PROYECTO: ANALIZADOR SINTÁCTICO

Desarrollo: Individual.

Estudiante: Sacaca Flores Edson Alain

Registro: 217190332

La BNF completa está en las ultimas páginas, y el desarrollo de cada producción esta debajo de cada tabla.

Fecha de Presentación: Enviar la carpeta del Proyecto al correo ginobarroso@uagrm.edu.bo, con Asunto: *PROY-COMP. Nombre del alumno (e.g. PROY-COMP. Arce Catacora, Luis),* hasta el jueves 28/julio/2022, 23:59:59 horas.

Escribir una BNF y luego **desarrollar un Analizador Sintáctico** (Parser) para el lenguaje MiniPASCAL, tomando en cuenta que el mismo **NO** es case-sensitive. Un Programa Pascal, consta de 3 secciones:

1) HEADER (el programador la puede obviar) //Header → PROGRAM ID; | λ	PROGRAM Factorial;
2) CUERPO: Mezcla de n ≥0 DECLARACIONES y PROCEDIMIENTOS (Esta sección puede ser vacía) //Cuerpo → …	VAR *//Declaración de* n ≥ 1 *líneas.* a, b, c : INTEGER; c, d : BOOLEAN; PROCEDURE Algo; *//Los procedimientos no tienen parámetros,* BEGIN *//ni variables locales.* Sentencias; *//El bloque puede estar vacío (λ)* END;
3) MAIN (Sección obligada. Un programa si o si, debe tener al menos esta sección) //Main -> BEGIN … END.	BEGIN Sentencias; *//El bloque puede estar vacío (λ)* END. *//Note que el END termina en PUNTO, no en PTOCOMA.*

Cuerpo

Cuerpo → Decl Cuerpo | Proc Cuerpo | λ

Conjunto first del cuerpo

F(Cuerpo) = F(Decl) | F(Proc) | { λ }
= { VAR } | { PROCEDURE } | { λ }

Declaracion de variables

VAR //Declaración de n ≥ 1 líneas.
a, b, c : INTEGER;
c, d : BOOLEAN;

Decl → VAR Linea masLinea
masLinea → Linea masLinea | λ
masID → , id masID | λ
Linea → ID masID : TIPODATO;

Conjuntos first de las declaraciones de variable

F(Linea) = { ID } //unica seccion

F(masID) = { , } | { λ }

F(masLinea) = { ID } | λ

F(Decl)= { VAR }

Procedimientos

Proc → PROCEDURE ID ; BEGIN Sentencias END ;

Conjuntos first de Procedimientos

F(Proc) = { PROCEDURE }

MAIN

Seccion MAIN

main → BEGIN Sentencias END .

Conjunto first de la sección main

F(main) = { BEGIN } //única seccion

SENTENCIAS DEL LENGUAJE MiniPASCAL

Las sentencias del lenguaje son 8: Asignación, Llamada, Condicional, BucleFor, BucleWhile, BucleRepeat, Lectura, Impresión.

// Sentencia → Asignación | Llamada | Condicional | BucleFor | BucleWhile | BucleRepeat | Lectura | Impresión

Como se sabe, las construcciones de programación: Condicional (IF-THEN-ELSE) y los Bucles WHILE y FOR, pueden tener una sola sentencia o un bloque BEGIN END; de sentencias. Recuerde que los bloques BEGIN END, pueden ser vacíos.

Sentencia // Para añadir N sentencias

Sentencia → Asignación Sentencia | Llamada Sentencia | Condicional Sentencia |
BucleFor Sentencia | Sentencia | BucleRepeat Sentencia |
Lectura Sentencia | Impresión Sentencia | λ

//vamos a tratar la ambigüedad que hay entre Asignación y llamada

Sentencia → ID := Expr ; Sentencia | ID ( ) ; Sentencia | Condicional Sentencia |
BucleFor Sentencia | BucleWhile Sentencia | BucleRepeat Sentencia |
Lectura Sentencia | Impresión Sentencia | λ

Factor-i

Sentencia → ID Sentencia1 Sentencia | Condicional Sentencia |
BucleFor Sentencia | BucleWhile Sentencia | BucleRepeat Sentencia |
Lectura Sentencia | Impresión Sentencia | λ

Sentencia1 → := Expr ; | ( ) ;

Sentencia → ID Sentencia1 Sentencia | Condicional Sentencia | BucleFor Sentencia |
BucleWhile Sentencia | BucleRepeat Sentencia | Lectura Sentencia |
Impresión Sentencia | λ //asi queda la BNF en LL1

Sentencia1 → := Expr ; | ( ) ;

Conjunto First

F(Sentencia1) = { : } | { (}

F(Sentencia) = { ID } | { IF } | { FOR } | { WHILE } | { REPEAT } | { READLN } | { WRITELN } | { λ }

Sentencia de una sola línea

/* Esta producción sera usada para las producciones de FOR WHILE IF que no tengan bloques BEGIN END, es decir que son Estructuras de una sola línea de código. En esta producción no habrá lambda ya que al ser solo una línea, es obligatorio poner esa línea única */

unaSentencia → ID Sentencia1 | Condicional | BucleFor |BucleWhile |
BucleRepeat | Lectura |Impresión //asi queda la BNF en LL1
unaSentencia1 → := Expr | ( )

Conjunto First
F(unaSentencia1) = { : } | { (}

F(unaSentencia) = { ID } | { IF } | { FOR } | { WHILE } | { REPEAT } | { READLN } | { WRITELN }

ASIGNACIÓN. Se refiere a la asignación de una Expr (Expresión aritmética) a una variable. //Asignación → ID **:=** Expr**;**	`altura := 25*Base + z*y;` *// **:=** es el token ASSIGN*
LLAMADA Para llamar a un procedimiento //Llamada → ID( );	`factorial();` `mostrar();`

CONDICIONAL.
Se refiere a las construcciones IF-THEN e IF-THEN-ELSE
 	
//Condicional → IF ExprBoole …
 	
Recuerde una regla de PASCAL: "Antes de un ELSE, no se escribe un punto y coma".
	`IF z=3*y and x+1<50 THEN Println("true");`
	
	`IF z>=0 OR (p+1< 0) THEN BEGIN END ELSE BEGIN READLN(x, y); END;`

BucleFor
Se refiere al bucle FOR de PASCAL en sus dos variantes: Una que usa TO y otra que usa DOWNTO
 	
//BucleFor → FOR ID:= Expr TO Expr DO …
	`FOR i:=1 TO n+1 DO Println("i=", i);`
	
	`FOR z:=2*n DOWNTO n+1 DO BEGIN Println("z*2=", z*2); READLN(p, q, s); END;`

BucleWhile
 	
//BucleWhile → While ExprBoole DO …
	`WHILE z=3*y and x+1<50 DO Println("Infinito");`
	
	`WHILE z <= 2*n DO BEGIN Println(z); READLN(p, q, s); z := z+1; END;`

BucleRepeat
 	
Se refiere a la construcción Repeat-Until ExprBoole; (El REPEAT-UNTIL puede estar vacío)
	`REPEAT Println("Infinito"); z := z-1; UNTIL z < 0;`
	
	`REPEAT UNTIL p-1 < z*3-5;`

Este bucle NO usa el bloque BEGIN-END; //BucleRepeat → REPEAT … UNTIL ExprBoole;	
LECTURA READLN(Uno o más ID's separados con comas); //Lectura → READLN(ID …);	`READLN(Altura);` `READLN(a, b, c);`

IMPRESION
 	
PRINTLN(Mezcla de n ≥ 1 STRINGctte y Expr, separados con comas);
 	
//Impresión → PRINTLN(…);
	`WRITELN("Hola Mundo");`
	`WRITELN(2*i-5);`
	`WRITELN("Hola", 2*10-5);`
	`WRITELN("Hola", "Mundo", z+3, a/20, "bye", (3*i+2) MOD z);`

Asignación

Asignacion → ID : = Expr ;
Expr → Expr + Termino | Expr – Termino | Termino
Termino → Termino * Factor | Termino / Factor | Termino MOD Factor | Factor
Factor → ID | NUM | - Factor | + Factor | (Expr)

Factor-i
Expr → Expr Expr1 | Termino
Expr1 → + Termino | - Termino

Termino → Termino Termino1 | Factor
Termino1 → * Factor | / Factor | MOD Factor

Recursion-i
Expr → Termino Expr2
Expr2 → Expr1 Expr2 | λ
Expr1 → + Termino | - Termino

Termino → Factor Termino2
Termino2 → Termino1 Termino2 | λ
Termino1 → * Factor t| / Factor | MOD Factor

Asignacion → ID : = Expr ;
Expr → Termino Expr2
Expr2 → Expr1 Expr2 | λ
Expr1 → + Termino | - Termino
Termino → Factor Termino2 //asi queda la BNF en LL1
Termino2 → Termino1 Termino2 | λ
Termino1 → * Factor | / Factor | MOD Factor
Factor → ID | NUM | - Factor | + Factor | (Expr)

Conjuntos first de la asignacion

F(Factor) = { ID } | { NUM } | { - } | { + } | { (}

F(Termino1) = { * } | { / } | { MOD }

F(Termino2) = F(Termino1) | { λ }
= { *, /, MOD } | { λ }

F(Termino) = F(Factor) //unica seccion

F(Expr1) = { + } | { - }

F(Expr2) = F(Expr1) | { λ }
= { +, - } | { λ }

F(Expr) = F(Termino) //unica seccion

F(Asignacion) = { ID } //unica seccion

Llamada

Llamada → ID ( ) ;

Condicional

Expresión Booleana

ExprBoole → ExprBoole OR TermBoole | TermBoole
TermBoole → TermBoole AND FactorBoole | FactorBoole

FactorBoole → Expr OPREL Expr | NOT FactorBoole

Recursion-i

ExprBoole → TermBoole ExprBoole1
ExprBoole1 → OR TermBoole ExprBoole1 | λ

TermBoole → FactorBoole TermBoole1
TermBoole1 → AND FactorBoole TermBoole1 | λ

ExprBoole → TermBoole ExprBoole1
ExprBoole1 → OR TermBoole ExprBoole1 | λ
TermBoole → FactorBoole TermBoole1 //asi queda la BNF en LL1
TermBoole1 → AND FactorBoole TermBoole1 | λ
FactorBoole → Expr OPREL Expr | NOT FactorBoole

Conjuntos First

F(FactorBoole) = F(Expr) | F(NOT)
= { ID, NUM, -, +, (} | { (} | { NOT }

F(TermBoole1) = { AND } | { λ }

F(TermBoole) = F(FactorBoole TermBoole1) //única sección

F(ExprBoole1) = { OR } | { λ }

F(ExprBoole) = F(TermBoole ExprBoole1) //única sección

BNF Condicional

Condicional → IF ExprBoole THEN BEGIN Sentencia END ; | IF ExprBoole THEN unaSentencia |
IF ExprBoole THEN BEGIN Sentencia END ELSE unaSentencia |
IF ExprBoole THEN BEGIN Sentencia END ELSE BEGIN Sentencia END ; |
IF ExprBoole THEN unaSentencia ELSE unaSentencia |
IF ExprBoole THEN unaSentencia ELSE BEGIN Sentencia END ;

Factor-i

Condicional → IF ExprBoole THEN Condicional1
Condicional1 → BEGIN Sentencia END ; | unaSentencia | BEGIN Sentencia END ELSE unaSentencia |
BEGIN Sentencia END ELSE BEGIN Sentencia END ; |
unaSentencia ELSE unaSentencia | unaSentencia ELSE BEGIN Sentencia END ;

Condicional → IF ExprBoole THEN Condicional1
Condicional1 → BEGIN Sentencia END Conidiconal2 | unaSentencia Condicional3
Condicional2 → ; | ELSE unaSentencia | ELSE BEGIN Sentencia END ;
Condicional3 → λ | ELSE unaSentencia | ELSE BEGIN Sentencia END ;

Condicional → IF ExprBoole THEN Condicional1
Condicional1 → BEGIN Sentencia END Conidiconal2 | unaSentencia Condicional3
Condicional2 → ELSE Condicional23 | ;
Condicional3 → ELSE Condicional23 | λ
Condicional23 → unaSentencia | BEGIN Sentencia END ;

Condicional → IF ExprBoole THEN Condicional1
Condicional1 → BEGIN Sentencia END Conidiconal2 | unaSentencia Condicional2
Condicional2 → ELSE Condicional3 | ; // asi queda la BNF en LL1
Condicional3 → unaSentencia ; | BEGIN Sentencia END ;

Conjuntos First

F(Condicional3) = { ID, IF, FOR, WHILE, REPEAT, READLN, WRITELN } | { BEGIN }

F(Condicional2) = { ELSE } | { ; }

F(Condicional1) = { BEGIN } | { ID, IF, FOR, WHILE, REPEAT, READLN, WRITELN }

F(Condicional) = { IF } // seguir única sección

Bucle FOR

BucleFor → FOR ID := Expr condFor Expr DO BEGIN Sentencia END; |
FOR ID := Expr condFor Expr DO unaSentencia ;
condFor → TO | DOWNTO

Factor-i
BucleFor → FOR ID := Expr condFor Expr DO BucleFor1
BucleForWhile → BEGIN Sentencia END; | unaSentencia ;

BucleFor → FOR ID := Expr condFor Expr DO BucleForWhile
BucleForWhile → BEGIN Sentencia END ; | unaSentencia ; //asi queda la BNF en LL1

condFor → TO | DOWNTO

Conjuntos First

F(condFor) = { TO } | { DOWNTO }

F(BucleForWhile) = { BEGIN } | { ID, IF, FOR, WHILE, REPEAT, READLN, WRITELN }

F(BucleFor) = { FOR } //seguir unica seccion

Bucle While

BucleWhile → WHILE ExprBoole DO BucleForWhile //unica seccion

Bucle Repeat

BucleRepeat → REPEAT BucleRepeat1 UNTIL ExprBoole ;
BucleRepeat1 → Sentencia | λ

Conjuntos First

F(BucleRepeat1) = { ID, :, =, NUM, -, +, (} | { λ }

F(BucleRepeat) = { REPEAT } //única sección

Bucle REPEAT sin PTOCOMA

BucleRepeatSinPC → REPEAT BucleRepeat1 UNTIL ExprBoole //seguir unica seccion

Lectura

Lectura → READLN (ID masID) ;

Impresion

Impresion → WRITELN (Elem masElem) ;
masElem → , Elem masElem | λ

Elem → STRINGctte | Expr

Conjuntos First

F(Elem) = { STRINGctte } | { ID, :, =, NUM, -, +, (}

F(masElem) = { , } | { λ }

F(Impresion) = { WRITELN } //seguir unica seccion

APÉNDICE

La producción para expresiones booleanas, ExprBoole, la definimos así:

ExprBoole → ExprBoole OR TermBoole | TermBoole
TermBoole → TermBoole AND FactorBoole | FactorBoole
FactorBoole → Expr OPREL Expr | (ExprBoole) | NOT FactorBoole //Expr = Expresiones aritméticas

Nota.- *Esta definición de ExprBoole, no es completa (Faltan: ID, TRUE y FALSE). Si la escribimos en forma completa, genera ambigüedad.*

La producción para expresiones Aritmética, Expr:

Expr → Expr + Termino | Expr – Termino | Termino
Termino → Termino * Factor | Termino / Factor | Termino MOD Factor | Factor
Factor → ID | NUM | - Factor | +Factor | (Expr)

La BNF de todo mi Proyecto es la siguiente:

Token´s Producciones

Programa	→	Header Cuerpo Main
Header	→	PROGRAM ID ; | λ
Cuerpo	→	Decl Cuerpo | Proc Cuerpo | λ
Decl	→	VAR Linea masLinea
masLinea	→	Linea masLinea | λ
Linea	→	ID masID : TIPODATO ;
masID	→	, ID masID | λ
Proc	→	PROCEDURE ID ; BEGIN Sentencia END ;
main	→	BEGIN Sentencia END .
Sentencia	→	ID Sentencia1 Sentencia | Condicional Sentencia | BucleFor Sentencia | BucleWhile Sentencia | BucleRepeat Sentencia | Lectura Sentencia | Impresión Sentencia | λ
Sentencia1	→	:= Expr ; | ( ) ;
unaSentencia	→	ID Sentencia1 | Condicional | BucleFor |BucleWhile | BucleRepeat | Lectura |Impresión
unaSentencia1	→	:= Expr | ( )
Asignacion	→	ID : = Expr ;
Expr	→	Termino Expr2
Expr2	→	Expr1 Expr2 | λ
Expr1	→	+ Termino | - Termino
Termino	→	Factor Termino2
Termino2	→	Termino1 Termino2 | λ
Termino1	→	* Factor | / Factor | MOD Factor
Factor	→	ID | NUM | - Factor | + Factor | (Expr)
Llamada	→	ID ( ) ;
ExprBoole	→	TermBoole ExprBoole1
ExprBoole1	→	OR TermBoole ExprBoole1 | λ
TermBoole	→	FactorBoole TermBoole1
TermBoole1	→	AND FactorBoole TermBoole1 | λ
FactorBoole	→	Expr OPREL Expr | NOT FactorBoole
Condicional	→	IF ExprBoole THEN Condicional1
Condicional1	→	BEGIN Sentencia END Conidiconal2 | unaSentencia Condicional2
Condicional2	→	ELSE Condicional3 | ;
Condicional3	→	unaSentencia ; | BEGIN Sentencia END ;
BucleFor	→	FOR ID := Expr condFor Expr DO BucleForWhile
BucleForWhile	→	BEGIN Sentencia END ; | unaSentencia ;
condFor	→	TO | DOWNTO

BucleWhile → WHILE ExprBoole DO BucleForWhile
BucleRepeat → REPEAT BucleRepeat1 UNTIL ExprBoole ;
BucleRepeat1 → Sentencia | λ
Lectura → READLN (ID masID) ;
Impresion → WRITELN (Elem masElem) ;
masElem → , Elem masElem | λ
Elem → STRINGctte | Expr