

# Requerimiento de la gramática

## Sistemas Computacionales

### Integrantes:

- Kevin Hans Juárez Pérez
- Emanuel Tolentino Santander
- Marly Galarza Acosta
- Citlali Martínez Sánchez

### Profesor:

Rodolfo Baumé

### Lenguajes Autómatas

GRUPO: B

# Introducción

En esta documentación, se presenta una descripción detallada de la gramática del lenguaje que se utilizará para nuestro proyecto. Esta gramática define las reglas y estructuras sintácticas necesarias para interpretar y evaluar expresiones aritméticas básicas, tales como la suma, resta, multiplicación y división.

El objetivo principal del analizador sintáctico es asegurar que las expresiones introducidas en la calculadora cumplan con las reglas gramaticales definidas, permitiendo así una correcta evaluación y ejecución de las operaciones aritméticas. El analizador sintáctico es una parte crucial del proceso de interpretación o compilación del lenguaje, ya que se encarga de validar la estructura del código antes de su ejecución o traducción a un lenguaje de máquina.

Esta documentación cubre todos los aspectos necesarios para la implementación del analizador sintáctico, incluyendo la definición de los símbolos terminales y no terminales, las producciones y reglas gramaticales, el símbolo inicial de la gramática, y las reglas de precedencia y asociatividad de los operadores. Además, se incluyen comentarios y anotaciones para clarificar las reglas y producciones, facilitando la comprensión y el desarrollo del analizador sintáctico.

# Símbolos terminales

Los símbolos terminales en programación incluyen operadores, palabras clave y delimitadores, cada uno con funciones específicas en el código.

- Operadores: (+ "suma", - "resta", \* "multiplicación", / "división", ^ "potencia", % "porcentaje", || "uno u otro", ! "diferente de", = "igual", > "mayor", >= "mayor o igual", < "menor", <= "menor o igual", & "y también")
- Palabras Clave: (Mod "método", Opera "librería", In "importar", Imp "imprimir", For "ciclo for", If "condicional", Else "condicional con dos casos o más", While "ciclo while", Switch "condicional", Do "ciclo con condicional", Case "condicional switch con dos casos o más", pi "valor de pi", rad "radianes", Sen "seno", Cos "coseno", Tan "tangente")
- Delimitadores: (( ) "agrupar sentencias", { } "encapsular parte de código", [ ] "determinar valores de")

Esta estructura es fundamental para escribir y entender códigos de programación.

# Producciones y Reglas

## Producción principal

```
<programa> ::= <importacion>
<operaciones_trigonometricas> <impresion>
<sentencia_if> <ciclo_for> <metodo>
<declaracion_variables> <uso_pi> <sentencia_switch>
```

## Reglas de Importación

```
<importacion> ::= "In" <libreria> ";" 
<libreria> ::= "Opera"
```

## Reglas de Operaciones Trigonométricas

```
<operaciones_trigonometricas> ::= <trig_op> <trig_op>
<trig_op> <trig_op>
<trig_op> ::= "Sen" "(" ")" | "Cos" "(" ")" | "Tan" "(" ")" | 
"Rad" "(" ")"
```

## Reglas de Impresión

```
<impresion> ::= "Imp" <caracter> ";" | "Imp" <cadena>
";"
<caracter> ::= "X" // Puedes expandir esta producción
según los caracteres permitidos
<cadena> ::= "\"" <texto> "\""
<texto> ::= "Hola Mundo" // Puedes expandir esta
producción para manejar cadenas más complejas
```

## **Reglas de Sentencia If**

```
<sentencia_if> ::= "If" <condicion> "{" <codigo> "}"  
<condicion> ::= "\"" <expresion> "\""  
<expresion> ::= "x > y" // Puedes expandir esta  
producción para manejar más expresiones  
<codigo> ::= <impresion>
```

## **Reglas de Ciclo For**

```
<ciclo_for> ::= "For" <argumentos_for> "{" <codigo> "}"  
<argumentos_for> ::= "\"" <inicio> "," <condicion> ","  
<incremento> "\""  
<inicio> ::= "i=0" // Puedes expandir esta producción  
para manejar diferentes inicios  
<condicion> ::= "i<10" // Puedes expandir esta  
producción para manejar diferentes condiciones  
<incremento> ::= "i++" // Puedes expandir esta  
producción para manejar diferentes incrementos  
<codigo> ::= <impresion>
```

## **Reglas de Métodos**

```
<metodo> ::= "Mod" <nombre_metodo> "{" <codigo> "}"  
<nombre_metodo> ::= "Suma" // Puedes expandir esta  
producción para manejar diferentes nombres de  
métodos  
<codigo> ::= <impresion>
```

## **Reglas de Declaración de Variables**

```
<declaracion_variables> ::= <variable> "=" <valor> ":"  
<variable> ::= "X"  
<valor> ::= "23"
```

## **Reglas de Uso de Pi**

<uso\_pi> ::= <numero> "\*" "Pi"

<numero> ::= "4" // Puedes expandir esta producción para manejar diferentes números

## **Reglas de Sentencia Switch**

<sentencia\_switch> ::= "Switch" <condicion\_switch> "(" <codigo> ")" <caso\_switch>

<condicion\_switch> ::= "\"" <expresion> "\"

<expresion> ::= "x < y" // Puedes expandir esta producción para manejar más expresiones

<codigo> ::= <impresion>

<caso\_switch> ::= "case" "(" <codigo> ")" | ε

# Símbolo Inicial

Para dar comienzo al programa será necesario inicializar con el símbolo o letra E, referida a las (expresiones) para comenzar con la redacción del código, esto debido a que se desea que el lenguaje sea interpretado en otros lenguajes de programación (Paradigma), para que sea más usado y accesible.

Usamos esta letra para que representara nuestro programa de los demás ya que queremos que sea único y marque la diferencia al resto. A lo largo de nuestra carrera nos hemos encontrado con diferentes lenguajes y sus distintas maneras de programar, muchas veces es diferente, en ocasiones son muy distintas las diferencias. Nosotros queremos que nuestro programa sea de fácil escritura manteniendo la sintaxis acostumbrada de modo que sea sencillo programar en nuestro lenguaje.

Así que, a partir de E, se pueden definir las reglas de derivación para las diferentes operaciones aritméticas como lo son (suma, resta, multiplicación, división) y los operandos (números, paréntesis).

# Precedencia y Asociatividad

El programa obedece las reglas jerárquicas

Expresiones

( )

Sen( ), Cos( ), tan( )

Raiz( )

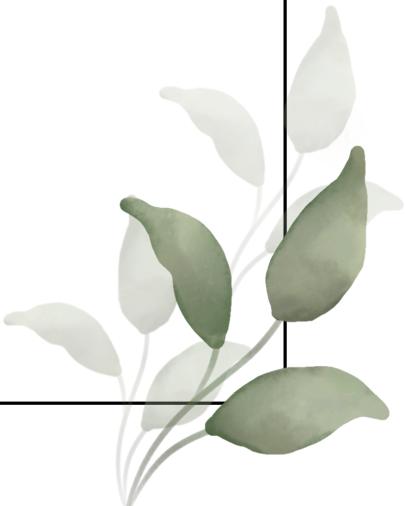
Pot( )

/

\*

-

+



# Comentarios y Anotaciones

Comentarios:

- ('left', 'Agrupar\_I','Agrupar\_F'): Los paréntesis ( y ) se utilizan para agrupar expresiones. La precedencia de agrupación asegura que las operaciones dentro de los paréntesis se realicen primero. La asociación left indica que las expresiones se evalúan de izquierda a derecha dentro de los paréntesis.
- ('left', 'Seno', 'Coseno','Tangente'): Las funciones trigonométricas Sen, Cos y Tan tienen una precedencia alta. La asociación left significa que si hay varias funciones trigonométricas en una expresión, se evaluarán de izquierda a derecha.
- ('left', 'Potencia'): El operador de potencia ^ tiene una alta precedencia para asegurar que las potencias se calculen antes que otras operaciones aritméticas como multiplicación y suma. La asociación left se refiere a evaluar potencias de izquierda a derecha, aunque en muchos lenguajes la potencia es asociativa a la derecha.

- ('left', 'Division','Multiplicacion'): Los operadores de multiplicación \* y división / tienen una precedencia media, inferior a la potencia pero superior a la suma y resta. La asociación left significa que se evalúan de izquierda a derecha.
- ('left', 'Resta','Suma'): Los operadores de suma + y resta - tienen la precedencia más baja entre los operadores aritméticos. La asociación left indica que las operaciones se realizan de izquierda a derecha.

