UTN CDU
Concepción del Uruguay

Proyecto Intérprete

SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

Trabajo Práctico

Integrantes:

- Forni, Lucas.
- Martinez, Mateo Samuel.
- Prediger, Enzo David.
- Quinteros, Elias.
- Trigos, Thomas.

Docentes:

- Pascal, Andres Jorge.
- Alvarez, Claudia Mabel.

Fecha de entrega: 05/08/2019

1.Documentación del programa: que hace y cómo se usa.

Cada programa debe comenzar llamando a la Función seguida de su nombre, entre paréntesis a continuación deben estar especificados los parámetros (ids) separados por coma. Luego se debe realizar una o más sentencias antes de concluir la función por un ‘Fin.‘.

Para utilizar este intérprete debemos escribir un código en un archivo de texto, luego abrir la aplicación INTÉRPRETE y colocar la dirección donde se encuentra el archivo de texto, en el archivo rar ya se adjunta un archivo Fuente.txt.

Definición de:

- Identificador (id): los identificadores deben estar conformados por una letra seguido de letras o números, no se tomarán en cuenta las mayúsculas como por ejemplo si escribimos número es lo mismo que escribamos NÚMERO.
- Constante real: están formados por los números reales.
- Constante cadena: estos deben seguir la siguiente forma debe comenzar y finalizar con el símbolo ' dentro de estos podremos escribir cualquier carácter que se desee.

Las posibles sentencias son:

- Asignación: Le da un valor a una variable determinada, la forma de la frase debe ser una Id seguida del operador ‘:=’ para asignar la expresión aritmética que se desee, una vez finalizada esta debe concluir con un punto.

- Escritura: Esta sentencia es requerida para mostrar por pantalla el valor de una variable como también una constante cadena. Para realizar esta misma debemos escribir en el código fuente la palabra reservada Mostrar seguida de paréntesis, dentro de los cuales pueden ir una constante cadena o un identificador, en el caso de querer mostrar una constante cadena y seguido un identificador, debemos escribir la constante cadena y el identificador separados por una coma (,), al final del paréntesis debe continuar un punto (.) para concluir esta sentencia.

- Lectura: La utilidad de esta sentencia es asignarle a un identificador un valor mediante la entrada por teclado, es decir, el usuario escribe el valor del identificador. Para realizar esta misma debemos escribir en el código fuente la palabra reservada Leer seguida de paréntesis, dentro de los cuales debe ir un identificador, al final del paréntesis debe continuar un punto (.) para concluir esta sentencia.

- Si: Este condicional debe estar seguido de una relación entre dos expresiones aritméticas , dos puntos (:), una o más sentencias y por último un ‘Fin.’, a no ser que se dé el caso de que se precise de otro condicional (‘sino’)

al que continuarán sentencias para luego terminar con ‘Fin.’. Será necesario para casos en los que se requiera de una diferenciación entre posibles alternativas que pueda tener el código.

- Para: Este ciclo debe comenzar con la palabra reservada ‘Para’ seguida de un identificador el cual se irá incrementando para poder controlar que finalice la sentencia en algún momento. Luego del identificador sigue el símbolo ‘=’ que le asigna al id una expresión aritmética que define el comienzo del ciclo, luego de la expresión aritmética se encuentra la palabra reservada ‘Hasta’ seguida de otra expresión aritmética que determina cuándo terminará el ciclo (la expresión aritmética que indica el comienzo del ciclo debe ser mayor que la que lo finaliza). Para especificar las sentencias que se encuentran el este ciclo se deben colocar dos puntos (:) luego de la expresión aritmética. La cantidad de sentencias que abarca el ciclo será determinado por un ‘Fin.’ que lo cerrara.

- Mientras: Esta estructura es llamada cuando se encuentra la palabra reservada ‘Mientras’ seguida de una o más expresiones aritméticas separadas por los operadores lógicos (And, Or). Para especificar las sentencias que se desea utilizar se deben colocar los dos puntos (:) luego de la última expresión aritmética, y una vez especificada esta se finaliza el ciclo con un ‘Fin.’ .

<u>Observaciones:</u>

1. No está definida la resta como operación, en su lugar la resta de dos números debe ser escrita como la suma de un número negativo y uno positivo, o viceversa como se quiera. Ejemplo
    a. -2 + 5 = 3. (suma de un número negativo con uno positivo).
    b. 5 + -2 = 3. (suma de un número positivo con uno negativo).
    c. -1 + -2 = -3. (suma de dos números negativos).
    d. 1 + 2 = 3. (suma de dos números positivos).

2. El programa no necesita instalación.

3. El compilador no diferencia entre mayúsculas y minúsculas.

4. Las operaciones aritméticas que se pueden realizar son: raiz, potencia, multiplicación, división, suma y se pueden llamar a funciones con sus parámetros.

5. La potencia y las raiz tienen prioridad respecto de todas las otras operaciones; la multiplicación y la división sobre la suma.
6. La raíz sólo puede ser cuadrada, y se llama anteponiendo a una expresión aritmética entre paréntesis, la palabra reservada ‘RAIZ’. Ejemplo
   RAIZ (4) = 2.
   RAIZ (2*2)= 2.

2. Definición de la sintaxis mediante una gramática en BNF

<Programa>::= <Programa> <Function> |<Function>
<Function>::= <Funcione> “**(**“ <Parámetros> “**)**” <Bloque>
<Funcione>::= “**Función**” “**Id**”
<Parámetros>::= <Parámetros> “**,**” “**Id**” | “**Id**”
<Parametros2>::= <Parametros2 “,” <EA> | <EA>
<Cuerpo>::= <Cuerpo> <Sentencia>|<Sentencia>
<Sentencia>::= <Asignación>| <Escritura> | <Lectura>| <If> | <For> | <While>
<Asignación>::= “**Id**” <OpAsig> <EA> “**.**”
<OpAsig>::= “**:=**”
<EA>::= <EA>“**+**”<EA>|<EA>“*****”<EA> |<EA>“**/**”<EA> |<EA>“**^**”<EA> |”√”<EA> |<D>
<D>::= “**(**“<EA>”**)**”| “**ConstanteReal**” |”**Id**” <R>
<R>::= (“<Parametros2>”)”|ε
<Escritura> ::= “ **Mostrar “ “ (“** <A> ”**)” “.”**
<A> ::= **“ConstanteCadena” <**B> | “**ID”**
<B>::= “,” “**ID”** |ε
<Lectura>::= “**Leer**” “**(**“ “**Id**” “**)**” “**.**”
<If>::= “**Si**” <Cond>“**:**” <Bloque>| “**Si**” <Cond> “**:**” <Cuerpo> “**Sino**” <Bloque>
<While>::= “**Mientras**” <Cond> “**:**” <Bloque>
<For>::= “**Para**” “**Id**” **“=”** <EA> “**HASTA**” <EA> “**:**” <Bloque>
<Cond>::= <EA> <OpRel> <EA> | <Cond> <OpLog> <Cond>
<OpRel>::= “**<**” | “**>**” | “**<>**” |”**>=**” |”**<=**” | “**=**”
<OpLog>::= “**and**”| “**or**”
<Bloque>::= <Cuerpo> “**Fin**” “**.**”

3. Gramática en LL(1) y TAS

Programa → Fuction Programa1

Programa1→ Fuction Programa1| Epsilon

Fuction → Funcióne **(**Parámetros**)** Bloque

Funcióne→ **Función id**

Parámetros→ **Id** Parametros1

Parametros1→ **, Id** Parametros1| Epsilon

Parámetros2→ EA Parametros3

Parametros3→ **,** EA Parametros3| Epsilon

Cuerpo → Sentencia Cuerpo1

Cuerpo1→ Sentencia Cuerpo1| Epsilon

Sentencia→ Asignación |Lectura |Escritura |If |while |For

Asignación→ **Id** OpAsig EA .

OpAsig→ **:=**

EA→ TH

H→ **+**TH|Epsilon

T → ZY

Y→ *****ZY|**/**ZY |Epsilon

Z → DX |√ DX

X→ **^**DX|Epsilon

D→ **ConstanteReal |ID** R **|(**EA**)**

R→ Epsilon | (Parametros2)

Lectura→ **Leer** (**Id**) **.**

Escritura → **Mostrar (**A**) .**

A → **ConstanteCadena** B |**ID**

B→ , **ID** | Epsilon

If → **Si** Cond : cuerpo if1

If1→ **Sino** Bloque | **Fin.**

While→ **Mientras** cond **:** bloque

For→ **Para Id** = EA **Hasta** EA : Bloque

Cond→ EE Cond1

Cond1→ OpLog Cond|Epsilon

EE→ EA P

P→ OpRel EA

OpRel→ **<|>|>=|<=|<>|=**

OpLog→ **And| Or**

Bloque→ Cuerpo **Fin .**

4. Descripción de la semántica asociada.

Programa → Fuction Programa1

EVALLAPRIMERA(ARBOL,ESTADO,LF)

CUERPO

EVALFUCTION(ARBOL^.HIJOS[1],ESTADO,LF)

EVALPROGRAMA1(ARBOL^.HIJOS[2],ESTADO,LF)

FIN

Programa1→ Fuction Programa1| Epsilon

EVALPROGRAMA1(ARBOL,ESTADO,LF)

CUERPO

SI ARBOL^.HIJOS[1]<>NIL ENTONCES

EVALFUCTION(ARBOL^.HIJOS[1],ESTADO,LF)

EVALPROGRAMA1(ARBOL^.HIJOS[2],ESTADO,LF)

FIN.

Fuction → Funcióne (Parámetros) Bloque

EVALFUCTION(ARBOL,ESTADO,LF)

CUERPO

SI EVALFUNCIONE(ARBOL^.HIJOS[1],ESTADO,LF)=TRUE ENTONCES

EVALBLOQUE(ARBOL^.HIJOS[5],ESTADO,LF)

FIN.

Funcióne→ Función id

EVALFUNCIONE(ARBOL,ESTADO,LF):BOOLEAN

VARIABLES

LEXEMA:CADENA

I:BYTE

```
CUERPO
        EVALFUNCIONE:=FALSE
        I:=1
        LEXEMA:=ARBOL^.HIJOS[2]^.LEXEMA
        MIENTRAS (I<=LF.TAM) AND (EVALFUNCIONE=FALSE)
                SI LF.INFO[I].LEXEMA=LEXEMA ENTONCES
                        EVALFUNCIONE:=TRUE
                SINO
                INCREMENTAR(I)
FIN.
```

Parámetros→ Id Parametros1

```
EVALPARAMETROS(ARBOL,ESTADO,LF,LP,I)
VAR OP:BYTE; DIR:TPUNTERO;
 CUERPO
   BUSCARLISTA(ESTADO,ARBOL^.HIJOS[1]^.LEXEMA,DIR)
   DIR^.INFO.LEXEMA:=LP[1]
   OP:=2.
   EVALPARAMETROS1(ARBOL^.HIJOS[2],ESTADO,LF,LP,I,OP)
FIN.
```

Parametros1→ , Id Parametros1| Epsilon

```
EVALPARAMETROS1(ARBOL,ESTADO,LF,LP,I,OP)
 VAR DIR:TPUNTERO
CUERPO
 SI ARBOL^.HIJOS[1]<>NIL
   BUSCARLISTA(DIR,ARBOL^.HIJOS[2]^.LEXEMA,DIR)
    SI OP<= I
      DIR^.INFO.LEXEMA:=LP[OP]
```

```
  INC(OP)
  EVALPARAMETROS1(ARBOL^.HIJOS[3],ESTADO,LF,LP,I,OP)
FIN.
```

Parámetros2→ EA Parametros3

```
EVALPARAMETROS2(ARBOL,ESTADO,LF,LP,I)
VAR VALOR:REAL
CUERPO
  EVALEA(ARBOL^.HIJOS[1],ESTADO,LF,VALOR)
  CARGARPARAMETROS(LP,VALOR,I)
  EVALPARAMETROS3(ARBOL^.HIJOS[2],ESTADO,LF,LP,I)
END.
```

Parametros3→ , EA Parametros3| Epsilon

```
EVALPARAMETROS3(ARBOL,ESTADO,LF, LP,I)
VALOR:REAL
CUERPO
 SI ARBOL^.HIJOS[1]<>NIL
  EVALEA(ARBOL^.HIJOS[2],ESTADO,LF,VALOR)
  CARGARPARAMETROS(LP,VALOR,I)
  EVALPARAMETROS3(ARBOL^.HIJOS[3],ESTADO,LF,LP,I)
FIN.
```

Cuerpo→ Sentencia Cuerpo1

```
EVALCUERPO(ARBOL,ESTADO,LF)

CUERPO

        EVALSENTENCIA(ARBOL^.HIJOS[1],ESTADO,LF)

        EVALCUERPO1(ARBOL^.HIJOS[2],ESTADO,LF)

FIN.
```

Cuerpo1→ Sentencia Cuerpo1| Epsilon

```
EVALCUERPO1(ARBOL,ESTADO,LF)

CUERPO

        SI ARBOL^.HIJOS[1]<>NIL ENTONCES

                EVALSENTENCIA(ARBOL^.HIJOS[1],ESTADO,LF)

                EVALCUERPO1(ARBOL^.HIJOS[2],ESTADO,LF)

FIN.
```

Setencia→ Asignación |Lectura |Escritura |If |while |For

```
EVALSENTENCIA(ARBOL,ESTADO,LF)
CUERPO
        SI ARBOL^.HIJOS[1]^.VOT=LECTURA ENTONCES
                EVALECTURA(ARBOL^.HIJOS[1],ESTADO,LF)
        SINO SI ARBOL^.HIJOS[1]^.VOT=ESCRITURA ENTONCES
                EVALESCRITURA(ARBOL^.HIJOS[1],ESTADO,LF)
        SINO SI ARBOL^.HIJOS[1]^.VOT=ASIGNACION ENTONCES
                EVALASIGNACION(ARBOL^.HIJOS[1],ESTADO,LF)
        SINO SI ARBOL^.HIJOS[1]^.VOT=WHIL ENTONCES
                EVALWHILE(ARBOL^.HIJOS[1],ESTADO,LF)
        SINO SI ARBOL^.HIJOS[1]^.VOT=II ENTONCES
                EVALIF(ARBOL^.HIJOS[1],ESTADO,LF)
        SINO SI ARBOL^.HIJOS[1]^.VOT=FO ENTONCES
           EVALFOR(ARBOL^.HIJOS[1],ESTADO,LF)
FIN.
```

Asignación→ Id OpAsig EA .

```
EVALASIGNACION(ARBOL,ESTADO,LF)
VARIABLES
        VALOR:REAL
        DIR:TPUNTERO
CUERPO
        VALOR:=0;
        EVALEA(ARBOL^.HIJOS[3],ESTADO,LF,VALOR)
        BUSCARLISTA(ESTADO,ARBOL^.HIJOS[1]^.LEXEMA,DIR)
        DIR^.INFO.VALOR:=VALOR
```

```
FIN.
```

OpAsig→ :=

EA→ T H

```
EVALEA(ARBOL,ESTADO,LF,VALOR)
VARIABLES
        VALOR1:REAL
CUERPO
        EVALT(ARBOL^.HIJOS[1],ESTADO,LF,VALOR1)
        EVALH(ARBOL^.HIJOS[2],ESTADO,LF,VALOR,VALOR1)
FIN.
```

H→ +T H|Epsilon

```
EVALH(ARBOL,ESTADO,LF,VALOR,VALOR1)
VARIABLES
        VALOR2:REAL
CUERPO
        SI ARBOL^.HIJOS[1]=NIL ENTONCES
                VALOR:=VALOR1
        SINO
                EVALT(ARBOL^.HIJOS[2],ESTADO,LF,VALOR2)
                VALOR1:=VALOR1+VALOR2
                EVALH(ARBOL^.HIJOS[3],ESTADO,LF,VALOR,VALOR1)
FIN.
```

T → ZY

```
EVALT(ARBOL,ESTADO,LF,VALOR1)
VARIABLES
        VALOR2:REAL
```

```
CUERPO
        EVALZ(ARBOL^.HIJOS[1],ESTADO,LF,VALOR2)
        EVALY(ARBOL^.HIJOS[2],ESTADO,LF,VALOR1,VALOR2)
FIN.
```

Y→ *ZY|/ZY |Epsilon

```
EVALY(ARBOL,ESTADO,LF,VALOR1,VALOR2 )
VARIABLES
        VALOR3:REAL
CUERPO
        SI ARBOL^.HIJOS[1]=NIL ENTONCES
                VALOR1:=VALOR2
        SINO
                EVALZ(ARBOL^.HIJOS[2],ESTADO,LF,VALOR3)
                SI ARBOL^.HIJOS[1]^.VOT=MULT ENTONCES
                        VALOR2:=VALOR2*VALOR3
                SINO SI ARBOL^.HIJOS[1]^.VOT=DIB ENTONCES
                        VALOR2:=VALOR2/VALOR3
                EVALY(ARBOL^.HIJOS[3],ESTADO,LF,VALOR1,VALOR2)
FIN.
```

Z → DX |√ DX

```
EVALZ(ARBOO,ESTADO,LF,VALOR1)
 VAR VALOR:REAL
CUERPO
  SI ARBOL^.HIJOS^.VOT=D
    EVALD(ARBOL.HIJOS[1],ESTADO,LF,valor)
    EVALX(ARBOL.HIJOS[2],ESTADO,LF,VALOR1,VALOR)
```

```
  END
  SINO SI ARBOL^.HIJOS^.VOT=RAIZ
    EVALD(ARBOL^.HIJOS[2],ESTADO,LF,valor)
    VALOR:=SQRT(VALOR)
    EVALX(ARBOL^.HIJOS[3],ESTADO,LF,VALOR1,VALOR)
FIN.
```

X→ ^DX|Epsilon

```
PROCEDURE EVALX (ARBOL,ESTADO,LF, VALOR,VALOR1);
VAR
        I:INTEGER;VALOR2:REAL;
CUERPO
        SI ARBOL^.HIJOS[1]=NIL
          VALOR:=VALOR1;
        ELSE
           EVALD(ARBOL^.HIJOS[2],ESTADO,LF,VALOR2);
           I:=ROUND(VALOR2);
           VALOR1:=POTENCIA(VALOR1,I);
           EVALX(ARBOL^.HIJOS[3],ESTADO,LF,VALOR,VALOR1);
FIN
```

D→ ConstanteReal |ID R |(EA)

```
 EVALD(ARBOL,ESTADO,LF,VALOR1)
        NUMERO:REAL;
        DIR:TPUNTERO;
   BUSCADO:TLF;
CUERPO
        SI ARBOL^.HIJOS[1]^.VOT=CONSTANTEREAL
```

```
            VAL(ARBOL^.HIJOS[1]^.LEXEMA,NUMERO);
            VALOR1:=NUMERO;
        SINO SI ARBOL^.HIJOS[1]^.VOT=ID
            BUSCADO.LEXEMA:=ARBOL^.HIJOS[1]^.LEXEMA
            BUSCARLF(LF,BUSCADO)
            IF BUSCADO.LEXEMA<>' '
               EVALR(ARBOL^.HIJOS[2],ESTADO,LF,BUSCADO)
            BUSCARLISTA(ESTADO,ARBOL^.HIJOS[1]^.LEXEMA,DIR)
            VALOR1:=DIR^.INFO.VALOR
         SINO SI ARBOL^.HIJOS[1]^.VOT=PARENTESISA
             EVALEA(ARBOL^.HIJOS[2],ESTADO,LF,VALOR1)
FIN
```

R→ Epsilon | (Parametros2)

```
  EVALR(ARBOL, ESTADO,LF,)
CUERPO
    SI ARBOL.HIJOS [1]<>NILL
       EVALPARAMETROS2(ARBOL^.HIJOS[2],ESTADO,LF,LP,I)
       EVALPARAMETROS(BUSCADO.APUNTADOR^.HIJOS[3],ESTADO,LF,LP,I)
       EVALBLOQUE(BUSCADO.APUNTADOR^.HIJOS[5],ESTADO,LF)
FIN
```

Lectura→ Leer (Id) .

```
EVALECTURA(ARBOL,ESTADO,LF)
VARIABLES
        DIR:TPUNTERO
        XA:REAL
CUERPO
```

```
        BuscarLista(ESTADO,ARBOL^.HIJOS[3]^.LEXEMA,DIR)
        READ(XA)
        DIR^.INFO.VALOR:= XA
FIN.
```

Escritura → Mostrar (A) .

```
EVALESCRITURA(ARBOL,STADO,LF)
CUERPO
        EVALA(ARBOL^.HIJOS[3],ESTADO,LF)
FIN.
```

A → ConstanteCadena B |ID

```
EVALA(ARBOL,ESTADO,LF)
VARIABLES
        VALOR:REAL
        DIR:TPUNTERO
CUERPO
        SI ARBOL^.HIJOS[1]^.VOT=ID ENTONCES
                BUSCARLISTA(ESTADO,ARBOL^.HIJOS[1]^.LEXEMA,DIR)
                ESCRIBIR(DIR^.INFO.VALOR)
        SINO SI ARBOL^.HIJOS[1]^.VOT=CONSTANTECADENA ENTONCES
                SI EVALB(ARBOL^.HIJOS[2],ESTADO,LF,VALOR)=TRUE
ENTONCES
                        ESCRIBIR (ARBOL^.HIJOS[1]^.LEXEMA,VALOR)
                SINO ESCRIBIR(ARBOL^.HIJOS[1]^.LEXEMA)
FIN.
```

B→ , ID | Epsilon

```
EVALB(ARBOL,ESTADO,LF,VALOR):BOOLEAN
```

```
VARIABLES
        DIR:TPUNTERO
CUERPO
        SI ARBOL^.HIJOS[1]<>NIL ENTONCES
                BUSCARLISTA(ESTADO,ARBOL^.HIJOS[2]^.LEXEMA,DIR)
                VALOR:=DIR^.INFO.VALOR
                EVALB:=TRUE
        SINO EVALB:=FALSE
FIN.
```

If → Si Cond : cuerpo if1

```
 EVALIF(ARBOL,ESTADO,LF)
CUERPO
        SI  EVALCOND(ARBOL^.HIJOS[2],ESTADO,LF)= TRUE ENTONCES
                EVALCUERPO(ARBOL^.HIJOS[4],ESTADO,LF)
        SINO EVALIF1(ARBOL^.HIJOS[5],ESTADO,LF)
FIN.
```

If1→ Sino Bloque | Fin.

```
EVALIF1(ARBOL,ESTADO,LF)
CUERPO
        SI ARBOL^.HIJOS[1]<>NIL ENTONCES
            EVALBLOQUE(ARBOL^.HIJOS[2],ESTADO,LF)
FIN.
```

While→ Mientras cond : bloque

```
 EVALWHILE(ARBOL,ESTADO,LF)
VARIABLES
```

```
        A:BOOLEAN
CUERPO
        A:=EVALCOND(ARBOL^.HIJOS[2],ESTADO,LF)
        MIENTRAS (A=TRUE)
                EVALBLOQUE(ARBOL^.HIJOS[4],ESTADO,LF)
                A:=EVALCOND(ARBOL^.HIJOS[2],ESTADO,LF)
FIN.
```

For→ Para Id = EA Hasta EA : Bloque

```
PROCEDURE EVALFOR(ARBOL,ESTADO,LF)
VARIABLES
VALOR,VALOR1:REAL;DIR:TPUNTERO; I,W,K:INTEGER
CUERPO
  EVALEA(ARBOL^.HIJOS[4],ESTADO,LF,VALOR)
  BUSCARLISTA(ESTADO,ARBOL^.HIJOS[2]^.LEXEMA,DIR)
  EVALEA(ARBOL^.HIJOS[6],ESTADO,LF,VALOR1)
  DIR^.INFO.VALOR:=VALOR
  I := round(VALOR)
  W:=ROUND(VALOR1)
  PARA K:=I HASTA W
  DIR^.INFO.VALOR:=ROUND(K)
  EVALBLOQUE(ARBOL^.HIJOS[8],ESTADO,LF)
FIN.
```

Cond→ EE Cond1

```
EVALCOND(ARBOL,ESTADO,LF):BOOLEAN
VARIABLES
```

```
        VALOR:BOOLEAN
CUERPO
        VALOR:=EVALEE(ARBOL^.HIJOS[1],ESTADO,LF)
        EVALCOND:=EVALCOND1(ARBOL^.HIJOS[2],ESTADO,LF,VALOR)
FIN
```

Cond1→ OpLog Cond|Epsilon

```
EVALCOND1(ARBOL, ESTADO, LF, VALOR): BOOLEAN
CUERPO
        SI ARBOL^.HIJOS[1]=NIL ENTONCES
                EVALCOND1:=VALOR
        SINO SI ARBOL^.HIJOS[1]<>NIL ENTONCES
                SI ARBOL^.HIJOS[1]^.HIJOS[1]^.VOT= AN ENTONCES
                        SI (VALOR=TRUE) AND
(EVALCOND(ARBOL^.HIJOS[2],ESTADO,LF)=TRUE) ENTONCES
                         EVALCOND1:=TRUE
                         SINO
                                EVALCOND1:=FALSE
                SINO SI ARBOL^.HIJOS[1]^.HIJOS[1]^.VOT = O ENTONCES
                        SI (VALOR=TRUE) OR
(EVALCOND(ARBOL^.HIJOS[2],ESTADO,LF)=TRUE) ENTONCES
                                EVALCOND1:=TRUE
                        SINO EVALCOND1:=FALSE
FIN.
```

EE→ EA P

```
EVALEE(ARBOL,ESTADO,LF):BOOLEAN
VARIABLES
        VALOR:REAL
```

```
CUERPO
        EVALEA(ARBOL^.HIJOS[1],ESTADO,LF,VALOR)
        EVALEE:=EVALP(ARBOL^.HIJOS[2],ESTADO,LF,VALOR)
FIN.
```

P→ OpRel EA

```
EVALP(ARBOL,ESTADO,LF,VALOR):BOOLEAN
VARIABLES
        VALOR1:REAL
        A:BYTE
CUERPO
        EVALEA(ARBOL^.HIJOS[2],ESTADO,LF,VALOR1)
        A:=EVALOPREL(ARBOL^.HIJOS[1],ESTADO,LF)
        EVALP:=FALSE
        CASO DE:
                1:SI VALOR < VALOR1 ENTONCES
                                EVALP:=TRUE
                2:SI VALOR <= VALOR1 ENTONCES
                                EVALP:=TRUE
                3:SI VALOR <> VALOR1 ENTONCES
                                EVALP:=TRUE
                4: SI VALOR > VALOR1 ENTONCES
                                EVALP:=TRUE
                5:SI VALOR >= VALOR1 ENTONCES
                                EVALP:=TRUE
                6:SI VALOR = VALOR1 ENTONCES
                                EVALP:=TRUE
```

```
                SINO EVALP:=FALSE
FIN.
```

OpRel <|>|>=|<=|<>|=

```
EVALOPREL (ARBOL,ESTADO,LF):BYTE
CUERPO
    SI ARBOL^.HIJOS[1]^.LEXEMA='<' ENTONCES
        EVALOPREL:=1
    SINO SI ARBOL^.HIJOS[1]^.LEXEMA='<='ENTONCES
        EVALOPREL:=2
    SINO SI ARBOL^.HIJOS[1]^.LEXEMA='<>' ENTONCES
        EVALOPREL:=3
    SINO SI ARBOL^.HIJOS[1]^.LEXEMA='>' ENTONCES
        EVALOPREL:=4
    SINO SI ARBOL^.HIJOS[1]^.LEXEMA='>=' ENTONCES
        EVALOPREL:=5
    SINO SI ARBOL^.HIJOS[1]^.LEXEMA='=' ENTONCES
        EVALOPREL:=6
FIN.
```

Bloque Cuerpo Fin .

```
EVALBLOQUE(ARBOL,ESTADO,LF)
CUERPO
    EVALCUERPO(ARBOL^.HIJOS[1],ESTADO,LF)
FIN.
```

6. Programa que calcula el minimo comun multiplo entre dos numeros ingresados por pantalla.

```
Funcion MCM (I)
J:=1.
I:=1.
K:=1.
Z:=0.
Mostrar('INGRESE EL PRIMER NUMERO PARA CALCULAR MCM: ').
Leer(A).
Leer(B).
Mostrar('INGRESE EL SEGUNDO NUMERO PARA CALCULAR MCM: ').
Mientras I<= A*B and Z<> 1:
I:=A*K.
Mientras J <= A*B AND Z<>1:.
Si B*J=I:
Z:=1.
Fin.
J:=J+1.
Fin.
J:=1.
K:=K+1.
Fin.
Mostrar('EL MULTIPLO COMUN MINIMO ES: ', I).
Fin.
```

7. Programa que contenga una función que calcule el n-ésimo número de la sucesión de Fibonacci.

```
Funcion Fibonacci (N)
Mostrar ('Ingrese la posición de la cual quiere conocer su numero en la sucesion de Fibonacci: ').
Leer(N).
A:=0.
B:=1.
n:=N+-1.
Para I=0 HASTA N:
 X:=A+B.
 A:=B.
 B:=X.
Fin.
Mostrar ('El numero de la sucesión de Fibonacci en la posición ingresada es: ', A).
Fin.
```