CSci 401 Introduction to Compilers
Final Exam

Spring 1998 W.L. Ruzzo

1. (15 points) Eliminate left recursion from the following grammar using the general method presented in the text.

   S → **(** L **)** | **atom**
   L → L **,** S | S

2. (20 points) For the grammar below with start symbol E:

   (a) Compute FIRST for each right hand side and FOLLOW(E) (in the table below).

Rule				FIRST	FOLLOW
(1)	E	$\rightarrow$	$E\,;E$		
(2)	E	$\rightarrow$	**a**		
(3)	E	$\rightarrow$	ϵ		

   (b) Fill in the "parsing table" below, showing for each nonterminal and each lookahead symbol which productions (if any) could be used to expand that nonterminal when parsing a string with that lookahead. Cells you leave blank are assumed to be cases where the parser should signal an error.

Nonterminal	Lookahead Symbol		
	a	**;**	**$**
E			

   (c) Is the grammar LL(1)? Why or why not?

3. (15 points) In general, can all array indexing operations be bounds-checked as part of compile-time static type checking? Why or why not? If not, outline when and how it can be done.

4. (15 points) Consider the following program:

```
Module M;
  var z:int;

  procedure foo(x:int, y:int);
  begin
    x := x + y;
    z := z + x + y;
  end foo;
```

```
begin
  z := 5;
  foo(z, z);
  output := z;
end M.
```

(a) What are the possible outputs of this program assuming *call-by-value* parameter passing?

(b) What are the possible outputs of this program assuming *call-by-reference* parameter passing?

(c) What are the possible outputs of this program assuming *call-by-value-result* (also called *copy in/copy out*) parameter passing?

5. (20 points) For the following program, draw the activation records, dynamic links, and static links for the program when execution reaches the point marked HERE. (You do not need to show offsets or other internal structure in the activation records.)

```
module M;
  procedure P();
    procedure Q();
    begin
      # HERE
      Q();
    end Q;
  begin
    Q();
  end P;
  procedure R();
  begin
    P();
  end R;
begin
  R();
end M.
```

6. (25 points) The marketing department wants you to add some debugging aids to help programmers writing Pl/0 programs (as opposed to the Pl/0 compiler writers). In particular, they want you to provide a library procedure called "dump()," which takes no parameters, and can be called at any point (or points) during the execution of a Pl/0 program to produce a listing of the names and values of all programmer-declared

variables lexically visible at the point of call. Describe how you would do this. In particular, note that the compiler would have to provide some information about the source program so that dump() could do its job. What information would it need to provide, where should that information be placed, how would dump() find it, and how would it use it? Does the compiled code need to be changed in any way, for example to maintain extra information in the run-time stack? Be as specific as possible. (Note that the same dump() procedure can be used by all source programs; the compiler doesn't produce a different version for each source program or call site.)

How would your answer change if dump() was to show all programmer-declared variables having values, rather just the lexically visible ones? (Remember to consider recursion.)

7. (25 points) Suppose the PL/0 language were modified to allow array bounds to be computed at run-time, e.g.

```
...
procedure foo(n:int);
  var a:array[n] of array[n] of int;
...
```

Briefly discuss how this would change run-time storage layout and code generation.

8. (25 points) The designers of the language Jaba, having missed my remarkably lucid explanation of block-structured scope rules, decided that they complicate the compiler and make a language somewhat error-prone (e.g. forgetting a local declaration of a variable name that happens to exist in an enclosing block) and harder to read ("where is *foobar*, anyway?"). Instead, they decided that all references to variables that are not local to the current procedure should be fully qualified. That is, if procedure P is declared inside procedure Q which is in turn declared inside the top-level procedure R, then inside P you would write `x + R.Q.y + R.z` to add together variables `x, y,` and `z` declared in P, Q, and R, respectively.

   (a) Discuss how you would change the symbol table and associated routines to accommodate such a requirement.

   (b) True or False: this change would allow you to eliminate static links and/or displays from the runtime environment, since you now know exactly where every variable comes from. Justify your answer.

9. (20 points) Intermediate code generally assumes an unlimited number of registers (or makes essentially no mention of registers at all). What advantages does this have?

10. (20 points) What is the difference between *peephole* and *local* optimization?

    Which is better? Which is easier? Justify.