Procesadores de lenguajes. 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> ::= <parametros>, <tipo> <identificador> | <tipo> <identificador>

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

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


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

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

<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 | ... | z |
            A | ... | Z
<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9


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

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

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


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

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

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

<sentencia_entrada> ::= input <lista_id> ;
<lista_id> ::= <lista_id>, <identificador> | <identificador>

<sentencia_salida> ::= output <lista_expresion_cadena> ;
<lista_expresion_cadena> ::=    <lista_expresion_cadena>, <expresion_cadena>
                                | <expresion_cadena>
<expresion_cadena> ::= <expresion> | <cadena>

<sentencia_return> :: return <expresion> ;

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

<sentencia_lista> ::= <expresion> >>
                    | <expresion> <<
                    | $ <expresion>

<expresion> ::= ( <expresion> )
                | <op_unario> <expresion>
                | <expresion> <op_binario> <expresion>
                | <expresion> ++ <expresion> @ <expresion>
```

```
                | <identificador>
                | <constante>
                | <funcion>

<op_unario> ::= | //
                | not
                | +
                | -
                | ++
                | --
                | #
                | ?

<op_binario> ::=  +
                | -
                | *
                | /
                | %
                | ^
                | ==
                | !=
                | >
                | >=
                | <
                | <=
                | and
                | or
                | xor
                | @
                | --
                | **

<constante> ::=   <entero>
                | <flotante>
                | <booleano>
                | <caracter>
                | <lista>

<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”	260	0: int, 1: float, 2: char, 3: bool
ID	[a-z|A-Z][a-z|A-Z|0-9|_]*	261	
PARIZQ	“(”	262	
PARDER	“)”	263	
PYC	“;”	264	
INPUT	“input”	265	
OUTPUT	“output”	266	
RETURN	“return”	268	
OPUNARIO	“//” | “not” | “-” | “”|||269|0 : //, 1 : *not*, 2 : −, 3 : ””		
OPUNARIOBINARIO	“_”	270	
OPER++	“++”	271	
OPERLISTA	“<<” | “>>”	272	
ARROBA	“@”	273	
OPBINARIO	“+” | “-” | "*" | “/” | “%” | “^” | “==” | “!=” | “>” | “>=” | “<” | “<=” | “and” | “or” | “xor” |	274	0: +, 1: -, 2: *, 3: /, 4: %, 5: ^, 6: ==, 7: !=, 8: >, 9: >=, 10: <, 11: <=, 12: and, 13: or, 14: xor
CONSTANTE	([0-9]+) | ([0-9].[0-9]) | (“true” | “false”)	270	0: int, 1: float, 2: bool, 3: char
ASIGN	“=”	274	
COMA	“,”	275	
MAIN	“main”	276	
REPEAT	“repeat”	277	
UNTIL	“until”	278	
IF	“if”	279	
WHILE	“while”	280	
ELSE	“else”	281	
CORCHETEIZQ	“[”	282	
CORCHETEDER	“]”	283	
CADENA	"[’]*"	284	
ELIF	“elif”	285	