

MIDTERM EXAM

- This test contains 7 questions with a total of 75 points
 - 6 15 points
 - 1-5, 7 10 points each
- You have 50 minutes to complete this midterm.
- You may use a one sided 8.5" x 11" sheet of notes. You must turn in your notes with the test.
- You may **not** use your text, or any other reference material.
- You may remove the staple so you can see both parts of the last questions at the same time.
- Do not turn this page or turn over the exam until instructed to do so.

Hints

- Don't spend too much time on short answer questions. If you don't know the answer right away, move on and come back.
- There are some questions that will require more time to think and organize your thoughts. If you find yourself stuck on one of these questions, save it until you have answered all the questions you can answer quickly. If you spend too much time on one question you may get frustrated and that will impair your ability to answer the other questions.
- Don't turn your test in early. If you have time left over, use it to review your answers. Students who turn tests in early often make trivial mistakes that they would catch if they reviewed their answers.
- Students that get the highest grades usually go over their answers several times.
- The space below a question **does not** indicate how long of an answer I am expected. I've given you lots of room so you can show your work.
- Double check that you understand the question before you answer it. If you don't understand the question, ask me.

- 1) (10 points) What needs to be accomplished in the action for the following gpl production. Include checking for errors.

```
variable_declaration:
    simple_type T_ID optional_initializer
    {
        // what goes here? (you don't have to write code, just explain the tasks to be done)
    }
```

- 1) if T_ID already in symbol table → error
- 2) if the type of simple_type and the type of optional_initializer are not compatible → error
- 3) evaluate the optional_initializer expression
- 4) insert T_ID into the symbol table using value in step (3) as the initial value

- 2) (10 points) What language does the following context-free grammar describe. The terminal tokens are 0 and 1.

```
S → S 0 S 1 S
S → S 1 S 0 S
S → ε
```

All strings of 0's and 1's (in any order) where there are an equal number of 0's as there are 1's.

- 3) (10 points) Demonstrate that the grammar from question 2 is ambiguous. Explain your answer.

The following two **left-most** derivations derive the same string. This is sufficient to demonstrate that the language is ambiguous.

$S \Rightarrow S0S1S \Rightarrow S0S1S0S1S \Rightarrow 0S1S0S1S \Rightarrow 01S0S1S \Rightarrow 010S1S \Rightarrow 0101S \Rightarrow 0101$

Note: after the first S is replaced with S 0 S 1, all the other S's are replaced with ε.

$S \Rightarrow S0S1S \Rightarrow 0S1S \Rightarrow 01S \Rightarrow 01S0S1S \Rightarrow 010S1S \Rightarrow 0101S \Rightarrow 0101$

4) (10 points) Consider a list of comma separated values such as those passed to a function call. For example:

f()
f(a)
f(a,a)
f(a,a,a)
...

Assuming all parameters are “a”, create a context free grammar that describes the parameters for a function call. That is, write a grammar that describes the language containing the strings ϵ “a” “a,a” “a,a,a” “a,a,a,a” ... Use “a” and “COMMA” as your terminal tokens. I will assume the first non-terminal token is the start symbol. Hint: this is easier if you think of ϵ as a special case to be handled separately.

The following context free grammar describes this language.

$parameters \rightarrow param_list \mid \epsilon$
 $param_list \rightarrow param_list \text{ COMMA } a \mid a$

5) (10 points) Left factor in the following grammar. The terminal tokens are id, (, and).

$S \rightarrow id \mid id() \mid id(id)$

There are two steps to removing the left factors in this grammar. The first step is to remove “id” from the right hand sides of all three S productions:

$S \rightarrow id S'$
 $S' \rightarrow \epsilon \mid () \mid (id)$

There is still a common factor “(“ that starts the RHS of two of the S' productions. Factor out this common string:

$S \rightarrow id S'$
 $S' \rightarrow \epsilon \mid (S''$
 $S'' \rightarrow) \mid id)$

6) (15 points) Compute the specified First and Follow sets for the following grammar, then construct a LL(1) parse table for the grammar. Put the sets in the First and Follow columns of the table. When filling in the table use the production number instead of writing out the production--for example, instead of writing "S → u B D z" in the table write **1**. The symbols u v w x y z are all terminal tokens.

- (1) S → u B D z
- (2) B → v B
- (3) B → w
- (4) D → E F
- (5) D → ε
- (6) E → x
- (7) E → ε
- (8) F → y

	First	Follow		u	v	w	x	y	z	\$
S	u	\$	S	1						
B	v w	x y z	B		2	3				
D	x ε	z	D				4	4	5	
E	x ε	y	E				6	7		
F	y	z	F					8		

7) (10 points) Use the above parse table to parse the given input string. This string may or may not be a legal string in the language. There are enough lines in this table to either parse the string or to determine it is an error. Put either the production or the production number in the action column. It will probably help prevent errors if you put the entire production.

Stack	Input	Action
\$S	u v w x y z \$	
\$ z D B u	u v w x y z \$	(1) $S \rightarrow u B D z$
\$ z D B	v w x y z \$	match u
\$ z D B v	v w x y z \$	(2) $B \rightarrow v B$
\$ z D B	w x y z \$	match v
\$ z D w	w x y z \$	(3) $B \rightarrow w$
\$ z D	x y z \$	match w
\$ z F E	x y z \$	(4) $D \rightarrow E F$
\$ z F x	x y z \$	(6) $E \rightarrow x$
\$ z F	y z \$	match x
\$ z y	y z \$	(8) $F \rightarrow y$
\$ z	z \$	match y
\$	\$	match z