CFG for Programming Language

Identifiers:

IdentifierList → Identifier [, IdentifierList]* | ϵ
Identifier → (Letter | _) (Letter | _ | Digit)*
Letter → [a-zA-Z]
Digit → [0-9]
NonZeroDigit → [1-9]
Fraction → “.” Digit+

Keywords:

Keyword → include, if, else, for, while, int, char, signed, unsigned, bool, return, void, string

Literals:

Literal → StringLiteral | IntegerLiteral | FloatLiteral
StringLiteral → "<any ASCII character except newline or the quote>"
IntegerLiteral → DecimalInteger
DecimalInteger → NonZeroDigit Digit* | [0]
FloatLiteral → IntegerLiteral Fraction

Operators:

Optr → LogicalOptr | RelationalOptr | ArithmeticOptr | BitwiseOptr | AssignmentOptr
LogicalOptr → and | or | not
RelationalOptr → < | > | == | <= | >= | !=
ArithmeticOptr → + | - | * | / | %
BitwiseOptr → << | >> | & | [|] | ~
AssignmentOptr → =
IndexOptr → [IntegerLiteral]

Expressions:

Atom → Identifier | Literal | FuncCall
ParenthesisExpr → (Expr) [Optr ParenthesisExpr]* | Expr
Expr → UnaryExpr | BinaryExpr
UnaryExpr → Atom | “-” Atom | “+” Atom | “~” Atom | “-” UnaryExpr | “+” UnaryExpr | “~” UnaryExpr | Identifier IndexOptr
BinaryExpr → UnaryExpr | UnaryExpr Optr BinaryExpr

Function Call

FuncCall → Identifier(Atom[, Atom]*)

Data Types

DataType → PrimitiveType | CompoundType
PrimitiveType → bool | char | int | float
CompoundType → PrimitiveType[IntegerLiteral] | PrimitiveType[IntegerLiteral] [IntegerLiteral]

Variables:

VariableList → DataType Identifier [, VariableList]* | ϵ
VarDec → DataType IdentifierList

Comments:

Comment → //<any character except newline>

Statements:

StmtList → SimpleStmt StmtList | CompoundStmt StmtList | ϵ
SimpleStmt → ExprStmt | PrintStmt | ReturnStmt | ContinueStmt | BreakStmt | IncludeStmt | VarDec
ExprStmt → ParenthesisExpr
PrintStmt → print(ParenthesisExpr)
ReturnStmt → return ParenthesisExpr
BreakStmt → break
ContinueStmt → continue
IncludeStmt → #include [StringLiteral, <Identifier>]
CompoundStmt → IfStmt | ForStmt | WhileStmt
IfStmt → if(ParanthesisExpr) Block | if(ParanthesisExpr) Block else Block | if(ParanthesisExpr) Block else IfStmt
ForStmt → for(Identifier, IntegerLiteral, IntegerLiteral [,IntegerLiteral]) Block
WhileStmt → while(ParanthesisExpr) Block

Blocks:

Block → { StmtList }

Functions:

FuncDef → ReturnType Identifier(VariableList) Block
ReturnType → PrimitiveType