Practica 2

Siguiente
Anterior

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

- Ejercicio 1 Complete el siguiente cuadro
- Ejercicio 2 ¿Cuál es la importancia de la sintaxis para un lenguaje?
- Ejercicio 3 ¿Explique a qué se denomina regla lexicográfica y regla sintáctica?
- Ejercicio 4 ¿En la definición de un lenguaje, a qué se llama palabra reservadas?
- Ejercicio 5 Dada la siguiente gramática escrita en BNF
- Ejercicio 6 Defina en BNF
- Ejercicio 7 Defina en EBNF la gramática para la definición de números reales
- Ejercicio 8 Utilizando la gramática que desarrolló en los puntos 6 y 7
- Ejercicio 9 Defina utilizando diagramas sintácticos la gramática para la definición
- Ejercicio 10
- Ejercicio 11 : La siguiente gramática intenta describir sintácticamente la sentencia for de ADA
- Ejercicio 12 Realice en EBNF la gramática para la definición un tag div en html 5
- Ejercicio 13 Defina en EBNF una gramática para la construcción de números primos
- Ejercicio 14 Sobre un lenguaje de su preferencia escriba en EBNF la gramática para la definición

BNF Para la Practica

- Pagina para testear
- Pagina para EBNF

En la gramática de Backus-Naur Form (BNF) se utilizan diferentes símbolos para representar elementos gramaticales y construcciones sintácticas. Aquí te presento los símbolos más comunes utilizados en la notación BNF:

- `::=` se utiliza para indicar la definición de una regla de producción.
- `|` se utiliza para separar opciones dentro de una misma regla de producción.
- `< >` se utilizan para indicar no terminales (símbolos no léxicos o variables).
- `" "` o `' '` se utilizan para indicar literales o símbolos terminales (símbolos léxicos o constantes).
- `[...]` se utiliza para indicar que un elemento es opcional.
- `{...}` se utiliza para indicar que un elemento se puede repetir cero o más veces.
- `(...)` se utiliza para agrupar elementos.

Ejercicio 1

Aca va el cuadro de la fotocopia

Ejercicio 2

¿Cuál es la importancia de la sintaxis para un lenguaje? ¿Cuáles son sus elementos?

La sintaxis es fundamental en cualquier lenguaje de programación ya que es la que permite definir las reglas y estructuras para escribir el código fuente de manera clara, concisa y coherente. La sintaxis es esencial para que el programa pueda ser interpretado y ejecutado correctamente por la máquina, ya que esta solo entiende un conjunto específico de instrucciones bien definidas. La sintaxis también ayuda a que el código sea más fácil de leer y entender por otros programadores.

Los elementos de la sintaxis en un lenguaje de programación incluyen:

- **`Palabras reservadas:`**
  son las palabras que tienen un significado especial en el lenguaje y que no pueden ser utilizadas como nombres de variables o funciones. Por ejemplo, en JavaScript, las palabras reservadas son if, else, for, while, function, etc.
- **`Identificadores:`**
  son los nombres de variables, funciones, objetos, clases, etc. Los identificadores deben seguir ciertas reglas de nomenclatura, como no contener espacios, comenzar con una letra o un guión bajo, etc.
- **`Operadores:`**
  son los símbolos que representan operaciones matemáticas, lógicas, de comparación, etc. Por ejemplo, en JavaScript, los operadores incluyen +, -, *, /, &&, ||, >, <, etc.
- **`Delimitadores:`**
  son los símbolos que se utilizan para separar o agrupar elementos en el código. Por ejemplo, en JavaScript, los delimitadores incluyen (), {}, [], ;, etc.
- Literales: son los valores constantes que aparecen directamente en el código fuente. Por ejemplo, en JavaScript, los literales pueden ser números, cadenas de texto, booleanos, objetos, etc.
- **`Comentarios:`**
  son los textos que se utilizan para explicar el código o hacer anotaciones para otros

programadores. Los comentarios no son interpretados por la máquina y no afectan el funcionamiento del programa.

En resumen, la sintaxis es crucial para cualquier lenguaje de programación, ya que define las reglas y estructuras para escribir código legible y coherente. Sin una sintaxis bien definida, el código sería difícil de interpretar y ejecutar por la máquina, y sería difícil de entender para otros programadores.

Ejercicio 3

¿Explique a qué se denomina regla lexicográfica y regla sintáctica?

Las reglas lexicográficas y sintácticas son fundamentales en la definición y construcción de lenguajes formales, incluyendo los lenguajes de programación. Las reglas lexicográficas definen cómo se deben estructurar y utilizar los elementos léxicos de un lenguaje, mientras que las reglas sintácticas definen cómo se deben estructurar y utilizar los elementos sintácticos del lenguaje. Ambas reglas son esenciales para garantizar que los programas escritos en un lenguaje formal sean interpretados y ejecutados correctamente.

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,..)

En la definición de un lenguaje, se llama palabras reservadas a aquellas palabras que tienen un significado especial y están reservadas para su uso por parte del lenguaje en sí mismo. Estas palabras no pueden ser utilizadas como identificadores o nombres de variables en el código del programa.

En la definición de una gramática, las palabras reservadas son equivalentes a los símbolos no terminales o las producciones que representan las estructuras sintácticas especiales del lenguaje.

En el lenguaje de programación Python, algunas palabras reservadas incluyen:

- `if:` se utiliza para declarar una condición que debe cumplirse para que se ejecute un bloque de código.
- `else:` se utiliza para declarar un bloque de código que se ejecutará si la condición del if no se cumple.
- `while:` se utiliza para declarar un bucle que se ejecutará mientras se cumpla una condición.
- `for:` se utiliza para declarar un bucle que se ejecutará un número determinado de veces.
- `def:` se utiliza para declarar una función en Python.
- `class:` se utiliza para declarar una clase en Python.

Estas palabras reservadas tienen un significado especial en Python y no pueden ser utilizadas como nombres de variables o funciones en el código.

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
}
```

Compilado

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

- a) Identifique las componentes de la misma
  - N: conjunto de símbolos no terminales, que en este caso son <numero_entero> y <digito>.
  - T: conjunto de símbolos terminales, que son los dígitos del 0 al 9.
  - S: símbolo inicial, que es <numero_entero>.
  - P: conjunto de producciones, que definen cómo se generan las cadenas válidas en el lenguaje. En este caso, hay dos producciones para <numero_entero> y una para <digito>. Las producciones indican que un <numero_entero> puede ser generado concatenando un <digito> y otro <numero_entero>, o bien concatenando un <numero_entero> y un <digito>, o bien siendo simplemente un <digito>. La producción para <digito> indica que un <digito> puede ser cualquiera de los dígitos del 0 al 9.
- b) Indique porqué es ambigua y corríjala

La gramática es ambigua porque una misma cadena puede ser generada por más de un árbol de derivación. Por ejemplo, la cadena "123" puede ser generada de dos formas diferentes:

- **<numero_entero>** →
  - <digito> <numero_entero> →
  - 1 <numero_entero> →
  - 1 <digito> <numero_entero> →
  - 1 2 <numero_entero> →
  - 1 2 3
- **<numero_entero>** →
  - <numero_entero> <digito>
  - → 12 <digito>
  - → 1 2 3

Para corregir la ambigüedad, se puede modificar la gramática de varias formas posibles. Aquí se presenta una opción:

```
N = {<numero_entero>, <digito>, <resto_numero>}
T = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
S = <numero_entero>
P = {
    <numero_entero> ::= <digito> <resto_numero>
```

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

Compilado

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

La principal diferencia es que se ha introducido un nuevo símbolo no terminal <resto_numero>, que se encargará de generar los dígitos restantes después del primer dígito. Además, se ha modificado la producción de <numero_entero> para que sea la concatenación de un <digito> y un <resto_numero>, y se ha añadido una nueva producción para <resto_numero> que indica que puede ser la concatenación de un <digito> y otro <resto_numero>, o bien ser la cadena vacía ε.

Con esta modificación, la gramática ya no es ambigua, ya que cada cadena generada solo tiene un árbol de derivación posible.

Ejercicio 6

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

```
<text> ::= <character> <text> | <character>
<character> ::= <letter> | <digit>
<letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N"
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

O tambien

```
<texto> ::= <letra> <texto> | <letra>
<letra> ::= [A-Z] | [a-z]
```

Ejercicio 7

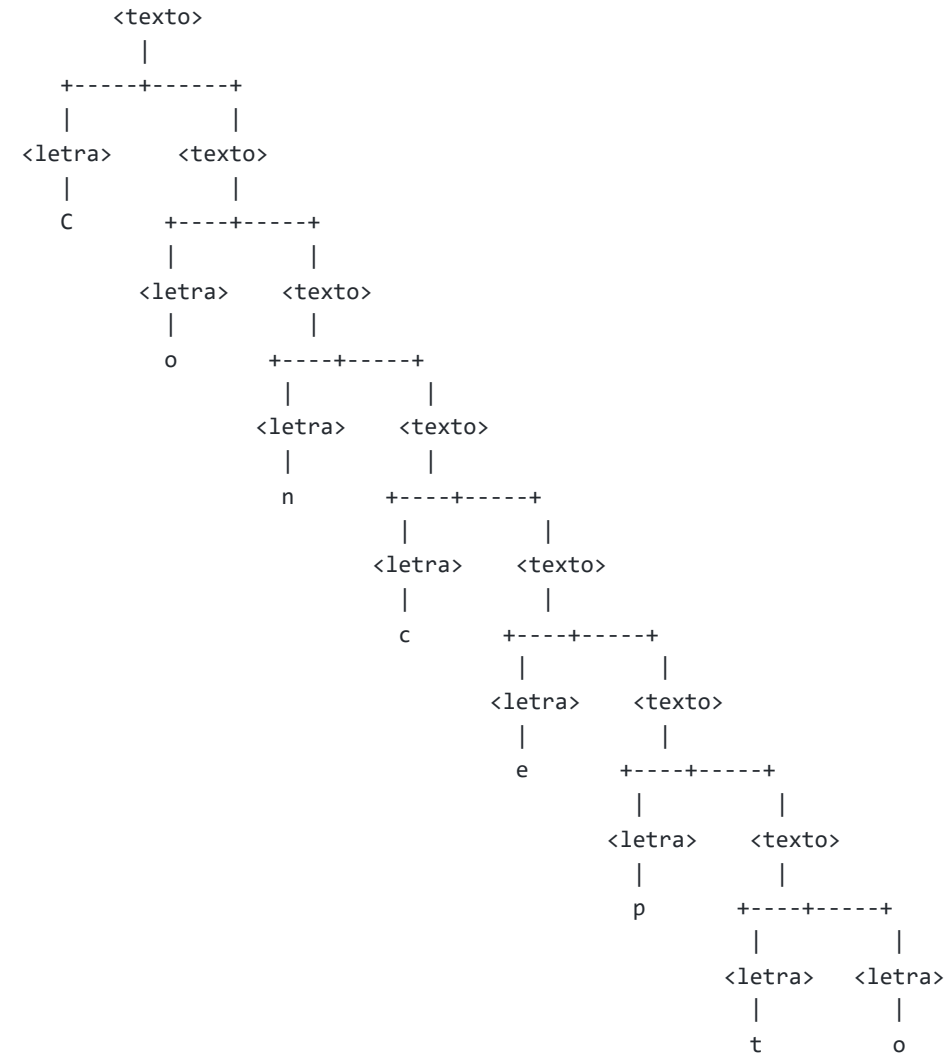
Defina en EBNF la gramática para la definición de números reales. Inténtelo desarrollar para BNF y explique las diferencias con la utilización de la gramática EBNF.

```
<real> ::= ([0-9] [0-9]*) ("." [0-9]+ )?
```

Ejercicio 8

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

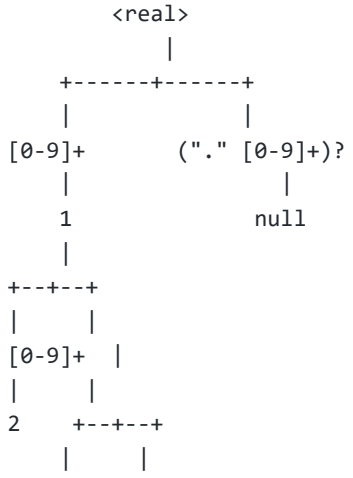
a) Conceptos

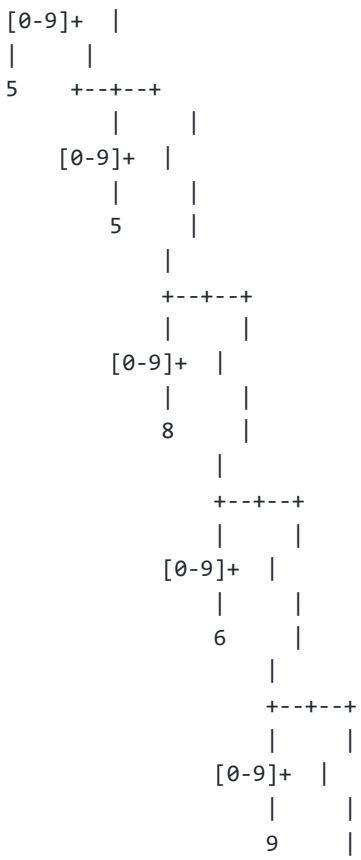


En este árbol, cada nodo interno representa una regla de la gramática, y los nodos hoja representan las letras individuales de la palabra "Conceptos". La palabra se genera siguiendo la regla `<texto> ::= <letra> <texto> | <letra>`, lo que significa que cada letra se agrega a la cadena de texto a través de la regla `<letra> <texto>` hasta que no quedan más letras para agregar.

b) Programación

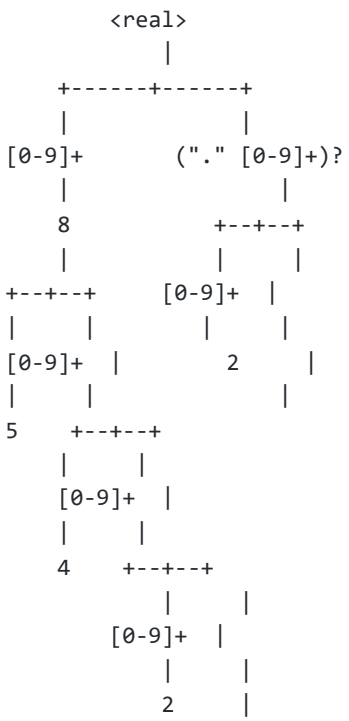
c) 1255869





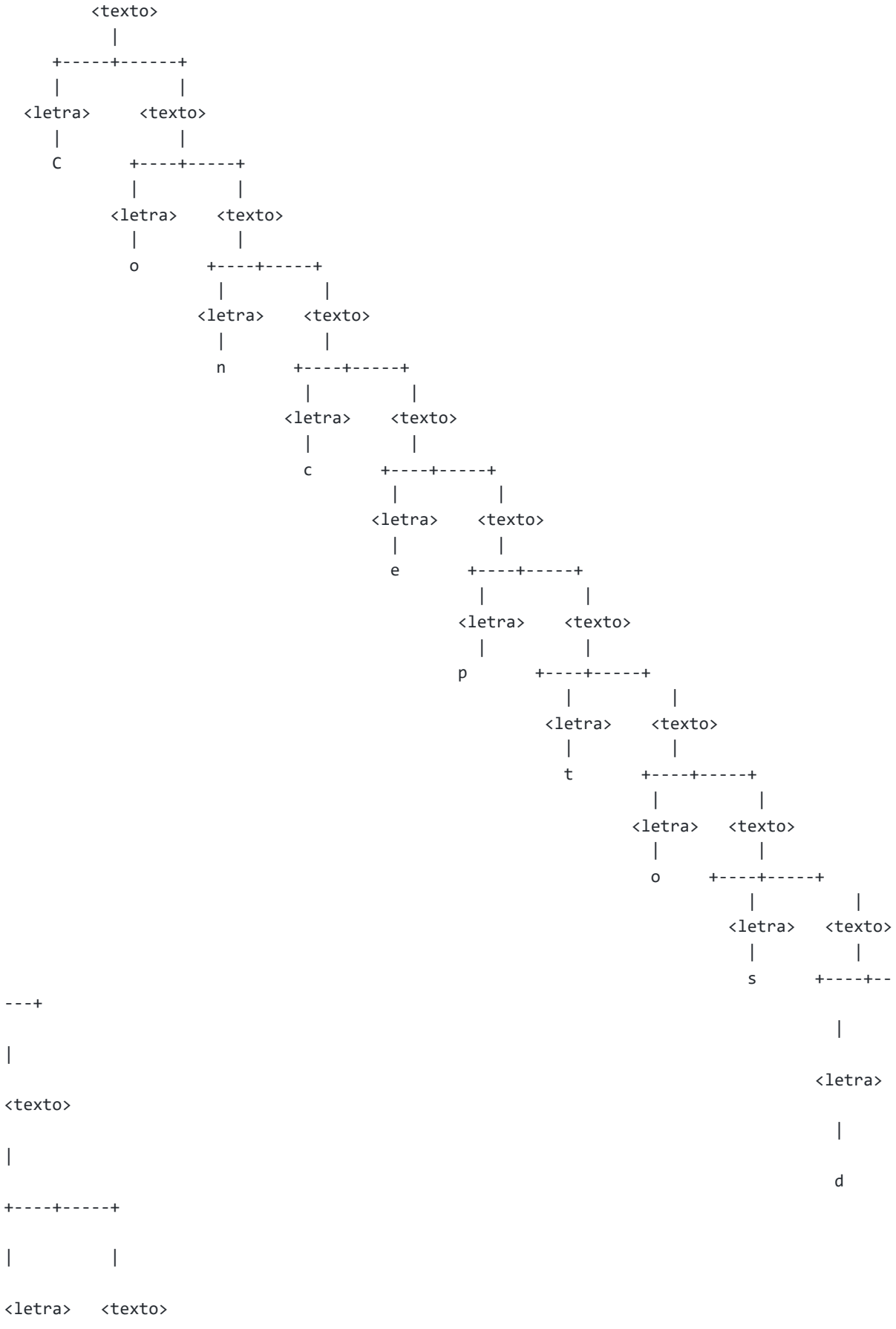
En este árbol, cada nodo interno representa una regla de la gramática, y los nodos hoja representan los valores reales. Aquí, el valor real es 1255869, que se compone de seis dígitos enteros. Entonces, la regla `<real>` se satisface simplemente tomando `[0-9]+` como la primera parte de la regla, y la segunda parte opcional `(("." [0-9]+)?)` se omite por completo.

d) 854,26



En este árbol, la primera parte de la regla `<real>` se satisface al tomar `[0-9]+` como la primera secuencia de dígitos (8, 5 y 4). Luego, la segunda parte de la regla se satisface al tomar el "." seguido de la secuencia de dígitos "26" como la parte decimal del número.

e) Conceptos de lenguajes



```
|          |

e     +----+-----+

|          |

<letra>    <texto>

|          |

l       +----+-----+

|          |

<letra>    <texto>

|          |

e       +----+-----+

|          |

<letra>   <letra>

|          |

n          g
```

En este árbol, cada nodo interno representa una regla de la gramática, y los nodos hoja representan las letras individuales de la frase "Conceptos de lenguajes". La frase se genera siguiendo la regla `<texto> ::= <letra> <texto> | <letra>` , lo que significa que cada letra y cada espacio se agregan a la cadena de texto a través de la regla `<letra> <texto>` hasta que no quedan más letras o espacios para agregar.

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.

```
<real> ::= <letra> ( (([0-9]) | <letra> )*)
<letra> ::= ([A-Z] | [a-z])
```

Arbol de definición

```
              identificador
                    |
                +---+----+
                |        |
               letra  letra/digito/_
                |
              +--+---+
```

```
        |      |
      letra  letra/digito/_
```

Ejercicio 10

- a) 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.

```
<operacion> ::= <numero> <signo> <numero>
<numero> ::= [0-9]+
<signo> ::= "+" | "-" | "*" | "/"
```

- b) A la gramática definida en el ejercicio anterior agregarle prioridad de operadores.

```
<operacion> ::= <termino> | <termino> <signo> <operacion>
<signo> ::= "+" | "-"
<termino> ::= <numero> | <numero> <multiplicativo> <termino>
<numero> ::= [0-9]+
<multiplicativo> ::= "*" | "/"
```

- c) Describa con sus palabras los pasos y decisiones que tomó para agregarle prioridad de operadores al ejercicio anterior.
  Basicamente lo primero que hago es determinar las multiplicaciones y divisiones del programa y dejo las sumas para el final.

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> ) }+
    <letra>::=( a | .. | z | A | .. | Z )
    <digito>::= ( 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 )
    <bloque>::=  <sentencia> | <sentencia> <bloque> | <bloque> <sentencia> ;
    <sentencia>::= <sentencia_asignacion> | <llamada_a_funcion> | <sentencia_if> |
    <sentencia_for> |  <sentencia_while> | <sentencia_switch>
}
```

- Para preguntar en clase pero yo hice los siguiente y mas o menos funciona. Lo unico que no mire bien bien es la sentencia `bloque`

```
<sentencia_for> ::=
  "for " <identificador> " = " " IN " " 1.." <rango>
  " loop "
      <bloque>
  " end loop;"
<identificador> ::= [a-z] | [A-Z]
<rango> ::= [1-9]+
<bloque> ::=  ([a-z] | [A-Z])*
```

- Machea `for n =  IN  1..6 loop as end loop;`

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)

```
<div> ::=
  "<div>"
    <bloque>
  "</div>"
<bloque> ::= [0-9]*
```

Ejercicio 13

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

No se puede porque tiene nros infinitos

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)

```
<div> ::= "funcion " <espacio> <identificador> <espacio> "(" <espacio> <parametros>? <espacio>
<espacio> ::= "\s"*
<letras> ::= ([a-z] | [A-Z])

<tipoDato> ::= "int" | "real" | "string"

<parametros> ::= <parametro> <parametros> | <parametro>

<parametro> ::= <tipoDato> <espacio> <identificador> ","
```

```
<identificador> ::= <letras> ( ([0-9] | <letras> )*)
```