CSE 505, Autumn 1996
D. Notkin
Homework #1
Assigned: 9/30/96
Due: 10/4/96

This assignment is intended to get us all on the same page with respect to some basic background you should have for the class. You may well have to look a few things up (in manuals, on the Web, etc.), but the basic concepts discussed shouldn't be too foreign to you (if they are, stop by and see me when you get a chance). You may work in pairs, turning in a single assignment with both names, receiving a single, shared grade. Assignments are due (preferably electronically) by the beginning of class on the due date; just email them to the TA in some reasonable format, or drop them off in the TA's mailbox (for non-electronic submissions or parts thereof). This assignment is worth 5% of the quarter's assignments (which in turn are worth 50% of the total grade).

1. (4 points) Dijkstra suggested the *structure principle:* "The static structure of a program should correspond in a simple way to the dynamic structure of the corresponding computations." In only a few sentences for each of the following language constructs, discuss whether they satisfy this principle:
   - `switch` statements in C;
   - single inheritance in any object-oriented programming language;
   - Fortran `DO` loops; and
   - exceptions in Ada.

2. (6 points) Consider the definition of a *strongly typed language* to be a language in which no program ever manipulates a bit pattern intended to represent a value of one type using operations intended to be applied to a different type. In only a few sentences for each, describe whether or not the language is strongly typed:
   - Pascal;
   - Lisp or Smalltalk-80; and
   - Eiffel or Java.

3. (10 points) Briefly define the term "coroutine.". Briefly present a problem for which a coroutine solution might be significantly clearer than one not using coroutines (in the same language).

4. (10 points) Böhm and Jacopini proved that all programs written with `goto` statements can be mechanically converted to programs that only use only sequencing, `while` loops, and `if-then-else` statements. Convert the Pascal program on the next page , which outputs the merge of two ordered arrays `x` and `y`, into a program without `goto`s and using only these three constructs. Do not rewrite the program from scratch, but instead convert it in a mechanical style; essentially, encode the program counter as a variable. Briefly compare the clarity of the two programs.

5. (4 points) Briefly distinguish a variable's *lifetime* and a variable's *scope*. Produce a fragment (from any programming language you wish) that shows a case where a live variable is out-of-scope.

6. (6 points) Functional languages generally provide a composition operator that composes any two compatible functions. Give two reasons why a composition operator cannot be written in Pascal. (Question modified from D. Watt.)

7. (5 points, extra credit) In any language you chose, write a self-replicating program. That is, write a program that takes no input and produces itself *exactly* as output. (Your program must contain at least one executable statement.)

```
program main (input,output);
label 2,3,5,6;
const n = 3; m = 2;
var i,j,k: integer;
    x: array[1..n] of integer;
    y: array[1..m] of integer;

begin
    x[1] := 5; x[2] := 7; x[3] := 11; y[1] := 1; y[2] := 9;

    i := 1; j := 1;
2:  if x[i] <= y[j] then goto 3
    else goto 5;
3:  writeln(x[i]);
    i := i+1;
    if i <= n then goto 2;
    for k := j to m do
        writeln(y[k]);
    goto 6;
5:  writeln(y[j]);
    j := j+1;
    if j <= m then goto 2;
    for k := i to n do
        writeln(x[k]);
6:
end.
```