Práctica 1 PL.

1.- Miembros del grupo.

- Jesús Alvarez Fernández.
- Manuel Castellón Reguero.
- Pablo Dueñas Alvarez.
- Yunhao Lin Pan.

2.- Equipo.

Equipo 5 del viernes de 11:30 a 13:30. Lenguaje BAABB.

3.- Características del lenguaje.

Las características de nuestro lenguaje son las siguientes:

1. Sintaxis inspirada en el lenguaje de programación C.
2. Palabras reservadas en castellano.
3. Las estructuras de datos consideradas como tipo elemental son las listas.
4. Como subprogramas tenemos los procedimientos.
5. La estructura de control adicional es el bucle for.

4.- Descripción formal de la sintaxis del lenguaje utilizando BNF.

<Programa> :== <Cabezera_programa><bloque>

<Cabecera_programa> :== “main” “(“ “)”

<bloque> :== <Inicio_de_bloque>
<Declarar_variables_locales>
<Declarar_procedimiento>
<Sentencias>
<Fin_de_bloque>
| <Sentencias>

<Inicio_de_bloque> :== “{“

<Fin_de_bloque> :== “}”

<Declarar_procedimientos> :== <Declarar_procedimientos> <Declarar_procedimiento>

<Declarar_procedimiento> :== <Cabecera_procedimiento> <bloque>
|

<Cabecera_procedimiento> :== <identificador> “(“ <Parametro> “)”

<Parametro> :== <Parametro> “,” <tipo_dato> <identificador>
| <tipo_dato> <identificador>
|

<Declarar_variables_locales> :== <Declarar_variables_locales><Variable_local>
|<Variable_local>
|

<Sentencias>:== <bucle_para>
| <bucle_mientras>
| <sentencia_si>
| <asignacion>
| <asignacion><operacion> “;”
| <identificador><asignacion><Cabecera_procedimiento> ";"
| <Sentencia_entrada> ";"
|<Sentencia_salida> ";"
| <bloque>
| <Sentencias>

<Sentencia_entrada> :== <nomb_entrada> <Lista_variables>

<Sentencia_salida> :== <nomb_salida> <Lista_exp_o_cadena>

<Lista_variables> :== <Lista_variables> “,” <identificador>
| <identificador>

<Lista_exp_o_cadena>:==<Lista_exp_o_cadena> ”,” <expresion>
| <Lista_exp_o_cadena> ”,” ”” <Cadena> ””
| <expresion>
| “” <Cadena> “”

<Cadena>:==<Cadena> <letra>
| <letra>

<Variable_local> :== <tipo_dato><identificador> ";"
| <tipo_dato><identificador><op_asignacion><expresion> ";"
| <tipo_dato><identificador>"," <identificador>";"

<bucle_para>:== ”para”<identificador>”:=”<número>”sube hasta”<número> “haz”
<bloque>
| ”para”<identificador> ”:=” <número> “baja hasta” <número> “haz”
<bloque>

<condicion> :== <Variable_local> <op_logico> <Variable_local>
| <Variable_local> <op_logico> <constante>

<sentencia_si>:== “si (" <condicion> ") “entonces”
<bloque>
“sino”
<bloque>

<bucle_mientras> :== “mientras” “(“<condicion>”)”
<bloque>

<tipo_dato>:== <tipo_basico>

| <tipo_complejo>

<tipo_basico> :== “entero”
| “real”
| “bool”
| “caracter”

<tipo_complejo> :== “lista de” <tipo_basico>

<constante> :== “constante” <tipo_dato> <identificador> “;”

<op_aritmetico> :== +
| -
| /
| *
| %
|@
| --

<op_logico> :== &&
| ||
| y
| o
| xor
| no
| !
| ==

<op_binario> :== <op_aritmetico>
| <op_logico>
| <op_binario>

<op_unaria> :== ++
| --
| -
| no
| !
| #
| ?
| **
| >>
| <<
| $

<op_asignacion> :== “=”

<asignacion> :== <expresion> <op_asignacion> <expresion> ";"
<expresion> <op_asignacion> <asignacion>

<expresion> :== <expresion> <op_binario> <expresion>
| <op_unaria> <expresion>
| <expresion> <op_unaria>

```
                    | <op_ternaria>
                    |<expresion> ”++” <expresion> ”@” <expresion>
                    | “(“ <expresion> “)”
                    |<número>
                    |<caracteres>
                    |<identificador>
                    |<agregados>

<agregados> :== <inicio_agregado> <agregados> <fin_agregado>
                    | <expresion> “,“
                    | <expresion>
                    |

<inicio_agregado> :== “[“
<fin_agregado> :== “]”

<numero> :== <real>
            | <entero>

<entero> :== <entero> <dígito>
              | <dígito>

<real>:== <entero> “.” <decimal>

<decimal> :== <decimal> <dígito>
                    | <dígito>

<dígito> :== 0 | 1 | … | 9

<identificador> :== <identificador> <letra>
                    | <identificador> <número>
                    | <letra>

<caracteres> :== <caracteres> | <letra>
                    |<letra>

<letra> :== a | b | …  | z
```

5.- Definición de la semántica utilizando lenguaje natural.

En este apartado explicaremos el uso de las distintas características del lenguaje, tanto las que son exclusivas de nuestro lenguaje como las que no.

Comencemos con lo más básico, el programa, para poder hacer un programa hay que hacer una cabecera que, en este caso, es el main propio de todo programa escrito en C. Tras la cabecera hay que incluir el bloque que conformará el programa en sí.

En el bloque se pueden definir variables, procedimientos y emplear sentencias. A continuación explicaré cada una con más detalle.

Para definir variables se emplean sentencias muy sencillas, para empezar se pone el tipo de dato que será la variable, que puede ser entero, real, bool o carácter, tras esto se pone el nombre de la variable y **;** . También es posible asignar el valor a la variable en la propia declaración poniendo el operador **=** tras el nombre de la variable, el valor que tendrá la variable y **;** de la misma forma que antes. Las listas tienen una peculiaridad en este campo que se explicará más adelante.

A la hora de definir procedimientos se hace lo siguiente: primero se define el nombre de la función, tras lo cual se abren y cierran unos paréntesis en los que se definirán las variables que emplean el procedimiento seguido por comas. Tras la cabecera se emplea otro bloque como el empleado en el programa en sí. Un ejemplo de procedimiento sería el siguiente: exponencial(entero numero1, entero numero2)**{bloque}** He de aclarar que esta es la definición, para usarla hay que hacerlo en el apartado de sentencias que viene a continuación.

Por último las sentencias que son el apartado más complejo de un bloque. En una sentencia se recogen las siguientes acciones:

1. Bucles **para**
2. Sentencias **si**
3. Bucles **mientras**
4. Asignaciones y operaciones.
   Por ejemplo variable1 = variable2; o variable1 = variable2+1;
5. Llamadas a procedimientos definidos con anterioridad. Esto se haría de la siguiente manera: variable = procedimiento(argumentos)
6. Sentencias de entrada por teclado o salida por pantalla
7. Otro bloque

De la lista anterior lo único que requiere una mayor profundidad son los bucles y la sentencia **si**.

Comencemos explicando el funcionamiento del bucle **para** (for) que está en nuestro lenguaje asignado, para emplearlo es necesario poner una cabecera seguida del cuerpo del bucle. La cabecera comienza con la palabra clave **para** seguida de una variable local que se emplea para contabilizar el número de iteraciones, una palabra clave para decidir si el bucle es incremental o decremental, la variable objetivo y por último la palabra clave **haz,** seguido del bloque que se hará en el bucle. Además el bucle aumenta o disminuye la variable de 1 en 1 lo que nos obliga a emplear variables numéricas exclusivamente para este bucle.

Un ejemplo de cabecera sería **para** variable := 0 **sube hasta** 10 **haz** , tras esto se incluiría un bloque de código. En este ejemplo se emplea un bloque incremental pero es posible usar uno decremental cambiando la palabra **sube** por **baja**.

Para el bucle **mientras** se emplea de la siguiente forma:
mientras (**condición**) {**bloque**} de forma que la condición es una operación lógica que emplea una variable y en bloque es lo que se desea repetir. Cabe destacar que a diferencia del bucle **para** este bucle no modifica la variable de la condición de forma automática sino que hay que hacerlo a mano, lo que además permite modificarlo de otras formas aparte de 1 en 1 o incluso permite el uso de otro tipo de variables.

Por último la sentencia **sí** que es simplemente
si (**condición**) **entonces {bloque} sino {bloque}**.

Por último una aclaración sobre el uso de las listas. Para emplear las listas basta con escribir **lista de** y a continuación incluir un tipo de dato básico, que será el tipo de valor que almacenará la lista. A la hora de declarar una lista es posible emplear agregados para su construcción, estos agregados funcionan de la siguiente forma: **lista de** entero = [0,1,2]; y se crearía una lista que contiene dichos elementos. Si no se emplean agregados para la creación de listas es necesario utilizar operadores para añadir dichos elementos de la misma forma en la que se emplean para su modificación.

6.- Identificación de los tokens con el máximo nivel de abstracción.

TOKEN	IDENTIFICADOR	PATRON	ATRIBUTOS
CARACTERES	256	([a-zA-Z])+	
NUMERO	257	([0-9])+	
INI_BLOQUE	258	“{“	
FIN_BLOQUE	259	“}”	
INI_AGRAGADO	260	“[“	
FIN_AGREGADO	261	“]”	
PUNTOYCOMA	262	“;”	
OP_ASIGNACION	263	“=”	
BUCLE_SI	264	“si”	
OP_ARITMETICA	265	(“+” | “-” | “*” | “/” | “%” | “@” | “--”)	
BUCLE_MIENTRAS	266	“mientras”	
OP_LOGICO	267	(“&&” | “||” | “y” | “o” | “xor”)	
OP_UNARIO	268	(“++” | “--” | “#” | “?” | “**” | “<<” | “>>” | “$”)	
BUCLE_PARA	269	“para”	
INI_PARENTESIS	270	“(“	
FIN_PARENTESIS	271	“)”	
SINO	272	“sino”	
ENTONCES	273	“entonces”	

DECL_LISTAS	274	“lista de”	
OP_UNI_BIN	275	(“no” | “!” | “-”)	
TIPO_VAR	276	(“bool”|”carácter”|”real”|” entero”)	0: bool 1: carácter 2: real 3: entero
ID	277		
DOSPUNTOS	278	“:”	
COMA	279	“,”	
LLEGADAFOR	280	“sube hasta” | “baja hasta”	0: sube hasta 1: baja hasta
FINFOR	281	“haz”	
CONSTANTE	282	“constante”	
INICOMILLAS	283	““	
FINCOMILLAS	284	““	