BNF for mona.jj

TOKENS

```
<DEFAULT> SKIP : {
" "
| "\t"
| "\n" : {
| "\r"
| "\f"
}



<DEFAULT> SKIP : {
"//" : IN_COMMENT
| <"//" (~["\n","\r"])* ("\n" | "\r" | "\r\n")> : {
}



<IN_COMMENT> SKIP : {
"/#" : {
| "#/" : {
| <~[]>
}
```

```
<DEFAULT> TOKEN [IGNORE_CASE] : {
<VAR: "var">
| <CONST: "const">
| <RETURN: "return">
| <CLASS: "class">
| <INT: "int">
| <BOOL: "bool">
| <T_FLOAT: "float">
| <VOID: "void">
| <MAIN: "main">
| <BREAK: "break">
| <IF: "if">
| <ELSE: "else">
| <ELSEIF: "else_if">
| <PRINT: "print">
| <IN: "in">
| <TRUE: "true">
| <FALSE: "false">
| <WHILE: "while">
| <FOR: "for">
| <GET: ".get">
| <GETCHAR: ".getChar">
| <LENGTH: ".length">
| <STRINGLEN: ".len">
| <INSERT: ".insert">
| <STRING: "String">
| <SKIP_mona: "skip">
}


<DEFAULT> TOKEN : {
<COMMA: ",">
| <SEMIC: ";">
| <COLON: ":">
| <LCBR: "{">
| <RCBR: "}">
| <LBR: "(">
| <RBR: ")">
| <LSBR: "[">
| <RSBR: "]">
| <DOT: ".">
}
```

```
<DEFAULT> TOKEN : {
<ASSIGN: "=">
| <PLUS_SIGN: "+">
| <MINUS_SIGN: "-">
| <MUL: "*">
| <DIV: "/">
| <POW: "^">
| <MOD: "%">
| <LOG_NEG: "~">
| <OR: "||">
| <AND: "&&">
| <EQUAL: "==">
| <NOT_EQUAL: "!=">
| <LESS_THAN: "<">
| <LESS_THAN_OR_EQUAL: "<=">
| <GREATER_THAN: ">">
| <GREATER_THAN_OR_EQUAL: ">=">
}
```

```
<DEFAULT> TOKEN : {
<#DIGIT: ["0"-"9"]>
| <#CHAR: ["a"-"z"] | ["A"-"Z"]>
| <NUM: (<MINUS_SIGN>)? ((["1"-"9"])+ (<DIGIT>)* | "0")>
| <FLOAT: <NUM> "." (<DIGIT>)* | "." (<DIGIT>)+>
| <IDENTIFIER: (<CHAR> | "_") (<CHAR> | "_" | <DIGIT>)*>
}
```

```
/*TOKEN: { : STRING }
 TOKEN: {  }
 TOKEN: { : DEFAULT }*/

<DEFAULT> TOKEN : {
<QUOTED_STRING: "\"" (~["\"","\\"] | "\\\"" | "\\\\")* "\"">
}
```

NON-TERMINALS

```
/*****************************************
* SECTION 4 - THE GRAMMAR &
PRODUCTION RULES *
```

**********/

program	::=	decl_list function_list class_list main
decl_list	::=	decl decl_list
	|	
decl	::=	(var_decl | const_decl)
var_decl	::=	<VAR> type identifier (assign expression |) <SEMIC>
const_decl	::=	<CONST> type identifier (assign expression |) <SEMIC>
function_list	::=	function function_list
	|	
function	::=	type identifier <LBR> parameter_list <RBR> <LCBR> decl_list statement_block <RCBR>
return_	::=	<RETURN> <LBR> return_list <RBR> <SEMIC>
return_list	::=	expression (<COMMA> return_list |)
type	::=	(<INT> | <BOOL> | <VOID> | <STRING> | <LSBR> type <RSBR> | <T_FLOAT>)
parameter_list	::=	(nemp_parameter_list |)
nemp_parameter_list	::=	type identifier (<COMMA> nemp_parameter_list |)
main	::=	<MAIN> <LCBR> decl_list statement_block <RCBR>
statement_block	::=	statement statement_block
	|	

statement	::=	(identifier statement_Left_factor_IDENTIFIER | <LCBR> statement_block <RCBR> | <IF> condition <LCBR> statement_block <RCBR> (else_if_list) (else_) | <WHILE> condition <LCBR> statement_block <RCBR> | <FOR> <LBR> identifier <IN> (values | array) <RBR> <LCBR> statement_block <RCBR> | <SKIP_mona> <SEMIC> | decl | <PRINT> <LBR> values <RBR> <SEMIC> | return_)
else_	::=	(<ELSE> <LCBR> statement_block (return_ |) <RCBR>)
	|	
else_if_list	::=	else_if else_if_list
	|	
else_if	::=	<ELSEIF> condition <LCBR> statement_block (return_ |) <RCBR>
statement_Left_factor_IDENTIFIER	::=	assign expression <SEMIC>
	|	(<LBR> arg_list <RBR> <SEMIC>) <RBR> <SEMIC>
	|	<INSERT> <LBR> values <RBR> <SEMIC>
expression	::=	(fragment arith_op)
	|	<LBR> expression <RBR> arith_op
	|	array
arith_op	::=	(<PLUS_SIGN> expression | <MINUS_SIGN> expression | <MUL> expression | <DIV> expression | <POW> expression | <MOD> expression)

	|	
fragment	::=	((<MINUS_SIGN> |) identifier ((<LBR> arg_list <RBR>) | (<GET> <LBR> (number | identifier) <RBR>) | (<LENGTH> <LBR> <RBR>) | (<STRINGLEN> <LBR> <RBR>) | (<GETCHAR> <LBR> (number | identifier) <RBR>) |) | number | bool | string | float_)
fragmentPrime	::=	arith_op expression fragmentPrime
	|	
condition	::=	(<LOG_NEG> condition | <LBR> condition <RBR> | fragment (comp_op expression |)) conditionPrime
conditionPrime	::=	(<AND> condition) conditionPrime
	|	(<OR> condition) conditionPrime
	|	
comp_op	::=	<EQUAL>
	|	<NOT_EQUAL>
	|	<LESS_THAN>
	|	<LESS_THAN_OR_EQUAL>
	|	<GREATER_THAN>
	|	<GREATER_THAN_OR_EQUAL>
arg_list	::=	(nemp_arg_list |)
nemp_arg_list	::=	(identifier | number | string | array | float_) (<COMMA> nemp_arg_list |)
array	::=	<LSBR> (element |) <RSBR>
element	::=	(string | number | identifier | float_) (<COMMA> element |)

values	::=	(identifier | float_ | string | number | bool)
d_structure	::=	(values <COLON> (values | array)) (<COMMA> d_structure |)
class_def	::=	<CLASS> identifier <LCBR> decl_list function_list <RCBR>
class_list	::=	class_def class_list
	|	
identifier	::=	(<IDENTIFIER>)
break_	::=	(<BREAK>)
assign	::=	(<ASSIGN>)
number	::=	(<NUM>)
float_	::=	(<FLOAT>)
string	::=	(<QUOTED_STRING>)
bool	::=	((<TRUE> | <FALSE>))