Programming Languages
CSCI-GA.2110.001 Spring 2017

Midterm Exam

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

1. True/False. Please Circle the correct answer on this sheet.

   (a) **T** A Turing complete language is a language that can be used to express any computable function.

   (b) **F** Static scoping means that the body of a function is evaluated in the environment of the function call.

   (c) **T** In pass by value-result, the formal parameter denotes a separate memory location from the corresponding actual parameter.

   (d) **T** The static semantics of a language defines the use of types within the language.

   (e) **F** The dynamic semantics of a language are the rules governing the organization of symbols in the language.

   (f) **F** The return address in a stack frame points to the code for the function represented by that stack frame.

   (g) **F** Each statement in the the body of an Ada task is executed concurrently with the other statements in the same task.

   (h) **T** In functional programming, variables denote values rather than memory locations.

   (i) **F** A high-level language is a language whose computational model reflects the underlying machine instructions.

   (j) **F** A procedure is an example of a module (package).

2. (a) Write a regular expression for the set of all names consisting of letters and digits such that there is no more than one upper-case letter and no more than one digit. 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.

   ([a-z]*([A-Z] | ϵ)[a-z]*([0-9] | ϵ)[a-z]*) | ([a-z]*([0-9] | ϵ)[a-z]*([A-Z] | ϵ)[a-z]*)

   (b) Describe precisely in words or mathematical notation the set defined by the following regular expression.

   [1-9][0-9]*(0 | 5)

   The set of strings representing all integers greater than 9 that are divisible by 5 (with no leading zeros).

   (c) Suppose you are given a context free grammar that already defines the non-terminals DECL, corresponding to a single declaration, and STATEMENT, corresponding to a single statement. Define the CFG grammar rule(s) for a block (such as that found in C, C++, or Java) where a block consists of an open bracket, {, followed by a zero or more declarations, followed by zero or more statements or nested blocks, followed by a close bracket, }. For example, a valid block would be:

```
{ int x, y;
  x = 3;
  y = 2 * x;
  { int z;
```

```
    z = x + y;
  }
  x = z;
}
```

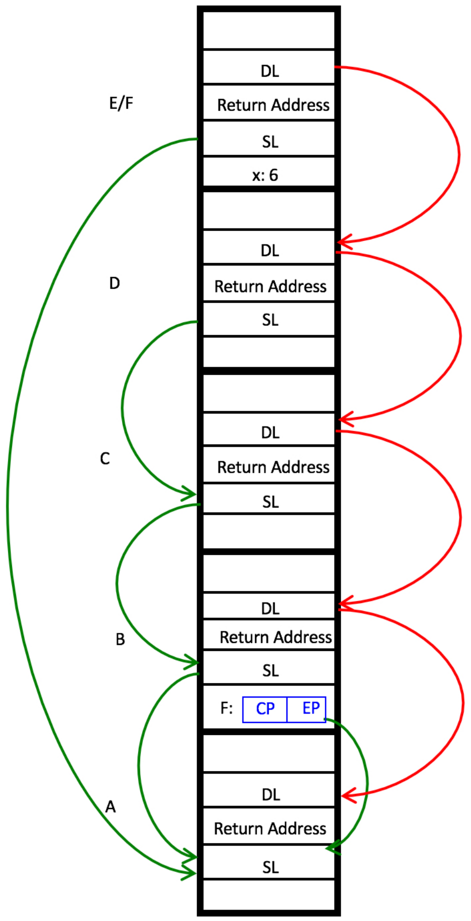
Remember, you do not need to define the syntax of a declaration or statement (just use DECL and STATEMENT in your grammar rules). Note: The grammar rules for the block should be roughly 3 lines.

BLOCK $\rightarrow$ **{ DECLS STBLS }**
DECLS $\rightarrow$ **DECL DECLS** $\mid \epsilon$
STBLS $\rightarrow$ **STATEMENT STBLS** $\mid$ **BLOCK STBLS** $\mid \epsilon$

(d) Based on your grammer rules, above, write the derivation for the sample block above, starting with your non-terminal representing a block. However, you can stop when the resulting string consists only of terminals and the DECL and STATEMENT non-terminals. You do not need to draw the parse tree.

BLOCK $\Rightarrow$ **{ DECLS STBLS }** $\Rightarrow$ **{ DECL DECLS STBLS }**
$\Rightarrow$ **{ DECL STBLS }** $\Rightarrow$ **{ DECL STATEMENT STBLS }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT STBLS }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT BLOCK STBLS }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT BLOCK STATEMENT STBLS }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT BLOCK STATEMENT }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT { DECLS STBLS} STATEMENT }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT { DECL DECLS STBLS} STATEMENT }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT { DECL STBLS} STATEMENT }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT { DECL STATEMENT STBLS} STATEMENT }**
$\Rightarrow$ **{ DECL STATEMENT STATEMENT { DECL STATEMENT } STATEMENT }**

3. Write a program (in any syntax you like) that corresponds to the call stack below. Indicate the point in the program at which the call stack would look like this. Keep the program as simple as possible.



Answer:

```
procedure A()

   procedure E(x:integer)
   begin
     (* This is where the call stack looks like the illustration *)
   end (*E*)

   procedure B(procedure F)

      procedure C()

         procedure D()
         begin
           F(6);
         end (* D *)
     begin (* C *)
       D();
     end (* C *)

   begin (* B *)
     C();
   end  (* B *)

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

4. (a) In Scheme, write the usual `map` function, such that `(map f L)` applies the function `f` to every element of the list `L`, returning a list of the results. You do not have to write down your recursive thinking. Your code should be roughly 3 lines. **Answer:**

```
(define (map f L)
  (cond ((null? L) '())
        (else (cons (f (car L)) (map f (cdr L))))))
```

   (b) Write a Scheme function `(compose-all L x)`, where L is a list of functions (f_1 f_2 ... f_{n-1} f_n), that returns the value of f_1(f_2 ... (f_{n-1}(f_n(x))) ...). For example,

```
  (compose-all (list car cdr car cdr) '((1 2) (3 4) (5 6)))
```

   would return the result of `(car (cdr (car (cdr '((1 2) (3 4) (5 6))))))`, which is 4. You do not have to write down your recursive thinking. Your code should be roughly 3 lines.
   Answer:

```
(define (compose-all L x)
  (cond ((null? L) x)
        (else ((car L) (compose-all (cdr L) x)))))
```

5. (a) In the following program, which pairs of "Put" statements execute concurrently with each other? For example, write "A/B" if `Put("A")` and `Put("B")` execute concurrently. There will be several such pairs.

```
procedure Main is
   task One is
      entry Go;
   end One;

   task body One is
   begin
         Put("A");
         accept Go do
            Put("B");
         end go;
         put("C");
         Put("D");
   end One;
begin -- main
      Put("E");
```

```
        One.Go;
        Put("F");
        Put("G");
    end Main;
```

Answer: A/E, C/F, C/G, D/F, D/G

(b) Write a very simple Ada package that declares three procedures, F, G, and H, but only G is visible outside of the package. Your procedures don't have to actually do anything.

Answer:

```
package P is
   procedure G;
end P;

package body P is
   procedure F is
   begin
      null;
   end F;
   procedure G is
   begin
      null;
   end G;
   procedure H is
   begin
      null;
   end H;
end P;
```

6. Given the following program,

```
program P;

   x: integer;
   y: integer;

   procedure f(a:integer, b:integer)
   begin
   x := x + a;
   y := y + b;
   end;

begin (* P *)
  x := 3;
  y := 5;
  f(x, x+y);
  print(x);
  print(y);
end;
```

What does the program print if pass-by-name parameter passing is used? Explain your answer.

The program prints 6 16. The semantics of pass-by-name says that the body of the above program is equivalent to the following code resulting from textual substitution of the call to `f` with the body of `f`, the substition of `a` with `x`, and the substitution of `b` with `x+y`.

```
x := 3;
y := 5;
x := x + x;   (* x gets assigned 6 *)
y := y + (x+y); (* y gets assigned 16 *)
print(x);
print(y);
```