Com Sci 221/321 — Programming Languages
Take-Home Midterm Exam, Autumn 1999

Due Monday, 8 November, 1:30 PM in class

5 problems on pages 2–6 plus 1 graduate problem on page 7

- This is a self-timed exam. Please limit yourself to 3 working hours from the time that you open the paper, until you complete the solutions that you hand in. You may take a break up to 30 minutes during the exam, but you should neither work on the problems nor study the class material during the break.
- You may use the textbook, my class materials, any other published material, and notes of your own. A *C* manual will be helpful for the graduate problem.
- There are 5 problems for Com Sci 221 and 321. They cover material in Chapters 1–4 of the text, and the class lectures through datatypes. There is 1 additional problem for Com Sci 321 only. Each numbered problem is weighted equally for grading purposes.
- The problem statements are a bit wordy, since I will not be nearby to answer questions. The wordiness does *not* not indicate subtlety.
- You may use whatever sort of paper you find convenient, but please make your answers legible, and write your name on every sheet.
- Please deliver your finished exam to me on paper at the beginning of class.

Com Sci 222 and 322

Problem 1: abstract syntax, postfix, evaluation

Draw the abstract syntax tree and postfix forms for the expressions

1. $a + (b * ((c + d) * e))$
2. $(((a + b) * c) + d) * e$
3. $a + ((b * (c + d)) * e)$
4. $a + (b * (c + (d * e)))$

The values of $a, \ldots, e$ are in memory, but our machine can only do arithmetic operations on registers. We can construct machine code to evaluate the expression by following the left-to-right sequence in the postfix form. Here is the machine code for expression 1, using 4 registers:

$$
\begin{aligned}
&R_1 := a \\
&R_2 := b \\
&R_3 := c \\
&R_4 := d \\
&R_3 := R_3 + R_4 \\
&R_4 := e \\
&R_3 := R_3 * R_4 \\
&R_2 := R_2 * R_3 \\
&R_1 := R_1 + R_2
\end{aligned}
$$

Every operation is done in an assignment of the form $R_i := R_i \circ R_{i+1}$ ($\circ$ stands for $+$ or $*$). The value of the whole expression always ends up in R_1. We reuse registers as much as possible while keeping strictly to the left-to-right evaluation dictated by the postfix expression. Depending on the form of the abstract syntax tree, an expression with $n \geq 2$ operands may require anything from 2 to n registers.

Write similar sequences of assignment commands to evaluate expressions 2, 3, and 4.

Explain very briefly how to determine the number of registers required from the shape of the abstract syntax tree. Look for a very simple property of the paths from root to leaf in the tree.

Problem 2: context-free grammar

Recall the context-free grammar for arithmetic expressions that gives precedence to $*$ and $/$ over $+$ and $-$:

$$
\begin{aligned}
E &\to E + T \\
E &\to E - T \\
E &\to T \\
T &\to T * F \\
T &\to T/F \\
T &\to F \\
F &\to C \\
F &\to V \\
F &\to (E)
\end{aligned}
$$

Modify the grammar to allow a single unary minus sign before any subexpression. For example, $x+-y$, $-w*z$, $-x+y$, $-(x-y)$, $-(-x)$, are allowed, but not $x+--y$, nor $x-+y$. A unary $-$ has the highest precedence.

Problem 3: control flow

Convert the following program schema with **goto**s into a pure conditional-**while** schema. Introduce new boolean variables as necessary. To get maximum points, you must preserve the structure of the schema as much as possible.

l :	SL_1
	if A **then goto** m
	SL_2
	goto l
m :	SL_3
	if B **then goto** n
	SL_4
	if C **then goto** m
n :	SL_5

Problem 4: reasoning with invariants

You are given two arrays A and B, both with indexes in the range $[0..99]$. A is sorted in increasing order, while B is sorted in decreasing order. That is,

$$\text{for all } k \text{ in } [0..98], A[k] \leq A[k+1] \text{ and } B[k] \geq B[k+1]$$

You are searching for the first i in $[0..99]$ such that $A[i] = B[i]$. Use two integer variables m and n, with the invariant

$$\begin{gathered}\text{for all } j \text{ in } [0..m-1],\ A[j] < B[j] \\ \text{and} \\ \text{for all } j \text{ in } [n+1..99],\ A[j] \geq B[j]\end{gathered}$$

Write a simple program with one **while** loop to find the desired i if it exists. To get nontrivial credit, your program must maintain precisely the invariant above—computing the right result is not enough. For full credit, your program should cut the distance between m and n in half each time.

Pay very close attention to the difference between strict comparisons $<$, $>$ vs. nonstrict comparisons $\leq$, $\geq$.

After the end of your loop write a very simple condition to test whether there exists an i with $A[i] = B[i]$.

Write a slightly different invariant that would lead you to the last i such that $A[i] = B[i]$, instead of the first one. Do not write the program, just the invariant.

Problem 5: datatypes

In C the following two assignments are precisely equivalent:

$$*C[i] = (*B[j]).first$$

$$*C[i] = B[j]\texttt{->}first$$

I list both of them since some of you may be more familiar with one form rather than the other. Describe the execution of the assignment above briefly but precisely, mentioning every l value and every r value involved, and every way in which an l value is treated as an r value or vice versa. You don't need to give the formula for calculating addresses of array components—merely mention that such a calculation must be done. Here are the first 3 steps to get you started. I'll abbreviate 'l value of ... ' by '$l(\ldots)$' and 'r value of ... ' by '$r(\ldots)$'.

1. dereference $l(j)$ to $r(j)$
2. calculate $l(A[j])$ from $l(A)$ and $r(j)$
3. dereference $l(A[j])$ to $r(A[j])$
4. ...

There are 11 steps total (including the 3 that I wrote above). If you follow the second version of the assignment, keep in mind that the '`->`' operator corresponds to a combination of a '.' and a '$*$'.

Com Sci 321 only

Problem G1: pointers and l values

In C, if x is a variable of type int$*$, then $*\&x$ is the same as x. The two expressions have the same l value, and may be substituted for one another anywhere. Briefly explain the expression $\& * x$, and its relation to x.