Tecnológico de Monterrey

Diseño de compiladores

Lenguaje Par++ - Diagramas y Gramática

César Barraza Aguilar

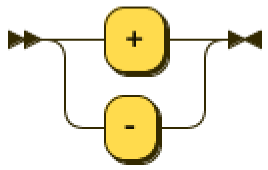
A01176786

Rogelio Martínez Martínez

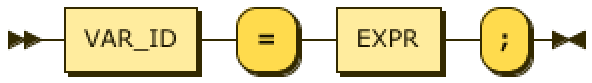
A01176740

Diagramas

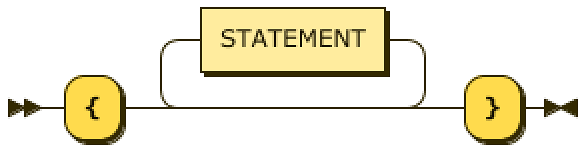
ADDSUB_OP



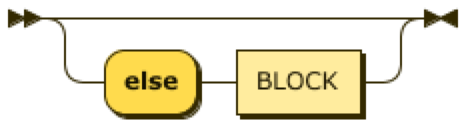
ASSIGNMENT



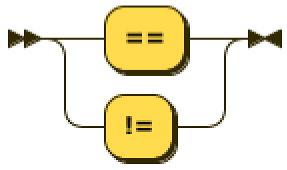
BLOCK



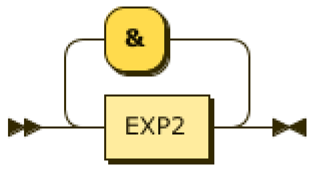
ELSE_STMT



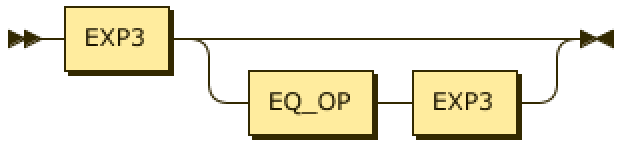
EQ_OP



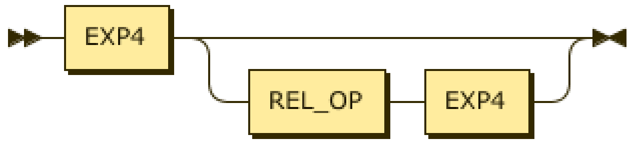
EXP1



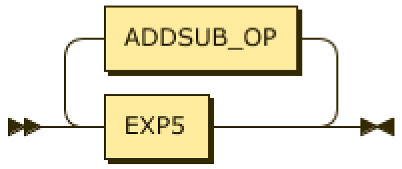
EXP2



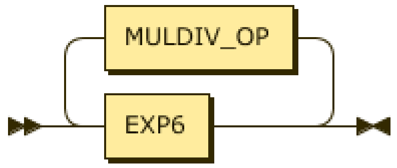
EXP3



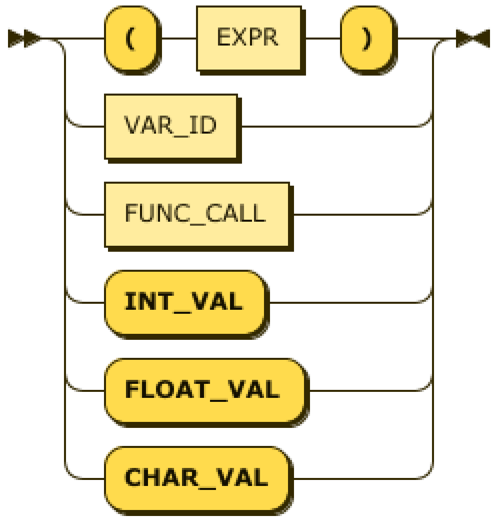
EXP4



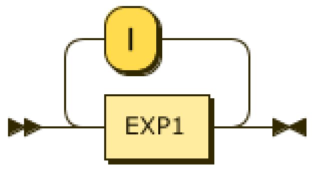
EXP5



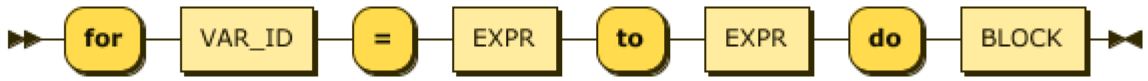
EXP6



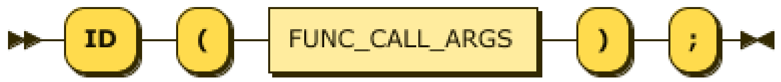
EXPR



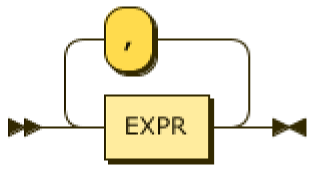
FOR_STMT



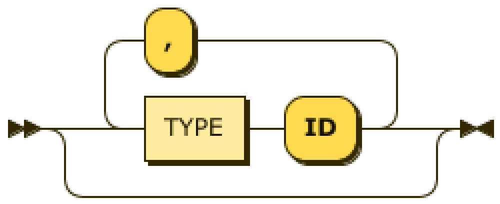
FUNC_CALL



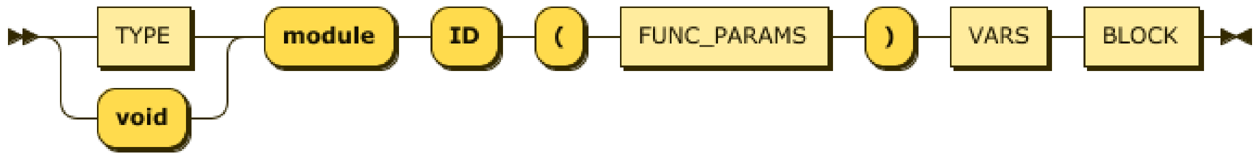
FUNC_CALL_ARGS



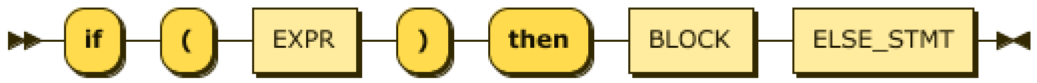
FUNC_PARAMS



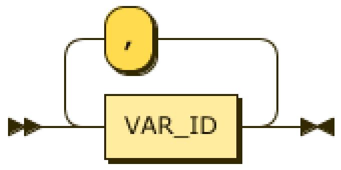
FUNCTION



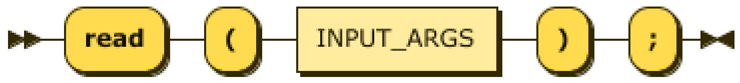
IF_STMT



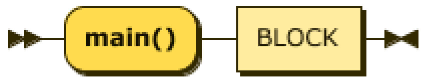
INPUT_ARGS



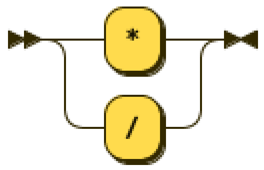
INPUT_STMT



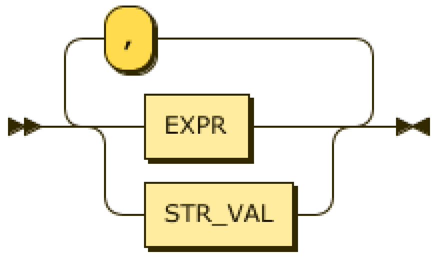
MAIN_FUNCTION



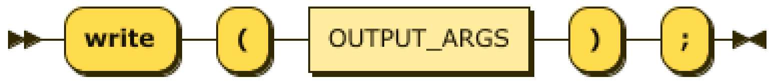
MULDIV_OP



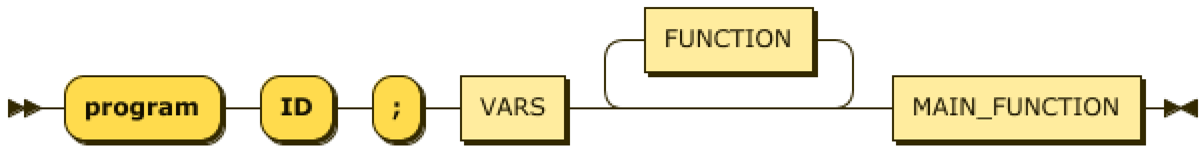
OUTPUT_ARGS



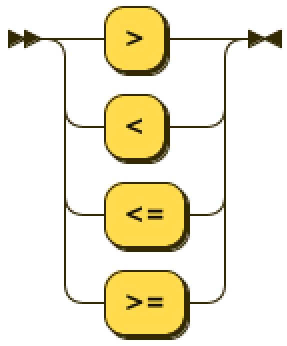
OUTPUT_STMT



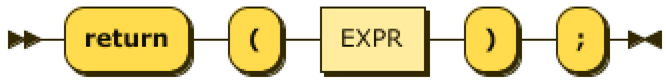
PROGRAM



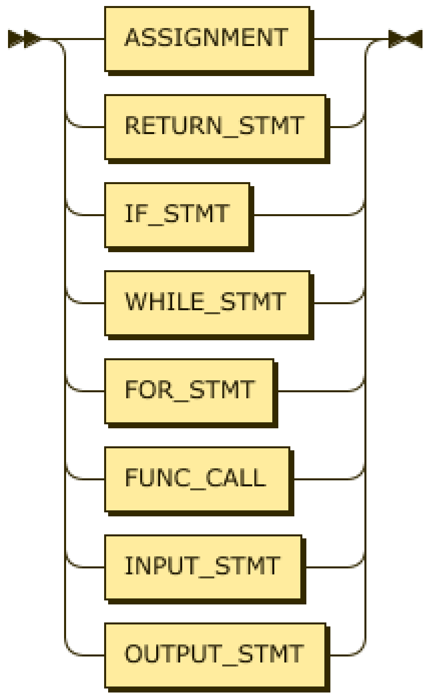
REL_OP



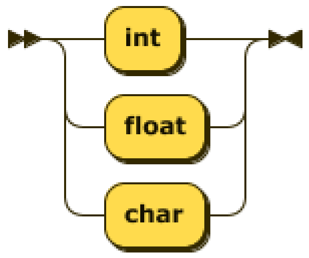
RETURN_STMT



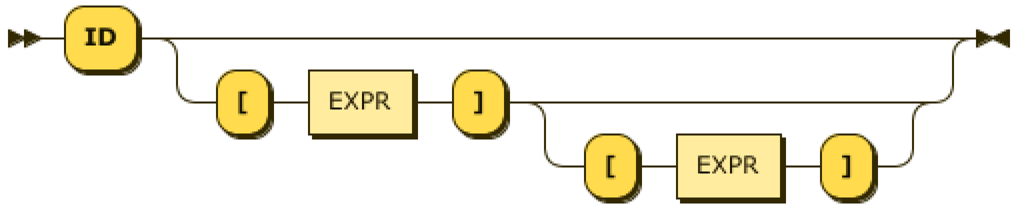
STATEMENT



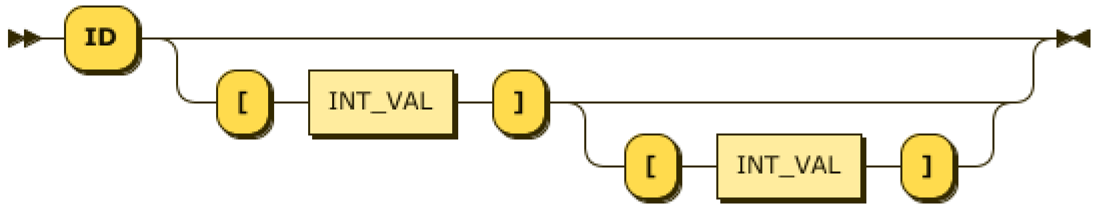
TYPE



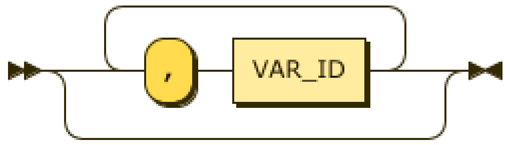
VAR_ID



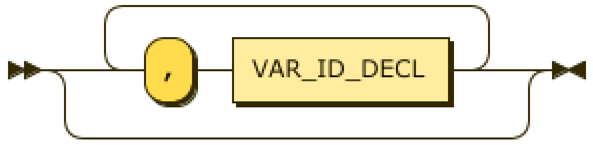
VAR_ID_DECL



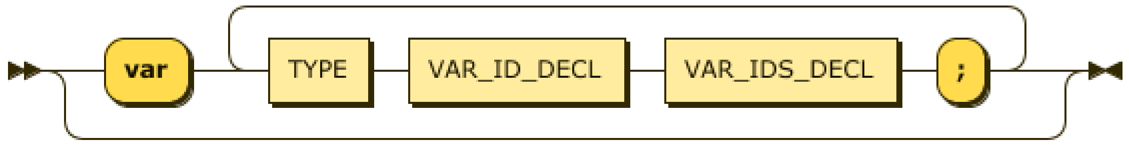
VAR_IDS



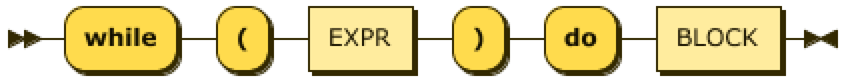
VAR_IDS_DECL



VARS



WHILE_STMT



Gramatica

```
grammar ParPlusPlus;

/*
  Parser Rules
*/
program : 'program' ID ';' vars function* main_function
  ;

vars : ( 'var' ( type var_id_decl var_ids_decl ';' )+ )?
  ;

var_id_decl : ID ( '[' INT_VAL ']' ( '[' INT_VAL ']')? )?
  ;

var_ids_decl : ( ',' var_id_decl )*
  ;

var_id : ID ( '[' expr ']' ( '[' expr ']' )? )?
  ;

var_ids : ( ',' var_id )*
  ;

type : 'int' | 'float' | 'char'
  ;

function : ( type | 'void' ) 'module' ID '(' func_params ')' vars block
  ;

func_params : ( type ID ( ',' type ID )* )?
  ;

block : '{' statement* '}'
  ;

statement : assignment
  | return_stmt
  | if_stmt
  | while_stmt
  | for_stmt
  | func_call
  | input_stmt
  | output_stmt
  ;

assignment : var_id '=' expr ';'
  ;

return_stmt : 'return' '(' expr ')' ';'
  ;

if_stmt : 'if' '(' expr ')' 'then' block else_stmt
  ;

else_stmt : ( 'else' block )?
  ;

while_stmt : 'while' '(' expr ')' 'do' block
  ;
```

```
for_stmt : 'for' var_id '=' expr 'to' expr 'do' block
  ;

rel_op : '>' | '<' | '<=' | '>='
  ;

eq_op : '==' | '!='
  ;

addsub_op : '+' | '-'
  ;

muldiv_op : '*' | '/'
  ;

expr : exp1 ( '|' expr )*
  ;

exp1 : exp2 ( '&' exp1 )*
  ;

exp2 : exp3 ( eq_op exp3 )?
  ;

exp3 : exp4 ( rel_op exp4 )?
  ;

exp4 : exp5 ( addsub_op exp4 )*
  ;

exp5 : exp6 ( muldiv_op exp5 )*
  ;

exp6 : '(' expr ')'
  | var_id
  | func_call
  | INT_VAL
  | FLOAT_VAL
  | CHAR_VAL
  ;

func_call : ID '(' func_call_args ')' ( ';' )?
  ;

func_call_args : expr ( ',' expr )*
  ;

input_stmt : 'read' '(' input_args ')' ';'
  ;

input_args : var_id ( ',' var_id )*
  ;

output_stmt : 'write' '(' output_args ')' ';'
  ;

output_args : ( expr | STR_VAL ) ( ',' ( expr | STR_VAL ) )*
  ;

main_function : 'main()' block
  ;
/*
```

```
  Lexer Rules
*/
WHITESPACE : [ \t\n\r] -> skip
  ;

ID : [a-zA-Z][a-zA-Z0-9]*
  ;

FLOAT_VAL : [0-9]* '.' [0-9]+
  ;

INT_VAL : [0-9]+
  ;

CHAR_VAL : '\'' [a-zA-Z] '\''
  ;

STR_VAL : '"' .*? '"'
  ;
```