**Programming Languages and Compilers Comprehensive Exam**

**September 1998**

Please answer each question in a separate booklet.

1. **Enter your answer in the booklet labeled ??.**

   The notation for a type rule,

   $$A \vdash \ e \ : \ \tau$$

   means "under type assumption A, expression e has type $\tau$". A type assumption is of the form $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$, binding variables to their types.

   In addition, a type rule can be of the form

   $$\frac{R_1 \ldots R_n}{R}$$

   which means that if type rules $R_1 \ldots R_n$ hold, then rule $R$ holds.

   (a) For a monomorphic, higher-order functional language, give the type rules for
      i. variable references (that is, when the expression is just $x$).
      ii. function application (when the expression is $(e1\ e2)$)
      iii. lambda abstraction (when the expression is $\lambda x.\ e$)

   (b) Describe informally how these type rules would be different in a polymorphic, higher-order functional language.

   (c) Give the type rules from part (a) for a polymorphic, higher order functional language such as ML.

2. **Enter your answer in the booklet labeled ??.**

   In the idealized logic programming model, if there is a clause that unifies with the current goal such that choosing that clause will lead to success, then that clause will be chosen. For example, if the current goal is `f(X,b)` and you have the clauses

```
f(A,B) :- ...
f(A,b) :- ...
f(a,B) :- ...
```

   then the clause that leads to the satisfaction of the goal will be chosen. If more than one clause would lead to success, then one is chosen non-deterministically.

   (a) Give the algorithm for an interpreter for logic programs in this idealized model. You can assume that unification has already been defined.

   (b) How does Prolog simulate this "angelic" non-determinism? Describe how the interpreter you described in part (a) would be different if it had to implement Prolog.

   (c) In prolog, describe the effect of the ! operator (as it relates to your answer to part (b)).

3. **Enter your answer in the booklet labeled ??.**

   (a) Define the term *dominator*.

(b) In a program, a statement S2 is *control dependent* on a statement S1 if - whenever S1 is executed - S1 solely determines whether S2 is subsequently executed or not. Give a formal definition of control dependence in terms of dominators.

(c) Why would a compiler compute control dependence information? What kind of program analysis or optimization would rely on it?

(d) Sketch out an algorithm for an analysis or optimization based on control dependence.

4. **Enter your answer in the booklet labeled ??.**

   Consider this Pascal program.

```
procedure A()
  var i: integer;

  procedure B(x: integer)
  begin (* B *)
    writeln(x+1);
  end (* B *)

  procedure C(procedure G(y: integer))

     procedure D()

         procedure E(z: integer)
         begin (* E *)
           G(z+2);
         end (* E *)

     begin (* D *)
        E(i);
     end (* D *)

  begin (* C *)
     i := 6;
     D();
  end (* C *)

begin (* A *)
  C(B);
end (* A *)
```

   (a) Draw the state of the stack during the call to G, including all dynamic links (aka *control* links) and static links (aka *access* links).

   (b) How was the static link for G created?

   (c) Give an algorithm that the compiler might use to generate the code that sets up the static link for the activation record (i.e. stack frame) when a procedure is called. Be sure to consider the case where the procedure itself was passed as a parameter (e.g. procedure G in the above example).

5. **Enter your answer in the booklet labeled ??.**

(a) Define the term *peephole optimization*.

(b) Name two optimizations that are generally performed by a peephole optimizer.

(c) Give an example demonstrating each of the optimizations you mentioned in part (b), showing the code before and after the optimization.

(d) Peephole optimizers are good at improving code that contains inefficiencies due to the way the compiler generated code, rather than the way the source program was written by the programmer. Describe in detail a scenario in which a compiler might generate poor code which a peephole optimizer could improve significantly.

6. **Enter your answer in the booklet labeled ??.**

   (a) State the two Church-Rosser theorems concerning the lambda calculus.

   (b) What are the relative advantages and disadvantages of normal order evaluation, compared to applicative order evaluation? For each disadvantage of normal order evaluation that you list, state how it's negative effects can be reduced by a smart implementation.

   (c) The Y combinator is used to express recursion in the untyped lambda calculus. It can be written

$$\lambda h.\ (\lambda x.\ h\ (x\ x))\ (\lambda x.\ h\ (x\ x))$$

   Can Y be written in the simply typed lambda calculus (i.e. a lambda calculus where each identifier is given a monomorphic type)? Explain.