

Problem Set 3—Procedural Languages

Due: Monday, 11 November at noon
Weight: 4% of your course grade.

1. Consider the following program in an imperative language:

```
main() {
    int i = 0, a[5] = { 1,2,3,4,5 }; /* ie, a[0]=1,a[1]=2,...,a[4]=5 */

    swap(int one, int two) {
        int tmp;
        tmp = one;
        one = two;
        two = tmp;
    }

    int f() { i=i+1; return i; } /* increments global variable i */
                                /* and returns its new value. */

    swap(a[i],i);
    write(i,a[0],a[1],a[2],a[3],a[4]); /* prints list of values */
                                        /* on a new line */

    swap(a[f()],i);
    write(i,a[0],a[1],a[2],a[3],a[4]); /* prints list of values */
                                        /* on a new line */
}
```

Give the output of the program, assuming the parameters are passed:

- (a) by value;
- (b) by value-result;
- (c) by reference;
- (d) by name.

Notes:

- Assume that for call-by-value, initial values of formals are copied sequentially (left to right) from the actual parameters.
- Assume that for call-by-value-result, initial values of formals are copied sequentially (left to right) from the actual parameters; final values of formals are copied sequentially (left to right) **at return time** into the actual parameters. (That is, the location to copy to is determined at return time.)

- Assume that for call-by-reference, the address of the actuals is determined sequentially (left to right) through the actuals at the time of the procedure call.
2. Below is a program in a block structured language like Pascal. In each procedure, variables are declared first, then nested procedures, and then the actual procedure body appears.

```

program
| A,B,C: integer;
| P:procedure
|   | C: integer;
|   | Q:procedure
|   |   | A,B: integer;
|   |   | begin
|   |   |   /* some code */
|   |   |   call R;           -----R
|   |   |   /* some code */
|   |   |   output := A,B,C; -----A -----B -----C
|   |   | end;-Q
|   | S:procedure
|   |   | C,D: integer;
|   |   | R:procedure
|   |   |   | begin
|   |   |   |   /* some code */
|   |   |   |   output := A,B,C; -----A -----B -----C
|   |   |   |   end;-R
|   |   |   begin
|   |   |   | /* some code */
|   |   |   | call Q;           -----Q
|   |   |   | /* some code */
|   |   |   | call R;           -----R
|   |   |   | /* some code */
|   |   |   | output := A,B,C; -----A -----B -----C
|   |   |   end;-S
|   |   | begin
|   |   |   /* some code */
|   |   |   call R;           -----R
|   |   |   /* some code */
|   |   |   call S;           -----S
|   |   |   /* some code */
|   |   |   output := A,B,C; -----A -----B -----C
|   | end-P
| R:procedure
|   | A: integer
|   | begin
|   |   /* some code */
|   |   output := A,B,C; -----A -----B -----C
|   | end;-R
| begin
|   /* some code */
|   call P;           Main.P
|   /* some code */
|   output := A,B,C; Main.A Main.B Main.C
end;-program

```

There are many instances of the same name used multiple times (the variables named A, B, C; the procedures named R). The correct association of the use of one of these names to the right declaration depends on whether the language uses static (lexical) or dynamic scope.

- (a) Assume the language uses static scope. We can uniquely identify each use of a name by writing it as `<proc>.<name>`, where `<proc>` is the name of the procedure (or `Main`, for main program) in which this use of `<name>` is declared. For example, the procedure call and variables used in the main program below have been labelled `Main.P`, `Main.A`, `Main.B`, and `Main.C`.

Using this convention, fill in the blanks below to uniquely label all other uses of names.

- (b) Assume the language uses dynamic scope. Begin to trace through the program and circle the **first** use of a name (use of a variable or call to a procedure) that refers to a different declaration under dynamic scope than the declaration indicated by static scope. Be precise: if it is a variable use, circle the first variable in the list which would be different.

3. Consider the following code in an imperative language that uses static scope:

```
Program M
Var g;
Var y;
Var b=true;
  procedure P
    Var x;
      Procedure R
        Var y;

        begin R

          y = Q();
          g = y;  /**** WHEN YOU GET HERE ****/
          return y;

        end R

      begin P

        y = 2;
        x = R();
        return x;

      end P

    procedure Q
      Var y;

      begin Q
        if b then {
```

```

        b = false;
        y = 3;
        g = P();
        return y; }
    else
        return 6;
end Q

begin M
    y = 1;
    g = P();

end M

```

- (a) Draw what the runtime stack would look like the first time you reach the assignment statement labelled **WHEN YOU GET HERE**. Show the stack frames including control links, access links, and local variable and procedure declarations. Remember this is a statically scoped language.
- (b) Repeat the above using a display instead of access links.

Silent Policy

A silent policy will take effect 24 hours before this assignment is due. This means that no question asked after noon on Sunday, 10 November will be answered, whether it is asked on the newsgroup, by e-mail or in person.

Handing It In

You can hand in this assignment on paper in the drop box (Sandford Flemming, 2nd floor, by the overpass to Pratt) or give it to your TA **before** the beginning of the day section tutorial. You may also hand it in electronically on CDF using the following command:

```
submit -c csc324h -a PS3 ps3.txt
```

We will accept your submission in plain ASCII (in which case your submission file must be called `ps3.txt`), in PostScript (`ps3.ps`), or in PDF (`ps3.pdf`). We recommend ASCII—the electronic submission option is meant to be convenient and easy, not to create additional work for you, so don't waste your time on a fancy presentation. You may draw your parse trees using any reasonable representation, as long as it is clear.

Type “`man submit`” on CDF for more information about the submit command.