

LITTLE Programming Language - Grammar

{x} : x is optional

CAPS : CAPS is a token (terminal) made up of one or more characters.

small case symbols are non-terminals.

```
/* Program */
program          -> PROGRAM id BEGIN pgm_body END
id               -> IDENTIFIER
pgm_body         -> decl func_declarations
decl             -> string_decl_list {decl} | var_decl_list {decl} | empty

/* Global String Declaration */
string_decl_list -> string_decl {string_decl_tail}
string_decl      -> STRING id := str ; | empty
str              -> STRINGLITERAL
string_decl_tail -> string_decl {string_decl_tail}

/* Variable Declaration */
var_decl_list    -> var_decl {var_decl_tail}
var_decl         -> var_type id_list ; | empty
var_type         -> FLOAT | INT
any_type         -> var_type | VOID
id_list          -> id id_tail
id_tail          -> , id id_tail | empty
var_decl_tail    -> var_decl {var_decl_tail}

/* Function Paramater List */
param_decl_list  -> param_decl param_decl_tail
param_decl       -> var_type id
param_decl_tail  -> , param_decl param_decl_tail | empty

/* Function Declarations */
func_declarations -> func_decl {func_decl_tail}
func_decl         -> FUNCTION any_type id ({param_decl_list}) BEGIN func_body END | empty
func_decl_tail    -> func_decl {func_decl_tail}
func_body         -> decl stmt_list

/* Statement List */
stmt_list        -> stmt stmt_tail | empty
stmt_tail        -> stmt stmt_tail | empty
stmt             -> assign_stmt | read_stmt | write_stmt | return_stmt | if_stmt | for_

/* Basic Statements */
assign_stmt      -> assign_expr ;
assign_expr      -> id := expr
read_stmt        -> READ ( id_list );
write_stmt       -> WRITE ( id_list );
return_stmt      -> RETURN expr ;

/* Expressions */
expr             -> factor expr_tail
expr_tail        -> addop factor expr_tail | empty
factor           -> postfix_expr factor_tail
factor_tail      -> mulop postfix_expr factor_tail | empty
postfix_expr     -> primary | call_expr
call_expr        -> id ( {expr_list} )
expr_list        -> expr expr_list_tail
expr_list_tail   -> , expr expr_list_tail | empty
primary          -> (expr) | id | INTLITERAL | FLOATLITERAL
addop            -> + | -
mulop            -> * | /

/* Complex Statements and Condition */
```

```

if_stmt      -> IF ( cond ) THEN stmt_list else_part ENDIF
else_part    -> ELSE stmt_list | empty
cond         -> expr compop expr
compop       -> < | > | =
for_stmt     -> FOR ({assign_expr}; {cond}; {assign_expr}) stmt_list ENDFOR

```

an IDENTIFIER token will begin with a letter, and be followed by up to 30 letters and numbers. IDENTIFIERS are case sensitive.

INTLITERAL: integer number

ex) 0, 123, 678

FLOATLITERAL: floating point number available in two different formats

yyyy.xxxxxx or .xxxxxxx

ex) 3.141592 , .1414 , .0001 , 456.98

STRINGLITERAL (Max 80 characters including '\0')

: anything sequence of character except '''

between ''' and '''

ex) "Hello world!" , "*****" , "this is a string"

COMMENT:

Starts with "--" and lasts till the end of line

ex) -- this is a comment

ex) -- any thing after the "--" is ignored

Keywords

PROGRAM, BEGIN, END, PROTO, FUNCTION, READ, WRITE,
IF, THEN, ELSE, ENDIF, RETURN, FOR, ENDFOR
FLOAT, INT, VOID, STRING,

Operators

:= + - * / = < > () ; ,