

### MIDTERM EXAM

- This test contains 7 questions with a total of 70 points
  - 1-3 5 points each
  - 5-7 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.
- 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) (5 points) In bison script, what do \$\$, \$1, \$2, etc, do? Give an example of how they are used in expressions.

These variables are used to pass values (called synthesized attributes) up the parse tree. Specifically, in an action for a production, a value can be passed up the parse tree so it can be used in an action above it in the parse (that is, an action associated with a production that comes before the current one in the derivation). This is illustrated by the following parse:

```
expression:
  expression T_PLUS expression
  {
    $$ = new Expression($1, T_PLUS, $3);
  }
```

A new *Expression object* is synthesized by this action. It is passed up the tree by assigning it to \$\$.

\$1 contains the Expression object synthesized by expanding the first expression in the RHS of this production (that is there was a \$\$ = in its actions). \$2 is the Expression object synthesized by the second expression.

2) (5 points) What does *%token* do in a bison script? Include an explanation of type. An example can help organize your answer.

*%token* declares a new terminal token that can be used on the RHS of any production (and grammar rule) and can be recognized by the scanner:

```
%token T_PLUS
```

In the generated parser (*y.tab.c* and *y.tab.h*) a new enumerated value is create for T\_PLUS. If a type is specified:

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

the token will have a value of this type associated with it (kept in a global union variable) that can be accessed in a production's action using a *\$n* variable:

```
variable_declaration:  
    simple_type T_ID optional_initializer  
    {  
        $2 is a <union_string> containing the value associated with T_ID  
    }
```

3) (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. In your project (thus far) what has been the most difficult aspect of each of these?

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

Creating the regular expressions for the lexical analysis was the hardest.

Specifying the precedence was the hardest part of the syntactic analysis.

Checking the validity of expressions was the hardest part of semantic analysis.

4) (10 points) Demonstrate that the following grammar is ambiguous. Explain your answer.

$S \rightarrow ABc$   
 $A \rightarrow aA \mid b \mid \epsilon$   
 $B \rightarrow bB \mid \epsilon$

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

$S \Rightarrow ABc \Rightarrow aABc \Rightarrow abBc \Rightarrow abc$

$S \Rightarrow ABc \Rightarrow aABc \Rightarrow aBc \Rightarrow abc$

The ambiguity arises from the fact that strings derived from  $A$  can end in a  $b$  and strings derived from  $B$  can start with a  $b$ . When a  $b$  is in the input it could be derived from either  $A$  or  $B$ .

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

G:  $S \rightarrow \langle S \rangle \mid \langle A \rangle$   
 $A \rightarrow ( A ) \mid B$   
 $B \rightarrow aBc \mid b$

Strings of the form  $\langle^n ({}^m a^l b c^l)^m \rangle^n$  where  $n \geq 1, m \geq 0, l \geq 0$

In words, all strings that start with the same number of  $\langle$  as the number of  $\rangle$  at the end. This number is 1 or more. In the middle the string has the same number of  $($  and  $)$ . This number is 0 or more. Inside the innermost  $( )$  is the string  $b$  or  $abc$  or  $aabcc$  or  $aaabccc$ , etc.

Examples:  $\langle b \rangle$   $\langle (b) \rangle$   $\langle (abc) \rangle$   $\langle abc \rangle$   $\langle \langle b \rangle \rangle$   $\langle \langle \langle b \rangle \rangle \rangle$   $\langle (((aabcc))) \rangle$

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

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow Aa \mid \epsilon \\ B &\rightarrow Bb \mid Sc \mid \epsilon \end{aligned}$$

Step 1: Remove the non-immediate left recursion

Note: The labeling of non-terminals is arbitrary. That means you can rearrange the productions before labeling. This is much easier if you swap the second and third productions.

$$\begin{aligned} S &\rightarrow A \mid B \\ B &\rightarrow Bb \mid Ac \mid Bc \mid \epsilon \quad \text{The "Sc" was replaced with "Ac | Bc"} \\ A &\rightarrow Aa \mid \epsilon \end{aligned}$$

Step 2: Remove the immediate left recursion in the second production

$$\begin{aligned} B &\rightarrow Bb \mid Ac \mid Bc \mid \epsilon \quad \alpha_1 = b \quad \alpha_2 = c \quad \beta_1 = Ac \quad \beta_2 = \epsilon \\ B &\rightarrow AcB' \mid B' \\ B' &\rightarrow bB' \mid cB' \mid \epsilon \end{aligned}$$

Step 3: Remove the immediate left recursion in the third production

$$\begin{aligned} A &\rightarrow Aa \mid \epsilon \quad \alpha_1 = a \quad \beta_1 = \epsilon \\ A &\rightarrow A' \\ A' &\rightarrow aA' \mid \epsilon \end{aligned}$$

These two are the same as:

$$A \rightarrow aA \mid \epsilon$$

Putting all this together, the new grammar is:

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aA \mid \epsilon \\ B &\rightarrow AcB' \mid B' \\ B' &\rightarrow bB' \mid cB' \mid \epsilon \end{aligned}$$

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 your parse table to parse the given input string. This string may or may not be a legal string in the language.

$$S \rightarrow AbB \mid BeA$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow cC \mid \epsilon$$

$$C \rightarrow B \mid d$$

$$\text{First}(S) = \{a, b, c, e\}$$

$$\text{First}(A) = \{a, \epsilon\}$$

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

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

$$\text{First}(AbB) = \{a, b\}$$

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

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

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

$$\text{Follow}(S) = \{\$\}$$

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

$$\text{Follow}(B) = \{e, \$\}$$

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

	a	b	c	d	e	\$
S	$S \rightarrow AbB$	$S \rightarrow AbB$	$S \rightarrow BeA$		$S \rightarrow BeA$	
A	$A \rightarrow aA$	$A \rightarrow \epsilon$				$A \rightarrow \epsilon$
B			$B \rightarrow cC$		$B \rightarrow \epsilon$	$B \rightarrow \epsilon$
C			$C \rightarrow B$	$C \rightarrow d$	$C \rightarrow B$	$C \rightarrow B$

Stack	Input	Action
\$S	abccde \$	
\$BbA	abccde \$	S --> AbB
\$BbAa	abccde \$	A --> aA
\$BbA	bccde \$	Match a
\$Bb	bccde \$	A --> ε
\$B	ccde \$	Match b
\$Cc	ccde \$	B --> cC
\$C	cde \$	Match c
\$B	cde \$	C --> B
\$Cc	cde \$	B --> cC
\$C	de \$	Match c
\$d	de \$	C --> d
\$	e \$	Match d
		error