Sintaxis y Semántica de los Lenguajes

Trabajo Práctico Final

Equipo Docente:

- Andrés Pascal
- Claudia Álvarez

Integrantes:

- Estanislao Gadea
- Baltazar Franz
- Juan Bautista Bonato

Fecha de entrega: 19 de julio de 2023

Documentación del Lenguaje: “Carlos”.

Con el lenguaje interpretado Carlos, se pueden realizar operaciones aritméticas con variables de tipo reales y vectores con componentes de tipo reales, hacer bucles, imprimir en pantalla y recibir datos numéricos con la consola.
El sistema solo acepta variables con valores reales pero admite dentro de su código fuente cadenas que se imprimen a la hora de mostrar o recibir datos.
Soporta las operaciones de suma, resta, multiplicación, división, potencia y raíz.

Estructura de un programa básico escrito en Carlos:

- Se escribe la palabra reservada “program”.
- Si lo desea, posteriormente puede escribir el título del programa después de la palabra reservada “title” (el título debe ser una cadena).
- Posteriormente podemos definir las variables que vamos a usar en el programa, iniciando con la palabra reservada “def” seguida por una secuencia de las variables (o vectores) separadas por una coma y después de la última se termina con un punto y coma.
    - definir variable: se coloca un identificador.
    - definir array: se coloca el identificador del array y posteriormente la palabra reservada “array” seguida por la cantidad máxima de elementos del vector entre corchetes (id array [real*]) .
      * Se lee solo la parte entera.
- Ahora es cuando se empieza a definir el cuerpo del programa empezando con una llave abierta, a partir de ahí se puede escribir las sentencias de su programa cada una finalizando con un “;” (a excepción de la última). Las sentencias que puede hacer en el programa son:
    - Asignación (=)
        - Para variables reales solo se escribe el id, un “=” y posteriormente una expresión aritmética, sólo un número, una variable o la componente de un vector.
        - Para vectores se escribe el identificador seguido de la posición a la cual se le va a asignar una expresión aritmética, una variable o un vector con posición, entre corchetes, el cual debe ser un número entero.
    - Condicional (if)
        - Sirva para evaluar relaciones entre números reales y decidir si se ejecuta un bloque de código. Empieza con la palabra

reservada “if” seguido de una una secuencia de una o más declaraciones que pueden ser verdaderas o falsas separadas por operadores lógicos. Las declaraciones son valores u operaciones aritméticas las cuales están separadas por relacionales que determinan un valor verdadero o falso de la relación. Seguido de la condición se escribe el bloque de códigos (con las mismas características del cuerpo original) a ejecutar entre llaves si esta es verdadera

- Lectura (read):
  - Se coloca la palabra reservada “read”, se abre un paréntesis, se escribe una cadena que se imprimirá en pantalla, y luego de una coma se coloca el identificador de la variable a la que se le asigna el valor leído.
- Impresión (print):
  - Está compuesto por la palabra reservada “print”, un paréntesis abierto, y una cadena y/o variables separadas por una coma. La función es mostrar en pantalla.
- Ciclo (while):
  - Se coloca la palabra reservada seguida de una condición con un valor booleano. Después de esta se abre una llave dentro de la cual se escribirá un cuerpo el cual se ejecutará una y otro vez siempre y cuando la condición se cumpla.

Gramática del lenguaje:

<Lenguaje> ::= "Program" <Titulo> “def” <Definiciones> “;” "{"<Cuerpo>"}" | "Program" <Titulo> "{"<Cuerpo>"}"

<Titulo> ::= "title" "cad" ";" | e

<Definiciones> ::= "id" “,” <Definiciones> | "id" "array" "[" <OpArit> "]" "," <Definiciones> | "id" | "id" "array" "[" <OpArit> "]"

<Cuerpo> ::= <Sent> ";" <Cuerpo> | <Sent>

<Sent> ::= <Asig> | <Condi> | <Leer> | <Imprimir> | <Ciclo>

<Asig> ::= <Asig_V> | <Asig_N> | <Asig_VC>

<Asig_V> ::= "id" "[" <opArit> "]" "=" <opArit>

<Asig_N> ::= "id" "=" <opArit>

<Asig_VC> ::= “id” “=” “[“<Elem_V> “]”

<Elem_V> ::= <OpArit> “,” <Elem_V> | <OpArit>

<opArit> ::= <opArit> "+" <OA2> | <opArit> "-" <OA2> | <OA2>

<OA2> ::= <OA2> "*" <OA3> | <OA2> "/" <OA3> | <OA3>

<OA3> ::= <Potencia> | "id"<arreglo> | "Creal" | "(" <opArit> ")" | “-”<OA3>

<arreglo> ::= "[" <opArit> "]" | e

<Potencia> ::= "pot" "(" "<Num_p>" ")" | "root" "(" "<Num_p>" ")"

<Num_p> ::= <opArit> "," <opArit>

<Condi> ::= "If" <valor_B> "{"<Cuerpo>"}" | "If" <valor_B> "{"<Cuerpo>"}" “else” "{"<Cuerpo>"}"

<valor_B> ::= <Valor_B> “|” <OL2> | <OL2>

<OL2> ::= <OL2> "&" <OL3> | <OL3>

<OL3> ::= “!” <OL3> | <opArit> “Relacional” <opArit> | "{" <valor_B> "}"

<Leer> ::= "read" "(" “cad” “,” “id” ")"

<Imprimir> ::= "print" "(" <Mostrar> ")"

<Mostrar> ::= <OpArit> | "cad" | <Mostrar>, "cad" | <Mostrar>”,” <OpArit>

<Ciclo> ::= "while" "<valor_B>" "{" "<Cuerpo>" "}"

Gramática del lenguaje (LL1):

<Lenguaje> -> "Program" <L_2>

<L_2> -> <Titulo> <L_3>

<L_3> -> “def” <Definiciones> “;” "{"<Cuerpo>"}" | "{"<Cuerpo>"}"

<Titulo> -> "title" "cad" ";" | epsilon

<Definiciones> -> "id" <D_2>

<D_2> -> “,” <Definiciones> | "array" "[" <OpArit> "]" <D_3> | epsilon

<D_3> -> "," <Definiciones> | epsilon

<Cuerpo> ::= <Sent> <C2>

<C2> ::= ";" <Cuerpo> | epsilon

<Sent> ::= <Asig> | <Condi> | <Leer> | <Imprimir> | <Ciclo>

<Asig> ::= “id”<Asig_2>

<Asig_2> ::= "[" <opArit> "]" "=" <Asig_3> | “=” <Asig_3>

<Asig_3> ::= <opArit> | “[“<Elem_V> “]”

<Elem_V> ::= <OpArit> <Elem_V2>

<Elem_V2> ::= “,” <Elem_V> | epsilon

<opArit> ::= <OA2> <SOA>

<SOA> ::= “+”<OA2> <SOA> | “-”<OA2> <SOA> | epsilon

<OA2> ::= <OA3> <SOA2>

<SOA2> ::= "*" <OA3><SOA2> | "/" <OA3><SOA2> | epsilon

<OA3> ::= <Potencia> | "id"<arreglo> | "Creal" | "(" <opArit> ")" | “-”<OA3>

<arreglo> ::= "[" <opArit> "]" | epsilon

<Potencia> ::= "pot" "(" "<Num_p>" ")" | "root" "(" "<Num_p>" ")"

<Num_p> ::= <opArit> "," <opArit>

<Condi> ::= "If" <valor_B> "{"<Cuerpo>"}" <Otro>

<Otro> ::= “else” "{"<Cuerpo>"}" | epsilon

<valor_B> ::= <OL2> <SOL>

<SOL> ::= “|” <OL2><SOL> | epsilon

<OL2> ::= <OL3><SOL2>

<SOL2> ::= "&" <OL3><SOL2> | epsilon

<OL3> ::= “!” <OL3> | <opArit> “Relacional” <opArit> | "{" <valor_B> "}"

<Leer> ::= "read" "(" “cad” “,” “id” ")"

<Imprimir> ::= "print" "(" <Mostrar> ")"

<Mostrar> ::= <OpArit><SM> | "cad"<SM>

<SM> ::= ”,”<SM2> | epsilon

<SM2> ::= "cad"<SM> | <OpArit><SM>

<Ciclo> ::= "while" <valor_B> "{" "<Cuerpo>" "}"

-Para <Lenguaje> -> "Program" <L_2>
Primero ("Program" <L_2>) = "Program"
-Para <L_2> -> <Titulo> <L_3>:
Primero (<Titulo> <L_3>) =
Primero (<Titulo>) =
Primero ("title" "cad" ";") = “title”
Primero (epsilon) =
Primero (<L_3>) =
Primero (“def” <Definiciones> “;” "{"<Cuerpo>"}") = def
Primero ("{"<Cuerpo>"}") = {
-Para <L_3> -> “def” <Definiciones> “;” "{"<Cuerpo>"}":
Primero (“def” <Definiciones> “;” "{"<Cuerpo>"}") = def
-Para <L_3> -> "{"<Cuerpo>"}":
Primero ("{"<Cuerpo>"}") = {
-Para <titulo> -> "title" "cad" ";":
Primero ("title" "cad" ";") = “title”
-Para <titulo> -> epsilon:
Primero (epsilon) = epsilon
Siguiente (<titulo>) = def, {
Siguiente (<L_2>) =
Siguiente (<Lenguaje>) = $
-Para <Definiciones> -> "id" <D_2>:
Primero ("id" <D_2>) = id

-Para <D_2> -> “,” <Definiciones>:
Primero (“,” <Definiciones>) = ,
-Para <D_2> -> “array” "[" <OpArit> "]" <D_3>:
Primero (“array” "[" <OpArit> "]" <D_3>) = “array”
-Para <D_2> -> epsilon
Primero (epsilon) = epsilon
Siguiente (D_2) =
Siguiente (<Definiciones>) = ;
Siguiente (L_2) =
Siguiente (<Lenguaje>) = $
-Para <D_3> -> “,” <Definiciones>:
Primero (“,” <Definiciones>) = ,
-Para <D_3> -> epsilon:
Primero (epsilon) = epsilon
Siguiente (D_3)= ;,$
Siguiente (D_2) =
Siguiente (<Definiciones>) = ;
Siguiente (L_2) =
Siguiente (<Lenguaje>) = $
-Para <Cuerpo> -> <Sent> <C2>:
Primero (<Sent> <C2>) = id, if, read, print, while
Primero (<Sent>) =
Primero (<Asig>) = id
Primero (“id”<Asig_2>) = id
Primero (<Condi>) =
Primero ("If" <valor_B> "{"<Cuerpo>"}" <Otro>) = if
Primero (<Leer>) =
Primero ("read" "(" “cad” “,” “id” ")") = read
Primero (<Imprimir>) =
Primero ("print" "(" <Mostrar> ")") = print
Primero (<Ciclo>) =
Primero ("while" <valor_B> "{" "<Cuerpo>" "}") = while
-Para <C2> -> ";" <Cuerpo>;
Primero (";" <Cuerpo>) = ;
-Para <C2> -> epsilon;
Primero (epsilon) = epsilon
Siguiente (C2) = },$
Siguiente (<Cuerpo>) = }
Siguiente (<Ciclo>) =
Siguiente (<Otro>) =
Siguiente (<Condi>) =
Siguiente (<L_3>) =
Siguiente (<L_2>) =
Siguiente (<Lenguaje>) = $
-Para <Sent> -> <Asig>:
Primero (<Asig>) = id
Primero (“id”<Asig_2>) = id
-Para <Sent> -> <Condi>:

Primero (<Condi>) =
Primero ("If" <valor_B> "{"<Cuerpo>"}" <Otro>) = if
-Para <Sent> -> <Leer>:
Primero (<Leer>) =
Primero ("read" "(" “cad” “,” “id” ")") = read
-Para <Sent> -> <Imprimir>:
Primero (<Imprimir>) =
Primero ("print" "(" <Mostrar> ")") = print
-Para <Sent> -> <Ciclo>:
Primero (<Ciclo>) =
Primero ("while" <valor_B> "{" "<Cuerpo>" "}") = while
-Para <Asig> -> “id”<Asig_2>
Primero (“id”<Asig_2>) = id
-Para <Asig_2> -> "[" <opArit> "]" "=" <Asig_3>
Primero ("[" <opArit> "]" "=" <Asig_3>) = [
-Para <Asig_2> -> “=” <Asig_3>
Primero (“=” <Asig_3>) = “=”
-Para <Asig_3> -> <OpArit>:
Primero (<OpArit>) = pot,root,id,Creal,(,-
Primero (<OA2> <SOA>) =
Primero (<OA2>) =
Primero (<OA3> <SOA2>) =
Primero (<OA3>) =
Primero (<Potencia>) =
Primero ("pot" "(" "<Num_p>" ")") = pot
Primero ("root" "(" "<Num_p>" ")") = root
Primero ("id"<arreglo>) = id
Primero ("Creal") = Creal
Primero ("(" <opArit> ")") = (
Primero (“-”<OA3>) = -
-Para <Asig_3> -> “[“<Elem_V> “]”:
Primero (“[“<Elem_V> “]”) = [
-Para <Elem_V> -> <OpArit> <Elem_V2>:
Primero (<OpArit> <Elem_V2>) = pot,root,id,Creal,(,-
Primero (<OpArit>) = pot,root,id,Creal,(,-
Primero (<OA2> <SOA>) =
Primero (<OA2>) =
Primero (<OA3> <SOA2>) =
Primero (<OA3>) =
Primero (<Potencia>) =
Primero ("pot" "(" "<Num_p>" ")") = pot
Primero ("root" "(" "<Num_p>" ")") = root
Primero ("id"<arreglo>) = id
Primero ("Creal") = Creal
Primero ("(" <opArit> ")") = (
Primero (“-”<OA3>) = -
-Para <Elem_V2> -> “,” <Elem_V>:
Primero (“,” <Elem_V>) = ,

-Para <Elem_V2> -> epsilon:

Primero (epsilon) = epsilon

Siguiente (<Elem_V2>) =],;,},$

Siguiente (<Elem_V>) =]

Siguiente (<Asig_3>) =

Siguiente (<Asig_2>) =

Siguiente (<Asig>) =

Siguiente (<Sent>) = ;

Siguiente (<Cuerpo>) = }

Siguiente (<Ciclo>) =

Siguiente (<Otro>) =

Siguiente (<Condi>) =

Siguiente (<L_3>) =

Siguiente (<L_2>) =

Siguiente (<Lenguaje>) = $

-Para <OpArit> -> <OA2> <SOA>:

Primero (<OA2> <SOA>) = pot, root, id, (,-

Primero (<OA2>) =

Primero (<OA3> <SOA2>) =

Primero (<OA3>) =

Primero (<Potencia>) =

Primero ("pot" "(" "<Num_p>" ")") = pot

Primero ("root" "(" "<Num_p>" ")") = root

Primero ("id"<arreglo>) = id

Primero ("Creal") = Creal

Primero ("(" <opArit> ")") = (

Primero (“-”<OA3>) = -

-Para <SOA> -> “+”<OA2> <SOA>

Primero (“+”<OA2> <SOA>) = +

-Para <SOA> -> “-”<OA2> <SOA>

Primero (“-”<OA2> <SOA>) = -

-Para <SOA> -> epsilon:

Primero (epsilon) = epsilon

Siguiente (<SOA>) =), , ,; , },],&,|,{,Relacional, $

Siguiente (<OpArit>) =), , ,; , },],Relacional

Siguiente (Valor_V) = {

Siguiente (OL2) = |

Siguiente (OL3) = &

Siguiente (asig_2) =

Siguiente (Asig) =

Siguiente (Sent) = ;

Siguiente (<Cuerpo>) = }

Siguiente (<Ciclo>) =

Siguiente (<Otro>) =

Siguiente (<Condi>) =

Siguiente (<L_3>) =

Siguiente (<L_2>) =

Siguiente (<Lenguaje>) = $

-Para <OA2> -> <OA3> <SOA2>:
Primero (<OA3> <SOA2>) = pot, root, id, Creal, (, -
Primero (<OA3>) =
Primero (<Potencia>) =
Primero ("pot" "(" "<Num_p>" ")") = pot
Primero ("root" "(" "<Num_p>" ")") = root
Primero ("id"<arreglo>) = id
Primero ("Creal") = Creal
Primero ("(" <opArit> ")") = (
Primero (“-”<OA3>) = -
-Para <SOA2> -> "*" <OA3><SOA2>
Primero ("*" <OA3><SOA2>) = *
-Para <SOA2> -> "/" <OA3><SOA2>
Primero ("/" <OA3><SOA2>) = /
-Para <SOA2> -> epsilon:
Primero (epsilon) = epsilon
Siguiente (SOA2) = +,-,), ; , },],&,|,{,Relacional, $
Siguiente (OA2) = +,-,), ; , },],&,|,{,Relacional, $
Siguiente (<SOA>) =), ; , },],&,|,{,,Relacional, $
Siguiente (<OpArit>) =), ; , },],Relacional
Siguiente (Valor_V) = {
Siguiente (OL2) = |
Siguiente (OL3) = &
Siguiente (asig_2) =
Siguiente (Asig) =
Siguiente (Sent) = ;
Siguiente (<Cuerpo>) = }
Siguiente (<Ciclo>) =
Siguiente (<Otro>) =
Siguiente (<Condi>) =
Siguiente (<L_3>) =
Siguiente (<L_2>) =
Siguiente (<Lenguaje>) = $
-Para <OA3> -> <Potencial>:
Primero (<Potencia>) = pot, root
Primero ("pot" "(" "<Num_p>" ")") = pot
Primero ("root" "(" "<Num_p>" ")") = root
-Para <OA3> -> "id"<arreglo>:
Primero ("id"<arreglo>) = id
-Para <OA3> -> "Creal":
Primero ("Creal") = Creal
-Para <OA3> -> "(" <opArit> ")":
Primero ("(" <opArit> ")") = (
-Para <OA3> -> “-”<OA3>:
Primero (“-”<OA3>) = -
-Para <arreglo> -> "[" <opArit> "]":
Primero ("[" <opArit> "]") = [
-Para <arreglo> -> epsilon:

Primero (epsilon) = epsilon

Siguiente (<Arreglo>) = *, /, +,-,), ; , },],&,|,{,Relacional, $

Siguiente (<OA3>) = *, /, +,-,), ; , },],&,|,{,Relacional, $

Siguiente (SOA2) = +,-,), ; , },],&,|,{,Relacional, $

Siguiente (OA2) = +,-,), ; , },],&,|,{,Relacional, $

Siguiente (<SOA>) =), ; , },],&,|,{,,Relacional, $

Siguiente (<OpArit>) =), ; , },],Relacional

Siguiente (Valor_V) = {

Siguiente (OL2) = |

Siguiente (OL3) = &

Siguiente (asig_2) =

Siguiente (Asig) =

Siguiente (Sent) = ;

Siguiente (<Cuerpo>) = }

Siguiente (<Ciclo>) =

Siguiente (<Otro>) =

Siguiente (<Condi>) =

Siguiente (<L_3>) =

Siguiente (<L_2>) =

Siguiente (<Lenguaje>) = $

-Para <Potencial> -> "pot" "(" "<Num_p>" ")"

Primero ("pot" "(" "<Num_p>" ")") = pot

-Para <Potencial> -> "root" "(" "<Num_p>" ")"

Primero ("root" "(" "<Num_p>" ")") = root

-Para <Num_p> -> <opArit> "," <opArit>

Primero (<opArit> "," <opArit>) = pot,root,id,Creal,(,-

Primero (<OpArit>) = pot,root,id,Creal,(,-

Primero (<OA2> <SOA>) =

Primero (<OA2>) =

Primero (<OA3> <SOA2>) =

Primero (<OA3>) =

Primero (<Potencia>) =

Primero ("pot" "(" "<Num_p>" ")") = pot

Primero ("root" "(" "<Num_p>" ")") = root

Primero ("id"<arreglo>) = id

Primero ("Creal") = Creal

Primero ("(" <opArit> ")") = (

Primero (“-”<OA3>) = -

-Para <Condi> -> "If" <valor_B> "{"<Cuerpo>"}" <Otro>

Primero ("If" <valor_B> "{"<Cuerpo>"}" <Otro>) = if

-Para <Otro> -> “else” "{"<Cuerpo>"}":

Primero (“else” "{"<Cuerpo>"}") = else

-Para <Otro> -> epsilon:

Primero (epsilon) = epsilon

Siguiente (<Otro>) = ; , },$

Siguiente (<Condi>)=

Siguiente (<Sent>) = ;

Siguiente (<Cuerpo>) = }

Siguiente (<L_3>) =
Siguiente (<L_2>) =
Siguiente (<Lenguaje>) = $

-Para <valor_B> -> <OL2> <SOL>:
Primero (<OL2> <SOL>) = !, pot,root,id,Creal,(,-, {
Primero (<OL2>) =
Primero (<OL3><SOL2>) =
Primero (<OL3>) =
Primero (“!” <OL3>) = = !
Primero (<opArit> “Relacional” <opArit>) =
Primero (<OpArit>) = pot,root,id,Creal,(,-
Primero (<OA2> <SOA>) =
Primero (<OA2>) =
Primero (<OA3> <SOA2>) =
Primero (<OA3>) =
Primero (<Potencia>) =
Primero ("pot" "(" "<Num_p>" ")") = pot
Primero ("root" "(" "<Num_p>" ")") = root
Primero ("id"<arreglo>) = id
Primero ("Creal") = Creal
Primero ("(" <opArit> ")") = (
Primero (“-”<OA3>) = -
Primero ("{" <valor_B> "}") = {

-Para <SOL> -> “|” <OL2><SOL>:
Primero (“|” <OL2><SOL>) = |

-Para <SOL> -> epsilon:
Primero (epsilon) = epsilon
Siguiente (<SOL>) = {,}
Siguiente (<Valor_B>) = {,}

-Para <OL2> -> <OL3><SOL2>:
Primero (<OL3><SOL2>) = !, pot,root,id,Creal,(,-, {
Primero (<OL3>) =
Primero (“!” <OL3>) = = !
Primero (<opArit> “Relacional” <opArit>) =
Primero (<OpArit>) = pot,root,id,Creal,(,-
Primero (<OA2> <SOA>) =
Primero (<OA2>) =
Primero (<OA3> <SOA2>) =
Primero (<OA3>) =
Primero (<Potencia>) =
Primero ("pot" "(" "<Num_p>" ")") = pot
Primero ("root" "(" "<Num_p>" ")") = root
Primero ("id"<arreglo>) = id
Primero ("Creal") = Creal
Primero ("(" <opArit> ")") = (
Primero (“-”<OA3>) = -
Primero ("{" <valor_B> "}") = {

-Para <SOL2> -> "&" <OL3><SOL2>:

Primero ("&" <OL3><SOL2>) = &

-Para <SOL> -> epsilon:

Primero (epsilon) = epsilon

Siguiente (SOL2) =

Siguiente (OL2) = |,{,}

Siguiente (<SOL>) = {,}

Siguiente (<Valor_B>) = {,}

-Para <OL3> -> “!” <OL3>

Primero (“!” <OL3>) = = !

-Para <OL3> -> <opArit> “Relacional” <opArit>

Primero (<opArit> “Relacional” <opArit>) = pot,root,id,Creal,(,-

Primero (<OpArit>) = pot,root,id,Creal,(,-

Primero (<OA2> <SOA>) =

Primero (<OA2>) =

Primero (<OA3> <SOA2>) =

Primero (<OA3>) =

Primero (<Potencia>) =

Primero ("pot" "(" "<Num_p>" ")") = pot

Primero ("root" "(" "<Num_p>" ")") = root

Primero ("id"<arreglo>) = id

Primero ("Creal") = Creal

Primero ("(" <opArit> ")") = (

Primero (“-”<OA3>) = -

-Para <OL3> -> "{" <valor_B> "}"

Primero ("{" <valor_B> "}") = {

-Para <Leer> -> "read" "(" “cad” “,” “id” ")"

Primero ("read" "(" “cad” “,” “id” ")") = read

-Para <Imprimir> -> "print" "(" <Mostrar> ")":

Primero ("print" "(" <Mostrar> ")") = print

-Para <Mostrar> -> <OpArit><SM>:

Primero (<OpArit><SM>) = pot,root,id,Creal,(,-

Primero (<OpArit>) = pot,root,id,Creal,(,-

Primero (<OA2> <SOA>) =

Primero (<OA2>) =

Primero (<OA3> <SOA2>) =

Primero (<OA3>) =

Primero (<Potencia>) =

Primero ("pot" "(" "<Num_p>" ")") = pot

Primero ("root" "(" "<Num_p>" ")") = root

Primero ("id"<arreglo>) = id

Primero ("Creal") = Creal

Primero ("(" <opArit> ")") = (

Primero (“-”<OA3>) = -

-Para <Mostrar> -> "cad"<SM>

Primero ("cad"<SM>) = cad

-Para <SM> -> ”,”<SM2>:

Primero (”,”<SM2>) = ,

-Para <SM> -> epsilon:

Primero (epsilon) = epsilon

Siguiente (<SM>) =)

Siguiente (<Mostrar>) =)

-Para <SM2> -> "cad"<SM>:

Primero ("cad"<SM>) = cad

-Para <SM2> -> <OpArit><SM> :

Primero (<OpArit><SM>) =pot,root,id,Creal,(,-

Primero (<OpArit>) = pot,root,id,Creal,(,-

Primero (<OA2> <SOA>) =

Primero (<OA2>) =

Primero (<OA3> <SOA2>) =

Primero (<OA3>) =

Primero (<Potencia>) =

Primero ("pot" "(" "<Num_p>" ")") = pot

Primero ("root" "(" "<Num_p>" ")") = root

Primero ("id"<arreglo>) = id

Primero ("Creal") = Creal

Primero ("(" <opArit> ")") = (

Primero (“-”<OA3>) = -

-Para <Ciclo> -> "while" <valor_B> "{" "<Cuerpo>" "}"

Primero ("while" <valor_B> "{" "<Cuerpo>" "}") = while

- La TAS se encuentra en archivo aparte.

Semántica:

<Lenguaje> -> "Program" <L_2>

```
Procedure evalLenguaje(Var arbol:TApuntNodo; Var estado:tEstado);
Begin
   evalL_2(arbol^.hijos.elem[2], estado);
End;
```

<L_2> -> <Titulo> <L_3>

```
Procedure evalL_2(Var arbol:TApuntNodo; Var estado:tEstado);
Begin
   evalTitulo(arbol^.hijos.elem[1], estado);
   evalL_3(arbol^.hijos.elem[2], estado);
End;
```

<L_3> -> “def” <Definiciones> “;” "{"<Cuerpo>"}" | "{"<Cuerpo>"}"

```
Procedure evalL_3(Var arbol:TApuntNodo; Var estado:tEstado);

Var a:   boolean;
Begin
   a := True;
   Case arbol^.hijos.elem[1]^.simbolo Of
      Tdef:
            Begin
               evalDefiniciones(arbol^.hijos.elem[2], estado);
               evalCuerpo(arbol^.hijos.elem[5], estado, a);
            End;
      TLlav_ab:   evalCuerpo(arbol^.hijos.elem[5], estado,a);


   End;
End;
```

<Titulo> -> "title" "cad" ";" | epsilon

```
Procedure evalTitulo(Var arbol:TApuntNodo; Var estado:tEstado);
Begin

End;
```

<Definiciones> -> "id" <D_2>

```
Procedure evalDefiniciones(Var arbol:TApuntNodo; Var estado:tEstado);
Begin
   If arbol^.hijos.elem[1]^.simbolo = Tid Then
      Begin
```

```
        evalD_2(arbol^.hijos.elem[2], estado,arbol^.hijos.elem[1]^.lexema);
      End;
End;
```

<D_2> -> “,” <Definiciones> | "array" "[" <OpArit> "]" <D_3> | epsilon

```
Procedure evalD_2(Var arbol:TApuntNodo; Var estado:tEstado; lexemaId:String);
 //evalúa si lo que sigue es otra definicion o si la nueva variable es un tarray

Var
   tipo:   ttipo;
   indice:   real;
Begin
   tipo := Treal;
   If arbol^.hijos.cant > 0 Then
      Begin
         Case arbol^.hijos.elem[1]^.simbolo Of
            Tcoma:
                  Begin
                     agregarVar(estado, lexemaId, tipo);
                     evalDefiniciones(arbol^.hijos.elem[2], estado);
                  End;
            Tarray:
                   Begin
                      tipo := Tarreglo;
                      evalOpArit(arbol^.hijos.elem[3], estado, indice);
                      //aca modificamos el coso para pasar de real a byte
                      agregarArray(arbol,estado, lexemaId, floor(indice), tipo);
                      evalD_3(arbol^.hijos.elem[5], estado);

                   End;
         End;
      End
   Else

      agregarVar(estado, lexemaId, tipo);
End;
```

<D_3> -> "," <Definiciones> | epsilon

```
Procedure evalD_3(Var arbol:TApuntNodo; Var estado:tEstado);
Begin
   If arbol^.hijos.cant > 0 Then
      evalDefiniciones(arbol^.hijos.elem[2], estado);
End;
```

<Cuerpo> -> <Sent> <C2>

```
Procedure evalCuerpo(Var arbol:TApuntNodo; Var estado:tEstado; resultado:boolean);
Begin
   If resultado Then
      Begin
         evalSent(arbol^.hijos.elem[1], estado);
         evalC2(arbol^.hijos.elem[2], estado);
      End;
End;
```

<C2> -> ";" <Cuerpo> | epsilon

```
Procedure evalC2(Var arbol:TApuntNodo; Var estado:tEstado);

Var a :   boolean;
Begin
   If arbol^.hijos.cant > 0 Then
      evalCuerpo(arbol^.hijos.elem[2], estado,True);
End;
```

<Sent> -> <Asig> | <Condi> | <Leer> | <Imprimir> | <Ciclo>

```
Procedure evalSent(Var arbol:TApuntNodo; Var estado:tEstado);
Begin
   Case arbol^.hijos.elem[1]^.simbolo Of
      VAsig:   evalAsig(arbol^.hijos.elem[1], estado);
      VCondi:   evalCondi(arbol^.hijos.elem[1], estado);
      VLeer:   evalLeer(arbol^.hijos.elem[1], estado);
      VImprimir:   evalImprimir(arbol^.hijos.elem[1], estado);
      VCiclo:   evalCiclo(arbol^.hijos.elem[1], estado);
   End;
End;
```

<Asig> -> “id” <Asig_2>

```
Procedure evalAsig(Var arbol:TApuntNodo; Var estado:tEstado);
Begin
   If arbol^.hijos.elem[1]^.simbolo = Tid Then
      Begin
         evalAsig_2(arbol^.hijos.elem[2], estado,arbol^.hijos.elem[1]^.lexema);
      End;
End;
```

<Asig_2> -> "[" <opArit> "]" "=" <Asig_3> | “=” <Asig_3>

```
Procedure evalAsig_2(Var arbol:TApuntNodo; Var estado:TEstado; lexemaId:String);

Var indice,valor:   real;
```

```
Begin

   Case arbol^.hijos.elem[1]^.simbolo Of
      TCorr_ab :
                Begin
                   evalopArit(arbol^.hijos.elem[2], estado, indice);
                   evalAsig_3(arbol^.hijos.elem[5], estado, valor);
                   AsignarValor(lexemaId,estado,floor(indice),valor);
                End;

      TAsig :
              Begin
                 evalAsig_3(arbol^.hijos.elem[2], estado, valor);
                 AsignarValor(lexemaId,estado,0,valor);

              End;
   End;

End;
```

<Asig_3> -> <opArit> | "["<Elem_V> "]"

```
Procedure evalAsig_3(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado:real);
Begin
   If arbol^.hijos.elem[1]^.simbolo = TCorr_ab Then
      evalElem_V(arbol^.hijos.elem[2], estado)
   Else
      evalopArit(arbol^.hijos.elem[1], estado, resultado);
End;
```

<Elem_V> -> <OpArit> <Elem_V2>

```
Procedure evalElem_V(Var arbol:TApuntNodo; Var estado:tEstado);

Var resultado:   real;
Begin
   evalOpArit(arbol^.hijos.elem[1], estado, resultado);
   evalElem_V2(arbol^.hijos.elem[2], estado);
End;
```

<Elem_V2> -> "," <Elem_V> | epsilon

```
Procedure evalElem_V2(Var arbol:TApuntNodo; Var estado:tEstado);
Begin
   If arbol^.hijos.cant > 0 Then
      evalElem_V(arbol^.hijos.elem[2], estado);
End;
```

<opArit> -> <OA2> <SOA>

```
Procedure evalopArit(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado:real);
Var op1:   real;
Begin
   evalOA2(arbol^.hijos.elem[1], estado, op1);
   evalSOA(arbol^.hijos.elem[2], estado, op1, resultado);
End;
```

<SOA> -> “+”<OA2> <SOA> | “-”<OA2> <SOA> | epsilon

```
Procedure evalSOA(Var arbol:TApuntNodo; Var estado:tEstado;Var op1:real ;Var
resultado:real);
Var op2:   real;
Begin
   If  arbol^.hijos.cant > 0 Then
      Begin
         Case arbol^.hijos.elem[1]^.simbolo Of
            Tmas:
                  Begin
                     evalOA2(arbol^.hijos.elem[2], estado, op2);
                     op1 := op1+op2;
                     evalSOA(arbol^.hijos.elem[3], estado, op1,resultado);
                  End;
            Tmenos:
                   Begin
                      evalOA2(arbol^.hijos.elem[2], estado, op2);
                      op1 := op1-op2;
                      evalSOA(arbol^.hijos.elem[3], estado, op1,resultado);
                   End;


         End;
      End
   Else
      resultado := op1;
End;
```

<OA2> -> <OA3> <SOA2>

```
Procedure evalOA2(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado:real);

Var op1:   real;
Begin
   evalOA3(arbol^.hijos.elem[1], estado, op1);
   evalSOA2(arbol^.hijos.elem[2], estado, op1,resultado);
End;
```

<SOA2> -> "*" <OA3><SOA2> | "/" <OA3><SOA2> | epsilon

```
Procedure evalSOA2(Var arbol:TApuntNodo; Var estado:tEstado;Var op1:real; Var
resultado:real);
Var op2:   real;
Begin
   If  arbol^.hijos.cant > 0 Then
      Begin
         Case arbol^.hijos.elem[1]^.simbolo Of
            Tmultip:
                  Begin
                     evalOA3(arbol^.hijos.elem[2], estado, op2);
                     op1 := op1*op2;
                     evalSOA2(arbol^.hijos.elem[3], estado, op1,resultado);
                  End;
            Tdiv:
                 Begin
                    evalOA3(arbol^.hijos.elem[2], estado, op2);
                    If op2<>0 Then
                       Begin
                          op1 := op1/op2;
                       End
                    Else
                       Begin
                          CLRSCR;
                          WriteLn('ERROR MATEMATICO');
                       End;
                    evalSOA2(arbol^.hijos.elem[3], estado, op1,resultado);
                 End;


         End;
      End
   Else
      resultado := op1;
End;
```

<OA3> -> <Potencia> | "id"<arreglo> | "Creal" | "(" <opArit> ")" | “-”<OA3>

```
Procedure evalOA3(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado:real);
Begin
   Case arbol^.hijos.elem[1]^.simbolo Of
      VPotencia:   evalPot(arbol^.hijos.elem[1], estado, resultado);
      Tid:   evalArreglo(arbol^.hijos.elem[2], estado, resultado,arbol^.hijos.elem[1]^.lexema);
      TCreal:   resultado :=   transformacionReal(arbol^.hijos.elem[1]^.lexema);
```

```
      TParen_ab:   evalOpArit(arbol^.hijos.elem[2], estado, resultado);
      Tmenos:
            Begin
               evalOA3(arbol^.hijos.elem[2], estado, resultado);
               resultado := resultado * -1;
            End;
   End;


End;
```

<Potencia> -> "pot" "(" "<Num_p>" ")" | "root" "(" "<Num_p>" ")"

```
Procedure evalPot(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado:real);
Var base, exponente:   real;
   i:   byte;
Begin
   Case arbol^.hijos.elem[1]^.simbolo Of
      Tpot:
            Begin
               evalNum_p(arbol^.hijos.elem[3], estado, base, exponente);
               resultado:=power(base,exponente);
               {resultado := base;
               For i:= 1 To floor(exponente) Do
                  resultado := resultado * base; }
            End;
      Troot:
            Begin
               evalNum_p(arbol^.hijos.elem[3], estado, base, exponente);
               if (base=0) and (exponente<>0) then
                  resultado:=0
               else
                  resultado := Exp((1/exponente)*Ln(base));
            End;
   End;
End;
```

<arreglo> -> "[" <opArit> "]" | epsilon

```
Procedure evalArreglo(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado:real;
lexemaId:String);

Var indice:   real;
Begin
   If  arbol^.hijos.cant > 0 Then
      Begin
         evalopArit(arbol^.hijos.elem[2], estado, indice);
```

```
        resultado := ValorDe(estado,lexemaid,floor(indice));
      End
   Else
      resultado := ValorDe(estado,lexemaid,0);

End;
```

<Num_p> -> <opArit> "," <opArit> *// base, exponente*

```
Procedure evalNum_p(Var arbol:TApuntNodo; Var estado:tEstado; Var base:real; Var
exponente:real);
Begin
   evalOpArit(arbol^.hijos.elem[1], estado, base);
   evalOpArit(arbol^.hijos.elem[3], estado, exponente);
End;
```

<Condi> -> "If" <valor_B> "{"<Cuerpo>"}" <Otro>

```
Procedure evalCondi(Var arbol:TApuntNodo; Var estado:tEstado);
Var resultado:   boolean;
Begin
   evalValor_B(arbol^.hijos.elem[2], estado, resultado);
   evalCuerpo(arbol^.hijos.elem[4], estado, resultado);
   evalOtro(arbol^.hijos.elem[6], estado, resultado);
End;
```

<Otro> -> “else” "{"<Cuerpo>"}" | epsilon

```
Procedure evalOtro(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado: boolean);
Begin
   If arbol^.hijos.cant > 0 Then
      Begin
         If Not resultado Then
            evalCuerpo(arbol^.hijos.elem[3], estado, True);

      End;
End;
```

<valor_B> -> <OL2> <SOL>

```
Procedure evalValor_B(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado: boolean);

Var aux:   boolean;
Begin
   evalOL2(arbol^.hijos.elem[1], estado, aux);
   evalSOL(arbol^.hijos.elem[2], estado, aux ,resultado);
End;
```

<SOL> -> “|” <OL2><SOL> | epsilon

```
Procedure evalSOL(Var arbol:TApuntNodo; Var estado:tEstado;Var aux:boolean ;Var
resultado:   boolean);

Var r2:   boolean;
Begin
   If arbol^.hijos.cant > 0 Then
      Begin
         evalOL2(arbol^.hijos.elem[2], estado, r2);
         aux := aux Or r2;
         evalSOL(arbol^.hijos.elem[3], estado, aux,resultado);
      End
   Else
      resultado := aux;
End;
```

<OL2> -> <OL3><SOL2>

```
Procedure evalOL2(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado:   boolean);

Var aux:   boolean;
Begin
   evalOL3(arbol^.hijos.elem[1], estado, aux);
   evalSOL2(arbol^.hijos.elem[2], estado, aux,resultado);
End;
```

<SOL2> -> "&" <OL3><SOL2> | epsilon

```
Procedure evalSOL2(Var arbol:TApuntNodo; Var estado:tEstado;Var aux:boolean; Var
resultado:   boolean);
Var r2:   boolean;
Begin
   If arbol^.hijos.cant > 0 Then
      Begin
         evalOL3(arbol^.hijos.elem[2], estado, r2);
         aux := aux And r2;
         evalSOL2(arbol^.hijos.elem[3], estado, aux, resultado);
      End
   Else
      resultado := aux;
End;
```

<OL3> -> “!” <OL3> | <opArit> “Relacional” <opArit> | "{" <valor_B> "}"

```
Procedure evalOL3(Var arbol:TApuntNodo; Var estado:tEstado; Var resultado:   boolean);

Var
   val1,val2:   real;
```

```
Begin
   Case arbol^.hijos.elem[1]^.simbolo Of
      Tnot:
            Begin
               evalOL3(arbol^.hijos.elem[2],estado,resultado);
               resultado := Not resultado;
            End;
      VopArit:
             Begin
                evalOpArit(arbol^.hijos.elem[1],estado,val1);
                evalOpArit(arbol^.hijos.elem[3],estado,val2);
                Case arbol^.hijos.elem[2]^.lexema Of
                   '<':   resultado := (val1 < val2);
                   '>':   resultado := (val1 > val2);
                   '==':   resultado := (val1 = val2);
                   '<>':   resultado := (val1 <> val2);
                   '<=':   resultado := (val1 <= val2);
                   '>=':   resultado := (val1 >= val2);
                End;
             End;
      TLlav_ab:   evalValor_B(arbol^.hijos.elem[2],estado,resultado);
   End;
End;
```

<Leer> -> "read" "(" “cad” “,” “id” ")"

```
Procedure evalLeer(Var arbol:   TApuntNodo; Var estado:   tEstado);

Var valEscrito:   real;
Begin
   write(arbol^.hijos.elem[3]^.lexema);
   readln(valEscrito);
   AsignarValor(arbol^.hijos.elem[5]^.lexema, estado,0,valEscrito);
   //puede ser que haya un
End;
```

<Imprimir> -> "print" "(" <Mostrar> ")"

```
Procedure evalImprimir(Var arbol:TApuntNodo; Var estado:tEstado);
Begin

   evalMostrar(arbol^.hijos.elem[3], estado);

End;
```

<Mostrar> -> <OpArit><SM> | "cad"<SM>

```
Procedure evalMostrar(Var arbol:TApuntNodo; Var estado:tEstado);
Var resultado:   real;
Begin
   Case arbol^.hijos.elem[1]^.simbolo Of
      VopArit:
            Begin
               evalopArit(arbol^.hijos.elem[1], estado, resultado);
               writeln(resultado:15:3);
               evalSM(arbol^.hijos.elem[2], estado);
            End;
      Tcad:
          Begin
             writeln(arbol^.hijos.elem[1]^.lexema);
             evalSM(arbol^.hijos.elem[2], estado);
          End;
   End;
End;
```

<SM> -> ”,”<SM2> | epsilon

```
Procedure evalSM(Var arbol:TApuntNodo; Var estado:tEstado);
Begin

   If arbol^.hijos.cant > 0 Then
      evalSM2(arbol^.hijos.elem[2], estado);

End;
```

<SM2> -> "cad"<SM> | <OpArit><SM>

```
Procedure evalSM2(Var arbol:TApuntNodo; Var estado:tEstado);

Var resultado:   real;
Begin
   Case arbol^.hijos.elem[1]^.simbolo Of
      Tcad:
          Begin

             writeln(arbol^.hijos.elem[1]^.lexema);
             evalSM(arbol^.hijos.elem[2], estado);

          End;
      VOpArit:
            Begin

               evalopArit(arbol^.hijos.elem[1],estado,resultado);
               writeln(resultado:15:3);
```

```
                evalSM(arbol^.hijos.elem[2], estado);

            End;
    End;
End;
```

<Ciclo> -> "while" <valor_B> "{" "<Cuerpo>" "}"

```
Procedure evalCiclo(Var arbol:TApuntNodo; Var estado:tEstado);
Var resultado:   boolean;
Begin
   evalValor_B(arbol^.hijos.elem[2],estado,resultado);
   While resultado Do
      Begin
         evalCuerpo(arbol^.hijos.elem[4],estado,true);
         evalValor_B(arbol^.hijos.elem[2],estado,resultado);
      End;
End;
```