FDMiniJava Grammar v1.4 (April 8, 2022)

Program → MainClass ClassDecl*

MainClass → **class** id { **public static int main** (**String** [ ] id) { VarDecl* Statement* } }

ClassDecl → **class** id { VarDecl* MethodDecl* } | **class** id **extends** id { VarDecl* MethodDecl* }

VarDecl → Type id ; | ArrayType id;

MethodDecl → **public** Type id (FormalList) { VarDecl* Statement* } |
public ArrayType id (FormalList) { VarDecl* Statement* }

FormalList → Type id FormalRest* | ε

FormalRest → , Type id

Type → **int** | **bool** | id

ArrayType → **int** [ ] | ArrayType [ ]

Statement → { Statement * } |
if (Exp) Statement **else** Statement |
while (Exp) Statement |
id = Exp ; |
id [ExpList] = Exp ; |
continue; |
break; |
return (Exp); |
putint (Exp) ; |
putch (Exp) ; |
putarray (Exp, Exp) ; |
starttime() ; |
stoptime() ;

Exp → Exp op Exp | Exp [ExpList] | ~~Exp . length~~ | Exp . id (ExpList) |
INTEGER_LITERAL | **true** | **false** | id | **this** | **new int** [ExpList] | **new** id () |
! Exp |- Exp | (Exp) | **getint**() |**getch**() | **getarray** (Exp)

ExpList → Exp ExpRest* | ε

ExpRest → , Exp

The lexical issues are as follows:

Identifiers: An identifier is a sequence of letters, digits, and underscores, starting with a letter. Uppercase letters are distinguished from lowercase. In the above grammar, the symbol id stands for an identifier.

Integer literals: A sequence of decimal digits is an integer constant that denotes the corresponding integer value. In the above grammar, the symbol INTEGER_LITERAL stands for an integer constant.

Binary operators: A binary operator is one of

“&&” (Boolean And), “ll” (Boolean Or), “<” (Less than), “<=” (Less than or equal), “==” (equal), “+” (Plus), “-” (Minus), “*” (Times), “/” (Divide)

The usual precedence rule is used. Unary operators (“!” and “-“) have higher priority than binary ones.

In the above grammar, the symbol op stands for a binary operator.

Comments: A comment may appear between any two tokens. There are two forms of comments: One starts with /*, ends with */, and may be nested; another begins with // and goes to the end of the line.