COP5556 Assignment 3

Modify your simple parser from Assignment 2 so that it returns an abstract syntax tree. The given abstract syntax is not, strictly speaking, the syntax, rather it indicates what the elements of the concrete syntax correspond to. How to deal with Expressions will be explained in the lecture.

|  |  |  |
| --- | --- | --- |
| Concrete Syntax | Abstract Syntax | Comments |
| Program ::= ( Declaration SEMI | Statement SEMI )\* | Program ::= (Dec | Statements)\* |  |
| Declaration :: = VariableDeclaration | ImageDeclaration | Dec ::= DecVar | DecImage |  |
| VariableDeclaration ::= VarType IDENT ( ASSIGN Expression | ϵ ) | DecVar ::= Type name (Expression | ϵ ) | For ϵ, use ExprEmpty. This is more convenient than using null. |
| VarType ::= KW\_int | KW\_string | Type ::= Int | String | Image | Boolean | Void | Use constants defined in Type enum |
| ImageDeclaration ::= KW\_image (LSQUARE Expression COMMA Expression RSQUARE | ϵ) IDENT (((LARROW | ASSIGN ) Expression) | ϵ ) | DecImage ∷= Type ( Expression0 Expression1 | ϵ) IDENT ( OP Expression2 | ϵ ) | Expression0 = width, Expression1 = height, Expression2 =source, IDENT = name. Add a constant NOP to Scanner.Kind, and use that for ϵ. |
| Statement ::= AssignmentStatement | ImageOutStatement | ImageInStatement | LoopStatement | Statement ∷= StatementAssign | StatementOutFile | StatementOutScreen |StatementImageIn | StatementLoop |  |
| ImageOutStatement ::= IDENT RARROW Expression | StatementOutFile ::= IDENT Expression | IDENT = name, Expression = filename |
| ImageOutStatement ::=IDENT RARROW KW\_SCREEN ( LSQUARE Expression COMMA Expression RSQUARE | ϵ ) | StatementOutScreen ::= IDENT (Expression0 Expression1 | ϵ ) | IDENT = name, Expression0= X, Expression1=Y |
| ImageInStatement ::= IDENT LARROW Expression | StatementImageIn ::= IDENT Expression |  |
| AssignmentStatement ::= IDENT ASSIGN Expression | StatementAssign ::= IDENT Expression |  |
| LoopStatement ∷= IDENT ASSIGN STAR ConstXYSelector COLON (Expression | ϵ ) COLON Expression | StatementLoop ::= IDENT (Expression0 | ϵ ) Expression1 | Expression0= cond, Expression1 = e |
| Expression ::= OrExpression Q Expression COLON Expression | ExprConditional ::= Expression0 Expression1 Expression2 | Expression0 = condition, Expression1 =trueCase, Expression2 = falseCase |
| Expression ::= OrExpression |  |  |
| OrExpression ::= AndExpression ( OR AndExpression)\* | BinaryExpr ::= Expression0 OP Expression1 | Expression0 = e0, Expression1=e1  See lecture for more explanation |
| AndExpression ::= EqExpression ( AND EqExpression )\* |  |  |
| EqExpression ::= RelExpression ( (EQ | NEQ ) RelExpression )\* |  |  |
| RelExpression ::= AddExpression ( ( LT | GT | LE | GE ) AddExpression)\* |  |  |
| AddExpression ::= MultExpression ( (PLUS | MINUS ) MultExpression )\* |  |  |
| MultExpression := UnaryExpression ( ( STAR | DIV | MOD ) UnaryExpression )\* |  |  |
| UnaryExpression ::= (PLUS | MINUS) UnaryExpression | UnaryExpressionNotPlusMinus | ExprUnary ::= OP Expression |  |
| UnaryExpressionNotPlusMinus ::= EXCL UnaryExpression | HashExpression |  |  |
| HashExpression ∷= Primary ( HASH Attribute)\* | ExprHash ::= Expression Attribute |  |
| Primary ::= INTLIT | ExprIntLit |  |
| Primary ::= IDENT | ExprVar |  |
| Primary ::= LPAREN Expression RPAREN |  |  |
| Primary ::= STRINGLIT | ExprStringLit |  |
| Primary ::= KW\_X | KW\_Y | ExprVar |  |
| Primary ::= CONSTANT | ExprConst |  |
| Primary ::= PixelConstructor |  |  |
| Primary ::= ArgExpression |  |  |
| Primary ::= (INTLIT | IDENT | LPAREN Expression RPAREN | STRINGLIT | KW\_X | KW\_Y | CONSTANT |PixelConstructor | ArgExpression ) (PixelSelector | ϵ )  (Now matches previous assignment) | ExprPixelSelector ::= Expression ExpressionX  ExpressionY | Expression = image, ExpressionX = X, ExpressionY =Y |
| PixelConstructor ∷= LPIXEL Expression COMMA Expression COMMA Expression RPIXEL | ExprPixelConstructor ::= Expressionr Expressiong Expressionb |  |
| PixelSelector ∷= LSQUARE Expression COMMA Expression RSQUARE |  | This is subsumed in ExprPixelSelector above |
| Attribute ∷= KW\_WIDTH | KW\_HEIGHT | KW\_RED | KW\_GREEN | KW\_BLUE |  |  |
| ArgExpression ∷= AT Primary | ExprArg |  |
| ConstXYSelector ::= LSQUARE KW\_X COMMA KW\_Y RSQUARE |  |  |

* As before, If an illegal sentence is encountered, your parser should throw a SyntaxException. The token where the error was manifested and an error message are parameters of the SyntaxException constructor. The contents of the message will not be graded, but you will appreciate it later if it is informative.
* If the given input is legal in the language, the parser returns an abstract syntax tree. It is recommended that each method return the most specific type possible. (For example, return Expression, if possible, rather than ASTNode.)
* The package cop5556fa20.AST has been provided. Use these classes for the AST.