1. **Breve descripción del lenguaje**

Nuestro lenguaje está basado en C y se caracteriza por las siguientes propiedades:

* Es fuertemente tipado.
* Se permite la creación de bloques en cualquier sentencia.
* Las variables deben de ser declaradas al inicio de un bloque y antes de ser usadas.
* Dispone de funciones como subprogramas
* Las palabras reservadas pertenecen al castellano.
* Dispone de las estructuras de control **si-sino**, **mientras** y **repetir hasta**.
* Para la entrada de datos, se realizará mediante teclado. Se podrá guardar dicha información utilizando una única sentencia para leer en múltiples variables. En cuanto a la salida, se hará por pantalla mediante una lista con expresiones, constantes y cadenas.
* Los tipos de datos básicos son **carácter**, **booleano**, **entero** y **real**. Además se permite la creación de listas sobre estos tipos básicos siempre que no sea de manera recursiva (listas de otras listas).
* Cada lista dispone de tres sentencias especiales que permiten manejar cuál es el elemento actual de la lista (avanzar, retroceder y volver al comienzo).
* Las listas disponen de una serie de operadores especiales que permiten:
  + Conocer su longitud.
  + Conocer el elemento actual..
  + Obtener el elemento que se encuentra en una posición dada.
  + Añadir un elemento en una posición dada.
  + Borrar el elemento que se encuentra en una posición dada.
  + Borrar los datos de la lista a partir de una posición dada.
  + Concatenar dos listas.
  + Realizar la suma, resta, multiplicación y división de una lista con otra, realizando dicha operación con los elementos de cada posición de la lista.
  + Realizar la suma, resta, multiplicación y división de una lista con una constante, aplicando dicha constante como operando para cada uno de los elementos de la lista.

1. **Especificación del lenguaje 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\_subprograma>

| ε

<Declar\_subprograma> ::= <Cabecera\_subprograma> <bloque>

<Declar\_de\_variables\_locales> ::= <Marca\_ini\_declar\_variables>

<Variables\_locales>

<Marca\_fin\_declar\_variables>

| ε

<Marca\_ini\_declar\_variables> ::= ‘**declaracion**’

<Marca\_fin\_declar\_variables> ::= ‘**findeclaracion**’

<Cabecera\_programa> ::= ‘**principal**’

<Inicio\_de\_bloque> ::= ‘**{**‘

<Fin\_de\_bloque> ::= ‘**}**’

<Variables\_locales> ::= <Variables\_locales> <Cuerpo\_declar\_variables>

| <Cuerpo\_declar\_variables>

<Cuerpo\_declar\_variables> ::= <tipo> <lista\_id>‘**;**’

<lista\_id> ::= <identificador>

| <lista\_id> ‘**,**’ <identificador>

<Cabecera\_subprograma> ::= <tipo> <identificador> ‘**(**‘ <lista\_parametros> ‘**)**’

<lista\_parametros> ::= <parametro>

| <lista\_parametros>‘**,**’ <parametro>

| ε

<parametro> ::= <tipo> <identificador>

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

| <Sentencia>

<Sentencia> ::= <bloque>

| <sentencia\_asignacion>

| <sentencia\_if>

| <sentencia\_while> | <sentencia\_entrada>

| <sentencia\_salida>

| <sentencia\_repeat\_until>

| <sentencia\_return>

| <retroceder\_lista>

| <avanzar\_lista>

| <comienzo\_lista>

<retroceder\_lista> ::= <expresion> ‘**<<**’‘**;**’

<avanzar\_lista> ::= <expresion> ‘**>>**’‘**;**’

<comienzo\_lista> ::= ‘**$**’ <expresion>‘**;**’

<sentencia\_repeat\_until> ::= ‘**repetir**’ <Sentencia> ‘**hasta**’ ‘**(**’ <expresion> ‘**)**’ ‘**;**’

<sentencia\_return> ::= ‘**devolver**’ <expresion>‘**;**’

<sentencia\_asignacion> ::= <identificador> ‘**=**’ <expresion>‘**;**’

<sentencia\_if> ::= ‘**si**’ ‘**(**’ <expresion> ‘**)**’ <Sentencia>

| ‘**si**’ ‘**(**’ <expresion > ‘**)**’ <Sentencia> ‘**sino**’ <Sentencia>

<sentencia\_while> ::= ‘**mientras**’ ‘**(**’ <expresion> ‘**)**’ <Sentencia>

<sentencia\_entrada> ::= ‘**leer**’ <lista\_id>‘**;**’

<sentencia\_salida> ::= ‘**escribir**’ <lista\_expresiones\_o\_cadena>‘**;**’

<lista\_expresiones\_o\_cadena> ::= <expr\_cad> |

<lista\_expresiones\_o\_cadena> ‘**,**’ <expr\_cad>

<expr\_cad> ::= <expresion> | <cadena>

<cadena> ::= ‘**“**’<caracteres>‘**”**’

<caracteres> ::= <caracter> | <caracteres> <caracter>

<expresion> ::= ‘**(**‘ <expresion> ‘**)**’

| <op\_unario> <expresion>

| <expresion> <op\_binario> <expresion>

| <expresion> ‘**++**’ <expresion> ‘**@**’ <expresion>

| <identificador>

| <constante>

| <funcion>

| <agregado\_lista>

<funcion> ::= <identificador> **‘(’**<lista\_f>**‘)’**

| <identificador> **‘(’‘)’**

<op\_unario> ::= ‘**#**’ | ‘**?**’ | ‘**!**’ | ‘**-**’ | ‘**+**’

<op\_binario> ::= ‘**+**’ | ‘**-**’ | ‘**\***’ | ‘**/**’ | ‘**%**’ | ‘**\*\***’ | ‘**--**’ | ‘**@**’ | ‘**==**’ | ‘**!=**’ | ‘**<**’ | ‘**>**’ | ‘**<=**’ | ‘**>=**’ |

‘**&&**’ | ‘**||**’

<tipo> ::= <tipo\_basico> | ‘**lista de**’ <tipo\_basico>

<tipo\_basico> ::= ‘**entero**’ | ‘**booleano**’ | ‘**real**’ | **‘caracter**’

<constante> ::= <entero> | <booleano> | <real> | “**‘**”<caracter>“**’**”

<agregado\_lista> ::= ‘**[**’ <lista> ‘**]**’

<lista\_f> ::= <expresion> | <lista\_f> ‘**,**’ <expresion>

<lista> ::= <expresion> | <lista>‘**,**’ <expresion>

<identificador> ::= <caracter> | <identificador> <caracter> | <identificador> <digito> | <digito>

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

| <digito>

<real> ::= <entero>’**.**’<entero>

<caracter> ::= [‘**a**’-’**z**’] | [‘**A**’-’**Z**’] | <digito>

<digito> ::= [‘**0’**-’**9**’]

<booleano> ::= ‘**V**’ | ‘**F**’

1. **Descripción de la semántica del lenguaje**

Nuestro lenguaje al estar basado en C, presentará ciertas similitudes con el mismo:

* Para iniciar el programa, se definirá la cabecera del mismo (equivalente en C al **“int main”**) con la palabra reservada **“principal”**. A continuación de la misma se escribirá un bloque de código.
* Los bloques de código comenzarán con el carácter reservado **“{”** y finalizarán con el carácter reservado **“}”**. En su interior en el siguiente orden: declaración de variables locales, declaración de subprogramas y sentencias.
* La declaración de variables locales se realizará escribiendo al inicio la palabra reservada **“declaracion”**, a continuación se colocan tantas variables locales como se deseen y se finalizará la declaración con la palabra reservada **“findeclaracion”**.
  + Para la declaración de variables se separarán los distintos tipos y se pondrán de la siguiente forma **<tipo> <variable>;** o **<tipo> <variable1>,<variable2>,<...>,<variableN>;** si se tratasen de varias variables del mismo tipo. En el caso de querer usar varios tipos de variables se utilizará de la siguiente forma: **<tipo1> <variable1>; <tipo2> <variable2>;** .
* La declaración de los distintos subprogramas que contenga nuestro programa comenzará con el tipo de dato que devuelve dicho subprograma, seguido del identificador del subprograma y, entre paréntesis, los distintos argumentos de este, quedando como resultado **<tipo> <identificador> (<lista\_argumentos>)**. A continuación el subprograma contendrá un bloque de código.
  + La lista de argumentos se realizará como en C, se escribirá entre los dos paréntesis, indicando el tipo de dato y a continuación el identificador del mismo, quedando los distintos argumentos separados por una coma. Aunque dos datos sean del mismo tipo, hay que indicar el tipo en ambos. Un ejemplo del mismo sería **(<tipo> <argumento1>, <tipo> <argumento2>)** .
* Las sentencias pueden contener un bloque de código (como los descritos anteriormente) o algunas de las operaciones elementales como son “si, sino, lectura, escritura, repetir hasta, devolver”, operaciones elementales de listas y operador de asignación, al cual se le asigna una expresión.
  + Para las expresiones se definen las operaciones con operadores unarios, binarios y ternarios, además se definen los identificadores de variables, constantes y funciones.

**Palabras Tokens**

declaracion findeclaracion **|** {declaracion}

principal **{** **|** {findeclaracion}

**}** ; **|** {principal}

, **(** **|** { **{** }

**)** **>>** **|** { **}** }

**<<** $ **|** { **;** }

repetir hasta **|** { **(** }

devolver = **|** { **)** }

si sino **|** { **>>** , **<<** }

leer mientras **|** { $, devolver}

escribir “ **|** { repetir }

# ? **|** { hasta }

| = **|** { = }

+ \* **|** { si }

/ % **|** { sino }

\*\* -- **|** { mientras }

@ == **|** { leer }

!= < **|** { escribir }

> <= **|** { #, ?, ! }

>= && **|** { - , +}

|| **|** { \*, /,%,\*\*,--,==,!=,<,>,<=, >=, &&, || }

++ lista de **|** { @ }

entero booleano **|** { ++ }

real caracter **|** { lista de }

[ ] **|** { entero, booleano, real, caracter }

V F **|** {V,F,([0-9])+, (‘[a-z]|[A-Z] | [0-9]’) ,

([0-9]+.[0-9]+) }

([0-9])+ (‘[a-z]|[A-Z] | [0-9]’) **|** {([a-z] | [A-Z])([a-z] | [A-Z] | [0-9])+ }

([0-9]+.[0-9]+) **|** (“[a-z] | [A-Z] | [0-9]”)

(“[a-z] | [A-Z] | [0-9]”) **|** { **,** }

([a-z] | [A-Z])([a-z] | [A-Z] | [0-9])+ **|**