

# PL - Práctica 1

## Lenguaje asignado: BBAAD

El lenguaje asignado está basado en la sintaxis de **C**, con las palabras reservadas en **inglés**, donde se añade a la lista de variables elementales la estructura de datos **lista**, los subprogramas son **funciones** y se incluye la estructura de control **do-until**.

## Descripción formal de la sintaxis del lenguaje usando BNF

```
# Programa principal
<programa> ::= <MAIN> <bloque>

# Bloque general
<bloque> ::= <INIBLOQUE>
            <declar_de_variables_locales>
            <declar_de_subprogs>
            <sentencias>
            <FINBLOQUE>

# Variables locales
<declar_de_variables_locales> ::= <LOCAL> <INIBLOQUE>
                                <variables_locales>
                                <FINBLOQUE>
                                |

<variables_locales> ::= <variables_locales> <cuerpo_declar_variables>
                       | <cuerpo_declar_variables>

<cuerpo_declar_variables> ::= <TIPO> <lista_variables> <PYC>

<lista_variables> ::= <ID> <COMA> <lista_variables>
                    | <ID>

<lista_expresiones> ::= <expresion>
                     | <expresion> <COMA> <lista_expresiones>

# Subprogramas (funciones)
<declar_de_subprogs> ::= <declar_de_subprogs> <declar_subprog>
                       |

<declar_subprog> ::= <cabecera_subprog> <bloque>

<cabecera_subprog> ::= <TIPO> <ID> <PARDER> <parametros> <PARIZQ>
                    | <TIPO> <ID> <PARDER> <PARIZQ>
```

```

<llamada_funcion> ::= <ID> <PARDER> <lista_expresiones> <PARIZQ>
                    | <ID> <PARDER> <PARIZQ>

<parametros> ::= <parametro> <COMA> <parametros>
                | <parametro>

<parametro> ::= <TIPO> <ID>

# Sentencia return
<sentencia_return> ::= <RETURN> <expresion> <PYC>

# Sentencias
<sentencias> ::= <sentencias> <sentencia>
                | <sentencia>
                |
<sentencia> ::= <bloque>
                | <expresion> <PYC>
                | <sentencia_asignacion>
                | <sentencia_lista>
                | <sentencia_if>
                | <sentencia_while>
                | <sentencia_entrada>
                | <sentencia_salida>
                | <sentencia_do_until>
                | <sentencia_return>

# Asignación
<sentencia_asignacion> ::= <ID> <ASIGN> <expresion> <PYC>

# IF
<sentencia_if> ::= <IF> <PARDER> <expresion> <PARIZQ> <sentencia> <bloque_else>
<bloque_else> ::= <ELSE> <sentencia>
                |

# While
<sentencia_while> ::= <WHILE> <PARDER> <expresion> <PARIZQ> <sentencia>

# Entrada
<sentencia_entrada> ::= <CIN> <lista_variables> <PYC>

# Salida
<sentencia_salida> ::= <COUT> <lista_expresiones_o_cadena> <PYC>

<lista_expresiones_o_cadena> ::= <lista_expresiones_o_cadena> <COMA> <expresion_cad
                                | <expresion_cadena>
<expresion_cadena> ::= <expresion>
                      | <CADENA>

# Sentencia Lista
<sentencia_lista> ::= <expresion> <SHIFT>
                    | <DOLLAR> <expresion>

# Do-Until
<sentencia_do_until> ::= <DO> <sentencia> <UNTIL> <PARIZQ> <expresion> <PARDER> <PY

```

```
# Expresión (devuelven un valor)
<expresion> ::= <PARDER> <expresion> <PARIZQ>
              | <OPUNARIO> <expresion>
              | <expresion> <OPBIN> <expresion>
# Operador ternario de listas
              | <expresion> <MASMAS> <expresion> <AT> <expresion>
              | <ID>
              | <CONST>
              | <lista>
              | <llamada_funcion>

# Lista
<lista> ::= <expresion>
          | <expresion>, <lista>
```

## Tabla de tokens

Para hacerlo más legible se han incluido las comillas dobles " pero no se leen, solo se leen en CADENA y las comillas simples ' en CHAR .

Nombre del token	Expresión regular	Código del token	Atributos
INIBLOQUE	"{"	257	
FINBLOQUE	"}"	258	
LOCAL	"local"	259	
TIPOSIMPLE	"int"   "float"   "char"   "bool"   "list_of int"   "list_of float"   "list_of char"   "list_of bool"	260	0: int 1: float 2: char 3: bool 4: list_int 5: list_float 6: list_char 7: list_bool
ID	[a-z A-Z][a-z A-Z 0-9 _]*	261	
PARIZQ	"("	262	
PARDER	")"	263	
PYC	";"	264	

CIN	"cin"	265	
COUT	"cout"	266	
CADENA	"[^\\']*"	267	
RETURN	"return"	268	
OPBIN	"*"   "/"   "%"   "***"   "=="   "!="   "&&"   "  "   "<"   ">"   "<="   ">="   "- -"	269	0: * 1: / 2: % 3: ** 4: == 5: != 6: && 7:    8: < 9: > 10: <= 11: == ">= 12: --
OPUNARIO	"!"   "#"   "?"   "+"   "-"	270	0: ! 1: # 2: ? 3: + 4: -
AT	"@"	271	
MASMAS	"++"	272	
BINYUN	"+"   "-"	273	0: + 1: -
CONST	([0-9]+)   ([0-9]*\\.[0-9]*)   ("true"   "false")   \\'[^\\']*\\'	274	0: int 1: float 2: bool 3: char
ASIGN	"="	275	
COMA	", "	276	
MAIN	"main"	277	
DO	"do"	278	
UNTIL	"until"	279	
IF	"if"	280	
WHILE	"while"	281	
ELSE	"else"	282	
CORIZQ	"["	284	
CORDER	"]"	285	
DOLLAR	"\$"	286	

SHIFT	"<<"   ">>"	287	0: << 1: >>
-------	-------------	-----	-------------

## Referencias

---

- Explicación de la expresión regular [ -~ ] : <https://catonmat.net/my-favorite-regex>.
- <https://import.viva64.com/docx/terminology/Priority/image1.png>