Lenguaje BABAA

Luis Antonio Ortega Andrés
Guillermo Galindo Ortuño
Johanna Capote Robayna
Antonio Coín Castro

1 Descripción del lenguaje

La sintaxis de nuestro lenguaje está inspirada en el lenguaje **C**. Por ello, tomaremos como referencia las reglas sintácticas de este para las instrucciones del nuevo lenguaje.
Para las palabras reservadas usaremos el lenguaje **castellano**.
En nuestro lenguaje implementaremos los **arrays 1D y 2D** como estructura de datos. Este tipo de datos tendrán las siguientes operaciones.

- Acceso a elemento.
- Producto.
- Suma y resta elemento a elemento.
- Producto externo (producto de un array por un escalar).
- Producto de matrices. Teniendo en cuenta que para multiplicar dos matrices, las dimensiones de estas deben ser las adecuadas.

Esta estructura solo puede tener tipos básicos en su interior (entero, carácter, real y booleano).
El tipo de subprograma que contendrá nuestro lenguaje son **funciones**, es decir, supondremos que nuestro subprograma siempre devuelve un valor.
Por último, el lenguaje incluirá una estructura de control tipo **switch**.

2 Descripción formal de la sintaxis del lenguaje usando BNF

```
#+BEGIN_SRC
<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>
                              |   vacio
<Declar_subprog>              ::= <Cabecera_subprograma> <Bloque>
<Declar_de_variables_locales> ::= <Marca_ini_declar_variables>
                                  <Variables_locales>
                                  <Marca_fin_declar_variables>
                              |   vacio
<Marca_ini_declar_variables>  ::= var
<Marca_fin_declar_variables>  ::= finvar
<Cabecera_programa>           ::= programa()
```

```
<Inicio_de_bloque>          ::= {
<Fin_de_bloque>             ::= }
<Variables_locales>         ::= <Variables_locales> <Cuerpo_declar_variable>;
                            |   <Cuerpo_declar_variable>;
<Cuerpo_declar_variable>    ::= <Tipo> <Lista_id>
<Cabecera_subprog>          ::= <Tipo_comp> <Identificador>(<Lista_argumentos>)
<Lista_argumentos>          ::= <Argumentos>
                            |
<Argumentos>                ::= <Argumento> , <Argumentos>
                            |   <Argumento>
<Argumento>                 ::= <Tipo> <Identificador_comp>
<Booleano>                  ::= verdadero | falso
<Digito>                    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<Natural>                   ::= <Digito> <Natural>
                            |   <Digito>
<Real>                      ::= <Natural>.<Natural>
<Caracter>                  ::= a | ... | z | A | ... | Z | _
<Alfanum>                   ::= <Caracter>
                            |   <Natural>
<Caracter_ascii>            ::= (Cualquier carácter ASCII menos las comillas (’ o "))
                            |   \"
                            |   \’
<Cadena>                    ::= <Cadena> <Caracter_ascii>
                            |   <Caracter_ascii>
<Cadena_const>              ::= "<Cadena>"
<Constante>                 ::= <Natural>
                            |   <Real>
                            |   <Booleano>
                            |   ’<Caracter_ascii>’
<Tipo>                      ::= entero
                            |   real
                            |   buleano
                            |   caracter
<Tipo_comp>                 ::= <Tipo>
                            |   <Tipo><Acceso_array>
<Identificador>             ::= <Identificador> <Alfanum>
                            |   <Caracter>
<Expresion>                 ::= (<Expresion>)
                            |   <Identificador>
                            |   <Constante>
                            |   <Op_unario_izquierda> <Expresion>
                            |   <Expresion> <Op_unario_derecha>
                            |   <Expresion> <Op_binario> <Expresion>
                            |   <Agregado1D>
                            |   <Agregado2D>
                            |   <Llamada_funcion>
<Agregado1D>                ::= {<Expresiones>}
<Agregado2D>                ::= {<Listas> ; <Expresiones>}
<Listas>                    ::= <Listas> ; <Expresiones>
                            |   <Expresiones>
<Expresiones>               ::= <Expresion>, <Expresiones>
                            |   <Expresion>
<Llamada_funcion>           ::= <Identificador>(<Expresiones>)
```

```
                              |   <Identificador>()
<Op_unario_izquierda>         ::= !
                              |   +
                              |   -
                              |   ++
                              |   --
<Acceso_array>                ::= [<Expresion>]
                              |   [<Expresion>,<Expresion>]
<Identificador_comp>          ::= <Identificador>
                              |   <Identificador><Acceso_array>
<Op_unario_derecha>           ::= ++
                              |   --
                              |   <Acceso_array>
<Op_binario>                  ::= ==
                              |   >=
                              |   <=
                              |   !=
                              |   +
                              |   -
                              |   *
                              |   /
                              |   **
                              |   ^
                              |   <
                              |   >
                              |   &&
                              |   ||
<Sentencias>                  ::= <Sentencias> <Sentencia>
                              |   vacio
<Sentencia>                   ::= <Bloque>
                              |   <Sentencia_asignacion>
                              |   <Sentencia_if>
                              |   <Sentencia_while>
                              |   <Sentencia switch>
                              |   <Sentencia_break>
                              |   <Sentencia_return>
                              |   <Sentencia_entrada>
                              |   <Sentencia_salida>
<Sentencia_asignacion>        ::= <Identificador_comp> = <Expresion>;
<Sentencia_if>                ::= si (<Expresion>) <Sentencia> <Sentencia_else>
<Sentencia_else>              ::= otro <Sentencia>
                              |
<Sentencia_while>             ::= mientras (<Expresion>) <Sentencia>
<Sentencia_switch>            ::= casos (<Expresion>) <Bloque_switch>
<Bloque_switch>               ::= { <Opciones> }
<Opciones>                    ::= <Opciones> <Opcion>
                              |   <Opcion> <Opcion_pred>
                              |   <Opcion_pred>
<Opcion>                      ::= caso <Entero>: <Sentencias>
<Opcion_pred>                 ::= predeterminado: <Sentencias>
<Sentencia_break>             ::= roto;
<Sentencia_return>            ::= devolver
<Sentencia_entrada>           ::= entrada <Lista_id>;
```

```
<Lista_id>                  ::= <Lista_id>, <Identificador_comp>
                            |   <Identificador_comp>
<Lista_exp_cad>             ::= <Lista_exp_cad>, <Exp_cad>
                            |   <Exp_cad>
<Exp_cad>                   ::= <Expresion>
                            |   <Cadena>
<Sentencia_salida>          ::= salida <Lista_exp_cad>;
#+END_SRC
```

3 Definición de la semántica en lenguaje natural

El programa comienza con una cabecera inicial y un bloque. La cabecera inicial esta formada por la palabra reservada "programa" seguida de de paréntesis sin argumentos. Por otro lado el bloque empieza con "{" y termina "}", en su interior pueden aparecer variables locales, subprogramas o sentencias.
Las variables locales deben declararse entre unas marcas de inicio y fin, notadas como "var" y "finvar" respectivamente. Una vez dentro de las marcas, podemos declarar variables al estilo de C, e inicializarlas *in situ*:

```
<tipo> <id>;
<tipo> <id> = <expr>;
```

Los subprogramas son siempre funciones que devuelven algo. Constan de una cabecera `<tipo> <nombre>(<argumentos>)`, donde los argumentos pueden ser 0 o más, separados por comas. El cuerpo de los subprogramas vuelve a ser un bloque, lo que permite anidamiento. Se devuelven los datos con la palabra clave 'devolver'.
Las sentencias pueden ser un bloque, una expresión o las instrucciones de control (si, otro, mientras, casos). Las instrucciones de control tienen una expresion y una sentencia. La sintaxis del 'switch' es una copia de la de C, teniendo en cuenta que los caso solo pueden ser números enteros.
La expresión puede encontrarse entre paréntesis, y puede ir precedido de una operación unaria o combinar dos expresiones con una operación binaria. Además puede ser una constante o un identificador.
Por último, tenemos un tipo especial similar a los arrays de C, que representa un agregado de datos del mismo tipo, ya sea unidimensional o bidimensional.

4 Tabla de tokens

Nombre	Expresión regular	Código	Atributos
CABECERA	"programa()"	257	
LLAVEIZQ	"{"	258	
LLAVEDCH	"}"	259	
INILOCAL	"var"	260	
FINLOCAL	"finvar"	261	
TIPO	"entero" | "real" | "buleano" | "caracter"	262	0:entero 1:real 2:buleano 3:caracter
IF	"si"	263	
ELSE	"otro"	264	
WHILE	"mientras"	265	
SWITCH	"casos"	266	
CASE	"caso"	267	
BREAK	"roto"	268	
CIN	"entrada"	269	
COUT	"salida"	270	
PREDET	"predeterminado"	271	
ASIG	"="	272	
CORCHIZQ	"["	273	
CORCHDCH	"]"	274	
COMA	","	275	
PYC	";"	276	
PYP	":"	277	
PARIZQ	"("	278	
PARDCH	")"	279	
SIGNO	"-"	280	
DIGITO	[0-9]	281	
PUNTO	"."	282	
IDENTIFICADOR	[a-zA-Z_][a-zA-Z0-9_]*	283	
LITERAL	-?[0-9]+(.[0-9]+)? | "verdadero" | "falso" | (\"(\\\" | \\\' | [^\"\'])+\") | (\'(\\\" | \\\' | [^\"\'])\')	284	0:real_num 1:verdadero 2:falso 3:cte_cadena 4:cte_caracter
RETURN	"devolver"	285	
OPREL	[<>]=? | [=!]=	286	
OPBIN	"+" | "-" | "*" | "/" | "**"	287	0:+ 1:- 2:* 3:/ 4:**
OPLOG	"^" | " || " | "&&"	288	0:^ 1:|| 2:&&
OPUNARIOIZQ	"!" | "-" | "++" | "--"	289	0:! 1:- 2:++ 3:--
OPUNARIODCH	"++" | "--"	290	0:++ 1:--
ARRAY1D	\[[0-9]+\]	291	
ARRAY2D	\[[0-9]+,[0-9+]\]	292	