Programming Languages
CSCI-GA.2110.001 Fall 2016

Midterm Exam

ANSWERS

Please write the answers to questions 1 and 5 on this sheet and all other answers in the blue book. Keep your answers brief!

1. True/False. Please Circle the correct answer.

   (a) T A compiler translates a program in one language to a program in another language.

   (b) T The term *static* refers to a property of a program that can be determined at compile time, rather than at run time.

   (c) F A Turing complete language is a language in which the most efficient way of implementing any computable function can be expressed.

   (d) F A regular expression can be used to describe any set that a CFG can describe.

   (e) T A CFG can be used to describe any set that a regular expression can describe.

   (f) T The semantics of a programming language gives meaning to syntactically-correct programs.

   (g) F Concurrency can only occur on a computer with multiple processors or cores.

   (h) F In functional programming languages, variables denote memory locations that can be overwritten.

   (i) T A closure is a data structure representing a function, so that, for example, functions can be passed as parameters.

   (j) F Recursive functions cannot be written in imperative languages like C or Ada, only in functional languages like Scheme.

2. (a) Write regular expressions that define the following sets. 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.

      i. The set of all strings representing integers and real numbers (e.g 27 and 34.68). There must be at least one digit before a decimal point and after a decimal point.
      `[0-9][0-9]* ((.[0-9][0-9]*) | ϵ )`

      ii. The set of all strings consisting of `a`'s and `b`'s whose length is an even number (i.e. consisting of an even number of characters).

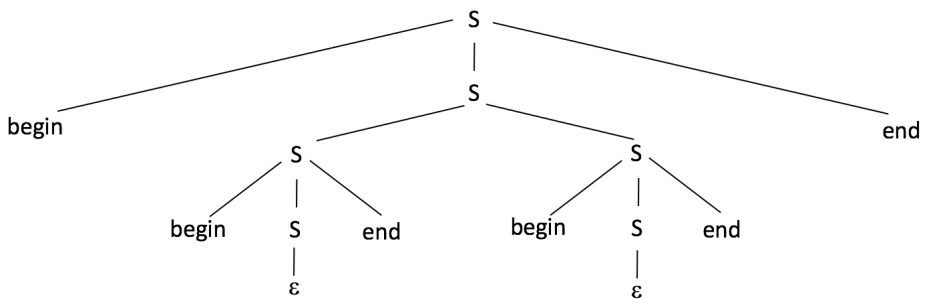
      ```
      (aa | ab | ba | bb)*
      ```

   (b) i. Write a CFG defining the set of all properly nested sequences of `begin` and `end` (e.g. `begin end`, `begin begin end end`, `begin end begin end`).
      `S -> begin S end | SS | ϵ`

      ii. Show the derivation of `begin begin end begin end end` starting from the start symbol of your CFG above.

      ```
      S => begin S end => begin SS end => begin begin S end S end =>
      begin begin end S end => begin begin end begin S end end =>
      begin begin end begin end end
      ```

iii. Draw the corresponding parse tree.



3. What does the following Ada code print? Be sure to indicate the cases where the printing order can't be determined.

```
procedure foo is
   task one;
   task two is entry go; end two;
   task body one is
   begin
      for i in 1..2 loop
         Put("One ");
         Two.Go;
         Put("Three ");
      end loop;
   end One;
   task body Two is
   begin
      for I in 1..2 loop
         Put("Two ");
         accept Go do
            Put("Four ");
         end Go;
         Put("Five ");
      end loop;
   end Two;
begin
   null;
end;
```

It prints the following, where the items with square brackets, "[...]" can appear in any order (i.e they are printed concurrently, so no assumption can be made about their order), and the items within curly brackets, "{ ... }" are printed sequentially.

```
{ [One Two] Four [{ Three One } { Five Two }] Four [ Three Five ] }
```

Stated another way:

- **One and Two are printed in any order,**

- **then Four**
- **then Three, One, Five, and Two in any order, as long as Three is printed before One and Five is printed before Two,**
- **then Four**
- **then Three and Five in any order.**

4. Consider the following program.

```
x: integer := 2;
y: array 1..3 of integer;
i: integer;

  procedure f(a: integer)
  begin
     x:= x + 1;
     print(a);
  end;

begin
  for i := 1 to 3 do y[i] := i*2;
  f(y[x]);
end
```

   (a) What does the program print if the language uses pass-by-reference parameter passing?
   4

   (b) What does the program print if the language uses pass-by-name parameter passing?
   6

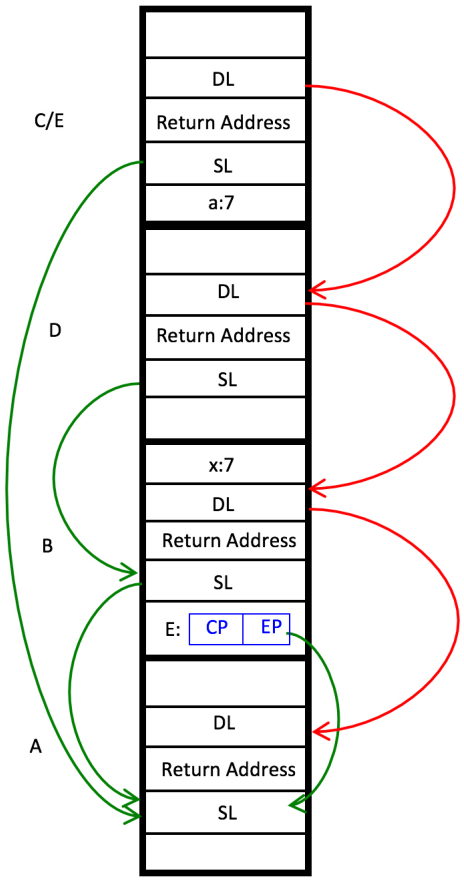
5. Given the following program written in a statically scoped language, draw in the static and dynamic links of the corresponding call stack shown on the right, at the point when the print statement is executed. Also draw in any closures required (including the EP pointer) and label the stack frames with the names of the corresponding functions. Assume `A()` is the main procedure of the program. Draw the labels, links, etc. in the stack shown below on this sheet.

```
procedure A()
    procedure B(procedure C)
       x: integer := 7;

       procedure D()
       begin
          C(x);
       end (* D *)

    begin (* B *)
       D();
    end; (* B *)

   procedure E(a: integer)
   begin
      print(a);  (* draw stack at this point *)
   end;
begin (* A *)
  B(E);
end (* A *)
```



6. (a) Write the Scheme function `(reduce f L)`, where `f` is a function and `L` is a list of the form $(x_1\ x_2\ \ldots\ x_{N-1}\ x_N)$, that computes `(f` x_1 `(f` x_2 `...` `(f` x_{N-1} x_N`)...))`. For example, `(reduce + '(1 2 3 4))` returns the value of `(+ 1 (+ 2 (+ 3 4)))` = 10. You can assume that `L` has at least one element (in which case the answer is just that element). `reduce` should be roughly three lines long. You don't need a comment describing your recursive thinking.

```
(define (reduce f L)
 (cond ((null? (cdr L)) (car L))
       (else (f (car L) (reduce f (cdr L))))))
```

(b) Write an Scheme expression that calls `reduce` to return the largest number in `L`, where `L` is a list of numbers. That is, your expression should look like:

```
(reduce ... L)
```

where you fill in the "...".

```
(reduce (lambda (x y) (if (> x y) x y)) L)
```

Since the `max` function is a built-in function in Scheme,

```
(reduce max L)
```

is acceptable, but I was hoping for the use of a lambda.