

Name: _____

USC loginid (no SSN): _____

CS 410 Midterm Exam
Fall 2002 [Bono]
October 18, 2002

There are 5 problems on the exam, with 50 points total available. There are 6 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 to look there).

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

	value	score
Problem 1&2	9 pts.	
Problem 3	10 pts.	
Problem 4AB	14 pts.	
Problem 4CD	7 pts.	
Problem 6	10 pts.	
TOTAL	50 pts.	

Problem 1 [5 pts.]

For each of the following language rules (numbered 1-5) for the Cool programming language, name the least powerful machine that can be used to check it. Give each answer as one of a, b, or c below:

- a. FSA
- b. PDA (the book calls this a stack automaton)
- c. Turing machine (this is equivalent to a program written in a general purpose language)

Language rules:

- ___ 1. matching **begin - end** pairs
- ___ 2. matching types of actual parameters to formal parameters
- ___ 3. checking that identifiers contain no illegal characters
- ___ 4. checking that string constants are properly terminated
- ___ 5. checking if a particular variable has been defined

Problem 2 [4 pts.].

For the first two parts, fill in the blanks.

- a) The _____ set of a nonterminal tells us for what inputs to reduce to that nonterminal in SLR parsing.
- b) Thompson's construction is a method for automatically building a _____ from a _____.
- c) Circle the correct choice in the parentheses.
 - a. **bison** creates a (bottom-up / top-down) parser.

Problem 3 [10 pts.]

Part A [8]. Use subset construction to build a DFA equivalent to the NFA below. Show your work. (Don't forget to do part B below.)

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

Part B [2]. Give a regular expression for the language accepted by your DFA.

Problem 4 [21 pts. total]

Part A [10]. Build the DFA to recognize viable prefixes for the grammar below. It has already been augmented for you. Show your work.

0) $E' \rightarrow E$

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

2) $E \rightarrow E = E$

Part B [4]. If we attempt to build an SLR parser with the grammar given above, we'll get a conflict. Show exactly where and how this conflict occurs (i.e., in terms of states from part A, inputs, and the relevant entries in the action table). Do not build the whole **action** or **goto** table. Show your work.

Problem 4 (continued)

Part C [5]. The grammar given in part A is ambiguous (and that's why we got the conflict). Show that it is ambiguous. The grammar is given again below.

0) $E' \rightarrow E$

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

2) $E \rightarrow E = E$

Part D [2]. The = (assignment) operator is normally right-associative. To get this interpretation which choice should the parser make where the conflict occurs (i.e., the conflict you got in part B):

Problem 5 [10 pts.]

Consider the grammar at the bottom of the page for arithmetic expressions. Write semantic rules to construct a version of the original expression, but that is the reverse of the original string. In the reverse string all the tokens will be in reverse order, but the characters that made up the token will not be reversed (i.e., names of identifiers don't change). Use a semantic attribute called `rev` for this value; `rev` will be type STL string. In case you are unfamiliar with the `string` class, here are some examples of what you can do with it:

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

Here are some examples of `E.rev`:

<u>expression parsed</u>	<u>E.rev</u>
a	a
a + b	b + a
a + b * c	c * b + a
(a + b) * (c + d)	(d + c) * (b + a)

Note: the subscripts below are only to distinguish different instances of the same nonterminal in a single production.

$E \rightarrow E_1 + T \quad \{ E.rev =$

| T

$T \rightarrow T_1 * F$

| F

$F \rightarrow id \quad \{ F.rev = id.lexval; \}$

| (E)