PL - Práctica 1 (v3)

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 (gramática abstracta)

```
# 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>

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

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

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

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

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

# Sentencias
<sentencias> ::= <sentencias> <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>

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

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

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

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

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

# Expresión (devuelven un valor)
<expresion> ::= <PARIZQ> <expresion> <PARDER>
              | <op_unarios> <expresion>
              | <expresion> <op_binarios> <expresion>
              | <expresion> <MASMAS> <expresion> <AT> <expresion>
              | <ID>
              | <constante>
```

```
               | <llamada_funcion>

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

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

<op_unarios> ::= <ADDSUB>
               | <EXCL>
               | <INTHASH>

<op_binarios> ::= <ANDLOG>
                | <ORLOG>
                | <EQN>
                | <REL>
                | <ADDSUB>
                | <MULDIV>
                | <ADDSUB>
                | <PORPOR>
                | <BORRLIST>

<constante> ::= <CONST>
               | <lista>

<lista> ::= <CORIZQ> <lista_expresiones> <CORDER>
          | <CORIZQ> <CORDER>
```

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	
TIPO	`"int" | "float" | "char" | "bool" | "list_of int" | "list_of float" | "list_of char"`	260	0: int 1: float 2: char 3: bool 4: list_int 5: list_float 6:

	`| "list_of bool"`		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	
ORLOG	`"||"`	269	
ANDLOG	`"&&"`	270	
EQN	`"=="|"!="`	271	$0: == 1: !=$
REL	`"<"|">"|"<="|">="`	272	$0: < 1: > 2: <= 3:=""">= $
ADDSUB	`"+"|"-"`	273	0: + 1: -
MULDIV	`"*"|"/"`	274	0: * 1: /
EXCL	`"!"`	275	
PORPOR	`"**"`	276	
BORRLIST	`"--"|"%"`	277	$ 0: -- 1: % $
INTHASH	`"?"|"#"`	278	$0: ? 1: # $
AT	`"@"`	279	
MASMAS	`"++"`	280	
DOLLAR	`"$"`	281	
SHIFT	`"<<"|">>"`	282	0: << 1: >>

CONST	`([0-9]+) | ([0-9]*\.[0-9]*) | \'[^\']\' | ("true" | "false")`	283	0: int 1: float 2: char 3: bool
ASIGN	`"="`	284	
COMA	`","`	285	
MAIN	`"main"`	286	
DO	`"do"`	287	
UNTIL	`"until"`	288	
IF	`"if"`	289	
WHILE	`"while"`	290	
ELSE	`"else"`	291	
CORIZQ	`"["`	292	
CORDER	`"]"`	293	

Referencias

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