

MIDTERM EXAM

- This test contains 7 questions with a total of 65 points
 - 1-4 5 points each
 - 5-6 10 points each
 - 7 25 points
- 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.
- 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 questions **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) (5 points) How is information about tokens passed from the scanner (gpl.l) to the parser (gpl.y)? For example, the input `int my_variable` produces the tokens `T_INT` and `T_ID`. In `gpl.y` we need the string "my_variable." How is this string passed from the scanner to the parser?

A global variable is used to return the values of tokens (e.g. 42 for an integer, "hello" for a string, 3.145 for a double). This variable is a union of all possible types of the values of tokens.

Members of the union are specified in the `.y` file and bison generates the union's declaration and puts it in `y.tab.h`.

The type of the token's value is specified in the `.y` file in the token statement:

```
%token <union_string> T_ID
```

Back in the `.y` file, the values in the union are retrieved using the `$1, $2, ... , $n` syntax:

```
symbol_type T_ID optional_initializer
{
    the value of T_ID is $2
}
```

- 2) (5 points) Briefly name and describe the three types of analysis performed by a compiler. If you don't remember names, put down what happens at each phase.

Linear/Lexical: group input characters into tokens

Hierarchical/Syntactic: is the input in the language (also called parsing)

Semantic: is the input meaningful, for example, type checking

- 3) (5 points) Draw a diagram of the data structure your program would create for the expression "2 + numbers[i]" in the following input. Your diagram will probably include the classes Expression, and Variable.

```
int i = 42;
int numbers[100];
int k = 2 + numbers[i];
```

4) (5 points) How does one demonstrate that a grammar is ambiguous. Make sure your answer is complete.

If there are two leftmost (or two rightmost) derivations of a string in the language described by the grammar, then the grammar is ambiguous.

5) (10 points) Describe the language generated by the following grammar.

G: $S \rightarrow + S * | + | *$

All strings with n +'s followed by $n-1$ *'s or $n+1$ *'s. $n \geq 0$.

When $n = 0$, there must be 1 * (you can't have -1 *'s).

Examples:

+ ++* +++** +++++*** ++++++****
* +** ++*** +++**** +++++*****

6) (10 points) Eliminate the left recursion in the following grammar.

$S \rightarrow Sab | SaS | X$
 $X \rightarrow Xc | a | b$

$S \rightarrow XS'$
 $S' \rightarrow abS' | aSS' | \text{epsilon}$
 $X \rightarrow aX' | bX'$
 $X' \rightarrow cX' | \text{epsilon}$

7) (25 points) Compute the specified First and Follow sets for the following grammar, then construct a LL(1) parse table for the grammar. Use the parse table to parse the input string $aebb\$$. This string may or may not be a legal string in the language.

NOTE: I made a typo in the grammar. E was suppose to go to ϵ . The first answer I posted assumed this rule was part of the grammar.

$$S \rightarrow aS \mid Ab$$

$$A \rightarrow CDE \mid \epsilon$$

$$C \rightarrow cS \mid \epsilon$$

$$D \rightarrow dS \mid \epsilon$$

$$E \rightarrow eS$$

$$\text{First}(A) = \{c, d, e, \epsilon\}$$

$$\text{First}(C) = \{c, \epsilon\}$$

$$\text{First}(D) = \{d, \epsilon\}$$

$$\text{First}(aS) = \{a\}$$

$$\text{First}(Ab) = \{c, d, e, b\}$$

$$\text{First}(CDE) = \{c, d, e\}$$

$$\text{First}(cS) = \{c\}$$

$$\text{First}(dS) = \{d\}$$

$$\text{First}(eS) = \{e\}$$

$$\text{Follow}(A) = \{b\}$$

$$\text{Follow}(E) = \{b\}$$

$$\text{Follow}(D) = \{e\}$$

$$\text{Follow}(C) = \{d, e\}$$

$$\text{Follow}(S) = \{\$, d, e, b\}$$

	a	b	c	d	e	\$
S	$S \rightarrow aS$	$S \rightarrow Ab$	$S \rightarrow Ab$	$S \rightarrow Ab$	$S \rightarrow Ab$	
A		$A \rightarrow \epsilon$	$A \rightarrow CDE$	$A \rightarrow CDE$	$A \rightarrow CDE$	
C			$C \rightarrow cS$	$C \rightarrow \epsilon$	$C \rightarrow \epsilon$	
D				$D \rightarrow dS$	$D \rightarrow \epsilon$	
E					$E \rightarrow eS$	

Stack	Input	Action
\$S	a e b b \$	
\$Sa	a e b b \$	S --> aS
\$S	e b b \$	Match a
\$bA	e b b \$	S --> Ab
\$bEDC	e b b \$	A --> CDE
\$bED	e b b \$	C --> ϵ
\$bE	e b b \$	D --> ϵ
\$bSe	e b b \$	E --> eS
\$bS	b b \$	Match e
\$bbA	b b \$	S --> Ab
\$bb	b b \$	A --> ϵ
\$b	b \$	Match b
\$	\$	Match b