**Programming Languages and Compilers**
**Comprehensive Exam**

**May 2000**

Please answer each question in a separate booklet.

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

   For this question, consider only the purely functional subset of Standard ML (i.e. without `ref`'s and without I/O).

   (a) Write a function in ML whose type would be displayed as `'a -> 'a`.

   (b) Write a function in ML whose type would be displayed as `'a -> 'b`

   (c) For our purposes, two functions, $f_1$ and $f_2$, are distinct iff there is at least one $x$ such that $f_1(x) \neq f_2(x)$. Can you write a function that is <u>not</u> distinct from the function you gave in part (b), but whose type would be displayed as `'a->'a`? If so, then do so. Otherwise, explain why not.

   (d) How many distinct functions are there in ML whose type would be displayed as `'a -> 'a`? Sketch a formal proof of your answer.

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

   Write in SETL, using the expressive power of the language, a procedure `BIN` that solves the one dimensional Bin Packing problem exactly. The Bin Packing problem can be stated as follows: Given a bin size and a set of objects of varying sizes, what is the minimum number of bins necessary to hold all the objects, such that the sum of the sizes of the objects in each bin is no greater than the size of the bin? Assume the input to `BIN` is an integer bin size and a set of integers representing the sizes of the objects. Do not worry about the computational complexity of `BIN`.

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

   (a) Define the terms *static scoping* and *dynamic scoping*.

   (b) Give an example in Ada of a simple program that would display a different result than usual if the compiler accidentally implemented dynamic scoping.

   (c) Is the lambda calculus statically scoped or dynamically scoped? Indicate which rule(s) in the lambda calculus determines the scoping and state such rule(s) precisely.

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

   (a) Give an intuitive explanation of why canonical LR parsers are more powerful than SLR parsers.

   (b) The algorithm for building a canonical LR parse table differs from that for an SLR parse table in how *items* are constructed and used. Describe precisely these differences.

   (c) Give a simple example of a grammar that can be parsed using a canonical LR parser but not an SLR parser.

   (d) Without building the entire parse table, show why the differences in the way that the *items* are constructed and used for the grammer you gave in part (c) would allow the grammar to be parsed by a canonical LR parser but not by an SLR parser. This question is just asking you to illustrate your answer to part (b) using the grammar in part (c).

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

   (a) Define the term *copy propagation* in the context of compiler optimizations.

   (b) Give the dataflow equations that need to be solved in order to support copy propagation.

   (c) Give the algorithm for performing copy propagation, assuming the dataflow equations have been solved,

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

   (a) Define precisely the term *back edge* in the context of a flow graph. If you use any other terms specific to flow graphs, define those as well.

   (b) Define precisely the term *loop* in the context of a flow graph. There are several definitions of a loop, you may choose any one of these.

   (c) Give an algorithm for finding loops in a flow graph.