i := i + 1;
    y := a[i] + 1;
  end

begin
  for j := 1 to 5 do a[j] = 10*j;
  i := 1;
  f(i,a[i]);
  for j := 1 to 5 do print(a[j]);
end.
```

   Pass by value: 10 20 30 40 50
   Pass by reference: 31 20 30 40 50
   Pass by value-result: 21 20 30 40 50
   Pass by name: 10 20 31 40 50

6. (a) In Ada, write a procedure that declares two tasks, task one and task two. When the procedure is called, task one and task two should print "one" and "two", respectively, over and over such that the printing of "one" and "two" is perfectly interleaved. Thus, the output should look like:

```
one
two
one
two
...
```

   There are many possible programs that satisfy this question. Here is a simple one.

```
procedure Hw1 is
   task One;
   task Two is
      entry Go;
   end Two;

   task body One is
   begin
      loop
         Put("One"); New_line;
         Two.Go;
      end loop;
   end One;

   task body Two is
   begin
      loop
         accept Go do
            Put("Two"); New_Line;
         end Go;
      end loop;
   end Two;
begin
   null;
end Hw1;
```

(b) Looking at the code you wrote for part (a), are the printing of "one" and the printing of "two" occurring concurrently? Justify your answer by describing what concurrency is and why these two events do or do not occur concurrently.

The printing of "one" and "two" is not concurrent. Two events, such as these print operations, are concurrent if no assumption can be made about the relative order in which they occur. In this case, we know that the print operations will always be interleaved, so we know exactly the order in which they occur. Note that this question did not ask if the tasks were concurrent.