Compiler Construction

Comprehensive Exam

May 1997

1 Parsing

a. Write down the algorithm (the "skeleton parser") for parsing a sentence using an LR(1) parse table.

b. Consider the following augmented grammar over the alphabet {c,d}:

```
0) S' → S
1) S → CC
2) C → cC
3) C → d
```

When fed to an LR parser generator, the following *action* and *goto* tables were produced:

	Action			Goto	
State	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Show the sequence of stack contents and actions that the machine will go through when given the string:

```
cdd$
```

2 Run-time Storage Management

In both parts of the question below, assume that you are compiling a language like **Pascal**, with recursive procedures and procedure and block nesting to arbitrary levels.

a. Describe the layout of an activation record for such a language and explain how it is possible for every simple *local* variable, even those declared in nested blocks, to be accessed as a constant offset from the activation record pointer, even when the procedure contains declarations of arrays whose size is not known until run time (i.e., the array dimensions are specified as expressions of variables known on entry to the procedure). Describe how storage is allocated on the stack for arrays. Explain how a compiler determines the locations within an activation record of scalar variables and arrays.

b. For the same language, describe the use of a *display* for accessing variables in the current activation records of procedures in which the current procedure's declaration is nested. Where is the display stored and how is it constructed on procedure invocation?

3 Code Generation

a. Describe the Sethi-Ullman "labeling algorithm" for "optimal" code generation from expression trees. Describe how code is generated from a labeled expression tree.

b. Explain how to adapt the labeling algorithm to a load-store architecture with no load delay.

4 Code Optimization

A variable `v` is said to be *live* at a given point *p* in a program if there is a control-flow path from *p* to a use of `v` which path contains no definition of `v`.

a. Give data-flow equations for the live problem on the control flow graph of a program.

b. Give one solution algorithm for these data flow equations and specify its complexity in time and space.