CSCE 531
Spring 2016
Final Exam

Do all six problems. You have 150 minutes. Write your solutions on the paper provided, leaving at least 0.5" blank in the left margin. This test is open book, open notes, but no electronic devices of any kind are allowed. That includes, but is not limited to, calculators, laptops, smartphones, tablets, smartwatches, and iAnything. For your own sake, please read *all* problems before trying *any* of them; the point value of a problem may not be commensurate with its difficulty.

You need not show your work unless explicitly asked to do so, but if your answer is incorrect, you are more likely to get partial credit if you show your work neatly and coherently.

There are 100 points total in the exam. 90 counts as full credit for graduates, and 81 for undergraduates. Any points earned beyond those limits counts as extra credit toward your exam grade.

1. (40 points total) NOTE: Read the entire question before giving your answers. You must get parts (a) and (b) correct to get any credit for parts (c) or (d).

   Consider the following grammar with start symbol S':

$$\begin{array}{rcl} S' & \to & S \\ S & \to & SaT \\ S & \to & T \\ T & \to & Tb \\ T & \to & cS \\ T & \to & \epsilon \end{array}$$

   (a) (10 points) Find $\mathit{FIRST}(S)$, $\mathit{FIRST}(T)$, $\mathit{FOLLOW}(S)$, and $\mathit{FOLLOW}(T)$. You need not justify your answer.

   (b) (15 points) Using the method described in class or the text, start to construct the set

$$\{I_0, I_1, I_2, \ldots\}$$

   of states for a simple LR (SLR) parser for this grammar, defining the transition function at the same time. (Note that this is *not* a canonical LR(1) parser! The states are sets of LR(0) items.) Compute the start state I_0 first, then *only* the possible transitions from I_0. This should produce some number of additional states $I_1, I_2, \ldots$, and you do not compute any further transitions. Omit the brackets for all items. Note that in class I denoted the transition function as *trans*. The textbook denotes the same function as *goto*. You may use the letter t to denote this function.

   (c) (10 points) Give as much of the action table for this parser as you can, based on your partial construction above. That is, define action$[i, a]$ for each state I_i and terminal a (including \$) that you can establish from your construction. Omit "error" entries and shifts to states that you did not construct.

(d) (5 points) Describe any conflicts you find, if any, giving the conflicting actions.

To ensure a unique correct answer, you must stick to the following two rules of order, which mirror the order I used for my example in class:

(a) List the kernel item(s) first in each state. List additional nonkernal items in the order that they enter the closure, or in case of a tie, the order that they appear in the grammar.

(b) When finding the transitions out of a state, or computing a closure, consider each item of the state in the order you listed it.

2. (15 points total) Given the following external declarations in C:

```
char c;
float f;
```

(a) (10 points) Draw the expression tree that is built for the expression `f = f + c` according to the rules for Project II. Be sure to include all implicit operators. Do not optimize the code by combining successive conversions. Include with each node of the tree (i) what variable or operator it corresponds to; (ii) the type of the corresponding subexpression; and (iii) "L" or "R" depending on whether the subexpression is an l- or r-value, respectively.

(b) (5 points) Give the sequence of back-end operations generated by traversing your tree. (You must get part (a) correct to get credit for this part.)

3. (15 points) The DO-loop in C has the following syntax (copied from gram.y):

```
iteration_statement
    : DO statement WHILE '(' expr ')' ';'
```

At runtime, the body (i.e., `statement`) is executed, then the test (i.e., `expr`) is evaluated. This sequence is repeated as long as the test evaluates to TRUE, and exits when the test evaluates to FALSE.

Describe what actions you would perform, and when, in the production above to emit code to execute the DO-loop, in the spirit of Project III. IMPORTANT: You may assume that the body of the loop does not include any `break`, `continue`, `goto`, or `return` statements, that is, nothing that could take control out of the loop without going through the test.

You may use pseudo-code or a concise English explanation, but you should mention by name any back-end routines you use. Also be clear about the value an action places on the semantic stack (if any).

4. (10 points total) Consider the following C declaration:

```
double a[8][3][5];
```

Assuming that a `double` is eight bytes,

(a) What is the size of `a`?

(b) Assuming that the base address of `a` is 1500, find the base address of the `double` variable `a[3][2][1]`.

5. (20 points total) Consider the following three-address code (line numbers added):

```
 1     L1:     i := a
 2             t1 := i + 5
 3     L2:     if t1 <= 10 then goto L6
 4     L3:     j := i + 1
 5             if j >= 0 then goto L1
 6             a := a + j
 7             goto L3
 8     L4:     if j <= 10 then goto L1
 9     L5:     i := j + 1
10             t1 := 2 * j
11             goto L4
12     L6:     a := t1 - 1
13             if a > 0 then goto L5
14
```

Assume that control enters at line 1 and that there are no other entry points.

(a) (5 points) Describe the basic blocks $B_1, B_2, \ldots$ by giving an inclusive range of line numbers for each block.

(b) (10 points) Draw the flow diagram as a directed graph, labeling the nodes $B_1, B_2, \ldots$. Give dangling arrows both for the entry point and for the exit point of the code as a whole.

(c) (5 points) Using the strict definition of a loop as defined in class and in the text, list any sets of vertices that constitute loops. Label any inner loop(s) as such.