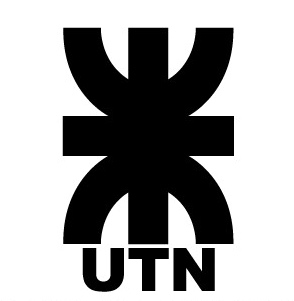
b  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**UTN-FRD**

**Trabajo Práctico Nro. 2**

**Sintaxis y semántica de los lenguajes**

**Alumno:** Contrera Agustin, Delgado Lucas, Julian Vazquez

**Docente:** Hernán Miranda.

**Corrección de la gramática:**

SD(ListaSentencias → ListaSentencias;Sentencia) = {“si”,“repetir”,“id”,“leer”,“mostrar”,“func”}

SD(ListaSentencias → Sentencia) = {“si”,“repetir”,“id”,“leer”,“mostrar”,“func”}

cómo   
SD(ListaSentencias → ListaSentencias;Sentencia)∩SD(ListaSentencias → Sentencia) ≠ ∅  
Luego la gramática no es LL1.

Probamos eliminar recursividad izquierda:  
ListaSentencias → ListaSentencias;Sentencia | Sentencia  
ListaPar → ListaPar “;” “id” | “id”  
Expresion2 → Expresion2 “opsuma” Termino | Termino  
Termino → Termino “opmult” Factor | Factor  
  
Estas producciones resultan luego:

ListaSentencias → Sentencia ListaSentencias\_prime  
ListaSentencias\_prime → ; Sentencia ListaSentencias\_prime / 𝝀

ListaPar → “id” ListaPar\_Prime  
ListaPar\_Prime → “;” “id” ListaPar\_Prime / 𝝀

Expresion2 →Termino Expresion2\_Prime  
Expresion2\_Prime →”opsuma” Termino Expresion2\_Prime / 𝝀

Termino → Factor Termino\_Prime  
Termino\_Prime → “opmult” Factor Termino\_Prime / 𝝀

Al hacer esto sigue sin ser LL1 puesto que:  
SD(SentenciaSi → “si” Expresion “entonces” ListaSentencias “sino” ListaSentencias “finsi”)∩SD(SentenciaSi → “si” Expresion “entonces” ListaSentencias “finsi”) ≠ ∅

Probamos Factorizar las siguientes producciones:

SentenciaSi → “si” Expresion “entonces” ListaSentencias “sino” ListaSentencias “finsi” | “si” Expresion “entonces” ListaSentencias “finsi”

Expresion → Expresion2 “oprel” Expresion2 | Expresion2

Estas producciones resultan:

SentenciaSi → “si” Expresion “entonces” ListaSentencias SentenciaSi\_Prime  
SentenciaSi\_Prime → “sino” ListaSentencias “finsi”/“finsi”

Expresion → Expresion2 Expresion\_Prime  
Expresion\_Prime → “oprel” Expresion2 / 𝝀

**La Gramática resultante es:**

VT = {id, num, si, entonces, sino, finsi, repetir, hasta, equal, leer, mostrar, func, finfunc, (, ), ;, oprel, opsuma, opmult}   
VN = {Program, ListaSentencias, Sentencia, SentenciaSi, SentenciaRepetir, SentenciaAsig, SentenciaLeer, SentenciaMostrar, SentenciaFun,Proc, ListaPar, Expression, Expresion2, Factor, Termino,ListaSentencias\_Prime,SentenciaSi\_Prime,ListaPar\_Prime,Expresion\_Prime, Expresion2\_Prime,Termino\_Prime}  
S = Program  
P= {  
Program →ListaSentencias  
ListaSentencias → Sentencia ListaSentencias\_Prime  
ListaSentencias\_Prime → “;” Sentencia ListaSentencias\_Prime | 𝝀  
Sentencia → SentenciaSi | SentenciaRepetir | SentenciaAsig | SentenciaLeer | SentenciaMostrar | SentenicaFun  
SentenciaSi → “si” Expresion “entonces” ListaSentencias SentenciaSi\_Prime  
SentenciaSi\_Prime → “sino” ListaSentencias “finsi” | “finsi”  
SentenciaRepetir → “repetir” ListaSentencias “hasta” Expresion  
SentenciaAsig → “id” “equal” Expresion  
SentenciaLeer → “leer” “id”  
SentenciaMostrar → “mostrar” Expresion  
SentenciaFun → “func” Proc “finfunc”  
Proc → “id” “(“ ListaPar “)” ListaSentencias  
ListaPar → “id” ListaPar\_Prime  
ListaPar\_Prime → “;” “id” ListaPar\_Prime | 𝝀  
Expresion → Expresion2 Expresion\_Prime  
Expresion\_Prime → “oprel” Expresion2 | 𝝀  
Expresion2 →Termino Expresion2\_Prime  
Expresion2\_Prime →”opsuma” Termino Expresion2\_Prime | 𝝀  
Termino → Factor Termino\_Prime  
Termino\_Prime → “opmult” Factor Termino\_Prime | 𝝀  
Factor → “(“ Expresion “)” | “num” | “id”  
}

Sus simbolos directrices son:

S.D(Program →ListaSentencias) : {“si”, “repetir”, “Id”, “leer”, “mostrar”, “func”}  
S.D(ListaSentencias → Sentencia ListaSentencias\_Prime):{ “si”, “repetir”, “Id”, “leer”,“mostrar”, “func”}  
S.D(ListaSentencias\_Prime → “;” Sentencia ListaSentencias\_Prime): {“;”}  
S.D(ListaSentencias\_Prime → 𝝀) : {“sino”,”finsi”,”hasta”,”finfunc”,#}  
S.D(Sentencia -> SentenciaSi) : {“si” }  
S.D(Sentencia -> SentenciaRepetir): {“repetir”}  
S.D(Sentencia -> SentenciaAsig) : {“id”}  
S.D(Sentencia -> SentenciaLeer) : {“leer”}  
S.D(Sentencia -> SentenciaMostrar): {“mostrar”}  
S.D(Sentencia -> SentenciaFun) : {“func”}  
S.D(SentenciaSi -> “si” Expresion “entonces” ListaSentencias SentenciaSi\_Prime): {“si”}  
S.D(SentenciaSi\_Prime → “sino” ListaSentencias “finsi”: {“sino”}  
S.D(SentenciaSi\_Prime → “finsi”): {“finsi”}  
S.D(SentenciaRepetir → “repetir” ListaSentencias “hasta” Expresion): {“repetir”}  
S.D(SentenciaAsig → “id” “equal” Expresion):{“id”}  
S.D(SentenciaLeer → “leer” “id”):{“leer”}  
S.D(SentenciaMostrar → “mostrar” Expresion):{“mostrar”}  
S.D(SentenciaFun → “func” Proc “finfunc”):{“func”}  
S.D(Proc → “id” “(“ ListaPar “)” ListaSentencias):{“id”}  
S.D(ListaPar → “id” ListaPar\_Prime): {“id”}  
S.D(ListaPar\_Prime → “;” “id” ListaPar\_Prime) :{“;”}  
S.D(ListaPar\_Prime →𝝀 ):{“)“}  
S.D(Expresion → Expresion2 Expresion\_Prime) :{“(”,”num”,”id”}  
S.D(Expresion\_Prime → “oprel” Expresion2 ):{“oprel”}  
S.D(Expresion\_Prime →𝝀 ):{“entonces”,”;”,#,”finsi”,”sino”,”hasta”,”finfunc”,”)”}  
S.D(Expresion2 →Termino Expresion2\_Prime): {“(“,”num”,”id”}  
S.D(Expresion2\_Prime →”opsuma” Termino Expresion2\_Prime) :{“opsuma”}  
S.D(Expresion2\_Prime → 𝝀) : {“oprel”,”entonces”,”;”,#,”sino”,”finsi”,”hasta”,”finfunc”,”)”}  
S.D(Termino → Factor Termino\_Prime) :{“(“,”num”,”id”}  
S.D(Termino\_Prime → “opmult” Factor Termino\_Prime) :{“opmult”}  
S.D(Termino\_Prime → 𝝀) :{ “opsuma”,“oprel”,”entonces”,”;”,#,”sino”,”finsi”,”hasta”,”finfunc”,”)”}  
S.D(Factor → “(“ Expresion “)”) : {“(“}  
S.D(Factor →“num”) : {“num”}  
S.D(Factor → “id”) :{“id“}  
  
Ejemplos:

parser(lexer("si 6 > 7 entonces var equal 8")) **La cadena pertenece al lenguaje**  
parser(lexer("leer vauxi ; vauxi equal 5")) **La cadena pertenece al lenguaje**  
parser(lexer('mostrar 3 \* 2 > 5 \* 6')) **La cadena pertenece al lenguaje**  
parser(lexer("si 4 > 3 entonces repetir leer id hasta 3 sino mostrar 4 finsi")) **La cadena pertenece al lenguaje**  
parser(lexer("si 6>7 entonces leer id finsi")) **La cadena pertenece al lenguaje**parser(lexer(“si 9 > 3”)) **Procesando Producción: entonces  
La cadena no pertenece al lenguaje,** esto se debe a que busca la produccion entonces y no la encuentra  
parser(lexer(“5 > 9 entonces”)) **Derivando Program ->   
La cadena no pertenece al lenguaje,** esto se debe a que “program” no puede derivarse en “num”  
parser(lexer("si 6 < 7 entonces hola equal 8")) **La cadena pertence al lenguaje**  
parser(lexer("hola aguante boca"))  **Procesando Producción: equal  
La cadena no pertenece al lenguaje,** parecido al caso anterior, “program” no puede derivarse de ninguna forma en “id”  
parser(lexer("(si 4 > 3 entonces leer id finsi)")) **Derivando Program ->   
La cadena no pertenece al lenguaje,** por lo mismo que en el ejemplo anterior, no hay forma de derivar program en “parentesisApertura”