PROYECTO FINAL SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

Integrantes
Bogado Magali
Gómez Noelia
Martinelli Iara
Dodera Sofía

27/07/2022

Contenido

Introducción

- Este proyecto consiste en la construcción de un Intérprete, el cual trabaja sobre un archivo de texto (.txt) -ubicado en la carpeta "Ejemplos"- e indica, a través de una serie de procedimientos, si el mismo respeta la estructura del lenguaje presentado.

Nombre del lenguaje → Byron

Sintaxis .

- El nombre del proyecto deberá estar escrito entre dos barras verticales (por ejemplo: |Sumatoria|), seguido de la palabra reservada "comienzo" para indicar el inicio del cuerpo principal. Este finaliza con "Final."

- Este archivo de texto contendrá una o más sentencias separadas por "."(punto).

- Estas sentencias pueden ser:
  - Asignación(id<<ExpArit1): el lado izquierdo de la sentencia asignación está formada únicamente por el terminal "id" que debe ser declarado previamente, el símbolo de asignación representado por "<<" y del lado derecho contiene una Expresión Aritmética que puede contener variables (id) y constantes reales.

  - Ciclo (Mientras Cond hacer '|'SecSent '|'): esta sentencia comienza con la palabra reservada "mientras", seguida de una condición determinada por el usuario, donde siempre que esta se cumpla, se dará paso a un bloque de sentencias, donde este mismo puede ser o no vacío.

  - Condicional (Si Q):el condicional comienza con la palabra reservada "si", seguida de cierta condición dada por el usuario, donde si esta se cumple, se da paso a determinado bloque de sentencias. Si esta o estas condiciones no se cumplen, se da lugar a la condicion dada por el "sino".

  - Lectura (cadena,ingresar(id)): la lectura en nuestro lenguaje está definida como→ una cadena (escrita entre comillas '""') seguida de una coma (,)y la instrucción de ingresar un identificador Ingresar(id).

  - Escritura (ImprimirR):esta sentencia comienza con la palabra reservada "Imprimir", seguida entre paréntesis por una

cadena o una ExpArit que puede ser o no vacía, separadas por coma (,).

- Definir (Define : Variables): la definición de variables comienza con la palabra reservada "define", seguida por dos puntos (:), y la o las variables a definir, separadas por una coma (,)

Aclaración: cuando se utilizan comillas dobles al nombrar palabras reservadas o variables, es solo a modo de referencia, no deben ser utilizadas a la hora de ejecutar un programa

Definición de la sintaxis mediante la CFG correspondiente

Gramática original

Programa → '|' id '|' **Cuerpo**.
Definir → Define: **Variables**
Variables → id**V**
V → ,id**V** | e

Cuerpo → Comienzo **SecSent** Final
SecSent → **SecSent Sentencia**. | ε
Sentencia → **Asignación** | **Ciclo** | **Condicional** | **Lectura** | **Escritura** | **Definir**

Asignación → id << **ExpArit1**
ExpArit1 → **ExpArit1** + **ExpArit2** | **ExpArit1** - **ExpArit2** | **ExpArit2**
ExpArit2 → **ExpArit2** * **ExpArit3** | **ExpArit2** / **ExpArit3** | **ExpArit3**
ExpArit3 → -**ExpArit3** | raiz(**ExpArit3**) | **ExpArit3**^**ExpArit3** | ConstReal | id | (**ExpArit1**)

Ciclo → Mientras **Cond** hacer '|' **SecSent** '|'
Condicional → Si **Cond** entonces '|' **SecSent** '|' | Si **Cond** entonces '|' **SecSent** '|' Sino '|' **SecSent** '|'

Cond → **Cond** and **ExpLog2** | **ExpLog2**
ExpLog2 → **ExpLog2** or **ExpLog3** | **ExpLog3**
ExpLog3 → not **ExpLog3** | **ExpRel**
ExpRel → **ExpArit1** opRel **ExpArit1** | [**Cond**]

Lectura → cadena, Ingresar(id)
Escritura → Imprimir(**ExpArit1**) | Imprimir(cadena**CAD**)
CAD → , cadena **CAD** | ε

Gramática en notación BNF

```
<Programa> ::= "|" "id" "|" <Cuerpo> .
<Definir> ::= "Define" ":" <Variables>
<Variables> ::= "id"<V>
<V> ::= ,"id"<V> | e

<Cuerpo>  ::= "Comienzo" <SecSent>  "Final"
<SecSent>  ::= <SecSent>  <Sentencia> "." | ε
<Sentencia>  ::= <Asignación>  | <Ciclo> | <Condicional> | <Lectura> | <Es-
critura> | <Definir>

<Asignación>  ::= "id" "<<" <ExpArit1>
<ExpArit1> ::= <ExpArit1> "+" <ExpArit2> | <ExpArit1> "-" <ExpArit2> |
<ExpArit2>
<ExpArit2> ::= <ExpArit2> "*" <ExpArit3> | <ExpArit2> "/" <ExpArit3> |
<ExpArit3>
<ExpArit3> ::= "-" <ExpArit3> | "raiz" "(" <ExpArit3> ")" | <ExpArit3> "^"
<ExpArit3> | "ConstReal" | "id" | "(" <ExpArit1> ")"

<Ciclo> ::= "Mientras" <Cond> "hacer" "|" <SecSent>  "|"
<Condicional> ::= "Si" <Cond>entonces "|" <SecSent>  "|" | "Si" <Cond> "en-
tonces" "|" <SecSent>  "|" "Sino" "|" <SecSent>  "|"

<Cond>::= <Cond> "and" <ExpLog2> | <ExpLog2>
<ExpLog2> ::= <ExpLog2> "or" <ExpLog3> | <ExpLog3>
<ExpLog3> ::= "not" <ExpLog3> | <ExpRel>
<ExpRel> ::= <ExpArit1> "opRel" <ExpArit1> | "[" <Cond> "]"

<Lectura> ::= "cadena" "," "Ingresar" "(" "id" ")"
<Escritura> ::= "Imprimir" "(" <ExpArit1> ")" | "Imprimir" "(" "cadena"
<CAD> ")"
<CAD> ::= "," "cadena" <CAD> | ε
```

Transformación de la gramática a LL(1)

Factorizar y sacar RI

```
Programa → '|' id '|' Cuerpo.
Definir → Define : Variables
Variables → idV
V → ,idV | e

Cuerpo → Comienzo SecSent Final
SecSent → SecSent Sentencia. | ε     (RI)
Sentencia → Asignación | Ciclo | Condicional | Lectura | Escritura | defi-
nir

Asignación → id << ExpArit1
ExpArit1 → ExpArit1 + ExpArit2 | ExpArit1 - ExpArit2 | ExpArit2 (RI)
ExpArit2 → ExpArit2 * ExpArit3 | ExpArit2 / ExpArit3 | ExpArit3 (RI)
ExpArit3 → raiz(ExpArit3) | ExpArit3^ExpArit3 | ConstReal | id | (ExpArit1)
(RI y ambiguedad)
```

Ciclo → Mientras **Cond** hacer '|' **SecSent** '|'
Condicional → Si **Cond** entonces '|' **SecSent** '|' | Si **Cond** entonces '|' **SecSent** '|' Sino '|' **SecSent** '|' (FI)

Cond → **Cond** and **ExpLog2** | **ExpLog2** (RI)
ExpLog2 → **ExpLog2** or **ExpLog3** | **ExpLog3** (RI)
ExpLog3 → not **ExpLog3** | **ExpRel**
ExpRel → **ExpArit1** opRel **ExpArit1** | [Cond]

Lectura → cadena, Ingresar(id)
Escritura → Imprimir(**ExpArit1**) | Imprimir(cadena**CAD**) (FI)
CAD → , cadena **CAD** | ε

LL1

Programa → '|' id '|' Cuerpo.

Definir → Define : Variables

Variables → idV

V → ,idV | e

Cuerpo → Comienzo SecSent Final

SecSent → Sentencia. SecSent | ε

Sentencia → Asignación | Ciclo | Condicional | Lectura | Escritura | Definir

Asignación → id << ExpArit1

ExpArit1 → ExpArit2 H

H → + ExpArit2 H | - ExpArit2 H | e

ExpArit2 → ExpArit3 J

J → * ExpArit3J | / ExpArit3J | e

ExpArit3 → ExpArit4L

ExpArit4→ -ExpArit4 | raiz(ExpArit1) | ConstReal | id | (ExpArit1)

L → ^ExpArit4L | e

Ciclo → Mientras Cond hacer '|' SecSent '|'

Condicional → Si Q

Q → Cond entonces '|' SecSent '|' A

A->Sino '|'SecSent '|' | e

Cond→ExpLog2K

K→and ExpLog2K | e

ExpLog2→ExpLog3M

M→or ExpLog3M | e

ExpLog3 → not ExpLog3 | ExpRel

ExpRel → ExpArit1 opRel ExpArit1 | [Cond]

Lectura → cadena, Ingresar(id)

Escritura → ImprimirR

R→(W

W->ExpArit1)| cadenaCAD)

CAD → , cadena CAD | ε

Construcción de la TAS

TAS

Para Programa → '|' id '|' Cuerpo.

Primero('|' id '|' Cuerpo.)= Primero (|)={|}

TAS[Programa,|]= | id | Cuerpo.

Para Definir → Define: Variables

Primero(Define: Variables)=Primero(Define)={Define}

TAS[Definir,Define]= Define: Variables

Para Variables → idV

Primero(idV)=Primero(id)={id}

TAS[Variables,id]= idV

Para V → ,idV | ϵ

Para V → ,idV

Primero(, idV)=Primero(,)={,}

TAS[V,,]= ,idV

Para V-> ϵ

Primero(ϵ)= ϵ →

Siguiente(V)= Siguiente(Variables)= Siguiente(Definir)= Siguiente(Sentencia)= Primero(. SecSent) = {.}

TAS[V,.]=ϵ

Para Cuerpo → Comienzo SecSent Final

Primero(Comienzo SecSent Final)= Primero(Comienzo)= {comienzo}

TAS[Cuerpo,comienzo]= Comienzo SecSent Final

Para SecSent → Sentencia. SecSent | ϵ

Para SecSent → Sentencia. SecSent

Primero(Sentencia. SecSent)= Primero(Sentencia)= {id, Mientras, Si, cadena, Imprimir, Define}

- Primero(Asignación)= {id}
- Primero(Ciclo)= {Mientras}
- Primero(Condicional)={Si}
- Primero(Lectura)={cadena}
- Primero(Escritura)={Imprimir}
- Primero(Definir)={Define}

TAS[SecSent,id]= Sentencia. SecSent

TAS[SecSent,Mientras]= Sentencia. SecSent

TAS[SecSent,Si]= Sentencia. SecSent

TAS[SecSent,cadena]= Sentencia. SecSent

TAS[SecSent,Imprimir]= Sentencia. SecSent

TAS[SecSent,Define]= Sentencia. SecSent

Para SecSent → ϵ

Primero(ϵ)= ϵ →

Siguiente(SecSent)=

- Primero(Final)= {Final}
- Primero(|)= { | }

TAS[SecSent,Final]= ϵ

TAS[SecSent, |]= ϵ

Sentencia → Asignación | Ciclo | Condicional | Lectura | Escritura | Definir

Para Sentencia→ Asignación

Primero(Asignacion)= Primero(id << ExpArit1)= Primero(id)={id}

Para Sentencia → Ciclo

Primero(Ciclo)= Primero(Mientras cond hacer '|' SecSent '|')= Primero(Mientras)= {Mientras}

Para Sentencia → Condicional

Primero(Condicional)= Primero(Si Q)= Primero(Si)={Si}

Para Sentencia → Lectura

Primero(Lectura)= Primero(cadena, Ingresar(id))=Primero(cadena)={cadena}

Para Sentencia → Escritura

Primero(Escritura)= Primero(ImprimirR)=Primero(Imprimir)={Imprimir}

Para Sentencia → Definir

Primero(Definir)=Primero(Define: Variables)=Primero(Define)={Define}

TAS[Sentencia,id]= Asignación

TAS[Sentancia,Mientras]= Ciclo

TAS[Sentencia,Si]= Condicional

TAS[Sentencia,cadena]= Lectura

TAS[Sentencia,Imprimir]= Escritura

TAS[Sentencia,Define]= Definir

Para Asignación → id << ExpArit1

Primero(id << ExpArit1)= Primero(id)= {id}

TAS[Asignacion,id]= id << ExpArit1

Para ExpArit1 → ExpArit2H

Primero(ExpArit2H)= Primero(ExpArit2)= Primero(ExpArit3J)= Primero(ExpArit3)=

{-,raíz,ConstReal,id,(}

- Primero(-ExpArit3L)={-}
- Primero(raíz(ExpArit3)L)={raiz}
- Primero(ConstRealL)={ConstReal}
- Primero(idL)={id}
- Primero((ExpArit1)L)={(}

TAS[ExpArit1,-]= ExpArit2H

TAS[ExpArit1,raiz]= ExpArit2H

TAS[ExpArit1,ConstReal]= ExpArit2H

TAS[ExpArit1,id]= ExpArit2H

TAS[ExpArit1,(]= ExpArit2H

Para H → + ExpArit2 H | - ExpArit2 H | e

Para H → + ExpArit2H

Primero(+ ExpArit2H)= Primero(+)={+}

Para H → - ExpArit2H

Primero(- ExpArit2H)= Primero(-)= {-}

TAS[H,+]= + ExpArit2H

TAS[H,-]= - ExpArit2H

Para H-> ϵ

Primero(ϵ)=ϵ →

Siguiente(H)= Siguiente(ExpArit1)= {. ,), opRel, or, and, hacer, entonces,]}

- Siguiente(Asignación)= Siguiente(Sentencia)= Primero(. SecSent)= Primero(.)= { . }
- Primero( )L)= Primero( ))= {) }
- Primero(opRel ExpArit1)= Primero(opRel)= {opRel}
- Siguiente(ExpRel)= Siguiente(ExpLog3)= Primero(M)= {or, and, hacer, entonces,]}

• Primero(or ExpLog3M)= Primero(or)= {or}

• Siguiente(M)= Siguiente(ExpLog2)= Primero(K)= {and, hacer, entonces,]}

  ◘ Primero(and ExpLog2K)= Primero(and)= {and}

  ◘ Siguiente(K)= Siguiente(Cond)= {hacer, entonces]}

    ○ Primero(hacer)={hacer}

    ○ Primero(entonces)={entonces}

    ○ Primero( ])= {] }

TAS[H,.]= ϵ

TAS[H,)]= ϵ

TAS[H,opRel]= ϵ

TAS[H,or]= ϵ

TAS[H,and]= ϵ

TAS[H,hacer]= ϵ

TAS[H,entonces]= ϵ

TAS[H,']']= ϵ

Para ExpArit2 → ExpArit3J

Primero(ExpArit3J)= Primero(ExpArit3)= {-,raíz,ConstReal,id,(}

TAS[ExpArit2,-]=ExpArit3J

TAS[ExpArit2,raiz]=ExpArit3J

TAS[ExpArit2,ConstReal]=ExpArit3J

TAS[ExpArit2,id]=ExpArit3J

TAS[ExpArit2,(]=ExpArit3J

Para J → *ExpArit3J | /ExpArit3J | ϵ

Para J → * ExpArit3J

Primero(* ExpArit3J) = Primero(*)={*}

TAS[J,*]= * ExpArit3J

Para J → / ExpArit3J

Primero(/ ExpArit3J)= Primero(/)={/}

TAS[J,/]= / ExpArit3J

Para J → ϵ

Primero(ϵ)= ϵ →

Siguiente(J)= Siguiente(ExpArit2)= {+, - , . ,), opRel, or, and, hacer, entonces,]}

- Primero(H)= {+, - , . ,), opRel, or, and, hacer, entonces,]}
  - Primero(+ ExpArit2 H)= Primero(+)= {+}
  - Primero(- ExpArit2 H)= Primero(-)= {-}
  - Primero(ϵ)=ϵ → Siguiente(H)= Siguiente(ExpArit1)= {. ,), opRel, or, and, hacer, entonces,]}

TAS[J,+]= ϵ

TAS[J,-]= ϵ

TAS[J,.]= ϵ

TAS[J,)]= ϵ

TAS[J,opRel]= ϵ

TAS[J,or]= ϵ

TAS[J,and]= ϵ

TAS[J,hacer]= ϵ

TAS[J,entonces]= ϵ

TAS[J,']']= ϵ

ExpArit3 → ExpArit4L

Primero(ExpArit4L)= Primero(ExpArit4)= {-,raiz,ConstReal,id,(}

- Primero(-ExpArit4L)= Primero(-)= {-}
- Primero(raiz(ExpArit4)L)= Primero(raiz)={raiz}
- Primero(ConstRealL)= Primero(ConstReal)= {ConstReal}
- Primero(idL)= Primero(id)={id}
- Primero((ExpArit1)L)= Primero(( )={(}

TAS[ExpArit3,-]= ExpArit4L

TAS[ExpArit3,raiz]= ExpArit4L

TAS[ExpArit3,ConstReal]= ExpArit4L

TAS[ExpArit3,id]= ExpArit4L

TAS[ExpArit3,(]= ExpArit4L

ExpArit4 → -ExpArit4 | raiz(ExpArit4) | ConstReal | id | (ExpArit1)

Para ExpArit4 → -ExpArit4L

Primero(-ExpArit4)= Primero(-)= {-}

Para ExpArit4 → raiz(ExpArit4)

Primero(raiz(ExpArit4))= Primero(raiz)={raiz}

Para ExpArit4 → ConstReal

Primero(ConstReal)= Primero(ConstReal)= {ConstReal}

Para ExpArit4 → id

Primero(id)= Primero(id)={id}

Para ExpArit4 → (ExpArit1

Primero((ExpArit1))= Primero(( )={(}

TAS[ExpArit4,-]= -ExpArit4

TAS[ExpArit4,raiz]=raiz(ExpArit4

TAS[ExpArit4,ConstReal]=ConstReal

TAS[ExpArit4,id]=id

TAS[ExpArit4,(]= (ExpArit1)

Para L → ^ExpArit4L | ϵ

Para L → ^ExpArit4L

Primero(^ExpArit4L)= Primero(^)= {^}

TAS[L,^]= ^ExpArit4L

Para L → e

Primero(ϵ)= ϵ →

Siguiente(L)= Siguiente(ExpArit3)=

- Primero(J)= {*,/,+, - , . ,), opRel, or, and, hacer, entonces,]}
  - Primero(*ExpArit4J)=Primero(*)={*}
  - Primero(/ ExpArit4J)=Primero(/)={/}

• Primero(ϵ)= ϵ →

Siguiente(J)= Siguiente(ExpArit2)= Primero(H)= {+, - , . ,), opRel, or, and, hacer, entonces,]}

TAS[L,*]= ϵ

TAS[L,/]= ϵ

TAS[L,+]= ϵ

TAS[L,-]= ϵ

TAS[L,.]= ϵ

TAS[L,)]= ϵ

TAS[L,opRel]= ϵ

TAS[L,or]= ϵ

TAS[L,and]= ϵ

TAS[L,hacer]= ϵ

TAS[L,entonces]= ϵ

TAS[L,']']= ϵ

Para Ciclo → Mientras Cond hacer '|' SecSent '|'

Primero(Mientras Cond hacer '|' SecSent '|')= Primero(Mientras)={Mientras}

TAS[Ciclo,Mientras]= Mientras Cond hacer '|' SecSent '|'

Para Condicional → Si Q

Primero(Si Q)= Primero(Si)={Si}

TAS[condicional, Si]= Si Q

Para Q → Cond entonces '|' SecSent '|' A

Primero(Cond entonces '|' SecSent '|' A)= Primero(Cond)= Primero(ExpLog2K)= Primero(ExpLog2)= Primero(ExpLog3)= {not, -,raíz,ConstReal,id,(, [}

- Primero(not ExpLog3)= Primero(not)= {not}
- Primero(ExpRel) = {-,raíz,ConstReal,id,(, [}
  • Primero(ExpArit1 opRel ExpArit1)= Primero(ExpArit1)= Primero(ExpArit2 H)= Primero(ExpArit2)= Primero(ExpArit4J)= Primero(ExpArit4)= {-,raíz,ConstReal,id,(}
  • Primero([cond])= Primero([ )= { [}

TAS[Q,not]= Cond entonces | SecSent | A

TAS[Q,-]= Cond entonces | SecSent | A

TAS[Q,raiz]= Cond entonces | SecSent | A

TAS[Q,ConstReal]= Cond entonces | SecSent | A

TAS[Q,id]= Cond entonces | SecSent | A

TAS[Q,(]= Cond entonces | SecSent | A

TAS[Q,[]= Cond entonces | SecSent | A

A → Sino '|'SecSent '|' | e

Para A → Sino '|'SecSent '|'

Primero(Sino '|'SecSent '|')= Primero(sino)={sino}

Para A → ϵ

Primero(ϵ)={ϵ} →

Siguiente(A)= Siguiente(Q)= Siguiente(Condicional)= Siguiente(Sentencia)= Primero(.T)= Primero(.)= {.}

TAS[A,sino]= Sino | SecSent |

TAS[A,.]= ϵ

Para Cond → ExpLog2K

Primero(ExpLog2)= Primero(ExpLog3M)= Primero(ExpLog3)= {not,-,raiz,constreal,id,(, [}

- Primero(not ExpLog3)= Primero(not)= {not}
- Primero(ExpRel) = {-,raíz,ConstReal,id,(, [}

TAS[Cond,not]= Explog2K

TAS[Cond,-]= Explog2K

TAS[Cond,raiz]= Explog2K

TAS[Cond,constreal]= Explog2K

TAS[Cond,id]= Explog2K

TAS[Cond,(]= Explog2K

TAS[Cond,'[']= Explog2K

Para K → and ExpLog2K | e

Para K → and ExpLog2K

Primero(and ExpLog2K)= Primero(and)={and}

Para K → ϵ

Primero(ϵ)= ϵ →

Siguiente(K)= Siguiente(Cond)= {hacer, entonces,]}

- Primero(hacer)={hacer}

- Primero(entonces)={entonces}
- Primero( ])= {] }

TAS[K,and]= and ExpLog2K

TAS[K,hacer]=ϵ

TAS[K,entonces]=ϵ

TAS[K, ‘]’]=ϵ

Para ExpLog2 → ExpLog3M

Primero(ExpLog3M)= Primero(ExpLog3)=

- Primero(not ExpLog3)= Primero(not)={not}
- Primero(ExpRel) = {-,raíz,ConstReal,id,(, [}

TAS[explog2,not]= explog3M

TAS[explog2,-]= explog3M

TAS[explog2,raiz]= explog3M

TAS[explog2,constreal]= explog3M

TAS[explog2,id]= explog3M

TAS[explog2,(]= explog3M

TAS[explog2,‘[’]= explog3M

Para M → or ExpLog3M | e

Para M → or ExpLog3M

Primero(or ExpLog3M)= Primero(or)= {or}

TAS[M,or]= or ExpLog3M

Para M → ϵ

Primero(ϵ)=ϵ →

Siguiente(M)= Siguiente(ExpLog2)= Primero(K)= {and, hacer, entonces}

Ya que K → ϵ,

Siguiente(K)= Siguiente(Cond)= {hacer, entonces,]}

TAS[M,and]= ϵ

TAS[M,hacer]= ϵ

TAS[M,entonces]= ϵ

TAS[M, ‘]’]= ϵ

Para ExpLog3 → not ExpLog3 | ExpRel

Para ExpLog3 → not ExpLog3

Primero(not ExpLog3)= Primero(not)= {not}

TAS[ExpLog3,not]= not ExpLog3

Para ExpLog3 → ExpRel

Primero(ExpRel)= {-,raíz,ConstReal,id,(, []}

TAS[ExpLog3,-]= ExpRel

TAS[ExpLog3,raiz]= ExpRel

TAS[ExpLog3,ConstReal]= ExpRel

TAS[ExpLog3,id]= ExpRel

TAS[ExpLog3,( ]= ExpRel

TAS[ExpLog3,'[']= ExpRel

Para ExpRel → ExpArit1 opRel ExpArit1 | [Cond]

Para ExpRel → ExpArit1 opRel ExpArit1

Primero(ExpArit1 opRel ExpArit1)= Primero(ExpArit1)= {-
,raíz,ConstReal,id,(}

TAS[ExpRel,-]= ExpArit1 opRel ExpArit1

TAS[ExpRel,raiz]= ExpArit1 opRel ExpArit1

TAS[ExpRel,ConstReal]= ExpArit1 opRel ExpArit1

TAS[ExpRel,id]= ExpArit1 opRel ExpArit1

TAS[ExpRel,(]= ExpArit1 opRel ExpArit1

Para ExpRel → [Cond]

Primero([Cond])=Primero([ ) = {[}

TAS[ExpRel,'[']= [Cond]

Para Lectura → cadena, Ingresar(id)

Primero(cadena, Ingresar(id))= Primero (cadena)={cadena}

TAS[Lectura,cadena]= cadena, Ingresar(id)

Para Escritura → ImprimirR

Primero(ImprimirR)= Primero(Imprimir)= {Imprimir}

TAS[Escritura,imprimir]= ImprimirR

Para R→(W

Primero((W) = Primero(( )= { (}

TAS[R,(]= (W

Para W → ExpArit1) | cadenaCAD)

Para W->ExpArit1)

Primero (ExpArit1))= {-,raíz,ConstReal,id,(}

TAS[W,-]= ExpArit1)

TAS[W,raiz]= ExpArit1)

TAS[W,ConstReal]= ExpArit1)

TAS[W,id]= ExpArit1)

TAS[W,(]= ExpArit1)

Para W → cadenaCAD)

Primero(cadenaCAD))= Primero(cadena)= {cadena}

TAS[W,cadena]= cadenaCAD)

Para CAD → , cadena CAD | ϵ

Para CAD → , cadena CAD

Primero(, cadena CAD)= Primero(,)= {,}

TAS[CAD,,]= , cadena CAD

Para CAD → ϵ

Siguiente(CAD)= Primero( ))= {) }

TAS[CAD,)]= ϵ

Especificación de la semántica asociada a cada variable de la CFG.

Programa → '|' id '|' Cuerpo.

EvaluaPrograma(Arbol,Estado);

EvaluaCuerpo(Arbol.hijos[4],Estado);

Definir → Define : Variables

EvaluaDdefinir(Arbol,Estado);

EvaluaVariables(Arbol.hijos[3],Estado);

Variables → idV

EvaluaVariables(Arbol,Estado);

EstadoAgregarVariable(Estado,Arbol.hijos[1].lexema);

EvaluaV(Arbol.hijos[2],estado

V → ,idV | e

EvaluaV(Arbol,Estado);

If not producción = epsilon

EstadoAgregarVariable(Estado,Arbol.Hijos[2].lexema);

EvaluaV(Arbol.hijos[3],Estado);

Cuerpo → Comienzo SecSent Final

EvaluaCuerpo(Arbol,Estado);

EvaluaSecSent(Arbol.Hijos[2],Estado);

SecSent → Sentencia. SecSent | ε

EvaluaSecSent(Arbol,Estado);

If not produccion = epsilon

EvaluaSentencia(Arbol.Hijos[1],Estado);

EvaluaSecSent(Arbol.hijos[3],Estado);

Sentencia → Asignación | Ciclo | Condicional | Lectura | Escritura | Definir

EvaluaSentencia(Arbol,Estado);

If produccion = Asignación

EvaluaAsignacion(Arbol.hijos[1],Estado);

```
If produccion = Ciclo
        EvaluaCiclo(Arbol.hijos[1],Estado);
If produccion = Condicional
        EvaluaCondicional(Arbol.hijos[1],Estado);
If produccion = Lectura
        EvaluaLectura(Arbol.hijos[1],Estado);
If produccion = Escritura
        EvaluaEscritura(Arbol.hijos[1],Estado);
If produccion = Definir
        EvaluaDefinir(Arbol.hijos[1],Estado);
```

Asignación → id << ExpArit1

```
EvaluaAsignacion(Arbol,Estado);
    EvaluaExpArit1(Arbol.hijos[3],Estado, Resultado);
    EstadoAsignarValor(Estado,Arbol.hijos[1].Lexema,Resultado);
```

ExpArit1 → ExpArit2 H

```
EvaluaExpArit1(Arbol,Estado,Resultado);
    EvaluaExpArit2(Arbol.hijos[1],Estado,Resultado1);
    EvaluaH(Arbol.hijos[2],Estado,Resultado1,Resultado);
```

H → + ExpArit2 H | - ExpArit2 H | e

```
EvaluaH(Arbol,Estado,PrimerOperando,Resultado)
    If produccion = suma
        EvaluaExpArit2(Arbol.hijos[2],Estado,SegundoOperando)
        Calc:= PrimerOperando + SegundoOperando
        EvaluaH(Arbol.hijos[3],Estado,Calc,Resultado)
    If produccion = resta
        EvaluaExpArit2(Arbol.hijos[2],estado,segundoOperando)
        Calc:= PrimerOperando - SegundoOperando
        EvaluarH(Arbol.hijos[2],Estado,Calc,Resultado)
```

```
        If produccion = epsilon
                        Resultado:=PrimerParametro
```

ExpArit2 → ExpArit3 J

```
EvaluaExpArit2(Arbol,Estado,Resultado);
        EvaluaExpArit3(Arbol.hijos[1],Estado,Resultado1)
        EvaluaJ(Arbol.hijos[2],Estado,Resultado1,Resultado)
```

J → * ExpArit3J | / ExpArit3J | e

```
EvaluaJ(Arbol,Estado,PrimerOperando,Resultado)
        If produccion = *
                EvaluaExpArit3(Arbol.hijos[2],Estado,SegundoOperando)
                Calc:= PrimerOperando* SegundoOperando
                EvaluaJ(Arbol.hijos[3],Estado,Calc,Resultado)
        If produccion = /
                EvaluaExpArit3(Arbol.hijos[2],Estado, Operando)
                Calc:= PrimerOperando / SegundoOperando
                EvaluaJ(Arbol.hijos[3],Estado,Calc,Resultado)
        If produccion = epsilon
                Resultado:=PrimerOperando
```

ExpArit3 → ExpArit4L

```
EvaluaExpArit3(Arbol,Estado,Resultado);
        EvaluaExpArit4(Arbol.hijos[1],Estado,Resultado1)
        EvaluaL(Arbol.hijos[2],Estado,Resultado1,Resultado)
```

ExpArit4→ -ExpArit4 | raiz(ExpArit1) | ConstReal | id | (ExpArit1)

```
EvaluaExpArit4(Arbol,Estado,Resultado);
        If produccion = -
```

EvaluaExpArit4(Arbol.hijos[2],Estado,PrimerOperando)

Resultado:=-1*primeroperando;

If produccion = raiz

EvaluaExpArit1(Arbol.hijos[3],Estado,primeroperando)

Resultado:= sqrt(primeroperando);

If produccion = constreal

Resultado := StrToFloat(Arbol.Hijos[1].Lexema)

If produccion = id

EstadoDevolverValor(Estado,Arbol.hijos[1].lexema,Resultado,mensaje)

If produccion = (

EvaluaExpArit1(Arbol.hijos[2],Estado,Resultado)

L → ^ExpArit4L | e

EvaluaL(Arbol,Estado,PrimerParametro,Resultado)

If produccion = ^

EvaluaExpArit4(Arbol.hijos[2],Estado,SegundoParametro)

Calc:= PrimerParametro^SegundoParametro

EvaluaL(Arbol.hijos[],estado,Calc,resultado)

else

Resultado:= PrimerParametro

Ciclo → Mientras Cond hacer '|' SecSent '|'

EvaluaCiclo(Arbol,estado);

EvaluaCond(Arbol.hijos[2],estado,resultado);

While resultado hacer

EvaluaSecSent(Arbol.hijos[5],estado);

evaluacond(Arbol.hijos[2],estado,resultado)

Condicional → Si Q

EvaluaCondicional(Arbol,estado)

evaluaQ(Arbol.hijos[2],estado,cond)

Q → Cond entonces '|' SecSent '|' A

```
EvaluaQ(Arbol,estado,resultado)
        evaluaCond(Arbol.hijos[1],estado,resultado)
                if resultado hacer
                evalsecsent(Arbol.hijos[4],estado)
sino
                evaluaA(Arbol.hijos[2],estado)
```

A->Sino '|'SecSent '|' | e

```
 EvaluaA(Arbol,estado)
        If not produccion = epsilon
        EvaluaSecSent(Arbol.hijos[3],estado)
```

Cond→ExpLog2K

```
EvaluaCond(Arbol,estado,resultado)
        evaluaExpLog2(Arbol.hijos[1],estado,primeroperando)
        evaluaK(Arbol.hijos[2],estado,primerOperando,resultado)
```

K→and ExpLog2K | e

```
EvaluaK(Arbol, Estado, resultado1,resultado);
        If produccion=and then
                EvaluaExpLog2(Arbol.hijos[2],estado,resultado2);
                aux:=resultado1 and resultado2;
                EvaluaK(Arbol.hijos[3],estado,aux,resultado);
        If produccion=epsilon then
                resultado:=resultado1;
```

ExpLog2→ExpLog3M

```
EvaluaExpLog2(Arbol,estado,resultado);
        EvaluaExpLog3(Arbol.hijos[1],estado,resultado1);
```

```
        EvaluaM(Arbol.hijos[2],estado,resultado1,resultado);
```

M→or ExpLog3M | e

```
EvaluaM(Arbol, Estado, resultado1,resultado);
        If produccion=or then
                EvaluaExpLog3(Arbol.hijos[2],estado,resultado2);
                aux:=resultado1 or resultado2;
                EvaluaM(Arbol.hijos[3],estado,aux,resultado);
        If produccion=epsilon then
                resultado:=resultado1;
```

ExpLog3 → not ExpLog3 | ExpRel

```
EvaluaExpLog3(Arbol, Estado,resultado);
        If produccion=not then
                EvaluaExpLog3(Arbol.hijos[2],estado,resultado);
        else
                EvaluaExpRel(Arbol.hijos[1],estado,resultado);
```

ExpRel → ExpArit1 opRel ExpArit1 | [Cond]

```
EvaluaExpRel(Arbol,estado,resultado);
        If produccion= ExpArit1 then
                EvaluaExpArit1(Arbol.hijos[1],estado,primeroperando);
                EvaluaExpArit|(Arbol.hijos[3],estado,segundooperando);
        según producción hacer
            '<':resultado:=primeroperando<segundooperando;
            '>':resultado:=primeroperando>segundooperando;
            '=':resultado:=primeroperando=segundooperando;
            '<=':resultado:=primeroperando<=segundooperando;
            '>=':resultado:=primeroperando>=segundooperando;
            '<>':resultado:=primeroperando<>segundooperando;
        else
```

EvaluaCond(Arbol.hijos[2],estado,resultado);

Lectura → cadena, Ingresar(id)

write(Arbol.hijos[1].lexema)

readln(X)

EstadoAsignaValor(estado,Arbol.hijos[5].lexema,X)

Escritura → ImprimirR

EvaluaEscritura(Arbol,estado)

evaluaR(Arbol.hijos[2],estado,resultado)

R→(W

EvaluaR(Arbol,estado,resultado)

evaluarW(Arbol.hijos[2],estado,resultado)

W->ExpArit1)| cadenaCAD)

EvaluaW(Arbol,estado,resultado)

If produccion = exparit1

EvaluaExpArit1(Arbol.hijos[1],estado,resultado)

write(resultado)

If produccion=CAD

write(árbol.hijos[1].lexema)

evaluaCad(Arbol.hijos[2],estado,resultado)

CAD → , cadena CAD | ε

evaluaCAD(Arbol,estado)

If not produccion = épsilon

write(árbol.hijos[2].lexema)

evaluarCAD(estado,Arbol.hijos[3])