UTN CDU
Concepción del Uruguay

Universidad Tecnológica Nacional Facultad Regional Concepción del Uruguay

Ingeniería en Sistemas de Información

Sintaxis y semántica de los lenguajes de programación

Proyecto Final Integrador - Proyecto Intérprete

Profesores:

- Pascal, Andres.
- Alvarez, Claudia.

Alumnos:

- Davezac, Emmanuel.
- Rodriguez, Franco.
- Villanueva, Nicolás.

Índice

Consignas

PROYECTO INTEGRADOR
Sintaxis y Semántica de los Lenguajes

Objetivo Construir un Intérprete para el lenguaje que se especifica a continuación

Características del lenguaje

1. Un programa es un conjunto de funciones.
2. Una función posee un nombre, una serie de parámetros y un cuerpo.
3. El cuerpo es una lista de sentencias.
4. Una sentencia es una asignación, una lectura, una escritura, un IF-THEN-ELSE, una estructura CASE, un ciclo WHILE y un FOR (los nombres de las palabras reservadas pueden ser otros).
5. El lado derecho de una asignación es una expresión aritmética sobre números reales, incluyendo suma, resta, producto, división, potencia y raíz cuadrada.
6. Las variables no se declaran.
7. Una lectura contiene una cadena que se mostrará por pantalla y la variable a leer.
8. Una escritura contiene una cadena que se mostrará por pantalla y una expresión aritmética.
9. Las condiciones de las estructuras IF, WHILE y CASE deben permitir operadores lógicos (además de los relacionales).
10. La estructura CASE contiene una lista de pares Condición:Bloque, de tal manera de que la primera condición de la lista que se cumpla, tendrá como resultado la ejecución del Bloque.
11. La estructura FOR contendrá una variable, su valor inicial, y su valor final, y el bloque se ejecutará para todos los valores de la variable de control desde el valor inicial al final.
12. Se permiten hacer modificaciones o agregados a esta descripción, siempre que tengan su justificación.

Ejemplo de un programa (de acuerdo a la CFG que se defina podría variar la sintaxis)

```
Funcion Promedio(x y)
  ^(x+y)/2.
Fin.

Funcion Principal()
  Leer("Ingresar un número: ", Elem).
  I =0.
  Mientras Elem<0 & Elem<=30:
    Leer(Elem).
    Para J=1..10:
       Primero:=Elem + Promedio(Elem*J, 2).
    Fin.
    I=I+1.
  Fin.
  Casos
    I>10:
        I=I+5.
        Primero=45.
      Fin.
    I<8:
        I=I-3.
      Fin.
  Fin.
  Si Elem=0:
```

```
    Escribir("La cantidad de números ingresados fue: ", I).
  Sino
    Escribir("El primero ingresado fue: ", Primero).
  Fin.
Fin.
```

Actividades a realizar

1. Definición de la sintaxis mediante la CFG correspondiente.
2. Definición de los componentes léxicos (terminales de la CFG) mediante expresiones regulares (cuando su estructura lo justifique)
3. Especificación de la semántica asociada a cada variable de la CFG.
4. Autómatas determinísticos para los componentes léxicos complejos.
5. Intérprete:
    5.1. Analizador Léxico.
    5.2. Analizador Sintáctico.
    5.3. Evaluador (ejecuta el programa, en base al árbol de análisis sintáctico).
6. Escribir un programa en este lenguaje que calcule el mínimo común múltiplo entre dos números ingresados por pantalla.
7. Escribir un programa que contenga una función que calcule el n-ésimo número de la sucesión de Fibonacci.

Entregables

1. Documentación del programa: qué hace y cómo se usa.
2. Definición formal de la sintaxis mediante una gramática en notación BNF.
3. Gramática modificada LL(1) y TAS.
4. Descripción de la semántica asociada.
5. Programas fuente y ejecutable.
6. Programas escritos en este nuevo lenguaje, correspondientes a los puntos 6 y 7.

Observaciones:

- El proyecto debe estar entregado y aprobado para acceder a la Promoción o la aprobación por examen final.
- El programa puede realizarse en cualquier lenguaje imperativo. Los que tienen aprobada la materia Paradigmas de Programación pueden utilizar cualquier lenguaje de programación, bajo cualquier paradigma, respetando las características del paradigma utilizado.
- **Fecha de Vencimiento de la entrega para promoción**: lunes 29/07/2019 (ante algún retraso, consultar)
- No es necesario presentar el proyecto para regularizar. En tal caso, se debe presentar antes de rendir el final.

Documentación del programa

Introducción

El siguiente contenido es la documentación del proyecto final integrador de la cátedra **Sintaxis y semántica de los lenguajes** de la **Universidad Tecnológica Nacional- Facultad regional Concepción del Uruguay.**
Este programa es un intérprete, este programa lee y ejecuta programas escritos en un lenguaje definido por nosotros. Un intérprete funciona línea por línea, ejecutando instrucciones a medida que las encuentra en el código fuente, en lugar de compilar todo el código antes de la ejecución.
Este intérprete permite realizar tareas comunes de programación como asignación de valores a variables, lectura y escritura en pantalla, creación de ciclos (tanto mientras como para) y toma de decisiones con condicionales (si-sino). También incluye una estructura para manejar casos múltiples (Casos).

¿Cómo usarlo?

Instrucciones para utilizar este intérprete:

- Escribe tu código fuente:
    - Se recomienda utilizar un editor de código como :
        - Visual Studio Code,
        - Atom,
        - Vim
        - Notepad++,
        - Sublime Text,
        - Gedit u otros para redactar el código de tu programa.
    - Este archivo contendrá las instrucciones y lógica de tu programa.
    - Guarda el archivo con la extensión ".TXT" para que el intérprete lo reconozca como un archivo fuente.

- Una vez que hayas redactado el código de tu programa, puedes iniciar la ejecución abriendo el archivo ejecutable "Interprete.exe". Al hacerlo, se te solicitará ingresar la ruta del archivo fuente que contiene tu código. Simplemente proporciona la ubicación del archivo fuente, y el intérprete se encargará de procesar y ejecutar tu programa.
  ACLARACIÓN IMPORTANTE: tanto la ruta como nombre del archivo no debe contener espacios.

Características del Intérprete-Lenguaje

Entre las principales características del intérprete y el lenguaje del mismo tenemos:

- Cada programa debe iniciar invocando la función, seguida de su nombre y paréntesis, donde se especifican los parámetros (identificadores) separados por comas.Toda función debe contener al menos Posteriormente, se deben incluir una o más sentencias dentro de la función antes de finalizar con la instrucción 'Fin.'. Este formato asegura una estructura clara y ordenada para definir y ejecutar funciones en el programa.
- El intérprete no distingue entre mayúsculas y minúsculas, lo que significa que el Intérprete no es *Case Sensitive*.
- Tanto el archivo fuente como las carpetas que lo contienen no pueden contener espacios en sus nombres, en caso contrario el intérprete tendrá problemas para detectar la ruta del archivo fuente.
- El intérprete fue creado utilizando el lenguaje de programación Pascal, siguiendo el paradigma imperativo.
- El intérprete utiliza un analizador sintáctico descendente predictivo no recursivo para verificar que la sintaxis del código fuente sea correcta, este se construye a partir de una gramática independiente del contexto (CFG) del tipo LL(1).
- No se requiere instalación para utilizar el intérprete, solo basta con abrir el ejecutable.
- El lenguaje que hemos diseñado sigue el paradigma imperativo, lo que implica que las sentencias se ejecutan de manera secuencial, una tras otra, a menos que se encuentren estructuras de control

condicionales (si-sino y casos) o bucles (mientras y para). Esta característica fundamental proporciona una estructura lógica y ordenada para la ejecución de programas, permitiendo la creación de lógica condicional y repeticiones controladas.

- Las instrucciones disponibles para ejecutar en nuestro lenguaje abarcan diversas funciones, entre ellas:
    - asignación: es una instrucción que asigna o guarda un valor en una variable. Se utiliza el signo **:=** para indicar que el valor a la derecha del signo se asigna a la variable a la izquierda del signo. Por ejemplo, en la expresión x := 10., el valor 10 se asigna a la variable x.
    - escritura en pantalla: es una instrucción que muestra información o valores en la pantalla. Puede mostrar texto o números a través de la consola, permitiendo al programa comunicar resultados o mensajes al usuario. En nuestro lenguaje se utiliza una la palabra reservada **mostrar** seguido entre paréntesis el mensaje o resultado a mostrar en la consola. Por ejemplo, en la expresión Mostrar('Hola mundo'). El programa muestra en la consola el mensaje "Hola mundo".,
    - lectura de datos: es una instrucción que permite que un programa reciba información que el usuario escribe a través de la consola.En nuestro lenguaje se utiliza una la palabra reservada **leer** seguido entre el identificador de la variable cuyo valor queremos solicitar.
      Esta operación permite al usuario ingresar valores que el programa puede utilizar durante su ejecución. Por ejemplo, en la expresión Leer (X). El usuario le proporciona al programa el valor de la variable X.
    - estructuras condicionales (Si-Sino): es una estructura condicional en la programación que permite ejecutar diferentes bloques de código según una condición. Si la condición es verdadera, se ejecuta el bloque de código asociado al "si"; de lo contrario, se ejecuta el bloque de código asociado al "sino". En resumen, se toma una decisión basada en la evaluación de una condición, y se ejecuta el bloque correspondiente según el resultado de esa evaluación.,
    - condicionales múltiples (case):es una estructura en programación que permite tomar decisiones entre varias opciones. Se evalúa una expresión y se ejecuta el bloque de código asociado al caso que coincida con el resultado de esa expresión.en pocas palabras,es una forma de ejecutar diferentes bloques de código según el valor de una expresión.,
    - Bucle Mientras: es una estructura de control de flujo en programación que repite un bloque de código mientras una condición especificada sea verdadera. En cada iteración, la condición se evalúa antes de ejecutar el bloque de código, y el ciclo continúa hasta que la condición se vuelva falsa. y por último
    - bucle Para: es una estructura de control de flujo en programación diseñada para ejecutar un bloque de código un número específico de veces. El bucle proporciona una manera concisa de realizar iteraciones controladas y es particularmente útil cuando se conoce de antemano la cantidad de repeticiones deseadas.

  Este conjunto de sentencias ofrece una amplia versatilidad para la construcción y ejecución de programas, permitiendo a los usuarios implementar lógica, control de flujo y manipulación de datos de manera efectiva.
- Las operaciones aritméticas disponibles de este lenguaje abarcan suma(+), resta(@), producto(*), división(/), potenciación(^) y raíz cuadrada (raiz).
- Los parámetros de las operaciones aritméticas pueden ser funciones.

- Los operadores aritméticos siguen un orden de prioridad en este lenguaje. La potencia y la raíz cuadrada tienen la máxima prioridad con respecto a todos los demás operadores, mientras que el producto y la división tienen un nivel superior de prioridad en comparación con la suma y la resta. Este orden asegura que las operaciones se realicen de manera coherente y sigan las reglas establecidas para la aritmética.
- En este lenguaje, la operación de raíz está limitada a la raíz cuadrada, y su sintaxis se define como sigue: `raíz(Expresión Aritmética)`.
- Las operaciones lógicas disponibles en este lenguaje son "and" (también conocida como "si") y "or" (también conocida como "o") .

    - and (si): Devuelve true si ambas expresiones son verdaderas; de lo contrario, devuelve false. Ejemplo: A and B es verdadero sólo si tanto A como B son verdaderos.
    - or (o):Devuelve true si al menos una de las expresiones es verdadera; devuelve false si ambas son falsas.Ejemplo: A or B es verdadero si al menos A o B son verdaderos.
    - Estos operadores lógicos son esenciales para construir condiciones y expresiones lógicas en la programación, permitiendo tomar decisiones basadas en evaluaciones booleanas.
    - La operación lógica NOT(NO) no se encuentra disponible como operación propia, pero podemos negar los operadores relacionales para negar el condicional.
      por ejemplo:
      Not(a=b) es equivalente a la expresión a<>b.
- Los operadores relacionales se utilizan para comprar dos expresiones aritméticas, y devuelven un resultado booleano (`true` o `false`). Sean A y B dos expresiones aritméticas cualesquiera, los operadores relacionales disponibles en nuestro lenguaje son:
    - Igual (=):
        - Se utiliza para verificar si dos expresiones son iguales.
        - Ejemplo: `A = B` es verdadero si A es igual a B.
    - Distinto(<>):
        - Comprueba si dos expresiones son diferentes.
        - Ejemplo: `A <> B` es verdadero si A no es igual a B.
    - Mayor que (>):
        - Verifica si la expresión de la izquierda es mayor que la expresión de la derecha.
        - Ejemplo: ‘A > B’ es verdadero si A es mayor que B.
    - Menor que (<):
        - Comprueba si la expresión de la izquierda es menor que la expresión de la derecha.
        - Ejemplo: ‘A < B’ es verdadero si A es menor que B.
    - Mayor o igual que (>=):
        - Determina si la expresión de la izquierda es mayor o igual que la expresión de la derecha.
        - Ejemplo: ‘A >= B’ es verdadero si A es mayor o igual que B.
    - Menor o igual que (<=):
        - Indica si la expresión de la izquierda es menor o igual que la expresión de la derecha.
        - Ejemplo: ‘A <= B' es verdadero si A es menor o igual que B.
- Estos operadores relacionales son fundamentales para construir condiciones y expresiones de comparación , facilitando la toma de decisiones basada en la relación entre valores.
- Las expresiones de comparación construidas a partir de expresiones aritméticas y operadores relacionales se pueden combinar a través de los operadores lógicos.
- Las variables no se declaran.
- Una función debe tener al menos un parámetro.
- El lenguaje se reserva algunas palabras que no pueden ser utilizadas por el programador como identificadores.

Algunos conceptos importantes:

Con la finalidad de ayudar a entender a más profundidad la documentación de nuestro programa elaboramos una lista de conceptos importantes relacionados al mismo

- **Intérprete:**
  Un intérprete es un programa que ejecuta código fuente de alto nivel directamente, sin necesidad de compilar previamente. Analiza y ejecuta instrucciones línea por línea en tiempo real, es decir, mientras el software se está ejecutando, y actúa como una interfaz entre ese proyecto y el procesador.
- **Código fuente:**
  Se refiere al conjunto de instrucciones escritas por un programador en un lenguaje de programación específico (en este caso nuestro lenguaje). Es el texto legible por humanos que luego se traduce a lenguaje de máquina para que la computadora pueda ejecutar las tareas definidas por el programador.

- **Editor de código:**
  Es una herramienta de software diseñada para crear, modificar y dar formato a archivos de código fuente de programas informáticos. Los editores de código proporcionan un entorno de trabajo especializado para programadores, permitiéndoles escribir y editar código de manera eficiente. Algunos editores de código populares incluyen Visual Studio Code, Sublime Text, Atom, Vim y Emacs, entre otros.
- **Constante Real:**
  Es la representación de un número real, su formato es el siguiente:
  Signo Parte Entera . Parte Decimal.
  En primer lugar, se encuentra el "Signo", que indica si el número es positivo o negativo. A continuación, se presenta la "Parte Entera " del número. Posteriormente, se utiliza el carácter "punto" para separar la "Parte Entera" de la "Parte Decimal". Finalmente, se incorpora la "Parte Decimal", que representa la porción fraccionaria del número.
  Ejemplos
  - 7.0
  - -7.0
  - 2.73
  - -2.73
- **Constante Entera:**
  Es la representación de un número número entero, su formato es el siguiente:
  Signo Parte Entera
  Ejemplos
  - 7
  - -7

- **Constante cadena:**
  Es una entidad en programación que sirve para representar y manipular texto. Se define mediante el uso de comillas simples al inicio y al final. Aquí algunos ejemplos:
  - 'Hola Mundo'
  - 'Error 404 not found'

  Este formato con comillas simples asegura que el contenido entre ellas sea interpretado como texto en lugar de otros elementos del lenguaje de programación.

- **Identificador (ID):**
  Es un nombre definido por el programador para designar elementos específicos dentro de un programa, como variables o funciones. Estos nombres deben seguir un formato particular, comenzando con al menos una letra seguida de letras o números adicionales. Un identificador no debe contener espacios. Es importante destacar que el lenguaje no es Case Sensitive, lo que significa que no hay distinción entre letras mayúsculas y minúsculas. Por ejemplo, "IMANID" y "imanid" se consideran equivalentes. Este enfoque permite flexibilidad en la elección de identificadores, ya que se consideran iguales independientemente de la combinación de mayúsculas y minúsculas utilizada.
- **Palabra reservada:**
  son términos específicos del lenguaje de programación que tienen un significado predefinido y están reservados para funciones o acciones particulares. Estas palabras tienen un propósito específico dentro del lenguaje y no pueden ser utilizadas como identificadores. Las palabras reservadas son esenciales para la sintaxis y estructura del código, y su uso incorrecto puede generar errores.

Definición de la sintaxis en notación BNF

La notación BNF (Backus-Naur Form) es una forma de representar formalmente las reglas gramaticales de un lenguaje de programación

En la notación BNF, se definen las reglas gramaticales mediante producciones que indican cómo se pueden construir las diferentes construcciones del lenguaje. Estas producciones siguen una estructura de la forma:

<nombre_no_terminal> ::= <definición>

- **<nombre_no_terminal>**: Representa un símbolo no terminal, que es un concepto o estructura dentro del lenguaje.
- **::=**: Se lee como "se define como".
- **<definición>**: Describe cómo se forma o construye el símbolo no terminal utilizando terminales (símbolos finales) y otros símbolos no terminales.

Sintaxis de nuestro lenguaje en notación BNF:

```
<PROGRAMA>::= <PROGRAMA> <FUNCION> |<FUNCION>
<FUNCION>::= "funcion" "id" "(" <PARAMETROS_1> ")" <BLOQUE>
<PARAMETROS_1>::= <PARAMETROS_1> "," "id" | "id"
<CUERPO>::= <CUERPO> <SENTENCIA>|<SENTENCIA>
<BLOQUE>::= <CUERPO> "fin" "."
<SENTENCIA>::= <ASIGNACION>| <ESCRITURA> | <LECTURA>| <SI> | <PARA> | <MIENTRAS> |
<CASOS>
<ASIGNACION>::= "id" ":=" <EXP_ARIT> "."
<EXP_ARIT>::= <EXP_ARIT>"+"<EXP_ARIT>|<EXP_ARIT>"*"<EXP_ARIT>
|<EXP_ARIT>"/"<EXP_ARIT> |<EXP_ARIT>"^"<EXP_ARIT> |"raiz"<EXP_ARIT> |<D>
<A>::= "("<EXP_ARIT>")"| "constante_real" | "constante_entera" | "id" <B>
<B>::= "("<PARAMETROS_2>")"|ε
<PARAMETROS_2>::= <PARAMETROS_2 "," <EXP_ARIT> | <EXP_ARIT>
<ESCRITURA> ::= "escribir" " (" <C> ")" "."
<C> ::= "constante_cadena" <D> | "id"
<D>::= "," "id" |ε
<LECTURA>::= "leer" "(" "id" ")" "."
<SI>::= "Si" <CONDICION>":" <BLOQUE>| "si" <CONDICION> ":" <CUERPO> " sino" <BLOQUE>
<PARA>::= "para" "id" "=" <EXP_ARIT> "HASTA" <EXP_ARIT> ":" <BLOQUE>
<MIENTRAS>::= "mientras" <CONDICION> ":" <BLOQUE>
<CASOS> "casos" ":" <CASOS_2> "fin" "."
<CASOS_2>::= <CASOS_2> <CASO> | <CASO>
<CASO>::=<CONDICION>":" <BLOQUE>
<CONDICION>::= <EXP_ARIT> <OP_REL> <EXP_ARIT> | <CONDICION> <OP_LOG> <CONDICION>
<OP_REL>::= "<" | ">" | "<>" |">=" |"<=" | "="
<OP_LOG>::= "and"| "or"
```

Gramática en LL(1)

Transformamos nuestra la gramática de nuestro lenguaje,de manera que esta sea del tipo LL(1). Esto es especialmente útil para construir un analizador sintáctico predictivo.
Los analizadores sintácticos predictivos basados en LL(1) son más simples de implementar y eficientes en términos de tiempo de ejecución.

```
PROGRAMA → FUNCTION PROGRAMA_1
PROGRAMA_1→ FUNCTION PROGRAMA_1| epsilon
FUNCTION_1 → FUNCTION_2 ( PARAMETROS ) BLOQUE
FUNCTION_2 → funcion id
PARAMETROS→ id PARAMETROS_1
PARAMETROS_1→ , id PARAMETROS_1 | epsilon
BLOQUE→ CUERPO fin .
CUERPO → SENTENCIA CUERPO_1
```

```
CUERPO_1→ SENTENCIA CUERPO_1| epsilon
SENTENCIA→ ASIGNACION | LECTURA | ESCRITURA | IF_1 | WHILE_1 | FOR_1 | CASOS_1
ASIGNACION→ id OP_ASIG EA .
OP_ASIG→ :=
EA→ TH
H→ + TH | epsilon
T → ZY
Y→ * ZY | / ZY | epsilon
Z → DX | √ DX
X→ ^DX | epsilon
D→ constante_real |ID R |( EA )
R→ epsilon | ( PARAMETROS_2 )
PARAMETROS_2→ EA PARAMETROS_3
PARAMETROS_3→ , EA PARAMETROS_3 | epsilon
LECTURA→ leer ( id ) .
ESCRITURA → mostrar ( A ) .
A → constante_cadena B | id
B→ , id | epsilon
IF_1 → si COND : CUERPO IF_2
IF_2→ sino BLOQUE | fin.
CASOS_1 -> casos : CASOS_2 fin .
CASOS_2 -> CASO CASOS_3
CASOS_3 -> CASO CASOS_3 | epsilon
CASO -> COND : BLOQUE
WHILE_1→ mientras COND : BLOQUE
FOR_1→ para id = EA hasta EA : BLOQUE
COND→ EE COND_1
COND_1→ OP_LOG COND | epsilon
EE→ EA P
P→ OP_REL EA
OP_REL→ < | > | >= | <= | <> | =
OP_LOG→ And | Or
```

Leyenda
■ : Variables ■ : Terminales

Tabla de Análisis Sintáctico (TAS)

Definición de las entradas de la TAS. También se adjuntó la TAS en forma de hoja de calculo..

```
TAS[PROGRAMA, funcion]:='PROGRAMA1 FUCTION';

TAS[PROGRAMA1, funcion]:='PROGRAMA1 FUCTION';
TAS[PROGRAMA1, pesos]:='epsilon';
```

```
TAS[FUCTION, funcion]:='BLOQUE ) PARAMETROS ( FUNCIONE';
TAS[FUNCIONE, funcion]:='id funcion';

TAS[PARAMETROS, id ]:='PARAMETROS1 id';
TAS[PARAMETROS1, ) ]:='epsilon';
TAS[PARAMETROS1, ,] :='PARAMETROS1 id ,';

TAS[PARAMETROS_2, id ]:='PARAMETROS_3 EA';
TAS[PARAMETROS_2, ) ]:='PARAMETROS_3 EA';
TAS[PARAMETROS_2,raiz]:='PARAMETROS_3 EA';
TAS[PARAMETROS_2,constante_real]:='PARAMETROS_3 EA';

TAS[PARAMETROS_3, , ]:='PARAMETROS_3 EA ,';
TAS[PARAMETROS_3, )]:='epsilon';

TAS[CUERPO,id]:='CUERPO_1 SENTENCIA';
TAS[CUERPO,leer]:='CUERPO_1 SENTENCIA';
TAS[CUERPO,mostrar]:='CUERPO_1 SENTENCIA';
TAS[CUERPO,si]:='CUERPO_1 SENTENCIA';
TAS[CUERPO,mientras]:='CUERPO_1 SENTENCIA';
TAS[CUERPO,para]:='CUERPO_1 SENTENCIA';
TAS[CUERPO,casos]:='CUERPO_1 SENTENCIA';

TAS[CUERPO_1,id]:='CUERPO_1 SENTENCIA';
TAS[CUERPO_1,leer]:='CUERPO_1 SENTENCIA';
TAS[CUERPO_1,mostrar]:='CUERPO_1 SENTENCIA';
TAS[CUERPO_1,si]:='CUERPO_1 SENTENCIA';
TAS[CUERPO_1,mientras]:='CUERPO_1 SENTENCIA';
TAS[CUERPO_1,para]:='CUERPO_1 SENTENCIA';
TAS[CUERPO_1,casos]:='CUERPO_1 SENTENCIA';
TAS[CUERPO_1,fin]:='epsilon';
TAS[CUERPO_1,sino]:='epsilon';

TAS[SENTENCIA,id]:='ASIGNACION';
TAS[SENTENCIA,leer]:='LECTURA';
TAS[SENTENCIA,mostrar]:='ESCRITURA';
TAS[SENTENCIA,si]:='IF_1';
TAS[SENTENCIA,mientras]:='WHILE_1';
TAS[SENTENCIA,para]:='FOR_1';
TAS[SENTENCIA,casos]:='CASOS_1';

TAS[ASIGNACION,id]:='. EA OP_ASIG id';

TAS[OP_ASIG, := ]:=' := ';

TAS[EA,id]:='H T';
TAS[EA, ( ]:='H T';
TAS[EA, raiz ]:='H T';
```

```
TAS[EA, constante_real ]:='H T';

TAS[H, ) ]:='epsilon';
TAS[H, . ]:='epsilon';
TAS[H, : ]:='epsilon';
TAS[H, + ]:='H T +';
TAS[H, > ]:='epsilon';
TAS[H, < ]:='epsilon';
TAS[H, >= ]:='epsilon';
TAS[H, <= ]:='epsilon';
TAS[H, <> ]:='epsilon';
TAS[H,= ]:='epsilon';
TAS[H, and ]:='epsilon';
TAS[H, or ]:='epsilon';
TAS[H, hasta ]:='epsilon';
TAS[H, , ]:='epsilon';

TAS[T,id]:='Y Z';
TAS[T, ( ]:='Y Z';
TAS[T,raiz]:='Y Z';
TAS[T,constante_real]:='Y Z';

TAS[Y,)]:='epsilon';
TAS[Y, . ]:='epsilon';
TAS[Y, : ]:='epsilon';
TAS[Y,+]:='epsilon';
TAS[Y,* ]:='Y Z *';
TAS[Y, D IB]:='Y Z /';
TAS[Y, > ]:='epsilon';
TAS[Y, < ]:='epsilon';
TAS[Y, >= ]:='epsilon';
TAS[Y, <= ]:='epsilon';
TAS[Y, <> ]:='epsilon';
TAS[Y,= ]:='epsilon';
TAS[Y,and]:='epsilon';
TAS[Y,o]:='epsilon';
TAS[Y,hasta]:='epsilon';
TAS[Y,,]:='epsilon';

TAS[Z,id]:='X D';
TAS[Z, ( ]:='X D';
TAS[Z,raiz]:='X D raiz';
TAS[Z,constante_real]:='X D';

TAS[X,)]:='epsilon';
TAS[X, . ]:='epsilon';
TAS[X, : ]:='epsilon';
TAS[X,+]:='epsilon';
```

```
TAS[X,* ]:='epsilon';
TAS[X,DIB]:='epsilon';
TAS[X, P OTEN]:='X D POTEN';
TAS[X, > ]:='epsilon';
TAS[X, < ]:='epsilon';
TAS[X, >= ]:='epsilon';
TAS[X, <= ]:='epsilon';
TAS[X, <> ]:='epsilon';
TAS[X,= ]:='epsilon';
TAS[X,and]:='epsilon';
TAS[X,o]:='epsilon';
TAS[X,hasta]:='epsilon';
TAS[X,,]:='epsilon';
TAS[D,id]:='R id';
TAS[D, ( ]:=') EA (';
TAS[D,constante_real]:='constante_real';
TAS[R, ( ]:=') PARAMETROS_2 (';
TAS[R,) ]:='epsilon';
TAS[R, . ]:='epsilon';
TAS[R, : ]:='epsilon';
TAS[R,+]:='epsilon';
TAS[R,* ]:='epsilon';
TAS[R,/]:='epsilon';
TAS[R, ^]:='epsilon';
TAS[R, > ]:='epsilon';
TAS[R, < ]:='epsilon';
TAS[R, >= ]:='epsilon';
TAS[R, <= ]:='epsilon';
TAS[R, <> ]:='epsilon';
TAS[R,= ]:='epsilon';
TAS[R,and]:='epsilon';
TAS[R,o]:='epsilon';
TAS[R,hasta]:='epsilon';
TAS[R,,]:='epsilon';

TAS[LECTURA,leer]:='. ) id ( leer';

TAS[ESCRITURA,mostrar]:='. ) A ( mostrar';

TAS[A,id]:='id';
TAS[A,constante_cadena]:='B constante_cadena';

TAS[B,)]:='epsilon';
TAS[B,,]:='id ,';

TAS[IF_1,si]:='IF_2 CUERPO : COND si';

TAS[IF_2,sino]:='BLOQUE sino';
```

```
TAS[IF_2,fin]:='. fin' ;

TAS[CASOS_1,casos]:='. fin CASOS_2 : casos';

TAS[CASOS_2,id]:='CASOS_3 CASO';
TAS[CASOS_2, ( ]:='CASOS_3 CASO';
TAS[CASOS_2,raiz]:='CASOS_3 CASO';
TAS[CASOS_2,constante_real]:='CASOS_3 CASO';

TAS[CASOS_3,id]:='CASOS_3 CASO';
TAS[CASOS_3, ( ]:='CASOS_3 CASO';
TAS[CASOS_3,raiz]:='CASOS_3 CASO';
TAS[CASOS_3,constante_real]:='CASOS_3 CASO';
TAS[CASOS_3,fin]:='epsilon';

TAS[CASO,id]:='BLOQUE : COND';
TAS[CASO, ( ]:='BLOQUE : COND';
TAS[CASO,raiz]:='BLOQUE : COND';
TAS[CASO,constante_real]:='BLOQUE : COND';

TAS[WHILE_1,mientras]:='BLOQUE : COND mientras';

TAS[FOR_1,para]:='BLOQUE : EA hasta EA =  id para';

TAS[COND,id]:='COND1 EE';
TAS[COND, ( ]:='COND1 EE';
TAS[COND,raiz]:='COND1 EE';
TAS[COND,constante_real]:='COND1 EE';

TAS[COND1, : ]:='epsilon';
TAS[COND1,and]:='COND OPLOG';
TAS[COND1,o]:='COND OPLOG';

TAS[EE, id ]:='P EA';
TAS[EE, ( ]:='P EA';
TAS[EE, constante_real]:='P EA';
TAS[EE, raiz ]:='P E A ';

TAS[P, < ]:='EA OPREL';
TAS[P, > ] :='EA OPREL';
TAS[P, <= ]:='EA OPREL';
TAS[P, >= ]:='EA OPREL';
TAS[P, <> ]:='EA OPR E L';
TAS[P, = ]:='EA OPREL' ;

TAS[OPREL, < ]:='<';
TAS[OPREL, >] :='>';
TAS[OPREL, <= ]:='<=';
```

```
TAS[OPREL, >=]:='>=';
TAS[OPREL, <> ]:='<>';
TAS[OPREL, = ]:='=';

TAS[OPLOG,and]:='and';
TAS[OPLOG,o]:='o';

TAS[BLOQUE,id]:='. fin CUERPO' ;
TAS[BLOQUE,leer]:='. fin CUERPO' ;
TAS[BLOQUE,mostrar]:='. fin CUERPO';
TAS[BLOQUE,si]:='. fin CUERPO';
TAS[BLOQUE,mientras]:='. fin CUERPO';
TAS[BLOQUE,para]:='. fin CUERPO';
TAS[BLOQUE,casos]:='. fin CUERPO';
```

Descripción de la semántica asociada.

```
PROGRAMA → FUNCTION_1 PROGRAMA_1
EVALPROGRAMA(ARBOL,ESTADO,LF)
CUERPO
    EVALFUNCTION_1(ARBOL^.HIJOS[1],ESTADO,LF)
    EVALPROGRAMA_1(ARBOL^.HIJOS[2],ESTADO,LF)
FIN

PROGRAMA_1 → FUNCTION_1 PROGRAMA_1| epsilon
EVALPROGRAMA_1(ARBOL,ESTADO,LF)
CUERPO
    SI ARBOL^.HIJOS[1]<>NIL ENTONCES
        EVALFUNCTION_1(ARBOL^.HIJOS[1],ESTADO,LF)
        EVALPROGRAMA_1(ARBOL^.HIJOS[2],ESTADO,LF)
FIN.

FUNCTION_1 → FUNCTION_2 ( PARAMETROS ) BLOQUE
EVALFUNCTION_1(ARBOL,ESTADO,LF)
CUERPO
    SI EVALFUNCTION_2(ARBOL^.HIJOS[1],ESTADO,LF)=TRUE ENTONCES
    EVALBLOQUE(ARBOL^.HIJOS[5],ESTADO,LF)
FIN.

FUNCTION_2 → funcion id
EVALFUNCTION_2(ARBOL,ESTADO,LF):BOOLEAN
VAR
LEXEMA:CADENA
I:BYTE
CUERPO
    EVALFUNCTION_2:=FALSE
    I:=1
    LEXEMA:=ARBOL^.HIJOS[2]^.LEXEMA
    MIENTRAS (I<=LF.TAM) AND (EVALFUNCTION_2=FALSE)
```

```
        SI LF.INFO[I].LEXEMA=LEXEMA ENTONCES
            EVALFUNCTION_2:=TRUE
        SINO
            INCREMENTAR(I)
FIN.

PARAMETROS → id PARAMETROS_1
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.
    EVALPARAMETROS_1(ARBOL^.HIJOS[2],ESTADO,LF,LP,I,OP)
FIN.

PARAMETROS_1 → , id PARAMETROS_1| epsilon
EVALPARAMETROS_1(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)
    EVALPARAMETROS_1(ARBOL^.HIJOS[3],ESTADO,LF,LP,I,OP)
FIN.

PARAMETROS_2 → EA PARAMETROS_3
EVALPARAMETROS_2(ARBOL,ESTADO,LF,LP,I)
VAR VALOR:REAL
CUERPO
    EVALEA(ARBOL^.HIJOS[1],ESTADO,LF,VALOR)
    CARGARPARAMETROS(LP,VALOR,I)
    EVALPARAMETROS_3(ARBOL^.HIJOS[2],ESTADO,LF,LP,I)
FIN.



PARAMETROS_3 → , EA PARAMETROS_3| epsilon
EVALPARAMETROS_3(ARBOL,ESTADO,LF, LP,I)
VALOR:REAL
CUERPO
    SI ARBOL^.HIJOS[1]<>NIL
    EVALEA(ARBOL^.HIJOS[2],ESTADO,LF,VALOR)
    CARGARPARAMETROS(LP,VALOR,I)
    EVALPARAMETROS_3(ARBOL^.HIJO_S[3],ESTADO,LF,LP,I)
FIN.
```

```
CUERPO → SENTENCIA CUERPO_1
EVALCUERPO(ARBOL,ESTADO,LF)
CUERPO
    EVALSENTENCIA(ARBOL^.HIJOS[1],ESTADO,LF)
    EVALCUERPO_1(ARBOL^.HIJOS[2],ESTADO,LF)
FIN.

CUERPO_1 → SENTENCIA CUERPO_1| epsilon
EVALCUERPO_1(ARBOL,ESTADO,LF)
CUERPO
    SI ARBOL^.HIJOS[1]<>NIL ENTONCES
        EVALSENTENCIA(ARBOL^.HIJOS[1],ESTADO,LF)
        EVALCUERPO_1(ARBOL^.HIJOS[2],ESTADO,LF)
FIN.

SENTENCIA → ASIGNACION | LECTURA | ESCRITURA | IF_1 | WHILE_1 | FOR_1
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=WHILE_1 ENTONCES
    EVALWHILE__1_1(ARBOL^.HIJOS[1],ESTADO,LF)
    SINO SI ARBOL^.HIJOS[1]^.VOT=IF_1 ENTONCES
    EVALIF_1(ARBOL^.HIJOS[1],ESTADO,LF)
    SINO SI ARBOL^.HIJOS[1]^.VOT=FOR_1 ENTONCES
    EVALFOR_1(ARBOL^.HIJOS[1],ESTADO,LF)
FIN.

ASIGNACION → id OP_ASIG EA .
EVALASIGNACION(ARBOL,ESTADO,LF)
VAR
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)
VAR
```

```
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)
VAR
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)
VAR
    VALOR2:REALCUERPO
    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 )
VAR
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)FIN
    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;
    SINO
        EVALD(ARBOL^.HIJOS[2],ESTADO,LF,VALOR2);
        I:=ROUND(VALOR2);
        VALOR1:=POTENCIA(VALOR1,I);
        EVALX(ARBOL^.HIJOS[3],ESTADO,LF,VALOR,VALOR1);
FIN.

D→ constante_real |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 | (PARAMETROS_2)
```

```
EVALR(ARBOL, ESTADO,LF,)
CUERPO
    SI ARBOL.HIJOS [1]<>NILL
        EVALPARAMETROS_2(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)
VAR
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 → constante_cadena B |id
EVALA(ARBOL,ESTADO,LF)
VAR
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):BOOLEANVARIABLES
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_1 → si COND : CUERPO IF_2
EVALIF(ARBOL,ESTADO,LF)
CUERPO
    SI EVALCOND(ARBOL^.HIJOS[2],ESTADO,LF)= TRUE ENTONCES
        EVALCUERPO(ARBOL^.HIJOS[4],ESTADO,LF)
    SINO
        EVALIF_2(ARBOL^.HIJOS[5],ESTADO,LF)
FIN.

IF_2→ sino BLOQUE | Fin.
EVALIF_2(ARBOL,ESTADO,LF)
CUERPO
    SI ARBOL^.HIJOS[1]<>NIL ENTONCES
        EVALBLOQUE(ARBOL^.HIJOS[2],ESTADO,LF)
FIN.


WHILE_1 → mientras COND : BLOQUE
EVALWHILE__1(ARBOL,ESTADO,LF)
VAR
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_1 → para id = EA hasta EA : BLOQUE
PROCEDURE EVALFOR_1(ARBOL,ESTADO,LF)
VAR
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 COND_1
EVALCOND(ARBOL,ESTADO,LF):BOOLEAN
VARIABLESVALOR:BOOLEAN
CUERPO
    VALOR:=EVALEE(ARBOL^.HIJOS[1],ESTADO,LF)
    EVALCOND:=EVALCOND_1(ARBOL^.HIJOS[2],ESTADO,LF,VALOR)
FIN.

COND_1 → OP_LOG COND |epsilon
EVALCOND_1(ARBOL, ESTADO, LF, VALOR): BOOLEAN
CUERPO
    SI ARBOL^.HIJOS[1]=NIL ENTONCES
    EVALCOND_1:=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
    EVALCOND_1:=TRUE
    SINO
    EVALCOND_1:=FALSE
    SINO SI ARBOL^.HIJOS[1]^.HIJOS[1]^.VOT = O ENTONCES
    SI (VALOR=TRUE) OR
    (EVALCOND(ARBOL^.HIJOS[2],ESTADO,LF)=TRUE) ENTONCES
    EVALCOND_1:=TRUE
    SINO EVALCOND_1:=FALSE
FIN.

EE→ EA P
EVALEE(ARBOL,ESTADO,LF):BOOLEAN
VAR
VALOR:REAL
CUERPO
    EVALEA(ARBOL^.HIJOS[1],ESTADO,LF,VALOR)
    EVALEE:=EVALP(ARBOL^.HIJOS[2],ESTADO,LF,VALOR)
FIN.

P→ OP_REL EA
EVALP(ARBOL,ESTADO,LF,VALOR):BOOLEAN
VAR
VALOR1:REAL
A:BYTE
CUERPO
    EVALEA(ARBOL^.HIJOS[2],ESTADO,LF,VALOR1)
```

```
    A:=EVALOP_REL(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:=TRUESINO EVALP:=FALSE
FIN.

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

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

Autómatas finitos de los componentes complejos

A continuación se encuentran los autómatas finitos deterministicos (AFD) para reconocer los componentes léxicos más complejos. Se encuentran tanto el diagrama de estados como su función de transición.
Recordemos que:

- Q: Conjunto de estados del autómata.
- $Q_0 \in Q$: Estado inicial del autómata.
- $F \subseteq Q$: Subconjunto de estados finales o de aceptación del autómata.
- Σ Alfabeto del autómata o conjunto de símbolos de entrada.

Autómata finito para Constante Real

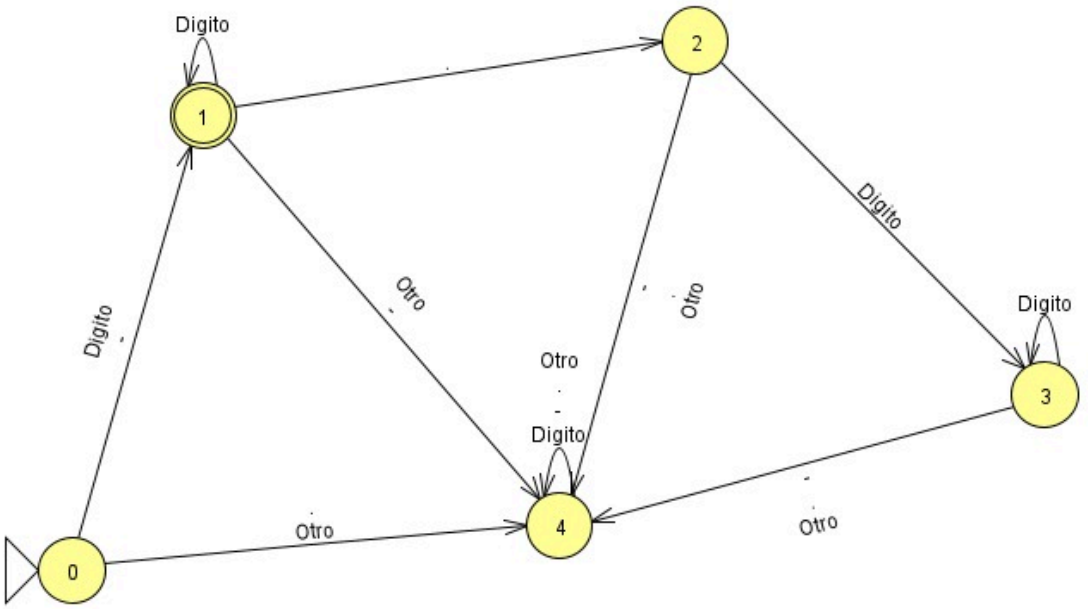
$Q = \{0,\ 1,\ 2,\ 3,\ 4\}$

$Q_0 = 0$

$F = \{1,\ 3\}$

$\Sigma = \{Dígito,\ \ -\ ,\ .\ ,\ Otro\}$

Función de transición				
	Dígito	**-**	**.**	**Otro**
0	1	1	4	4
1	1	4	2	4
2	3	4	4	4
3	3	4	4	4
4	4	4	4	4



Autómata finito para Identificador

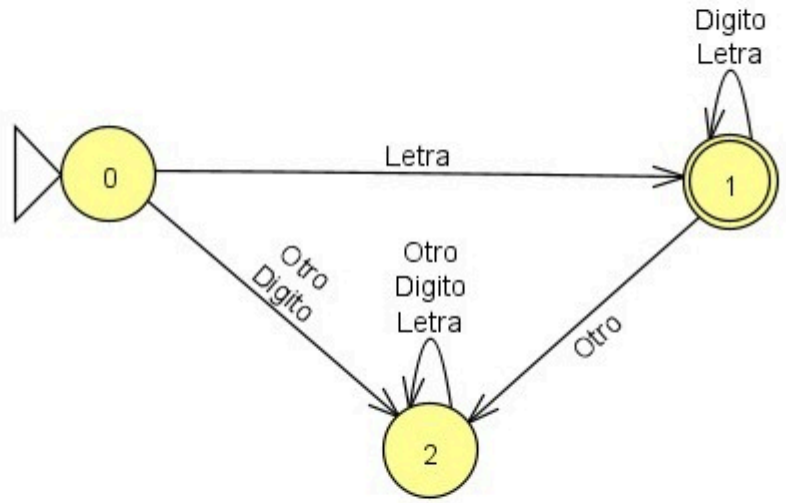
$Q = \{0,\ 1,\ 2\}$

$Q_0 = 0$

$F = \{1\}$

$\Sigma = \{Letra,\ \ Dígito,\ \ Otro\}$

Función de transición			
	Letra	**Dígito**	**Otro**
0	1	2	2
1	1	1	2
2	2	2	2



Autómata finito para Operador relacional

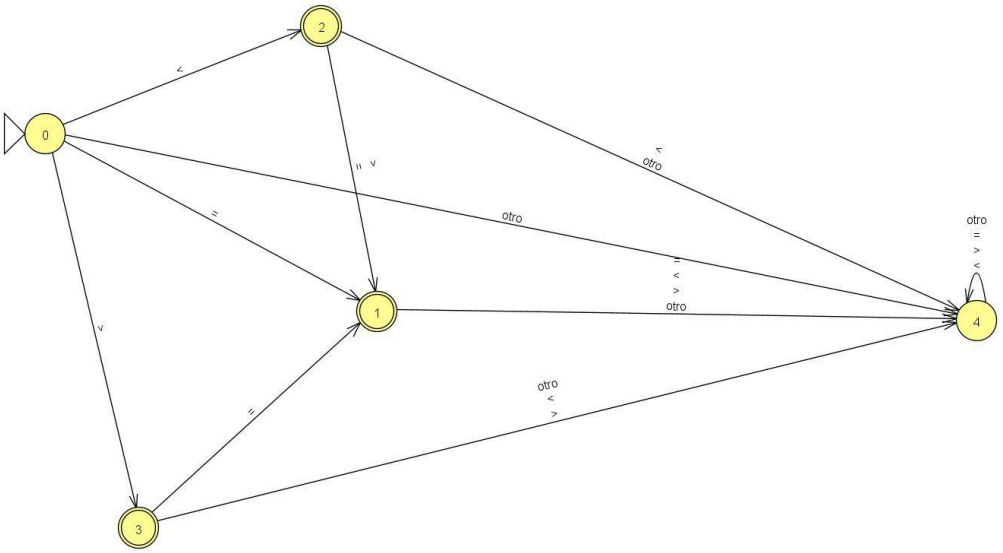
$Q = \{0, 1, 2, 3, 4\}$

$Q_0 = 0$

$F = \{1, 2, 3\}$

$\Sigma = \{>,\ <,\ =\ , otro\}$

Función de transición				
	<	>	=	otro
0	2	3	1	4
1	4	4	4	4
2	4	1	1	4
3	4	4	1	4
4	4	4	4	4



Autómata finito para Operador de asignación

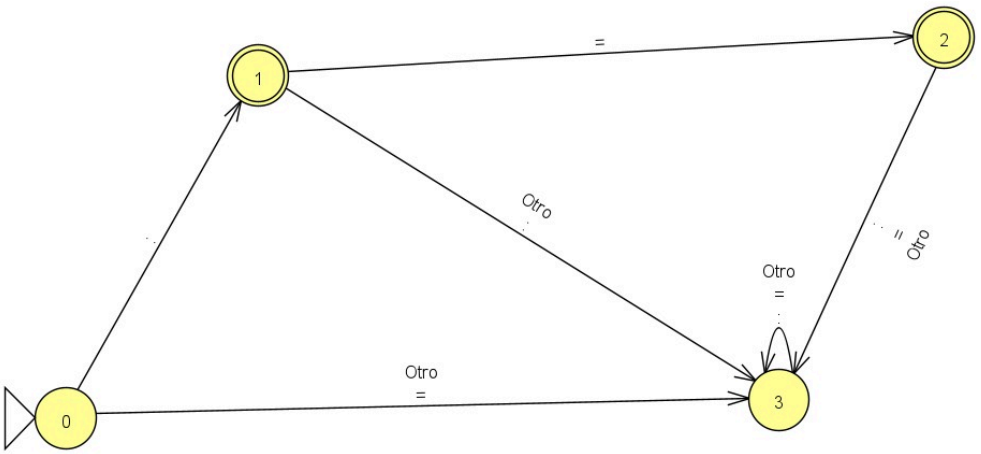
$Q = \{0,\ 1,\ 2,\ 3\}$

$Q_0 = 0$

$F = \{1,\ 2\}$

$\Sigma = \{:\ ,\ \ =,\ Otro\}$

Función de transición			
	:	**=**	**Otro**
0	1	3	3
1	3	2	3
2	3	3	3
3	3	3	3



Escribir un programa en este lenguaje que calcule el mínimo común múltiplo entre dos números ingresados por pantalla.

```
Funcion mcm (a , b)
    mostrar('ingrese el primer numero').
    leer(a).
    aux2:=a.
    mostrar('ingrese el segundo numero numero').
    leer(b).
    aux3:=b.
    mientras  b<> 0:
        q:=0.
        r:=0.
        aux:=a.
        mientras aux >= b :
            q:=q+1.
            aux:=aux + (b  * -1).
        fin.
        r:=a+( -1 * ( b*q)).
        a:=b.
        b:=r.
        Fin.
    mcm:=(aux2*aux3)/a.
    mostrar('El minimo comun multiplo es: ',mcm).
Fin.
```

Escribir un programa que contenga una función que calcule el n-ésimo número de la sucesión de Fibonacci.

```
Funcion Fibonacci (N)
    A:=0.
    B:=1.
    N:=N+(-1*1).
    Para I=0 HASTA N:
        C:=A+B.
        A:=B.
        B:=C.
        Fin.
    Fibonacci:=A.
Fin.

funcion main(N)
    Mostrar ('Ingrese la posicion del numero').
    Leer(N).
    C:=Fibonacci(N).
    Mostrar ('El resultado es: ', C).
fin.
```