Tecnológico de Monterrey

Desarrollo de aplicaciones avanzadas de ciencias computacionales (503)

Patito - Entrega #0

TC3002B.503

Mario Alberto González Méndez - A00832313

Prof. Carlos Acuña Ocampo
Prof. José Carlos Ortiz Bayliss
Profa. Elda Guadalupe Quiroga González

15 de Octubre del 2025

Introducción:

Este documento define la Etapa 0 del compilador para el mini-lenguaje "Patito". En esta fase inicial, se establece el léxico (tokens y expresiones regulares) y la sintaxis (gramática libre de contexto, CFG) del lenguaje. El diseño se basa en los diagramas de sintaxis proporcionados y sirve como la especificación formal sobre la cual se construirán las siguientes etapas de análisis. Se incluyen, además, notas sobre las decisiones de diseño y extensiones implementadas para mejorar la funcionalidad del lenguaje base.

Léxico (Tokens y Expresiones Regulares)

Palabras reservadas:

Las siguientes palabras tienen un significado fijo en el lenguaje "Patito" y no pueden ser usadas como identificadores (ID):

programa, inicio, fin, vars, entero, flotante, escribe, letrero, si, sino, mientras, haz, nula

Notas sobre el Léxico (Extensiones)

Para mejorar la usabilidad del lenguaje, se han añadido los siguientes tokens que extienden la especificación original de los diagramas:

- Literales de Cadena (Strings): Se añade el token LETRERO (ej. "hola mundo") para permitir la impresión de cadenas de texto fijas.
- Operadores Relacionales Adicionales: Se añaden los tokens MAYORIG (>=) y MENORIG (<=) para completar el conjunto de comparaciones relacionales.

Tabla de Tokens y Regex Principales

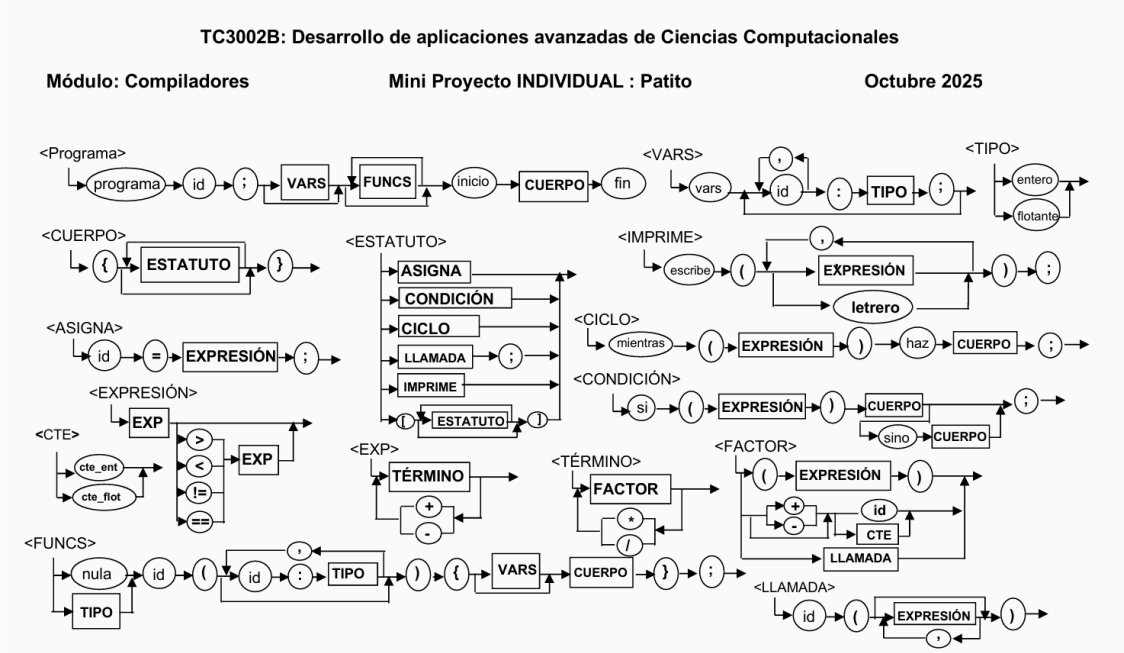
Token	Regex (simplificada)	Ejemplo	Descripción
ID	[A-Za-z_][A-Za-z_0-9]*	suma1	Identificadores
CTE_ENT	\d+	123	Constante entera
CTE_FLOT	\d+\.\d+	3.14	Constante flotante
LETRERO	"[^"\n]*"	"hola"	Cadena (extensión)
MAS	\+	+	Suma
MENOS	-	-	Resta
POR	*	*	Multiplicación
DIV	/	/	División
ASIG	=	=	Asignación
MAYORIG	>=	>=	Relacional (extensión)
MENORIG	<=	<=	Relacional (extensión)
IGUALDAD	==	==	Igualdad
DIF	!=	!=	Diferente
MAYOR	>	>	Relacional
MENOR	<	<	Relacional
LPAREN	\(	(	Paréntesis izq.
RPAREN	\)	)	Paréntesis der.
LBRACE	\{	{	Llave izq.
RBRACE	\}	}	Llave der.
COMA	,	,	Separador
PTOCOMA	;	;	Fin de sentencia
DOSPTOS	:	:	En id: tipo
PROGRAMA	(por palabra)	programa	Reservada
INICIO	(por palabra)	inicio	Reservada
FIN	(por palabra)	fin	Reservada
VARS	(por palabra)	vars	Reservada
ENTERO	(por palabra)	entero	Tipo
FLOTANTE	(por palabra)	flotante	Tipo
ESCRIBE	(por palabra)	escribe	I/O
LETRERO_KW	(por palabra)	letrero	Keyword en IMPRIME

Token	Regex (simplificada)	Ejemplo	Descripción
ID	[A-Za-z_][A-Za-z_0-9]*	suma1	Identificadores
CTE_ENT	\d+	123	Constante entera
CTE_FLOT	\d+\.\d+	3.14	Constante flotante
SI	(por palabra)	si	Condición
SINO	(por palabra)	sino	Else
MIENTRAS	(por palabra)	mientras	Ciclo
HAZ	(por palabra)	haz	Cuerpo del ciclo
NULA	(por palabra)	nula	Tipo de retorno

Espacios y comentarios: se ignoran espacios y \t; se contabilizan líneas con \n.
Comentario de linea opcional: //[^\n]*.

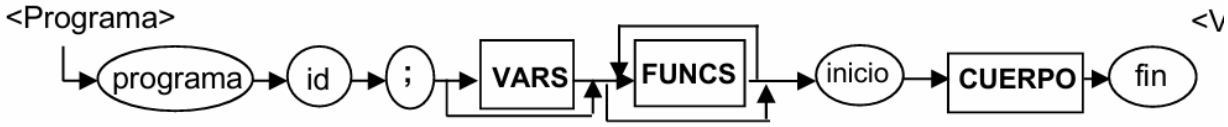
Keywords vs ID: primero se reconoce ID, luego se retipa si el lexema coincide con las reservadas.

Sintaxis (CFG)



Idea clave: <EXP> es la parte aritmética aditiva; <EXPRESION> permite opcionalmente un operador relacional entre dos EXP. Así se respeta la precedencia * // > + / - > relacionales.

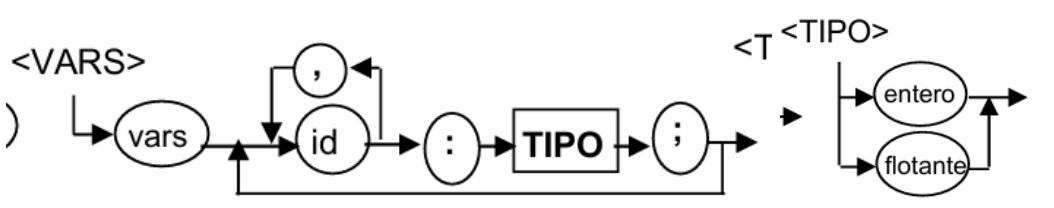
Estructura General:



<Programa>
Esta es la regla principal que define la estructura de todo el archivo.

Programa → PROGRAMA ID PTOCOMA VarsOpcional FuncsOpcional INICIO Cuerpo FIN

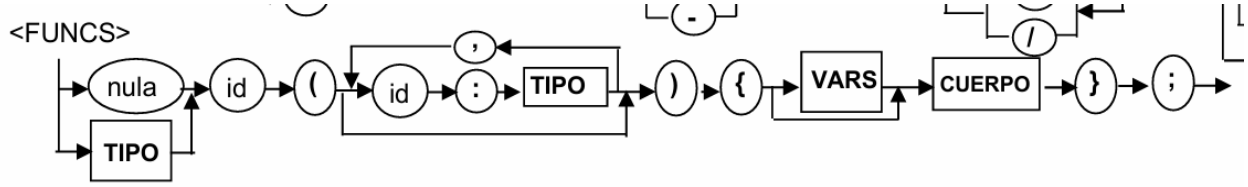
// --- Reglas auxiliares para opcionales ---
VarsOpcional → VARS ListaDeclVar | ε
FuncsOpcional → ListaFuncs | ε



<VARS> y <TIPO>
Estas reglas definen cómo se declara un bloque de variables.

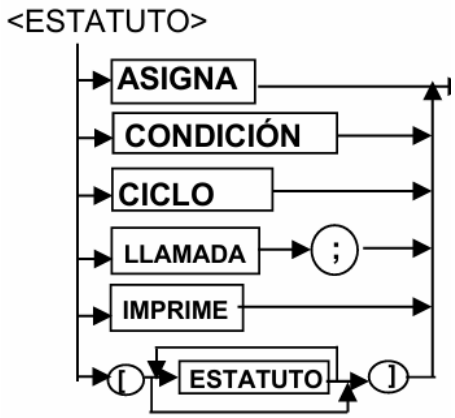
ListaDeclVar -> DeclVar | DeclVar ListaDeclVar
DeclVar -> Ids DOSPTOS Tipo PTOCOMA
Ids -> ID | Ids COMA ID

// <TIPO> define los tipos de datos.
Tipo -> ENTERO | FLOTANTE



<FUNCS>
Estas reglas definen la declaración de funciones, sus parámetros y tipo de retorno.

ListaFuncs -> FuncDef | FuncDef ListaFuncs
FuncDef -> TipoRetorno ID LPAREN Params RPAREN LBRACE VarsOpcional Cuerpo RBRACE PTOCOMA
TipoRetorno -> Tipo | NULA
Params -> ListaParams | ε
ListaParams -> ID DOSPTOS Tipo | ListaParams COMA ID DOSPTOS Tipo



<CUERPO> y <ESTATUTO>

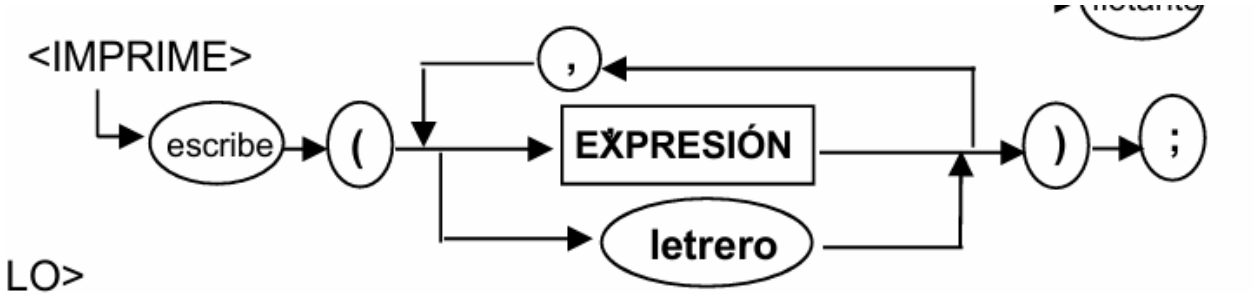
<CUERPO> es un bloque de código , y <ESTATUTO> es cada acción individual .

Cuerpo -> LBRACE ListaEstatuto RBRACE
ListaEstatuto -> Estatuto ListaEstatuto | ε

Estatuto -> Asigna
| Condicion
| Ciclo
| Imprime
| Llamada PTOCOMA
| Cuerpo // (Extensión: bloques anidados)

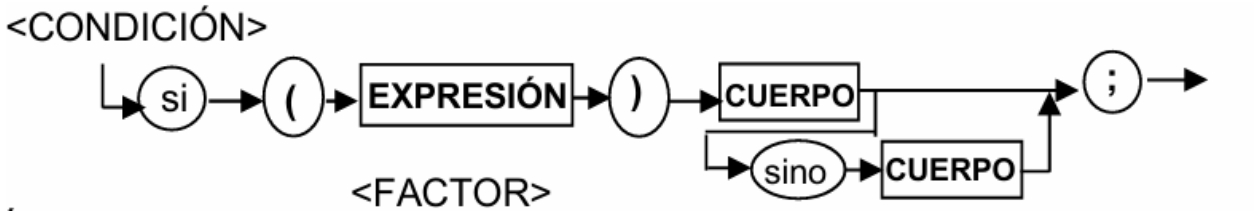
<ASIGNA>
Esta regla define la asignación de un valor a una variable.

Asigna -> ID ASIG Expresion PTOCOMA



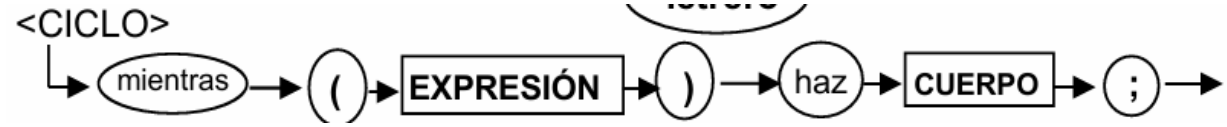
<IMPRIME>
Define el estatuto escribe, que puede imprimir expresiones, la palabra letrero o un string (extensión).

Imprime -> ESCRIBE LPAREN ListaImprime RPAREN PTOCOMA
ListaImprime -> ItemImprime | ListaImprime COMA ItemImprime
ItemImprime -> Expresion | LETRERO | LETRERO_KW // (LETRERO es extensión)



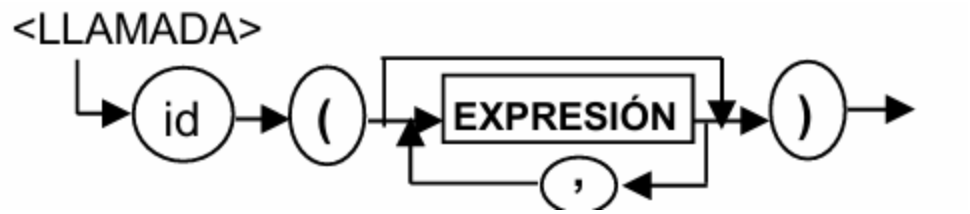
<CONDICIÓN>
Define el estatuto si / sino .

Condicion -> SI LPAREN Expresion RPAREN Cuerpo PTOCOMA
| SI LPAREN Expresion RPAREN Cuerpo SINO Cuerpo PTOCOMA
// (Decisión: añadir PTOCOMA al final por consistencia)



<CICLO>
Define el estatuto mientras .

Ciclo -> MIENTRAS LPAREN Expresion RPAREN HAZ Cuerpo PTOCOMA

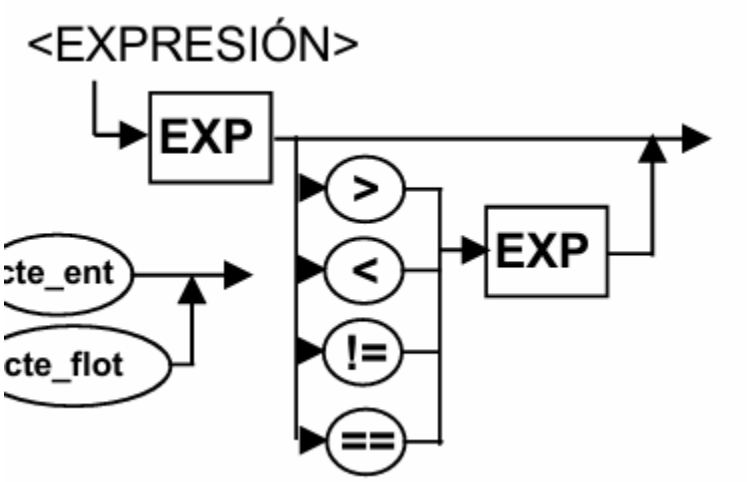


<LLAMADA>
Define la sintaxis para invocar una función, con o sin parámetros .

Llamada -> ID LPAREN RPAREN
| ID LPAREN ListaArgs RPAREN
ListaArgs -> Expresion | ListaArgs COMA Expresion

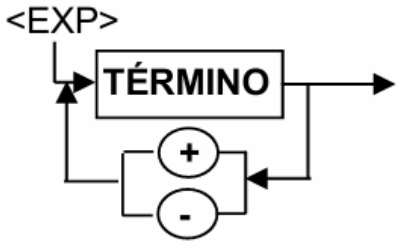
Expresiones (<EXPRESIÓN>, <EXP>, <TÉRMINO>, <FACTOR>, <CTE>)

Este es el grupo de reglas más importante para definir la precedencia de operadores.



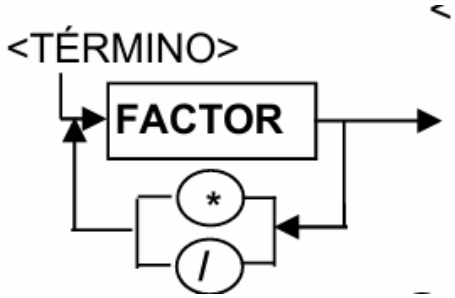
<EXPRESIÓN> (Nivel 4: Relacional)
Define comparaciones .

Expresion -> Exp
| Exp Oprel Exp
Oprel -> MAYOR | MENOR | DIF | IGUALDAD | MAYORIG | MENORIG //
(MAYORIG/MENORIG son extensión)



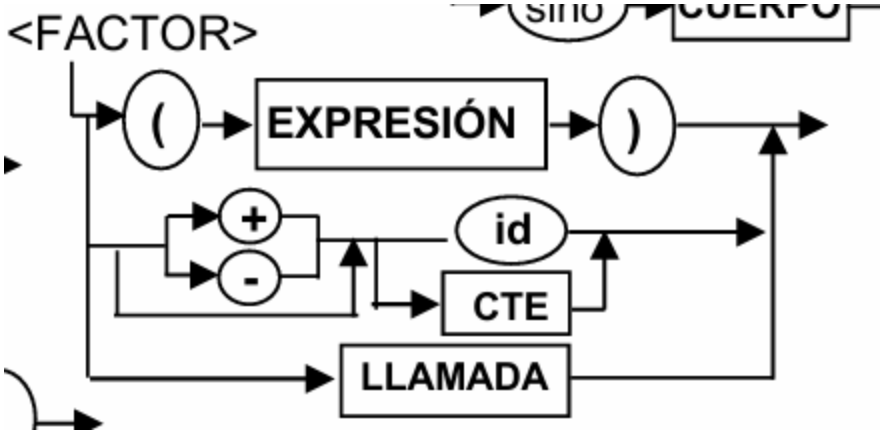
<EXP> (Nivel 3: Suma / Resta)
Define sumas y restas.

Exp -> Termino
| Exp MAS Termino
| Exp MENOS Termino



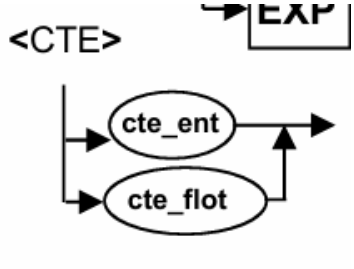
<TÉRMINO> (Nivel 2: Mult. / Div.)
Define multiplicaciones y divisiones .

Termino -> Factor
| Termino POR Factor
| Termino DIV Factor



<FACTOR> (Nivel 1: Base)
Define los elementos base de una expresión: paréntesis, unarios, llamadas, IDs y constantes .

```
Factor      -> LPAREN Expresion RPAREN
            | MAS Factor        // (Extensión: unario)
            | MENOS Factor      // (Extensión: unario)
            | Llamada           // (¡CORRECCIÓN CLAVE!)
            | ID
            | Cte               // Agrupa las constantes
```



<CTE> (Constantes)
Define las constantes numéricas.

```
Cte         -> CTE_ENT
            | CTE_FLOT
```

4. Precedencia de Operadores

La gramática ha sido diseñada para respetar una precedencia de operadores estándar. Esto se logra mediante la jerarquía de las reglas de expresión (Expresion, Exp, Termino, Factor), asegurando que las operaciones se evalúen en el orden correcto.

El orden de precedencia, de mayor a menor, es el siguiente:

1- Nivel 1 (Más Alto): Agrupación y Unarios

- ( ) (Paréntesis)
- + - (Operadores unarios, ej. -5)
- Llamada (Llamadas a función, ej. miFunc())

2- Nivel 2: Multiplicativos

- * (Multiplicación)
- / (División)
- Asociatividad: Izquierda (ej. 10 / 2 * 3 se evalúa como (10 / 2) * 3)

Nivel 3: Aditivos

- + (Suma)
- - (Resta)
- Asociatividad: Izquierda (ej. 5 - 2 + 1 se evalúa como (5 - 2) + 1)

Nivel 4 (Más Bajo): Relacionales

- > (Mayor que)
- < (Menor que)
- == (Igualdad)
- != (Diferente)
- >= (Mayor o igual que) - Extensión
- <= (Menor o igual que) - Extensión

Nota: Se permite sólo una comparación por expresión, (ej. a > b es válido, pero a > b > c no lo es).

5. Decisiones de Diseño y Extensiones

Para mejorar la funcionalidad y consistencia del lenguaje "Patito" base, se implementaron las siguientes extensiones y decisiones de diseño que difieren o aclaran los diagramas originales:

📈 Extensiones del Lenguaje

- Literales de Cadena (Strings): Se añadió el token LETRERO (ej. "hola mundo") como una opción válida en la regla Imprime. El diagrama original de <IMPRIME> solo contemplaba imprimir la palabra clave letrero o el resultado de una EXPRESIÓN.

- Operadores Relacionales Adicionales: Se añadieron los tokens MAYORIG (>=) y MENORIG (<=) a la regla Oprel. El diagrama de <EXPRESIÓN> original solo especificaba >, <, != y ==.

- Operadores Unarios: Se añadieron las producciones MAS Factor y MENOS Factor a la regla Factor. Esto permite el manejo de números positivos o negativos explícitos (ej. x = -10;). El diagrama de <FACTOR> original no incluía esta capacidad.

- Bloques de Estatutos Anidados: Se añadió la producción Cuerpo a la regla Estatuto. Esto permite anidar bloques {...} dentro de otros bloques (ej. inicio { { ... } } fin), lo cual es estándar en lenguajes procedurales pero no estaba explícito en el diagrama <ESTATUTO> .

🔧 Correcciones y Decisiones de Diseño

- Punto y Coma en Condición: Se decidió añadir un PTOCOMA (;) al final de las reglas de Condicion (estatuto si/sino). El diagrama original de <CONDICIÓN> no lo mostraba , pero sí lo hacía en <ASIGNA>, <CICLO> e <IMPRIME>. Se añade por consistencia sintáctica con el resto de los estatutos.

- Bloques Vacíos: La regla ListaEstatuto -> ... | ε permite que un <CUERPO> esté vacío (ej. {}). Los diagramas implicaban que un cuerpo debía contener al menos un estatuto. Se implementa de esta forma por flexibilidad.

Ejemplos validos

```
from parser import parser

codigo = '''
programa p;
vars
  a: entero;
  b: flotante;
inicio {
  a = 3 + 2 * 5;
  si (a >= 10) { b = a - 1; } ;
  escribe("hola", a <= 20, a == 12, a != 7);
} fin
'''

arbol = parser.parse(codigo)
print(arbol)
```

Fragmentos mínimos de implementación

Lexer (extracto)

- ID : [A-Za-z_][A-Za-z_0-9]*
- CTE_ENT : \d+
- CTE_FLOT : \d+\.\d+
- Relacionales: >= <= == != > < (de dos caracteres primero)
- Delimitadores: () {} , ; :
- Palabras reservadas: ver arriba

Parser (extracto)

- Precedencia: relacionales < + - < * /
- Reglas: ver secciones arriba la tabla y cfg

Trabajo Futuro:

Añadir funciones Funcs, tipos de retorno enteros, flotantes, nulos y parámetros.

Mejorar mensajes de error y realizar pruebas unitarias.

Código realizado para esta entrega (revisar en github):

https://github.com/elshavo/Compiladores-ML/tree/main/Compiladores/LenguajePatito