# CS 536

## Final Exam

Tuesday, May 11, 1999

12:25 PM— 2:25 PM

6104 Social Sciences

### Instructions

Answer any *five* questions. (If you answer more, only the first five will count.) Each question is worth 20 points. Please try to make your answers neat and coherent. Remember, if we can't read it, it's wrong. Partial credit will be given, so try to put something down for each question (a blank answer always gets 0 points!).

1. Assume that we add a simple "for loop" to CSX:

   ```
   for (id = init to limit)
     stmt
   ```

   `id` is any integer variable. `init` and `limit` are integer-valued expressions that are evaluated once, at the beginning of the loop. Iteration terminates as soon as `id` is greater than `limit`. Zero iterations are possible if `init` is greater than `limit`. At the end of each iteration `id` is incremented by 1.

   Show an outline of the JVM code you'd generate for this kind of loop. Design an AST structure appropriate for this for loop.

2. Just as variables and fields may be initialized, some programming languages allow formal parameters to be initialized. An initialized parameter provides a *default* value. In a call of a method, a user may choose to not provide an explicit parameter value, choosing the default instead. For example, given

   ```
   int power(int base, int expo = 2) {
     /* compute base**expo */}
   ```

   the calls `power(100,2)` and `power(100)` both compute the same value ($100^2$).

   What changes would be needed in your CSX type checker to correctly handle initialized formal parameters?

3. Recall that CSX allows no overloading. That is, in each scope each identifier must be uniquely defined. However identifiers used as labels are very different from identifiers used as variables, constants, parameters and methods. Hence allowing an identifier to be used as both a label and “something else” within a scope might be reasonable.

   Explain the changes that would have to be made to CSX symbol tables and type checkers to allow the same identifier to be used as a label and as one other thing (a variable, constant, parameter or method) within the same scope. For example, the following would be legal in extended CSX:

```
{ int i=21;
   i: while (i > 0){
         i = i - 1;
         if (i == 17)
            break i;
   }
}
```

4. Let G be some context-free grammar and let A be a nonterminal symbol in G. Explain how to test whether *any* of the terminal strings derived from A is odd in length.

5. (a) Consider the following context free grammar:

   S → L a b
   S → L a c
   L → λ

   Is this grammar LL(1)? Why? Is this grammar LALR(1)? Why?

   (b) Consider the following context free grammar:

   S → L S a
   S → b
   L → λ

   Is this grammar LL(1)? Why? Is this grammar LALR(1)? Why?

   (c) Consider the following context free grammar:

   S → L a S
   S → b
   L → λ

   Is this grammar LL(1)? Why? Is this grammar LALR(1)? Why?

6. A common compiler optimization is *constant folding*. This optimization replaces an expression whose operands are all literal constants with the value the expression must compute. For example, `1+2` can be folded into the value `3`.

   What changes would be needed in your code generation routines for CSX to support folding of binary operators whose operands are both literals? Are any further changes needed to implement folding when an operand is not a literal constant but is an expression that can be folded into a literal value (for example, `(1+2)*3`)?

7. Assume we are translating a function or procedure call in CSX. Explain how we could determine the exact number of stack locations that will be needed to evaluate and store (on the stack) the parameters for the call. Illustrate your technique on the following call:

   ```
   p(a+g(1,2), x-(y-(p+1)));
   ```