# CIS (4|5)61
## Winter 2005 Final Exam (Take Home

Everyone should complete the first two questions. Undergraduates should complete one of the remaining questions; graduates should complete all.
You may submit your answers to this exam on hardcopy or by emailing a PDF file. In either case, completed exams are due no later than 4pm on Wednesday.
You must spend no more than 4 hours on the exam, and you must work alone. You may not use a lexical analyzer generator (lex, flex, jflex, etc.) or a parser generator (yacc, bison, cup, etc.) to find answers. You may, however, refer to reference materials describing those tools, as well as textbooks and other information.

1. [10 points] Recall that tools in the lex family of lexer generators use a "greedy" or "maximum munch" rule in pattern matching, always matching the longest possible prefix of the input stream.

(a) Show the DFA that will be generated if the input to the lexical analyzer generator is

```
"ab"   { return new Symbol(tokens.SHORT);  }
"abaa" { return new Symbol(tokens.LONG);   }
```

(b) Consider how the generated lexical analyzer processes the input string "abab". Describe each step of the processing, including state transitions, calling action routines, and maintaining the input buffer (e.g., advancing to the next character). For example, the first line of your answer might be "Looking at 'a'; advance from state 0 to state 1 and advance to second character of input."

2. [10 points]

Languages with syntax derived from C, including C++ and Java, use ”cast” notation like

```
x = (FooBar) y.foo();
```

The following grammar is a very simple model of expressions with optional casts. The terminal symbols are left and right parentheses, identifiers ( *I* ), and the subtraction operator (“−”).

$\langle S \rangle \longrightarrow \langle E \rangle$ \$

$\langle E \rangle \longrightarrow \langle E \rangle$ “−” $\langle P \rangle$

$\langle E \rangle \longrightarrow$ “(” $I$ “)” $\langle E \rangle$

$\langle E \rangle \longrightarrow \langle P \rangle$

$\langle P \rangle \longrightarrow$ “(” $\langle E \rangle$ “)”

$\langle P \rangle \longrightarrow I$

Is it LALR(1)? Give enough of the CFSM to justify your answer.

3. [10 points]

Some of you generated code for the stack-oriented Java virtual machine, and some of you generated code for a physical processor with registers. If you generated Java byte code, you had to calculate the maximum number of stack slots required for expression evaluation. It is relatively simple to translate stack-oriented code into register-oriented code, provided you have as many available registers as the maximum required stack depth. Briefly describe how, and illustrate with an example of translation from postfix or stack-oriented code to register-oriented code for the expression $a + b - (c + d)$.

4. [10 points]

Creating a new object is typically much more expensive than re-initializing an existing object. Some programmers use "object pooling" to avoid the extra overhead of memory allocation, keeping track of objects that can be recycled. Like many hand-optimizations this one makes source code harder to understand and more error-prone.

Since the job of the optimizer is to remove the temptation to write ugly code, devise a transformation and a safety check (using data flow analysis) to recycle objects that are created each time through a loop. Describe it briefly but in enough detail that someone with a basic familiarity with data flow analysis and optimization could implement your scheme.