Spring, 1999 Name: _____

My Section Letter: _____ My Section Day and Time : _____

Com S 342 — Principles of Programming Languages

Test on *EOPL* Sections 5.6–5.7 and 6.1–6.3

This test has 6 questions and pages numbered 1 through 8.

Special Instructions for this Test

Diagrams should be drawn as we discussed in class. If your diagrams do not follow the conventions used in class, or if they are sloppy or hard to read, you will lose points.

Reminders

For this test, you can use one (1) page (8.5 by 11 inches, one (1) side, no less than 9pt font) of notes. Handwriting is okay. No photo-reduction is permitted. Don't use anything with printing on the other side, please. These notes are to be handed in at the end of the test. Have your name in the top right corner. Use of other notes or failure to follow these instructions will be considered cheating.

During the test, if you need more space for an answer, use the back of a page. Note when you do that on the front.

This test is timed. We will not grade your test if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

For programs, indentation is important to us for "clarity" points; if your code is sloppy or hard to read, you will lose points. Correct syntax also matters. Check your code over for syntax errors. You will lose points if your code has syntax errors.

1. (10 points) Consider the following expression in the defined language.

```
letrecproc
  helper(ls, acc) = if null(ls) then acc
                    else helper(cdr(ls), addToEnd(add1(car(ls)), acc));
  add1all(ls) = helper(ls, emptylist)
in add1all(mylist)
```

   Write, in the defined language's concrete syntax, an equivalent desugared form of the above expression, which does not use `letrecproc` or `letrec`. (You may use `let` in your answer.)

2. (5 points) In the Unix shell, the environment maps names to strings. Commands written in the Unix shell can associate strings with names in the environment. When a command P calls another command C, the environment that is active at the time of the call is used as the starting environment for C, but when C returns, P's environment is unaffected by whatever changes C may have made.

   What is the scope rule used in the Unix shell?

3. (15 points) This is a question about dynamic scoping. Consider the following expression in the defined langauge (using call-by-value and the indirect array model).

```
let x = 3;
    y = 5
in let p = proc(i, k)
         begin
             x := x + 1;
             %% draw the picture at this point of the execution
             list(x, y, i, k)
         end;
       x = 7
   in let x = 9
      in let ls = p(let y = 11 in y, x)
         in cons(x, cons(y, ls))
```

Using dynamic scoping, (a) draw a picture of the run-time stack when execution reaches the point indicated (with the stack growing down the page), and (b) give the result (if any) of the above expression. If the expression has no result, or encounters an error, write that and briefly explain what the problem is.

4. (10 points) This is a question about dynamic assignment. Using call-by-value and the indirect array model, what is the value of the following expression?

```
let size = 9;
    color = 440000;
    weight = 120;
    ls = emptylist
in let f = proc(x)
             begin
              ls := list(size, color, weight);
              color := 3300
             end
   in begin
        color := 22 during f(weight);
        cons (color, ls)
      end
```

5. (15 points) Assuming static scoping and the direct array model, consider the following session with the defined language interpreter's read-eval-print loop

```
--> definearray a[2];
--> begin a[0] := 342; a[1] := 541 end;
--> define i = 100;
--> define f = proc(x,b) begin x := +(9,i); a[0] := i; b[1] := b[0] end;
--> f(i, a);
```

Fill in the following table with the final values of `i`, `a[0]`, and `a[1]` after running whole of the the above session, in each of the given parameter mechanisms. (If need be, use "?" to represent an undefined (i.e., unspecified) value.)

	ending value of		
calling mechanism	`i`	`a[0]`	`a[1]`
call-by-value			
call-by-reference			
call-by-value-result			

6. This is a problem about parameter passing mechanisms and array models. Throughout this problem use static scoping. Consider the following expression.

```
letarray a[2]; b[2]
in begin
     a[0] := 10; a[1] := 50; b[0] := 100; b[1] := 500;
     let add = proc(r, temp, u, v, index, acc)
                  begin
                    acc := 0;
                    temp := v;
                    temp[index] := +(temp[index], u[index]);
                    acc := +(acc, temp[index]);
                    index := +(index, 1);
                    temp[index] := +(temp[index], u[index]);
                    acc := +(acc, temp[index]);
                    r := temp;
                    %%% draw a picture at this point of the execution
                    acc
                  end
     in let r = add(a, b, a, b, 0, b[1])
        in
            list(a[0], a[1], b[0], b[1], r)
   end
```

For each of the following combinations of parameter passing mechanism and array model: (i) draw a picture of the execution (as discussed in class) for the point noted by the comment, and (ii) give the result of the expression. The combinations you are to do are as follows (there are more on the following pages).

(a) (15 points) Call-by-value with the indirect model.

Here is another copy of the expression, for your convenience.

```
letarray a[2]; b[2]
in begin
     a[0] := 10; a[1] := 50; b[0] := 100; b[1] := 500;
     let add = proc(r, temp, u, v, index, acc)
                  begin
                    acc := 0;
                    temp := v;
                    temp[index] := +(temp[index], u[index]);
                    acc := +(acc, temp[index]);
                    index := +(index, 1);
                    temp[index] := +(temp[index], u[index]);
                    acc := +(acc, temp[index]);
                    r := temp;
                    %%% draw a picture at this point of the execution
                    acc
                  end
     in let r = add(a, b, a, b, 0, b[1])
        in
            list(a[0], a[1], b[0], b[1], r)
   end
```

(b) (15 points) Call-by-reference with the direct model.

Here is another copy of the expression, for your convenience.

```
letarray a[2]; b[2]
in begin
     a[0] := 10; a[1] := 50; b[0] := 100; b[1] := 500;
     let add = proc(r, temp, u, v, index, acc)
                  begin
                    acc := 0;
                    temp := v;
                    temp[index] := +(temp[index], u[index]);
                    acc := +(acc, temp[index]);
                    index := +(index, 1);
                    temp[index] := +(temp[index], u[index]);
                    acc := +(acc, temp[index]);
                    r := temp;
                    %%% draw a picture at this point of the execution
                    acc
                  end
     in let r = add(a, b, a, b, 0, b[1])
        in
            list(a[0], a[1], b[0], b[1], r)
   end
```

(c) (15 points) Call-by-value-result with the indirect model. Don't show in your picture the copying back of the results; copy results back left-to-right.