Assignment-Chapter 4-Parser

Version: Refer to the version in the file name

Contents

1 Problem to Solve

Consider the following grammar, which is the same as the one given on page 175 of the textbook with the exception of the first rule. The first rule, <assign> is added for assignment statements. <assign> will be the starting symbol of your grammar.

<assign> → id = <expr>
<expr> → <term> {(+ | -) <term>}
<term> → <factor> {(* | /) <factor>}
<factor> → id | int_constant | (<expr>)

The respective parser program modules for all the rules except for the assignment statement are given on pages 175-178 of the textbook. You will have to add a module for the assignment statement.

Convert the recursive-descent parser program given on pages 175-178 of the textbook to Java and modify it to parse assignment statements.

Your program must be well written and formatted.

2 Optional Lexical Analyzer

You may use the supplied lexical analyzer called LexicalAnalyzer.java, if you wish to do so. front_end.txt is the input file for this lexical analyzer only, not for the parser.

3 Naming of Program Modules

Use the following names for your program modules:

Module	Name
Java Project	ParserProject
Package	parserPackage
Class	Parse
Input file	statements.txt

4 Input

1. Test the parser using the supplied text file, which is called statements.txt. You must use the file as it is given without modifying it.
2. Store the input file in the same location as that of your program file.
3. In your program, open the input file using the path "src\\parserPackage\\statements.txt"

5 Output

1. Separate the parses of the statements from each other using a line of asterisks (*).
2. Precede the result of each parse by the text of the corresponding statement.
3. The output must be well formatted and readable.
4. To keep the format of the output intact when you insert it in the D2L comment section, you may use the following steps in Eclipse: (Thanks to Anna Blasingame for providing the instructions)
   a. Run-> Run Configurations -> Common -> Output file (check it) -> File System (specify where you want to store it and the name you want for the text file).
   b. Open the file using the Microsoft Word.
   c. Copy the contents and paste them in the D2L comment section.

5. The output of your parser ought to be similar to the sample output below.

```
****************************************************
Parsing the statement: sumTotal = (sum + 47    ) / total

Next token is: IDENT          Next lexeme is sumTotal
Enter <assign>
Next token is: ASSIGN_OP      Next lexeme is =
Next token is: LEFT_PAREN     Next lexeme is (
Enter <expr>
Enter <term>
Enter <factor>
Next token is: IDENT          Next lexeme is sum
Enter <expr>
Enter <term>
Enter <factor>
Next token is: ADD_OP         Next lexeme is +
Exit <factor>
Exit <term>
Next token is: INT_LIT        Next lexeme is 47
Enter <term>
Enter <factor>
Next token is: RIGHT_PAREN    Next lexeme is )
Exit <factor>
Exit <term>
Exit <expr>
Next token is: DIV_OP         Next lexeme is /
Exit <factor>
Next token is: IDENT          Next lexeme is total
Enter <factor>
Exit <factor>
Exit <term>
Exit <expr>
Exit <assign>
```

Note: after all the statements are parsed and the end of file is detected, the following line is to be printed:

```
Next token is: END_OF_FILE    Next lexeme is EOF
```

6 What to turn in

a. Attach your java program, Parse.java

b. Attach your input file, statements.txt

c. Include the result of the execution of your program in the comment section of D2L, as described in the Output section above.

Please do not zip your files.