

Name: \_\_\_\_\_

USC ID: \_\_\_\_\_

## Midterm Exam CS 410, Spring 2000 [Bono]

March 2, 2000

There are 7 problems on the exam, with 90 points total available. There are 8 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!

	<b>value</b>	<b>score</b>
Problem 1	9 pts.	
Problem 2	15 pts.	
Problem 3	6 pts.	
Problem 4	12 pts.	
Problem 5	20 pts.	
Problem 6	12 pts.	
Problem 7	16 pts.	
<b>TOTAL</b>	90 pts.	

## Problem 1 [9 pts.]

**Part A [2].** What does it mean for a nonterminal,  $S$ , to be *nullable*?

**Part B [2].** `bison` is a tool for automatically building a

---

from a \_\_\_\_\_.

**Part C [5].** Give a regular expression to recognize string constants that have the following rules:

- uses the alphabet  $\{a, b, c, @, \backslash, \text{"}\}$
- surrounded by double-quotes
- may not contain double quotes
- may only contain `@` if preceded by a `\`
- may contain any other combination of chars in the alphabet (including empty string)

Here are some examples:

<code>"\aa\@bb\@\@"</code>	legal
<code>"aabb@"</code>	illegal -- @ not preceded by \
<code>"aa\"</code>	illegal -- contains a quote

**Problem 2 [15 pts.]**

Consider the following regular expression:  $(ab)^*$

**Part A [5].** Use Thompson's construction to build an NFA for the regular expression.

**Part B [10].** Use subset construction to build a DFA equivalent to the NFA you built for part A. Show your work.

### Problem 3 [6 pts.]

Show that the following grammar is ambiguous.

$$\begin{aligned} E &\rightarrow E + T \\ &\quad | T \\ T &\rightarrow T + E \\ &\quad | \mathbf{id} \end{aligned}$$

### Problem 4 [12 pts.]

The following grammar suffers from the dangling else problem.

**Part A.** If we attempt to build an SLR parser with the grammar, we'll get a shift-reduce error. Show exactly where and how this error occurs. I.e., show the relevant sets of items in the DFA, and the details and rationale for the shift and for the reduce. You do not have to build the whole DFA or parse tables to answer this question.

**Part B.** If we want every **else** to match the closest **if** (normal C/C++, etc. interpretation), should we choose *shift* or *reduce* at the point of the conflict?

- 1)  $S \rightarrow \text{if ( expr ) } S$
- 2)  $S \rightarrow \text{if ( expr ) } S \text{ else } S$
- 3)  $S \rightarrow \text{other}$

### Problem 5 [20 pts.]

Consider the following grammar for arithmetic expressions in reverse-polish notation:

$E \rightarrow E E +$

$E \rightarrow E E -$

$E \rightarrow E E *$

$E \rightarrow E E /$

$E \rightarrow id$

**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.** Show the first and follow sets for all the non-terminals in the grammar you gave in part B.

**Part D.** Show the LL(1) parse table for the grammar you gave in part B.

Circle and label your answers to each of the parts.

### Problem 6 [12 pts.]

Consider the following grammar for arithmetic expressions in reverse-polish notation (less operators than in the previous problem):

$E \rightarrow E E +$

$E \rightarrow E E -$

$E \rightarrow id$

Show the DFA for recognizing viable prefixes for this grammar, including the sets of items associated with each state.

## Problem 7 [16 pts.]

One of the rules for a legal switch statement in C is that it have at most one "default option". You are going to write semantic rules for the grammar below such that it computes the attribute `SwStmt.legal`, which is true if the switch statement has at most 1 "default option", false otherwise. `SwStmt` being legal means *that* switch statement is legal, it says nothing about the legality of any switch statements nested inside it (see ex 4 below). Hint: you may use additional semantic attributes.

A little more information about the C switch statement syntax. A "default option" is a kind of statement. Default options, and other parts of the switch statement may be nested inside of other statements in the switch. A default option belongs to the closest enclosing switch statement. (I'm not making any of this up!) Here are some examples:

```
switch ( expr ) /* legal */
{
}

switch ( expr ) { /* legal */
  while (expr) {
    default : expr;
  }
}

switch ( expr ) { /* legal */
  switch ( expr ) { /* legal */
    default : expr;
  }
  default : expr;
}

/* ex. 4 */
switch ( expr ) { /* legal */
  switch ( expr ) { /* not legal */
    default: expr;
    default: expr;
  }
  default : expr;
}

switch ( expr ) { /* not legal */
  default : expr;
  default : expr;
}
```

---

A simplified grammar (terminals are in bold, and punctuation; other identifiers are nonterminals;  $\epsilon$  denotes the empty string):

`SwStmt`  $\rightarrow$  **switch** ( **expr** ) `Stmt`

`Stmt`  $\rightarrow$  **expr**

| `SwStmt`

| **default** : `Stmt`<sub>1</sub>

| **while** ( `expr` ) `Stmt`<sub>1</sub>

| { `Slist` }

`SList`  $\rightarrow$   $\epsilon$

| `Stmt` `SList`<sub>1</sub>