Programming Languages
CSCI-GA.2110.001 Fall 2023

Homework 1

Due Monday, October 30 at 11:55pm

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) Strings consisting of any number of lower case letters, exactly two upper case letters, and exactly one digit, such that the digit is somewhere between the two upper case letters. They can be of any length (obviously at least three characters). For example, "doxFexo7wvsQdle" would be a valid string.

   (b) Floating point literals that specify an exponent, such as the following: 243.876E11 (representing 243.867×10^{11}). There must be at least one digit before the decimal point and one digit after the decimal point (before the "E").

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

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).

```
x: int;

fun f(x: int, y: int)
  z:int;
{
    z = x+y-1;
    z = z + 1;
    return z;
}

proc g()
  a:int;
{
  a = 3;
  x = f(a, 2);
}
```

   You only have to create grammar rules that are sufficient to parse the above program. Your starting non-terminal should be called PROG (for "program") and the above program should be able to be derived from PROG. To get you started, here are some productions you might use.

```
PROG → DECLS
DECLS → DECL DECLS | DECL
DECL → VARDECL ; | FUNDECL | PROCDECL
```

(b) Draw the parse tree for the above program, based on a derivation using your grammar. Note: It will be quite large.

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

   (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.

   (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)?

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

4. Draw the state of the stack, including all relevant values (e.g. variables, return addresses, dynamic links, static links, closures), at the time that the `writeln(y)` is executed.

```
procedure A;

    procedure B(procedure C)
       procedure  D(procedure I);
         x: integer := 6;
       begin (* D *)
         I(x);
       end;
    begin (* B *)
      C(D);
    end;

    procedure F(procedure H)
      procedure G(y: integer)
      begin (* G *)
         writeln(y); (* draw state of stack when this is executed *))
      end;
    begin (* F *)
         H(G);
    end;

begin (* A *)
  B(F);
end;
```

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] = j*2;
  i := 2;
  f(i,a[i]);
  for j := 1 to 5 do print(a[j]);
end.
```

6. (a) In Ada, define a procedure containing two tasks, each of which contains a single loop. The loop in the first task prints the numbers from 1 to 100, the loop in the second task prints the numbers from 201 to 300. The execution of the procedure should cause the tasks to alternate printing 10 numbers at a time, so that the user would be guaranteed to see:
   `1 2...10 201 202...210 11 12...20 211 212...220 21...`
   Be sure there is only one loop in each task.

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

7. As you know, the sequence of Fibonacci numbers can be defined as follows:
   fib(0) = 0, fib(1) = 1, fib(j) = fib(j-1) + fib(j-2).

   (a) Write in Scheme the function `(fib n)` which returns the n^{th} Fibonacci number (i.e. fib(n) in the above definition). This Scheme code should reflect the above definition of Fibonacci numbers.

   (b) If you wrote the Scheme code to directly reflect the definition of the Fibonacci sequence (which you were supposed to), the complexity of the program would be exponential (i.e. $O(2^n)$), due to two recursive calls in the body of the function. For this part, write a Scheme function, `(lfib n)`, that computes the n^{th} Fibonacci number in linear time (with only a single recursive call). Do not define any function outside of `lfib`, but you can use `letrec` within `lfib`.