**Gramática bnf**

Tarea 1

Raúl Alfaro y Diego Araya

Curso de Lenguajes de Programación

<programa> : := <decl\_glob>\* <funcion>\* <principal>

<decl\_glob> : := "let" <decl\_var> "$"

<principal> : := "procedure" "principal" "є" "э" <bloque> /\* void, sin parámetros \*/

<funcion> : := "function" <id> "є" <param\_formales\_opt> "э" ":" <tipo\_ret>

<bloque>

<param\_formales\_opt> ::= /\* vacío \*/

| <lista\_params>

<lista\_params> ::= <param> ("," <param>)\*

<param> ::= <id> ":" <tipo>

<tipo\_ret> ::= "int" | "float" | "char" | "bool"

<tipo> ::= "int"

| "float"

| "char"

| "bool"

| <tipo\_arreglo>

<tipo\_arreglo> ::= "int" "[" <tam\_arreglo> "]"

| "char" "[" <tam\_arreglo> "]"

<tam\_arreglo> ::= <lit\_entero> /\* tamaño estático, positivo \*/

/\* ===================== BLOQUES Y SENTENCIAS ===================== \*/

<bloque> ::= "¿" <sentencia>\* "?"

| <sentencia> /\* bloque reducido: una sola sentencia \*/

<sentencia> ::= <sent\_decl\_local>

| <sent\_asig\_o\_expr>

| <sent\_decide>

| <sent\_loop>

| <sent\_for>

| <sent\_return>

| <sent\_break>

| <sent\_io>

| <comentario> /\* comentarios como sentencia (se ignoran) \*/

<sent\_decl\_local> ::= "let" <decl\_var> "$"

<decl\_var> ::= <id> ":" <tipo> <init\_opt>

| <id> ":" <tipo\_arreglo> <init\_arr\_opt>

<init\_opt> ::= /\* vacío \*/

| "=" <expr>

<init\_arr\_opt> ::= /\* vacío \*/

| "=" <lit\_arreglo> /\* creación con asignación \*/

<asignacion> ::= <lvalue> "=" <expr>

<lvalue> ::= <id>

| <id> "[" <expr> "]" /\* acceso/modificación de arreglo \*/

<expr\_sin\_asignacion> ::= <expr> /\* expresiones permitidas como sentencia \*/

/\* ===================== ESTRUCTURAS DE CONTROL ====================\*/

<sent\_decide> ::= "decide" "of"

( "є" <cond\_bool> "э" "->" <bloque> )+

( "else" "->" <bloque> )?

"end" "decide" "$"

<sent\_loop> ::= "loop"

<sentencia>\*

"exit" "when" <cond\_bool> "$"

"end" "loop" "$"

<sent\_for> ::= "for" <asignacion> "step" <valor\_step>

( "to" | "downto" ) <expr> "do"

<bloque>

<valor\_step> ::= <lit\_entero> | <lit\_flotante>

/\* ===================== RETURN / BREAK ===================== \*/

<sent\_return> ::= "return" <expr> "$"

| "return" "$" /\* en funciones void NO se usa; sólo por flexibilidad \*/

<sent\_break> ::= "break" "$"

/\* ===================== I/O ===================== \*/

<sent\_io> ::= "input" "є" <id> "э" "$" /\* lee a variable int o float \*/

| "output" "є" <lista\_out> "э" "$"

<lista\_out> ::= <expr> ("," <expr>)\* /\* literales o variables de tipo permitido \*/

/\* ===================== EXPRESIONES ===================== \*/

/\* Precedencia (alta → baja):

1) Postfijo: [] ++ --

2) Prefijo: - Σ

3) Potencia: ^

4) \* / // %

5) + -

6) Relacionales: < <= > >= == !=

7) Lógicas: @ ~

Paréntesis de agrupación: є … э

\*/

<expr> ::= <expr\_log\_or>

<expr\_log\_or> ::= <expr\_log\_and> ( "~" <expr\_log\_and> )\*

<expr\_log\_and> ::= <expr\_rel> ( "@" <expr\_rel> )\*

<expr\_rel> ::= <expr\_aditiva>

| <expr\_aditiva> <op\_rel> <expr\_aditiva>

<op\_rel> ::= "<" | "<=" | ">" | ">=" | "==" | "!="

<expr\_aditiva> ::= <expr\_mult> ( ("+" | "-") <expr\_mult> )\*

<expr\_mult> ::= <expr\_pot> ( ("\*" | "/" | "//" | "%") <expr\_pot> )\*

<expr\_pot> ::= <expr\_unaria> ( "^" <expr\_unaria> )\*

<expr\_unaria> ::= <op\_unario> <expr\_unaria>

| <expr\_postfijo>

<op\_unario> ::= "-" | "Σ"

<expr\_postfijo> ::= <primario> ( <postfix\_op> )\*

<postfix\_op> ::= "++" | "--" | "[" <expr> "]"

<primario> ::= <lit>

| <lvalue>

| <llamada\_func>

| "є" <expr> "э"

<llamada\_func> ::= <id> "є" <lista\_args\_opt> "э"

<lista\_args\_opt> ::= /\* vacío \*/

| <lista\_args>

<lista\_args> ::= <expr> ("," <expr>)\*

<cond\_bool> ::= <expr> /\* debe tipar a boolean en control de flujo \*/