

Name: _____

USC ID: _____

Midterm Exam CS 410, Spring 2001 [Bono]

March 1, 2001

There are 6 problems on the exam, with 80 points total available. There are 7 pages to the exam, including this one; make sure you have all of them. If you need additional space to write any answers, you may use the backs of pages (just direct us where to look).

Put your name and USC ID number at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	12 pts.	
Problem 2	10 pts.	
Problem 3	6 pts.	
Problem 4	20 pts.	
Problem 5	20 pts.	
Problem 6	12 pts.	
TOTAL	80 pts.	

Problem 1 [12 pts.]

Circle all that apply about the compiler tool `bison` (same things are also true of the `yacc` tool):

1. Generates a parser whose actions correspond to a leftmost derivation
2. Generates a table-driven parser
3. Generates a recursive-descent parser
4. Generates a predictive parser (i.e., LL(1))
5. Takes a grammar as input
6. Generates a LALR(1) parser
7. Generates a bottom-up parser
8. Applies Thompson's construction
9. Takes regular expressions as input
10. Generates a top-down parser
11. Takes AST's as input
12. Generates a parser whose actions correspond to a rightmost derivation in reverse

Problem 2 [10 pts.]

Part A [10]. Use subset construction to build a DFA equivalent to the NFA below. Show your work.

	a	b	ϵ
1	{ }	{ }	{ 2 }
2	{ 2 }	{ 3, 4 }	{ 5 }
3	{ 5 }	{ }	{ }
4	{ }	{ 4 }	{ 1, 5 }
5	{ }	{ }	{ }

Problem 3 [6 pts.]

Show that the following grammar is ambiguous:

$$\begin{array}{l} X \rightarrow y X X \\ \quad | y X \\ \quad | m \end{array}$$

Problem 4 [20 pts.]

Part A [15]. Build the DFA to recognize viable prefixes for the grammar below. We have already augmented the grammar for you. The set of terminals is $\{\mathbf{id}, =, [,]\}$

$$0) S' \rightarrow S$$

$$1) S \rightarrow \mathbf{id}$$

$$2) S \rightarrow V = E$$

$$3) V \rightarrow \mathbf{id}$$

$$4) V \rightarrow \mathbf{id} [E]$$

$$5) E \rightarrow V$$

Part B [5]. If we attempt to build an SLR parser with the grammar given above, we'll get a reduce-reduce error. Show exactly where and how this error occurs (i.e., in terms of states from part A, inputs, and reductions).

Problem 5 [20 pts.]

Consider the following grammar (a, b, c, and d are the terminals, ϵ is the empty string):

$$\begin{aligned} S &\rightarrow a Z b \\ &\quad | a S Y b \\ Y &\rightarrow d Y Y \\ &\quad | Z \\ Z &\rightarrow c Z \\ &\quad | \epsilon \end{aligned}$$

Part A. Eliminate left recursion from the grammar, if necessary, or write "Same", if not necessary.

Part B. Left-factor the grammar you gave for part A, or write "Same", if not necessary.

Part C. Attempt to build an LL(1) parse table for the grammar you gave in part B. It is not LL(1): for places where we can put multiple values in a table entry, show all the values.

Note: show your work.

Circle and label (A,B,C) your answers to each of the parts.

Problem 6 [12 pts.]

Consider the following grammar for arithmetic expressions. Write semantic rules to construct an RPN (Reverse Polish Notation) version of the expression. RPN is a postfix notation, that is we write the operator after the two operands instead of between them. With RPN you don't ever need parentheses to specify an arithmetic expression. Hint: use a semantic attribute called `rpn` which is a string containing the RPN version of the expression. The `rpn` attribute will be of type `string` from the STL library. Here are some examples of what you can do with strings:

```
string s = "water"; // assign a C-string to a string
string t = "melon";
string u;
u = s + t; // concat two strings: u is "watermelon"
s = s + "boy"; // concat with type conversion:
// s has value "waterboy"
u[0] = 'l'; // access/change individual chars
// u has value "latermelon"
```

Here are some examples of RPN form:

<u>(infix) expression parsed</u>	<u>E.rpn</u>
3 + 5	3 5 +
4 + 3 * 7	4 3 7 * +
3 + 5 - (4 + 3 * 7)	3 5 + 4 3 7 * + -
3 + 5 - 4 + 3 * 7	3 5 + 4 - 3 7 * +

$E \rightarrow E_1 + T_1 \quad \{ E.rpn =$

| $E_1 - T_1$

| T

$T \rightarrow T_1 * F$

| F

$F \rightarrow \text{num} \quad \{ F.rpn = \text{num.lexval} \}$

| (E_1)