Procesadores de Lenguajes

Práctica 1

Ismael Sánchez García

Juan Manuel Fajardo Sarmiento

Francisco Javier Caracuel Beltrán

1. Descripción del lenguaje a desarrollar:

   El lenguaje se puede dividir en dos secciones: común y específica. La sección común es similar para todos los grupos y la específica es única, siguiendo unas reglas definidas por un código.
   En este caso, el código es “BABAD”.

   - Común:
     - Es un subconjunto de un lenguaje de programación estructurado.
     - Los identificadores se declaran antes de ser usados.
     - Tipos de datos mínimos: entero, real, carácter, booleano.
       Los enteros y reales podrán realizar las siguientes operaciones: suma, resta, producto, división, operaciones de relación.
       Los booleanos pues realizar las siguientes operaciones: and, or, not, xor.
     - Todas las expresiones disponen de la sentencia de asignación.
     - Permite expresiones aritméticas lógicas.
     - Tiene una sentencia de entrada y otra de salida.
     - Dispone de las estructuras de control: if-then-else, while.
     - La estructura sintáctica es: <Programa> ::= <Cabecera_programa> <bloque>
     - Se permite el anidamiento de bloques.
     - La comprobación de tipos es fuertemente tipado.
     - En los argumentos de un subprograma los parámetros se pasan por valor.
     - No se permiten declaraciones fuera de los bloques, teniendo que ir entre marcas de inicio y de fin.

   - Específico (lenguaje “BABAD”):
     - La sintaxis está inspirada en lenguaje C.
     - Las palabras reservadas se encuentran en castellano.
     - Debe disponer de un Array de una y dos dimensiones con las operaciones: acceso a un elemento, producto, suma, resta elemento a elemento, producto de un array por un escalar y producto de matrices.
     - Deben existir las constantes de tipo Array.
     - Los subprogramas deben ser funciones.
     - Las estructuras de control adicional son: do-until.

2. Descripción formal de la sintaxis del lenguaje usando BNF:

```
//
// BABAD (Lenguaje C - Castellano - vector 1D y 2D - Funciones - Do-Until)
//

<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> ::= <Marca_ini_declar_variables>
<Variables_locales> <Marca_fin_declar_variables>
        |

<Cabecera_programa> ::= principal

<Inicio_de_bloque> ::= {

<Fin_de_bloque> ::= }

<Marca_ini_declar_variables> ::= ini_var_local

<Marca_fin_declar_variables> ::= fin_var_local

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

<Cuerpo_declar_variables> ::= <tipo_basico> <lista_variables> ;

<Cabecera_subprograma> ::= <tipo_basico> <variable> ( <lista_parametros>
)
        | <tipo_basico> <variable> ( )

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

<Sentencia> ::= <bloque>
        | <sentencia_asignacion>
        | <sentencia_si>
        | <sentencia_mientras>
        | <sentencia_entrada>
        | <sentencia_salida>
        | <sentencia_devolver>
        | <sentencia_hacer_hasta>

<sentencia_asignacion> ::= <var_array> = <expresion> ;
```

```
<sentencia_si> ::= si ( <expresion> ) <sentencia>
                  |  ( <expresion> ) <sentencia> si_no <sentencia>

<sentencia_hacer_hasta> ::= hacer <sentencia> hasta ( <expresion> )

<sentencia_mientras> ::= mientras ( <expresion> ) <sentencia>

<sentencia_entrada> ::= leer <lista_variables> ;

<sentencia_salida> ::= escribir <lista_expresiones_o_cadena> ;

<sentencia_devolver> ::= devolver <expresion> ;

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

<tipo_basico> ::= entero
          | booleano
          | caracter
          | flotante

<lista_variables> ::= <lista_variables> , <variable>
          | <variable>

<identificador> ::= "cadena que empieza por _ o una letra"

<variable> ::= <identificador>
           | <identificador> [ <const_entero_sin_signo> ]
           | <identificador> [ <const_entero_sin_signo> ] [
<const_entero_sin_signo> ]

<var_array> ::= <identificador>
           | <identificador> [ <expresion> ]
           | <identificador> [ <expresion>, <expresion> ]

<lista_parametros> ::= <lista_parametros> , <tipo_basico> <variable>
          | <tipo_basico> <variable>

<lista_entero> ::= <lista_entero> , <const_entero>
           | <const_entero>

<lista_booleano> ::= <lista_booleano> , <const_booleano>
           | <const_booleano>

<lista_flotante> ::= <lista_flotante> , <const_flotante>
           | <const_flotante>
```

```
<lista_caracter> ::= <lista_caracter> , <const_caracter>
          | <const_caracter>

<lista_expresiones_o_cadena> ::= <lista_expresiones_o_cadena> ,
<expresion>
          | <lista_expresiones_o_cadena> , <cadena>
          | <expresion>
          | <cadena>

<cadena> ::= "cualquier secuencia de caracteres"

<op_unario> ::= &
          | +
          | -
          | ~
          | !

<op_binario> ::= +
          | -
          | *
          | /
          | ==
          | !=
          | <=
          | >=
          | <
          | >
          | &&
          | ||

<signo> ::= -
          | +
          |

<constante> ::= <const_entero>
          |<const_entero_sin_signo>
          | <const_matriz>
          | <const_booleano>
          | <const_flotante>
          | <const_flotante_sin_signo>
          | <const_caracter>

<funcion> ::= <identificador> ( <lista_expresiones_o_cadena> ) ;
                              | <identificador ( ) ;

<vector> ::= <tipo_basico> <identificador> [ <const_entero_sin_signo> ] ;
                              | <tipo_basico> <identificador> [
<const_entero_sin_signo> , <const_entero_sin_signo> ] ;

<const_entero_sin_signo> ::= <const_entero_sin_signo> [0-9]
          | [0-9]
```

```
<const_entero> ::= <signo> <const_entero_sin_signo>

<const_matriz> ::=  <matriz_entero>
            | <matriz_booleano>
            | <matriz_flotante>
            | <matriz_caracter>

<matriz_entero> ::=  { <lista_entero> }

<matriz_booleano> ::=  { <lista_booleano> }

<matriz_flotante> ::=  { <lista_flotante> }

<matriz_caracter ::=  { <lista_caracter> }

<const_booleano> ::= verdadero
          | falso

<const_flotante> ::= <const_entero> . <const_entero_sin_signo>

<const_flotante_sin_signo> ::= <const_entero_sin_signo> .
<const_entero_sin_signo> //Se cree que no es necesario al tener
const_entero -, +, o vacío

<const_caracter> ::= [a-z]
          | [A-Z]
```

3. Definición de la semántica en lenguaje natural:

   El programa comienza con una cabecera inicial y un bloque.

   La cabecera inicial está formada por un entero, la palabra reservada *principal* seguida de paréntesis sin argumentos.

   El bloque comienza con la palabra reservada “ini_bloque” y termina con la palabra reservada “fin_bloque”. Entre la apertura y cierre del bloque pueden aparecer variables locales, subprogramas o sentencias.

   Las variables locales se componen del tipo al que pertenecen (entero, booleano, carácter, flotante) y una lista de identificadores, pudiendo ser uno o varios separados por comas, terminando en ; (Ej: entero a , b , c;).

   Los subprogramas se definen como funciones, indicando primero el tipo que devuelve, el nombre de la función, seguido de un paréntesis de apertura y de cierre, siendo opcional añadir una lista de parámetros entre ambos.

   Lista de parámetros es una lista de tipo y nombre de la variable, separados por comas.

   Tras la declaración de la cabecera de la función se escribe un bloque.

   Sentencia puede ser un bloque, una asignación, las instrucciones de control: si, hacer hasta, mientras; entrada/salida (teclado/pantalla) o devolver (return). Las instrucciones de control, devolver y la de asginación hacen uso de una expresión.

   Expresión puede encontrarse entre paréntesis y es una operación unaria, binaria, una constante, una función o una variable.

   Una variable puede ser una variable básica, un vector o una matriz de 2 dimensiones.

4. Identificación de los tokens:

a) Palabras:

{	=	booleano	*	verdadero
}	[	caracter	/	falso
;	]	flotante	==	.
,	si	leer	!=	principal
(	si_no	escribir	<=	
)	hacer	&	>=	
ini_var_local	hasta	+	<	
fin_var_local	mientras	-	>	
ini_bloque	devolver	~	&&	
fin_bloque	entero	!	||	

b) Tokens:

Tokens	Código	Palabra	Atributo
PRINCIPAL	256	principal	
INI_BLOQUE	257	{	
FIN_BLOQUE	258	}	
PUNTO_Y_COMA	259	;	
COMA	260	,	
PARENT_IZQUIERDO	261	(	
PARENT_DERECHO	262	)	
INI_VAR_LOCAL	263	ini_var_local	
FIN_VAR_LOCAL	264	fin_var_local	
ASIGNACION	265	=	
INI_DIM_MATRIZ	266	[	
FIN_DIM_MATRIZ	267	]	
SI	268	si	
SI_NO	269	si_no	
HACER	270	hacer	
HASTA	271	hasta	
MIENTRAS	272	mientras	
DEVOLVER	273	devolver	
TIPO_BASICO	274	entero booleano caracter flotante	0: entero 1: booleano 2: carácter 3: flotante
ENTRADA	275	leer	
SALIDA	276	escribir	

SIGNO	277	+ -	0: + 1: -
OP_UNARIO	278	& ~ !	0: & 1: ~ 2: !
OP_BINARIO	279	* / == != <= >= < > && ||	0: * 1: / 2: == 3: != 4: <= 5: >= 6: < 7: > 8: && 9: ||
CONST_LOGICA	280	verdadero falso	0: verdadero 1: falso
CADENA	281	\"[^\"]+\"	
CONST_ENTERO	282	[0-9]+	
CONST_FLOTANTE	283	[0-9]*\.([0-9]+)?	
CONST_CARACTER	284	\'[^\']\'	