Programming Languages
CSCI-GA.2110.001 Fall 2014

Midterm Exam
ANSWERS

Please write all answers in the blue book. Keep your answers brief!

1. (a) Can the following set be specified using a regular expression?
   - The set of all strings containing only the letters `a`, `b`, and `c` such that the `a`s come before the `b`s, and the `b`s come before the `c`s.

   If so, provide the regular expression. If not, explain why not.

   Yes, the regular expression is: `a*b*c*`.

   (b) Can the following set be specified using a context free grammar (CFG)?
   - The set of all strings, containing only the letters `a`, `b`, and `c`, that are palindromes (are identical when read forward and backward, e.g. `abcba`).

   If so, provide the CFG. If not, explain why not.

   Yes, the CFG is:

$$S \rightarrow \texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \texttt{a}S\texttt{a} \mid \texttt{b}S\texttt{b} \mid \texttt{c}S\texttt{c} \mid \epsilon$$

2. (a) For each of pass-by-value, pass-by-reference, pass-by-value-result and pass-by-name, what would the following program print if that parameter passing mechanism was used?

```
program one;
  a: array[1..5] of integer;
  i: integer;

  procedure f(x: integer; y: integer)
  begin
    y := y + 1;
    x := x * 2;
    a[i] := a[i] + 15;
  end

begin (* main program *)
  for j := 1 to 5 loop
     a[j] := j;
  end loop;
  i := 1;
  f(a[i], i);
  for j := 1 to 5 loop
     print(a[j]);
  end loop;
end (* main program *)
```

Pass by value: 16 2 3 4 5
Pass by reference: 2 17 3 4 5
Pass by value-result: 2 2 3 4 5
Pass by name: 1 19 3 4 5

(b) What do I mean when I say "Java uses pass-by-value with pointer semantics for objects"?

It means that Java uses pass-by-value, but the value of an object is its address.

3. (a) What does concurrency mean in the context of programming languages? Be sure your answer covers the situation where a program is executed on a single processor.

Two portions of a program (e.g. threads, tasks, etc.) are concurrent if no assumption can be made about the relative order of their execution with respect to each other.

(b) Write a simple Ada program that exhibits concurrency, so that the user might see different output in different runs of the program.

Almost any answer involving a task (and some output) will do. For example,

```
with text_io;
use text_io;

procedure Test is

   task One;

   task body One is
   begin
      Put("In One"); New_Line;
   end One;

begin -- Test
   Put("In Test"); New_Line;
end Test;
```

4. (a) Write a `map-list` function in Scheme, where `map-list` takes a list of functions, `funs`, and a list of values, `L`, and returns a list of the results of applying each function in `funs` to the corresponding value in `L`. For example,

```
> (map-list (list (lambda (x) (+ x 1)) (lambda (x) (* x 3))) '(2 5))
(3 15)
```

Answer:

```
(define (map-list funs L)
 (cond ((null? funs) '())
       (else (cons ((car funs) (car L))
                   (map-list (cdr funs) (cdr L))))
       ))
```

(b) Write a function `ho-map-list` (where "ho" stands for "higher-order") that takes a list of functions, `funs`, and returns a function that takes a list of values, `L`, and applies each function in `funs` to the corresponding element of `L`. For example,

```
> (define f (ho-map-list (list (lambda (x) (+ x 1)) (lambda (x) (* x 3)))))
> (f '(2 5))
(3 15)
```

In writing `ho-map-list`, you cannot call any external user-defined function (such as `map-list` above). That is, `ho-map-list` has to be self-contained.

Answer:

```
(define (ho-map-list funs)
  (lambda (values)
     (letrec ((m-l (lambda (fs L)
                     (cond ((null? fs) '())
                           (else (cons ((car fs) (car L))
                                       (m-l (cdr fs) (cdr L))))
                           ))))
       (m-l funs values))))
```

5. (a) Assuming the following program is written in a statically scoped language, what would it print?

```
program two;
  x: integer;
  y: integer;

  procedure A()
  begin
    print(x,y);
  end;

  procedure B(procedure C())
    x:integer;
  begin
    x := 7;
    C();
  end;

begin (* main program *)
  x := 1;
  y := 2;
  B(A);
end;
```

It would print `1 2`

(b) What would the above program print if it were written in a dynamically scoped language?
It would print `7 2`**.**

6. Note: The λ-calculus will not be on our midterm (but will be on the final exam).

(a) What does "normal form" mean in the context of the λ-calculus?
A term is in normal form if it contains no reducible expressions (redexes)

(b) Give an example of a term that is in normal form.
$\lambda x.x$ **(one example of an infinite number of terms in normal form)**

(c) Give an example of a term that has no normal form and justify your answer.
The term $(\lambda x.(x\ x))(\lambda x.(x\ x))$ **has no normal form. This is so because applying** β**-reduction to that term results in the same term. That is,**

$$(\lambda x.(x\ x))(\lambda x.(x\ x)) \Rightarrow (\lambda x.(x\ x))(\lambda x.(x\ x)) \Rightarrow ...$$

(d) Give an example, if possible, of a term that would be reduced to normal form under normal order evaluation but would not be reduced to normal form under applicative order evaluation. If an example is not possible, state why not, and if it is possible, explain it.

The term

$$(\lambda y.3)((\lambda x.(x\ x))(\lambda x.(x\ x)))$$

would reduce to normal form under normal order evaluation (reducing the outermost redex first) but not under applicative order evaluation (reducing the innermost redex first). Specifically, the reduction under normal order evaluation would be

$$(\lambda y.3)((\lambda x.(x\ x))(\lambda x.(x\ x))) \Rightarrow 3$$

The reduction sequence under applicative order evaluation would be

$$(\lambda y.3)((\lambda x.(x\ x))(\lambda x.(x\ x))) \Rightarrow (\lambda y.3)((\lambda x.(x\ x))(\lambda x.(x\ x))) \Rightarrow ...$$

since trying to reduce the innermost redex, $((\lambda x.(x\ x))(\lambda x.(x\ x)))$, first would never lead to a normal form (as discussed above).

(e) Give an example, if possible, of a term that would be reduced to normal form under applicative order evaluation but would not be reduced to normal form under normal order evaluation. If an example is not possible, state why not, and if it is possible, explain it.

No, that is not possible. Church-Rosser Theorem II says that if any order of evaluation results in a normal form, then normal order evaluation will also result in a normal form.