

Práctica 1: Diseño del lenguaje

Alejandro Borrego Megías

Blanca Cano Camarero

José Luis Ruiz Benito

Curso 2021/2022

El lenguaje que describimos a continuación (**BBAAC**) está basado en la sintaxis de **C**, con palabras reservadas en **inglés**, a cuyos tipos de datos primitivos añadimos la estructura **lista**, los subprogramas son **funciones** y además de las estructuras de control básicas incluimos la estructura **repeat-until**

Descripción de la sintaxis en BNF

```
<Programa> ::= <Cabecera_programa> <bloque>

<bloque> ::= <Inicio_de_bloque>
            <Declar_de_variables_locales>
            <Declar_de_subprogs>
            <Sentencias>
            <Fin_de_bloque>

<Declar_de_subprogs> ::= <Declar_de_subprogs> <Declar_subprog> |

<Declar_subprog> ::= <Cabecera_subprograma> <bloque>

<Declar_de_variables_locales> ::= var
                                <Inicio_de_bloque>
                                <Variables_locales>
                                <Fin_de_bloque>
                                |

<Cabecera_programa> ::= main (<parametros>)
<parametros> ::= <variables>

<Inicio_de_bloque> ::= {
<Fin_de_bloque> ::= }

<Variables_locales> ::= <Variables_locales> <Cuerpo_declar_variables>
                        | <Cuerpo_declar_variables>

<Cuerpo_declar_variables> ::= <tipo> <variables> ;

<tipo> ::= <primitivo> | <estructura>
<primitivo> ::= bool | char | float | int | string
<estructura> ::= list of <primitivo>

<variables> ::= <identificador>, <variables> | <identificador>
<identificador> ::= <letra> | <letra> <letra_o_digito>
```

```

<letra_o_digito> ::= <letra_o_digito> <letra> | <letra_o_digito> <digito> | <letra> | <digito>

<letra> ::= _ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v |
          A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |
<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Cabecera_subprog> ::= <tipo> <identificador> (<parametros>) | void <identificador> (<parametros>)

<Sentencias> ::= <Sentencias> <Sentencia> | <Sentencia>

<Sentencia> ::= <bloque>
               | <sentencia_asignacion>
               | <sentencia_if>
               | <sentencia_while>
               | <sentencia_entrada>
               | <sentencia_salida>
               | <sentencia_return>
               | <sentencia_repeat_until>

<sentencia_asignacion> ::= <identificador> = <expresion>

<sentencia_if> ::= if (<expresion>) <bloque> <sentencia_else>
<sentencia_else> ::= else <bloque> | elif (<expresion>) <bloque> | elif (<expresion>) <bloque> <sentencia_else>

<sentencia_while> ::= while (<expresion>) <bloque>

<sentencia_entrada> ::= input(<id>) ;

<sentencia_salida> ::= output(<expresion>) ;

<sentencia_return> ::= return <expresion> ;

<sentencia_repeat_until> ::= repeat <bloque> until (<expresion>) ;

<expresion> ::= ( <expresion> )
               | <op_unario> <expresion>
               | <expresion> <op_binario> <expresion>
               | <identificador>
               | <constante>
               | <funcion>

<op_unario> ::= | //
               | not
               | -
               | ++
               | --
               | get_back      # Último elemento de una lista
               | get_front     # Primer elemento de una lista
               | pop_back      # Elimina último elemento de una lista
               | pop_front     # Elimina primer elemento de una lista

<op_binario> ::= +

```

```

| -
| *
| /
| %
| ^
| ==
| !=
| >
| >=
| <
| <=
| and
| or
| xor
| push_back      # Inserta elemento al final de la lista
| push_front     # Inserta elemento al principio de la lista

<constante> ::=  <entero>
                  | <flotante>
                  | <booleano>
                  | <caracter>
                  | <literal_lista>
                  | <cadena>

<entero> ::= <digito> <entero>
            | <digito>

<flotante> ::= <entero> . <entero>
              | . <entero>

<caracter> ::= '<Cualquier caracter ASCII>'

<booleano> ::=  true
              | false

<lista> ::= [ <lista_expresiones> ]
<lista_expresiones> ::= <expresion>, <lista_expresiones> | <expresion> |

<cadena> ::= "<Cadena con cualquier caracter ASCII>"

<funcion> ::= <id> (<lista_expresiones>)

```

Tabla de tokens

| Nombre del token | Expresión regular | Código del token | Atributos |
|------------------|---|------------------|---|
| INICIOBLOQUE | "{" | 257 | |
| FINBLOQUE | "}" | 258 | |
| VAR | "var" | 259 | |
| PRIMITIVO | "int" "float" "char" "bool" "string" | 260 | 0: int, 1: float, 2: char, 3: bool, 4: string |
| ID | [a-z A-Z][a-z A-Z 0-9 _]* | 261 | |
| PARIZQ | "(" | 262 | |
| PARDER | ")" | 263 | |

| Nombre del token | Expresión regular | Código del token | Atributos |
|------------------|---|------------------|---|
| PYC | “,” | 264 | |
| INPUT | “input” | 265 | |
| OUTPUT | “output” | 266 | |
| RETURN | “return” | 268 | |
| OPUNARIO | “//” “not” “-” “++” “-” “get_back” “get_front” “pop_back” “pop_front” | 270 | 0: !, 1: #, 2: ?, 3: +, 4: -, |
| OPBINARIO | “+” “-” “*” “/” “%” “^” “==” “!=” “>” “>=” “<” “<=” “and” “or” “xor” “push_back” “push_front” | 269 | 0: +, 1: -, 2: *, 3: /, 4: %, 5: ^, 6: ==, 7: !=, 8: >, 9: >=, 10: <, 11: <=, 12: and, 13: or, 14: xor, 15: push_back, 16: push_front |
| CONSTANTE | ([0-9]+) ([0-9].[0-9]) (“true” “false”) ‘[^\\]’ | 270 | 0: int, 1: float, 2: bool, 3: char |
| ASIGN | “=” | 271 | |
| COMA | “,” | 272 | |
| MAIN | “main” | 273 | |
| REPEAT | “repeat” | 274 | |
| UNTIL | “until” | 275 | |
| IF | “if” | 276 | |
| WHILE | “while” | 277 | |
| ELSE | “else” | 278 | |
| CORCHETEIZQ | “[” | 279 | |
| CORCHETEDER | “]” | 280 | |