Stanford University
Computer Science Department

Fall 2005 Comprehensive Exam in Compilers

1. **Closed Book, - NO laptop. Write only in the Blue Book provided.**
2. **The exam is timed for one hour.**
3. **Write your Magic Number on this sheet and the Blue Book; DO NOT WRITE YOUR NAME.**

The following is a statement of the Stanford University Honor Code:

A. The Honor Code is an undertaking of the students, individually and collectively:

1. that they will not give or receive aid in examinations; that they will not give or receive un-permitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;

2. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

B. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

By writing my Magic Number below, I certify that I acknowledge and accept the Honor Code.

Magic Number----------

Compilers Comprehensive, November 2005

This is a 60 minute, closed book exam. Please mark your answers in the blue book.

1. (10 points)

   Suppose you were implementing a lexical analyzer for the C programming language in a tool like Lex or Flex. Suppose the entire input to the compiler were:

```
divbypointer(double num, double *pdenom)
{
  return num/*pdenom;
}
```

   Describe two ways of handling comments.

   (a) Method 1 uses a regular expression that matches a complete comment as a single lexeme.

   (b) Method 2 recognizes `/*` and then enters a special "start condition," in which `*/` and any single character are recognized as lexemes. When `*/` is recognized, it returns to the default start condition, in which C language tokens are recognized.

   Briefly discuss how each method would handle the example above, and discuss the practical merits of each.

2. (15 points) These questions are about the handling of variables by a compiler for a simple language like C. In your answers, address only the compiler behavior that is *necessary for code generation.* Do not address type checking or other aspects of semantic analysis are not strictly necessary to emit code for correct programs.

   (a) Describe the compiler's processing of a global variable g of type `int`, both at the point of *declaration* and at the point of *use*.

   (b) How is the handling of a local variable declaration of type `int` different from the handling of the global variable of the same type?

   (c) What information does the compiler have to maintain in the symbol table to generate code for `A[i].f[j]`?

3. (10 points)

   Why would it be useful for an optimizing compiler to have optimizations on an intermediate representation (such as 3-address code) *and* peephole optimization at the instruction level?

4. (35 points) Consider the following context-free grammar:

$$
\begin{aligned}
S &\rightarrow Sa \\
S &\rightarrow bS \\
S &\rightarrow c
\end{aligned}
$$

(a) (3 points) Show that the grammar is ambiguous.

(b) (10 points) Write the canonical collections of LR(1) items for this grammar.

(c) (2 points) Identify all conflicting items, and the types of the conflicts (e.g., "shift-reduce conflict in state 3 on d").

(d) (5 points) Could the original grammar be converted into an LALR(1) parser that parses all input correctly by resolving conflicts, in the way that YACC and similar parser generators allow? If so, how should they be resolved? In either case, please explain (briefly).

(e) (5 points) Rewrite the grammar in an equivalent form that is suitable for LL parsing and minimizes the use of stack space.

(f) (5 points) Rewrite the grammar in an equivalent form that is directly suitable for LR parsing (i.e., does not result in conflicts) and minimizes the use of stack space.

(g) (5 points) In your modified LL(1) grammar, show the sequence of stack contents and inputs when parsing the input *bbcaa*.