Com Sci 221 — Programming Languages
Midterm Exam — Winter 1995
Due Monday, 20 February
3 pages of problems, 5 problems
Open book, open notes, open mind

Take this exam at a time and place of your own choosing, but in time to hand it in at the beginning of class on Monday, 20 February. Give yourself a contiguous 3-hour block of time to do the problems, starting when you open the exam paper. You may refer to your text, notes or any other written materials gathered in advance. You may not communicate with anyone except the instructor about the exam or class material during your 3-hour exam period. Each numbered problem is equally valuable.

1. Draw the abstract syntax tree and postfix form for each of the expressions

   (a) $((((a * b) + c) + d) * e) + f$

   (b) $a * ((b + ((c + d) * e)) + f)$

   (c) $a + (b * (c + (d * (e + f))))$

   (d) $((a + b) * c) + (d + (e * f))$

   How tall does the stack get when evaluating each of these expressions in postfix form? State a general principle that determines the height of stack required to evaluate a given abstract syntax tree. The principle is quite simple, but very few people have expressed it correctly in the past. Although the stack method of evaluation uses postfix, it is easier to calculate the stack height intuitively by looking at the abstract syntax tree. Hint: you must distinguish between tree branches going to the right and those going to the left. They have completely different impact on the stack.

2.
   - $A \to B$ denotes the set of functions from A to B (**array** A **of** B)
   - $A \times B$ denotes the set of ordered pairs from A and B (records with two fields, one containing a value in A and the other a value in B)
   - $A + B$ denotes the marked union of A and B (variant record field containing either a member of A or a member of B)
   - A^* denotes the set of lists containing members of A.

   For each of the following pairs of types, state whether the two are equivalent ($\cong$), the first can represent the second ($\succ$), the second can represent the first ($\prec$), or the two are incomparable ($\perp$). You may use the symbols in parentheses to express your answer, or use English phrases if you prefer.

(a)	$A \to (B \to C)$	$(A \times B) \to C$
(b)	$(A + B)^*$	$A^* + B^*$
(c)	$(A \to C) \times (B \to C)$	$(A \times B) \to C$
(d)	$(A \to B) \to C$	$A \to (B \times C)$
(e)	$(A^*)^*$	$(A + \{0\})^*$
(f)	$(A \to B) \times (A \to C)$	$A \to (B \times C)$

3. Consider the usual iterative program fragment to reverse a list s_0:

$s := s_0$;
$r := empty$;
while $s \neq empty$ **do begin**
    $r := cons(head(s), r)$;
    $s := tail(s)$
end

cons adds a new first element to a list, *head* returns the first element of a list (the same thing as *car* in Scheme), and *tail* returns all but the first element of a list (the same thing as *cdr* in Scheme).

Give appropriate precondition, postcondition, and loop invariant that could be used to show that the program fragment is correct (but don't do the proof). Use the mathematical operations p^R (reversal of p) and $p \,.\, q$ (p concatenated with q—equivalently q appended to the end of p) to express your assertions.

4. Draw the abstract syntax tree and the flowchart for the following program

$max := 0$;
$i := 1$;
while $i < n$ **do begin**
    if $A[i] > max$ **then begin**
        $max := A[i]$
    end;
    $i := i + 1$
end

Draw a box around the portion of the flowchart that corresponds to each nontrivial subtree in the abstract syntax tree (skip the trivial ones consisting of a single assignment statement). The subtrees include the whole **while** loop, the body of the **while** loop, and the whole **if-then** statement. Depending on how you choose to treat sequential composition in the abstract syntax tree, there may be another subtree. How many flowchart edges go into each box? How many go out of each box?

5. Flowcharts represent the flow of control in conventional procedural programs (using assignments, sequential composition, conditionals, and **while**-loops) very well. But, there are unusual commands whose impact on flow of control is not presented very clearly and naturally by flowcharts. Evaluate each proposed command and program construct below according to how well or poorly its impact on flow of control can be presented by flowcharts. If some of them are clearly worse than others, point that out. Justify your evaluation very briefly—perhaps one sentence for each command.

   - Parallel composition with **cobegin**-**coend**. **cobegin** $\alpha \| \beta$ **coend** causes α and β to be executed simultaneously.
   - Indirect addressing (fetch or store from memory at the address given in another memory location).
   - The **come**-**go** feature: every time **come** is executed, the current control point is saved. When a **go** is executed, control is returned to the point just after the last **come** to be executed. The state does not change. (The other constructs have been used in real programming languages, but this is a silly one that I made up.)
   - The computed **goto** command, which branches to the address given by the value of some expression.
   - The **break** command, which exits the innermost **while**-loop.
   - Program self-modification: the program may compute machine instructions, store them into memory and pass control to them.