CSC 460/660, Midterm
March 22, 2006

Name: ____________________________________________
Student ID: ________________________________

Signature: ________________________________________

• You have 75 minutes to write the 9 questions on this examination. A total of 100 marks is available.

• **Justify all of your answers**

• You may use **one** sheet of handwritten notes.

• Keep your answers short. If you run out of space for a question, you have written too much.

• The number in square brackets to the right of the question number indicates the number of marks allocated for that question. Use these to help you determine how much time you should spend on each question.

• Use the back of the page for rough work.

• **Good luck**

**UNIVERSITY REGULATIONS:**

• No candidate shall be permitted to enter the examination room after the expiration of one half hour, or to leave during the first half hour of the examination.

• **CAUTION:** Candidates guilty of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action.

  1. Making use of any books, papers or memoranda, electronic equipment, or other memory aid devices, other than those authorized by the examiners.

  2. Speaking or communicating with other candidates.

  3. Purposely exposing written papers to the view of other candidates. The plea of accident or forgetfulness shall not be received.
First & Follow Sets

Question 1 [16 points]
Consider the following grammar:

\[
\begin{align*}
A & \rightarrow C q \mid B m \\
B & \rightarrow C \mid x A \mid \epsilon \\
C & \rightarrow w \mid z \mid y B w
\end{align*}
\]

• [6/16] Compute \(FIRST(A), FIRST(B), FIRST(C)\).

• [6/16] Compute \(FOLLOW(A), FOLLOW(B), FOLLOW(C)\).

(Question 1 continued on next page)
First & Follow Sets

• Recall that a grammar is LL(1) if it can be parsed by a predictive parser (e.g., a recursive descent) with one look ahead. Why is this grammar not LL(1)?
Scanners

**Question 2** [13 points]
Consider the following (slightly altered) Espresso grammar snippet (terminals are in boldface):

- `switchStatement ::= switch ( id ) switchBlock`
- `switchBlock ::= \{ switchBlockStatementGroupsOpt \}`
- `switchBlockStatementGroupsOpt ::= switchBlockStatementGroups | \epsilon`
- `switchBlockStatementGroups ::= switchBlockStatementGroups switchBlockStatementGroup`
- `switchBlockStatementGroup ::= switchLabelBlockStatements`
- `switchLabels ::= switchLabel | switchLabels switchLabel`
- `switchLabel ::= case integer : | default :`

- [3/13] Under the assumption that identifiers (id) consist of only the letters `a,c,d,e,f,h,i,l,s,t,w`, and `integer` being a non-empty sequence of digits (it is ok to start with 0, don’t worry about that), give the alphabet that your scanner will be operating with, i.e., specify the correct value for \( \Sigma \).

(Question 2 continued on next page)
Scanners

Assume that you have the following definitions available (These are like the constants defined in sym.java):

```java
int NUMBER = ...;
int LBRACE = ...;
int SWITCH = ...;
int RBRACE = ...;
int CASE = ...;
int LPAREN = ...;
int DEFAULT = ...;
int RPAREN = ...;
int COLON = ...;
int ID = ...;
```

as well as the following constructor for a Token class, boolean function and int function:

```java
Token(int kind) { ... } // Creates a new token
boolean isKeyword(String lex) { ... } // Returns true if lex is a keyword
int lexeme2KeywordConstant(String lex) { ... } // Return the corresponding token kind
```

The last of the function returns the appropriate constant if given a string that represents a valid keyword (i.e. the constant to be passed to Token).

- [10/13] Draw the scanner (not parser) diagram. Mark the accepting states with a double circle, and provide the correct call to Token() and mark pushback states with a *. (You can assume that the current lexeme is available in the global variable lexeme).

(End of question 2)
Question 3 [12 points]

Consider the following grammar:

\[
\begin{align*}
A & \rightarrow A \alpha \mid A \beta \mid A C C \mid \delta \\
B & \rightarrow \alpha B \mid \alpha \gamma B \mid \delta \\
C & \rightarrow C \gamma \mid \omega \mid B
\end{align*}
\]

- [6/12] Eliminate the left recursion in the grammar.

- [6/12] Eliminate the common sub-expressions from the result you got from removing the left recursion.

(End of question 3)
Question 4 [15 points]
Consider the following context-free grammar:

\[
\begin{align*}
A & \rightarrow \text{id} = E \mid \text{id} = A \\
E & \rightarrow T + E \mid T \\
T & \rightarrow \text{id} \mid \text{num} \mid f(E)
\end{align*}
\]

• [4/15] Give a **left-most** derivation of the string \(a = b = c + f(a + 3)\).

• [4/15] Give a **right-most** derivation of the string \(a = b = c + f(a + 3)\).

(Question 4 continued on next page)
Derivations & Parse Trees


• [3/15] Is this grammar ambiguous? Why/Why not?

(End of question 4)
LL(1) or not?

Question 5 [8 points]
Consider a general grammar $G$:

$$G_i \rightarrow \alpha_{i_1} \mid \alpha_{i_2} \mid \ldots \mid \alpha_{i_n}, \quad i = 0, \ldots, m$$

where

$$\forall i : 0 \leq i \leq m : \bigcap_{j=1}^{n_i} \text{FIRST}(\alpha_{i_j}) = \emptyset.$$ 

Is $G$ always LL(1), that is, can you always write a predictive parser with a look-ahead of one and with no back-tracking for $G$ if the above equation is satisfied? Explain.

(End of question 5)
Parsing

**Question 6** [15 points]
Recall this illustration of a predictive non-recursive parser from the textbook.

Let $M[A, a]$ be the parsing table indexed by a non-terminal $A$ and a terminal $a$.

When parsing we can be in one of the following 3 situations ($X$ is the symbol on the top of the stack):

1. $X = a = \$$.  
2. $X = a \neq \$$.  
3. $X$ is a non-terminal.

• [6/15] Write down the action that the parser must take for each of the 3 cases above. You may assume that $M[X, a] = \{X \rightarrow UVW\}$.  

(Question 6 continued on next page)
Parsing

Assume we have the following parsing table for a grammar G.

<table>
<thead>
<tr>
<th></th>
<th>Input (terminals)</th>
<th>id</th>
<th>+</th>
<th>*</th>
<th>(</th>
<th>)</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>$E \rightarrow T E'$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E'$</td>
<td>$E' \rightarrow + T E'$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T$</td>
<td>$T \rightarrow F T'$</td>
<td>$T' \rightarrow \epsilon$</td>
<td></td>
<td>$T' \rightarrow * F T'$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T'$</td>
<td>$F \rightarrow id$</td>
<td></td>
<td></td>
<td></td>
<td>$F \rightarrow ( E )$</td>
<td>$E' \rightarrow \epsilon$</td>
<td>$E' \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T' \rightarrow * F T'$</td>
<td>$T' \rightarrow \epsilon$</td>
<td>$T' \rightarrow \epsilon$</td>
<td>$T' \rightarrow \epsilon$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• [9/15] Show the parsing of $(id + id) \ast id$ in the table below. The output column shows which rule you used, e.g. $E \rightarrow + T E'$.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>$(id + id) \ast id$</td>
<td>$E \rightarrow T E'$</td>
</tr>
<tr>
<td>$E' T$</td>
<td>$(id + id) \ast id$</td>
<td>$E \rightarrow T E'$</td>
</tr>
</tbody>
</table>

(End of question 6)
Symbol Tables

**Question 7** [8 points]
Consider the following program:

```java
public class foo {
    public foo foo;
    public foo(foo foo) {
        foo.foo = foo;
    }
}
```

This is a valid Espresso program. Explain what in the Espresso symbol table structure allows this to work, that is, classes, fields and parameters/locals all having the same name without name conflicts during name resolution.

(End of question 7)
Espresso

**Question 8** [5 points]
Give at least one good reason to have all parse tree nodes in Espresso inherit from the same super-class (AST)?

(End of question 8)
Derivations & Ambiguities

Question 9 [8 points]
Let $G$ be a non-ambiguous context-free grammar and let

$$S \Rightarrow^*_L \omega$$
be a left-most derivation of $\omega$, and

$$S \Rightarrow^*_R \omega$$
be a right-most derivation of $\omega$.

• [4/8] Why is the lengths of the derivations always the same?

Now, let $G'$ be an ambiguous context-free grammar.

• [4/8] Is the length of $S \Rightarrow^*_L \alpha$ always the same as $S \Rightarrow^*_R \alpha$ for all $\alpha \in L(G')$?

(End of question 9)
(End of the exam)