# Práctica Nro. 2 Sintaxis

**Objetivo**: conocer como se define léxicamente un lenguaje de programación y cuales son las herramientas necesarias para hacerlo

**Ejercicio 1**: Complete el siguiente cuadro:

|  |  |  |  |
| --- | --- | --- | --- |
| **Meta símbolos utilizados por** | | **Símbolo utilizado en Diagramas sintacticos** | **Significado** |
| **BNF** | **EBNF** |  |  |
| palabra terminal | palabra terminal | Ovalo | Definición de un elemento terminal |
|  |  | rectángulo | Definición de un elemento no terminal |
| ::= | ::= | diagrama con rectángulos, óvalos y flechas | Se lee como “Significa” / “Se expande en” / “Puede ser” |
| | | (|) | flecha que se divide en dos o más caminos | Significa “o” (Alternativa) |
| < p > < p1 > | {} |  | Repetición |
|  | \* |  | Repetición de 0 o más veces |
|  | + |  | Repetición de 1 o más veces |
|  | [] |  |  |

Nota: p y p1 son producciones simbólicas

**Ejercicio 2:** ¿Cuál es la importancia de la sintaxis para un lenguaje? ¿Cuáles son sus elementos?

RTA {

### **Importancia de la Sintaxis**

1. **Precisión y claridad:** Permite que el código sea comprensible tanto para la máquina como para los programadores.
2. **Ejecución sin errores:** Evita errores de interpretación y facilita la detección de fallos en el código.
3. **Facilita el mantenimiento:** Un código bien estructurado y con sintaxis clara es más fácil de modificar y depurar.
4. **Compatibilidad:** Permite que los programas funcionen correctamente en los entornos adecuados.

### **Elementos de la Sintaxis**

■ Alfabeto o conjunto de caracteres

■ Identificadores

■ Operadores

■ Comentarios y uso de blancos

■ Palabra clave y palabra reservada

}

**Ejercicio 3:** ¿Explique a qué se denomina regla lexicográfica y regla sintáctica?

RTA {

■ Reglas léxicas: Conjunto de reglas para formar las “word”, a partir de los caracteres del alfabeto.

Respuesta de ChatGPT: “Define cómo se deben formar los **tokens** (las unidades mínimas de significado) a partir de los caracteres en el código fuente. Estas reglas son establecidas por el **analizador léxico (lexer o scanner)** en un compilador o intérprete.”

“•Diferencias entre mayúsculas y minúsculas •Símbolo de distinto. En C != en Pascal <>”

■ Reglas sintácticas: Conjunto de reglas que definen cómo formar a partir de esas palabras, las “expresiones” y “sentencias”.

Respuesta de ChatGPT: “Las reglas sintácticas definen la **estructura válida del código** en función de los tokens generados por el análisis léxico. Son establecidas por el **analizador sintáctico (parser)** del compilador.”

“•El If en C no lleva “”then””, en Pascal si”

}

**Ejercicio 4:**¿En la definición de un lenguaje, a qué se llama palabra reservadas? ¿A qué son equivalentes en la definición de una gramática? De un ejemplo de palabra reservada en el lenguaje que más conoce. (Ada,C,Ruby,Python,..)

RTA{

Palabra reservada, son palabras claves que además no pueden ser usadas por el programador como identificador de otra entidad.

En la definición de una gramática, son equivalentes al conjunto de símbolos terminales.

Ejemplos en Java: if, else, public, private, protected, int, double, float, char, boolean.

}

**Ejercicio 5:** Dada la siguiente gramática escrita en BNF:

G= ( N, T, S, P)

N = {<numero\_entero>, <digito> } T = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

S = <numero\_entero> P = {

<numero\_entero>::=<digito><numero\_entero> | <numero\_entero><digito> | <digito>

<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

}

1. Identifique las componentes de la misma
2. Indique porqué es ambigua y corríjala

**Ejercicio 6:** Defina en BNF (Gramática de contexto libre desarrollada por Backus- Naur) la gramática para la definición de una palabra cualquiera.

RTA {

<palabra> ::= <letra> | <letra> <palabra>

<letra> ::= “a”|…|”z”|”A”|…|”Z”|

}

**Ejercicio 7:** Defina en EBNF la gramática para la definición de números reales. Inténtelo desarrollar para BNF y explique las diferencias con la utilización de la gramática EBNF.

RTA {

BNF:

<numero\_real> ::= <signo> <dígitos> | <signo> <dígitos> <decimal> | <signo> <dígitos> <exponente> | <signo> <dígitos> <decimal> <exponente>

<signo> ::= "+" | "-" | ""

<dígitos > ::= <digito> | <digito> <dígitos>

<digito> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<decimal> ::= "." <dígitos>

<exponente> ::= "e" <signo> <dígitos> | "E" <signo> <dígitos>

EBNF

numero\_real ::= [ "+" | "-" ] dígitos [ "." dígitos ] [ ("e" | "E") [ "+" | "-" ] dígitos ]

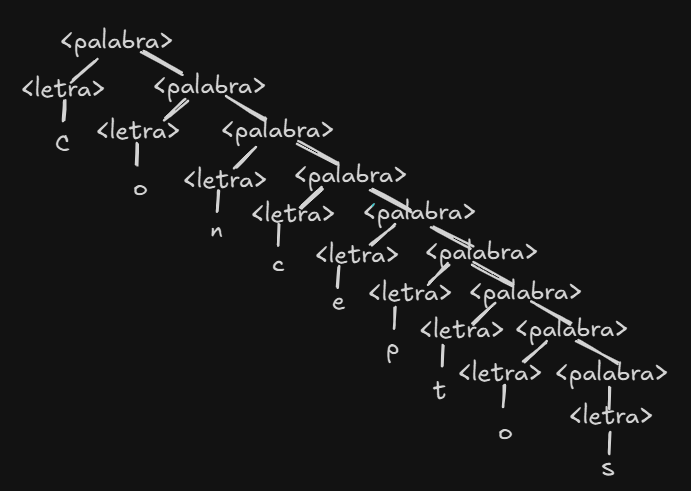
dígitos ::= digito { digito }

digito ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

}

**Ejercicio 8:** Utilizando la gramática que desarrolló en los puntos 6 y 7, escriba el árbol sintáctico de:

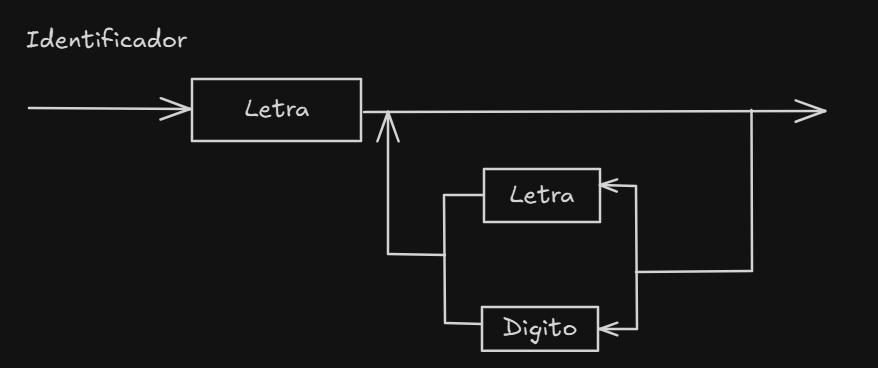
* 1. Conceptos



* 1. Programación
  2. 1255869
  3. 854,26
  4. Conceptos de lenguajes

**Ejercicio 9:** Defina utilizando diagramas sintácticos la gramática para la definición de un identificador de un lenguaje de programación. Tenga presente como regla que un identificador no puede comenzar con números.

RTA {



}

# Ejercicio 10:

1. Defina con EBNF la gramática para una expresión numérica, dónde intervienen variables y números. Considerar los operadores +, -, \* y / sin orden de prioridad. No considerar el uso de paréntesis.

RTA {

G=(N,T,S,P)

N={<expresion\_numerica>}

T={+,-,\*,/}

S={<expresión\_numerica>}

P={

<expresión\_numerica> ::= <operando> {<operador> <operando>}+

<operando> ::= <numero>|<variable>

<numero> ::= [-]{<digito>}\*

<digito> ::= “0”|”1”|”2”|”3”|”4”|”5”|”6”|”7”|”8”|”9”

<variable> ::= <letra>{<letra>|<digito>}+

<letra> ::= “a”|…|”z”|”A”|…|”Z”

<operador> “”= “+”|”-”|”\*”|”/”

}

}

1. A la gramática definida en el ejercicio anterior agregarle prioridad de operadores.

RTA {

CONSULTAR

}

1. Describa con sus palabras los pasos y decisiones que tomó para agregarle prioridad de operadores al ejercicio anterior.

**Ejercicio 11:** La siguiente gramática intenta describir sintácticamente la sentencia for de ADA, indique cuál/cuáles son los errores justificando la respuesta.

N= {<sentencia\_for>, <bloque>, <variable>, <letra>, <cadena>, <digito>, <otro>, <operacion>,

<llamada\_a\_funcion>, <numero>, <sentencia> }

P= {

<sentencia\_for>::= for (i= IN 1..10) loop <bloque> end loop;

<variable>::= <letra> | <cadena>

<cadena>::= { ( <letra> | <digito> | <otro> ) }+ //Deberia ser “\*” y no “+”.

<letra>::=( a | .. | z | A | .. | Z ) //No usa comillas y el paréntesis no existe en BNF/EBNF.

<digito>::= ( 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 ) //Idem anterior.

<bloque>::= <sentencia> | <sentencia> <bloque> | <bloque> <sentencia> ; //Es ambiguo.

<sentencia>::= <sentencia\_asignacion> | <llamada\_a\_funcion> | <sentencia\_if> |

<sentencia\_for> | <sentencia\_while> | <sentencia\_switch> //No están definidos ninguno de los elementos no terminales.

}

**Ejercicio 12:** Realice en EBNF la gramática para la definición un tag div en html 5. (Puede ayudarse con el siguiente enlace (<https://developer.mozilla.org/es/docs/Web/HTML/Elemento/div>)

RTA {  
div ::= "<div" [atributos] ">" <contenido> "</div>"

atributos ::= {<atributo>}\*

atributo ::= <nombre\_atributo> "=" '"'<valor\_atributo>'"' ;

nombre\_atributo ::= letra { letra | digito | "\_" | "-" }+

valor\_atributo ::= {carácter}\*

contenido ::= { <texto> | <otro\_elemento> }

otro\_elemento ::= "<" <nombre\_elemento> <atributos> ">" <contenido> "</" <nombre\_elemento>">"

nombre\_elemento ::= letra { letra | digito | "\_" | "-" }+

texto ::= { <carácter> }\* ;

caracter ::= “ “|..|”/”|”:”|..|”>”|”{”|..|”~”|””

letra ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z" ;

digito ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;  
}

**Ejercicio 13:** Defina en EBNF una gramática para la construcción de números primos ¿Qué debería agregar a la gramática para completar el ejercicio?

**RTA {QUE CARAJOS ESTE EJERCICIO, NO SE PUEDE HACER}**

**Ejercicio 14:** Sobre un lenguaje de su preferencia escriba en EBNF la gramática para la definición de funciones o métodos o procedimientos (considere los parámetros en caso de ser necesario)

RTA {

<metodo> ::= <modificadores> <tipo\_retorno> <nombre\_metodo> "(" <parametros> ")" "{" <bloque\_codigo> "}" ;

<modificadores> ::= { "public" | "private" | "protected" | "static" | "final" | "abstract" | "synchronized" } ;

<tipo\_retorno> ::= "void" | <tipo> ;

<nombre\_metodo> ::= <letra> { <letra> | <digito> | "\_" } ;

<parametros> ::= [ <parametro> { "," <parametro> } ] ;

<parametro> ::= <tipo> <nombre\_variable> ;

<tipo> ::= "int" | "double" | "float" | "boolean" | "char" | "String" | "long" | "short" | "byte" | <nombre\_clase> ;

<nombre\_variable> ::= <letra> { <letra> | <digito> | "\_" } ;

<bloque\_codigo> ::= { <sentencia> } ;

<sentencia> ::= <declaracion> | <asignacion> | <retorno> | <llamada\_metodo> | <estructura\_control> ;

<declaracion> ::= <tipo> <nombre\_variable> ";" ;

<asignacion> ::= <nombre\_variable> "=" <expresion> ";" ;

<retorno> ::= "return" <expresion> ";" ;

<estructura\_control> ::= <if> | <while> | <for> | <switch> ;

<if> ::= "if" "(" <expresion> ")" "{" <bloque\_codigo> "}" [ "else" "{" <bloque\_codigo> "}" ] ;

<while> ::= "while" "(" <expresion> ")" "{" <bloque\_codigo> "}" ;

<for> ::= "for" "(" <asignacion> <expresion> ";" <asignacion> ")" "{" <bloque\_codigo> "}" ;

<switch> ::= "switch" "(" <expresion> ")" "{" { "case" <expresion> ":" <bloque\_codigo> "break" ";" } [ "default" ":" <bloque\_codigo> ] "}" ;

<expresion> ::= <termino> { ("+" | "-" | "\*" | "/") <termino> } ;

<termino> ::= <factor> { ("\*" | "/") <factor> } ;

<factor> ::= <numero> | <nombre\_variable> | "(" <expresion> ")" ;

<numero> ::= [ "+" | "-" ] <digito> { <digito> } ;

<nombre\_clase> ::= <letra> { <letra> | <digito> | "\_" } ;

<letra> ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z" ;

<digito> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

}