# Gramática de Proyecto

Alcántara Estrada Kevin Isaac
Menchaca Carrillo Rodolfo Josue
Sandoval Mendoza Angel

Diciembre 2024

## 1 Gramática Original

```
programa → decl proto decl var decl func
decl proto → proto tipo id ( argumentos ) ; decl proto | epsilon
decl var → tipo lista var ; decl var | epsilon
tipo → basico compuesto | struct { decl var } | puntero
puntero → ptr basico
basico → int | float | double | complex | rune | void | string
compuesto → [ literal entera ] compuesto | epsilon
lista var → lista var , id | id
decl func → func tipo id ( argumentos ) bloque decl func | epsilon
argumentos → lista args | epsilon
lista args → lista args, tipo id | tipo id
bloque → { declaraciones instrucciones }
instrucciones → instrucciones sentencia | sentencia
sentencia → parte izquierda = exp ;
            | if( exp ) sentencia | if( exp ) sentencia else sentencia | while( exp ) sentencia
            | do sentencia while( exp ) | break ; | bloque | return exp ; | return;
            | switch( exp ) { casos } | print exp ; | scan parte izquierda
casos → caso casos | epsilon | predeterminado
caso → case opcion : instrucciones
opcion → literal enter | literal runa
predeterminado → default: instrucciones
parte izquierda → id localizacion | id
exp → exp || exp
     | exp && exp | exp == exp | exp != exp | exp < exp | exp <= exp | exp >= exp | exp > exp
     | exp + exp | exp - exp | exp  exp | exp / exp | exp % exp | exp // exp | ! exp |  exp | (exp)
     | id localizacion | false | literal cadena | true | literal runa | literal entera
     | literal flotante | literal doble | literal compleja | id(parametros) | id
parametros → lista param | epsilon
lista param → lista param , exp | exp
localizacion → arreglo | estructurado
arreglo → arreglo [ exp ] | [ exp ]
estructurado → estructurado . id | . id
```

## 2 Simbolos Terminales

proto, id, (,), struct, {, }, ptr, int, float, double, complex, rune, void, string, [, ], func, if, else, while, do, break, ;, return, switch, print, scan, case, default, :, |||, &&, ==, !=, <, >, <=, >=, + , -, *, /, % , //, !, false, true, literal.cadena, literal.runa, literal.entera, literal.flotante, literal.doble, literal.compleja

## 3 Gramática Final

```
programa → decl_proto decl_var decl_func
decl_proto → proto tipo id ( argumentos ) ; decl_proto'
decl_proto' → decl_proto | epsilon
decl_var → tipo lista_var ; decl_var'
decl_var' → decl_var | epsilon
tipo → basico tipo' | struct { decl_var } | puntero
tipo' → compuesto | epsilon
puntero → ptr basico
basico → int | float | double | complex | rune | void | string
compuesto → [ literal_entera ] compuesto'
compuesto' → [ literal_entera ] compuesto' | epsilon
lista_var → id lista_var'
```

```
lista_var' → , id lista_var' | epsilon
decl_func → func tipo id ( argumentos ) bloque decl_func'
decl_func' → decl_func | epsilon
argumentos → lista_args | epsilon
lista_args → tipo id lista_args'
lista_args' → , tipo id lista_args' | epsilon
bloque → { declaraciones instrucciones }
instrucciones → sentencia instrucciones | epsilon
sentencia → parte_izquierda = exp ;
           | if( exp ) sentencia elseif
           | while( exp ) sentencia
           | do sentencia while( exp ) ;
           | break ;
           | bloque
           | return exp ;
           | return ;
           | switch( exp ) { casos }
           | print exp ;
           | scan parte_izquierda
elseif → else sentencia | epsilon
casos → caso casos | epsilon | predeterminado
caso → case opcion : instrucciones
opcion → literal_entera | literal_runa
predeterminado → default : instrucciones
parte_izquierda → id localizacion | id
exp → exp_or
exp_or → exp_and exp_or'
exp_or' → || exp_and exp_or' | epsilon
exp_and → exp_eq exp_and'
exp_and' → && exp_eq exp_and' | epsilon
exp_eq → exp_rel exp_eq'
exp_eq' → == exp_rel exp_eq' | != exp_rel exp_eq' | epsilon
exp_rel → exp_add exp_rel'
exp_rel' → < exp_add exp_rel'
            | <= exp_add exp_rel'
            | >= exp_add exp_rel'
            | > exp_add exp_rel'
            | epsilon
exp_add → exp_mul exp_add'
exp_add' → + exp_mul exp_add' | - exp_mul exp_add' | epsilon
exp_mul → exp_unary exp_mul'
exp_mul' → * exp_unary exp_mul' | / exp_unary exp_mul' | % exp_unary exp_mul'
         | // exp_unary exp_mul'| epsilon
exp_unary → ! exp_unary | - exp_unary | primary
primary → ( exp ) | id localizacion | false | literal_cadena | true | literal_runa | literal_entera
            | literal_flotante | literal_doble | literal_compleja | id ( parametros ) | id
parametros → lista_param | epsilon
lista_param → exp lista_param'
lista_param' → , exp lista_param' | epsilon
localizacion → arreglo | estructurado
arreglo → [ exp ] arreglo'
arreglo' → [ exp ] arreglo' | epsilon
estructurado → . id estructurado'
estructurado' → . id estructurado' | epsilon
```