**LL Parsing Example**
**Martha Kosa**

A parser is a critical part of a compiler for a programming language. Parsing must be done efficiently. The parser must be able to determine quickly the applicable production rule or if no production rule is applicable at all (which means the program is not grammatically correct). It is simpler if the leftmost nonterminal symbol is replaced at each step in the parse. How can we determine which production rule to apply? We need some terminal lookahead symbols. In an LL(1) grammar, only one lookahead symbol is needed. The production rules can be stored in a parse table with rows corresponding to the current nonterminal symbol and columns corresponding to the current input symbol. Each entry of the parse table contains at most one rule; accessing an empty entry in the table indicates an error situation. Using the production rules, the FIRST and FOLLOW sets are computed to fill this table. The applicable lookahead symbols are in the FIRST set for the current nonterminal symbol. If a lambda rule is applied, the applicable lookahead symbols are in the FOLLOW set for the current nonterminal symbol. $ is a special terminal symbol to mark the end of the input string. A stack is used to keep track of the current sentential form, one symbol at a time. If the top of the stack is a terminal symbol and the current input symbol does not match it, an error is detected. If no error is detected, the stack is popped, and the current input symbol is consumed. If the top of the stack is a nonterminal symbol and no production rule appears in the parse table entry corresponding to that top of the stack as the row identifier and the current input symbol as the column identifier, an error is detected. If no error is detected, the stack is popped, and the reverse of the righthand side of the corresponding production rule is pushed on the stack, one symbol at a time. The righthand side is reversed because of the LIFO property of the stack. If the entire input string has been consumed and the stack is empty, the string is accepted. Otherwise, the string is rejected.

A grammar for a typical complex programming language such as C++ or Java has hundreds of rules. Our example will be a smaller one. Let us consider a language with an arbitrary number of simple assignment statements, separated by semicolons. We will assume that our assignment statements begin with the keyword **int,** followed by a variable name (for simplicity, assume variable names begin with an arbitrary number of a's and/or b's followed by a c), followed by an equals sign, and finally followed by a number (for simplicity, assume our possible digits are 0, 1, and 2).

For your convenience, this grammar has been created for you.

***Try It!***

1. Open the grammar file **SimpleDeclarationAndAssignment.CFG.flap**.
2. Select ***Input > Build LL(1) Parse Table***. You should see a window similar to the following.

1. Fill in the values for the FIRST sets for each of the nonterminal symbols. If the

nonterminal symbol appears as the lefthand side of a lambda rule, lambda will be in its first set. Do not type any commas! You can check your work as you go along by clicking the **Next** button after you fill in a set. If you have already practiced a lot with FIRST sets, you can click the **Do Step** button.

2. Fill in the values for the FOLLOW sets for each of the nonterminal symbols. Do not type any commas! Don't forget about the $! You can check your work in the same way as you did previously.
3. Fill in the entries in the parse table. Enter only the righthand sides of the production rules!
4. You can check your work in the same way as you did previously. Your window should look like the following.

1. Click the **Parse** button. Resize the window and/or subwindows if necessary to see the entire parse table. Your window should look similar to the following.

**1.** **intc=0;** is one of the shortest valid strings. How many other valid strings are the same length? What are those strings? Enter **intc=0;** in the input box, and then click the **Start** button. Your window should look similar to the following.

1. Click the **Step** button. Your window should look similar to the following (after resizing if necessary).

1. Click on the one applicable righthand side from the parse table. What row and column are you using? Click the **Step** button the correct number of times to put the entire righthand side on the stack. The top of the stack is the leftmost symbol.
2. Keep selecting the one applicable righthand side from the parse table and clicking the **Step** button until a terminal symbol is on top of the stack. Resize the window to see the current parse tree. Your window should look similar to the following.

1. Click the **Step** button until a nonterminal symbol is on top of the stack.
2. Click on the one appropriate righthand side from the parse table and click the **Step** button until the entire righthand side has been pushed on the stack.
3. Repeat the previous two steps until **S** is on top of the stack.
4. Click on the one appropriate righthand side from the parse table and click the **Step** button twice. What does the stack have on it now?
5. Click the **Step** button. Your window should look similar to the following.

1. You have successfully parsed a string. Select **Derivation Table** from the combo box to see all the rules applied with the corresponding sentential forms. Your window should look similar to the following.

Now let's see what happens when we attempt to parse an invalid string. All variable names in the language corresponding to our grammar must end with a c. What happens if we forget that?

***Try It!***

1. If it is not already open, open the file **SimpleDeclarationAndAssignment.CFG.flap**.
2. If the LL(1) parse table is not already built, build it as before. You can click the **Do All!** button and then the **Parse** button to save time.
3. Enter **intab=0;** in the input box, and then click the **Start** button.
4. As you did in the previous section, when a nonterminal symbol is on top of the stack, click on the appropriate righthand side and then click on the **Step** button enough times to push the entire righthand side on the stack. When a terminal symbol is on top of the stack, click on the **Step** button enough times to match terminal symbols until a nonterminal symbol is on top of the stack. You will be able to apply 5 production rules before you get stuck. As ou click, notice the output label at the very bottom of your window. Your window should look similar to the following (with possible resizing).

1. What is in the parse table entry with row corresponding to the top stack symbol and column corresponding to the current input symbol? Click the parse table entry, and then the **Step** button. Your window should look similar to the following.

1. Click on the **Step** button. You should get the error message **String rejected.** at the bottom of your window.

The parser stops when the first error is detected. For more practice, attempt to parse some other invalid strings, such as strings without **int**, strings without semicolons separating assignments, etc.