CMSC 330 Parameters Fall 2012

Note: the use of call–by–name (or call–by–need) in a language which has side effects, as several questions below illustrate, is generally considered to be a poor programming language choice and a bad idea. However, it's included in these homework questions just to give you the opportunity to make sure you understand what call–by–name is and what its effects would be; if we didn't include any questions about call–by–name, or only had questions in which call–by–name produced the same results as call–by–value, then you wouldn't be able to test your understanding of call–by–name. (In the absence of side effects, call–by–name would produce the same results as call–by–value, which is why it's used in a language like Haskell, which has no side effects.)

1. For each of the three parameter transition techniques call–by–value, call–by–reference, and call–by–name, give the output of the following program, written using C syntax.

```
#include <stdio.h>

int a, b, i;
int c[4]= {100, 150, 175, 190};

void f(int d, int e, int f) {
  d= 5;
  b= a + 3;
  e= 3;
  i= 0;
  f= 20;
  c[i]= 200;
}

int main() {
  a= 10;
  b= 5;
  i= 1;
  f(a, i, c[i]);
  printf("%d %d %d %d %d %d %d\n", a, b, i, c[0], c[1], c[2], c[3]);
  return 0;
}
```

2. Consider the following code written in OCaml syntax:

```
let value = ref 1;;

let f n = value := !value + n ; !value;;

let g v w =
  let x = (f 2) in
    let y = v + w in
      let z = v * w in
        x + z - y;;

g (f 1) (if !value > 4 then 6 else 5)
```

   a. What output would the program produce if parameters were passed by value (as they actually are in OCaml)?

b. What output would the program produce if parameters were passed by need, as they are in Haskell? Recall that in call–by–need an actual parameter is evaluated only once, at the point where it is first used in the called procedure.

c. What output would the program produce if parameters were passed by name?

3. What results would the following program, written in C syntax, produce if parameters are passed by value? If C adopted call–by–need as its default parameter passing mechanism, what would the program's results be?

```
#include <stdio.h>

int func(int a, int b) {
  if (b == 0)
    return 0;
  else return func(a, b);
}

int main() {
  printf("%d\n", func(func(1, 1), func(0, 0)));
  return 0;
}
```

4. The following procedure will exchange the values of two integer variables if they are passed by reference. If the parameter transmission mechanism were changed to call–by–name, is there a set of input variables whose values cannot be exchanged?

```
void swap(int x, int y) {
  int temp;

  temp= x;
  x= y;
  y= temp;
}
```

5. Give the least restrictive limits that can be placed on actual parameters so that call–by–name and call–by–reference always give the same results in all cases.

6. (This problem is not about parameters, but we covered scoping as part of the same topic as parameters.) In Pascal, blocks are surrounded by `begin`/`end`, and comments are enclosed in curly braces (`{ }`). The built–in function `writeln` prints things. Semicolons are statment separators, not statement terminators, so the last statement in a block does not need to be terminated by a semicolon. Procedures can be nested inside other procedures, and a procedure that does not have parameters does not have empty parentheses after the procedure name in its definition; likewise a call to a procedure that has no parameters does not have empty parentheses. A procedure's local variables follow its header line, then its local procedures, then the block that is its body. (Nested procedures are commonly indented farther than the procedure that they are declared inside.) The main program's name is not `main`; it is just the outermost procedure, whose body is the last block in the program.

Given the above, give the output that the following program would produce if Pascal used dynamic scoping, and give its output if Pascal used static scoping (as it actually does). Note that the program would be correct regardless of which type of scoping were to be used.

```
program scope(output);

    var x, y: integer;  { the main program's variables }

    procedure a;
        var w, z: integer;

        procedure b;
        begin { b }
            writeln('b:  ', w, '  ', x, '  ', y, '  ', z)
        end;  { b }

        procedure c(y: integer);

            procedure d;
                var w: integer;
            begin { d }
                w:= y + x;
                z:= z + 1;
                b
            end;  { d }

        begin { c }
            x:= x * 2;
            d
        end;  { c }

        procedure e(x, z: integer);
        begin { e }
            w:= y + x;
            writeln('e:  ', w, '  ', x, '  ', y, '  ', z);
            c(6)
        end;  { e }

    begin { a }
        w:= 4;
        z:= 7;
        e(1, 2);
        writeln('a:  ', w, '  ', x, '  ', y, '  ', z)
    end;  { a }

begin { scope }
    x:= 3;
    y:= 8;
    a
end.  { scope }
```