

Assignment 2

Implement a parser for the language specified by the following context-free grammar.

For readability, terminal symbols are red, and in most cases are given by their text values, not the Token kind. For example, the grammar uses **boolean** rather than KW_boolean, and **(** instead of LPAREN. Your parser should use KW_boolean and LPAREN.

Program ::= **Identifier** Block

Block ::= { ((Declaration | Statement) ;) * }

Declaration ::= Type **IDENTIFIER** | **image** **IDENTIFIER** [Expression , Expression]

Type ::= **int** | **float** | **boolean** | **image** | **filename**

Statement ::= StatementInput | StatementWrite | StatementAssignment
| StatementWhile | StatementIf | StatementShow | StatementSleep

StatementInput ::= **input** **IDENTIFIER** from @ Expression

StatementWrite ::= **write** **IDENTIFIER** to **IDENTIFIER**

StatementAssignment ::= LHS **:=** Expression

StatementWhile ::= **while** (Expression) Block

StatementIf ::= **if** (Expression) Block

StatementShow ::= **show** Expression

StatementSleep ::= **sleep** Expression

LHS ::= **IDENTIFIER** | **IDENTIFIER** PixelSelector | Color (**IDENTIFIER** PixelSelector)

Color ::= **red** | **green** | **blue** | **alpha**

PixelSelector ::= [Expression , Expression]

Expression ::= OrExpression ? Expression : Expression
| OrExpression

OrExpression ::= AndExpression (| AndExpression) *

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

AndExpression ::= EqExpression (& EqExpression)
 EqExpression ::= RelExpression ((= | ! =) RelExpression)
 RelExpression ::= AddExpression ((< | > | < = | > =) AddExpression)
 AddExpression ::= MultExpression ((+ | -) MultExpression)
 MultExpression ::= PowerExpression ((* | / | %) PowerExpression)
 PowerExpression ::= UnaryExpression (* PowerExpression | ε)
 UnaryExpression ::= + UnaryExpression | - UnaryExpression | UnaryExpressionNotPlusMinus
 UnaryExpressionNotPlusMinus ::= ! UnaryExpression | Primary
 Primary ::= INTEGER_LITERAL | BOOLEAN_LITERAL | FLOAT_LITERAL |
 (Expression) | FunctionApplication | IDENTIFIER | PixelExpression |
 PredefinedName | PixelConstructor

PixelConstructor ::= << Expression , Expression , Expression , Expression >>
 PixelExpression ::= IDENTIFIER PixelSelector
 FunctionApplication ::= FunctionName [Expression , [FunctionName [Expression , Expression]
 FunctionName ::= sin | cos | atan | abs | log | cart_x | cart_y | polar_a | polar_r
 int | float | width | height | color
 PredefinedName ::= Z | default_height | default_width

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Use the provided SimpleParser.java and SimpleParserTest.java as a starting point. Your Scanner.java from Assignment 1 (with any errors corrected) will also be needed. As given, the code should compile, and two of the test cases should pass. All three should all pass when your parser has been completely and correctly implemented. Of course, you will need to create many more Junit tests. The SimpleParser will simply determine whether a given sentence is legal or not. If not, a `SyntaxException` should be thrown. If the sentence is legal, the parse method should simply return.

Use the approach described in the lectures to systematically build the parser. If the grammar is not LL(1), you may need to transform it, or use an ad hoc solution.

Turn in a jar file containing your source code for Parser.java, Scanner.java, and ParserTest.java. Your ParserTest will not be graded, but may be looked at in case of academic honesty issues.

We will subject your parser to our set of unit tests and your grade will be determined solely by how many tests are passed. Name your jar file in the following format:

firstname_lastname_ufid_hw2.jar

Additional requirements:

This code must remain in package `cop5556sp18` (case sensitive): do not create additional packages. Names (of classes, method, variables, etc.) in starter code must not be changed. Unless otherwise specified, your code should not import any classes other than those from the standard Java distribution or your Scanner.java.

All code, including the Scanner code must be your own work. Using someone else's Scanner code is not permitted.

Submission Checklist

See the checklist from Assignment 1.

Comments and suggestions:

Work incrementally. Note that you can call the routines corresponding to fragments of the grammar in Junit tests. For example, you should have a method name `Expression`, and you could call that directly, passing in an input that satisfies that grammar fragment. It is useful when working incrementally to ensure that you are not calling an unimplemented procedure without realizing it. To this end, the sample code throws an `UnsupportedOperationException` in every method that is needed for compilation, but has not yet been implemented. In your completed parser, this exception should never be thrown.

You will want to provide better error messages than given in the sample code. In particular, you will want to output the location of the offending token.

This assignment is not only about coding, it is also about understanding and working with context-free grammars. Think about what we have discussed in class as you work. Is the grammar ambiguous? Is it LL(1)? If not, what can you do?

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder