EJERCICIO 1:

· **Reglas principales**:

* · Un **identificador** se compone de un carácter inicial (<c>), seguido de un carácter (<m>), y opcionalmente un **resto** (<resto>).

· **Subreglas**:

* · <c> puede ser un número (<n>) o una letra (<m>).
* <n> son dígitos entre 0 y 9.
* <m> son letras A, B o C.
* <resto> puede ser un solo carácter (<c>) o una cadena de caracteres recursiva.

### ****Parte a) Verificación de identificadores****

Analizamos cada ocurrencia con las reglas dadas.

a1: 1AB

* 1. El primer carácter (<c>) es 111 (válido: está en <n>).
  2. El segundo carácter (<m>) es A (válido: está en <m>).
  3. El resto (<resto>) es B, que es un carácter válido (<c> en <m>).
  4. **Resultado**: Válido.

a2: ABAc

* 1. El primer carácter (<c>) es A (válido: está en <m>).
  2. El segundo carácter (<m>) es B (válido: está en <m>).
  3. El resto comienza con A, que es un carácter válido (<c> en <m>), pero el último carácter es c, que no pertenece a \<n>∪\<m>
  4. **Resultado**: No válido.

a3: 4278

* 1. El primer carácter (<c>) es 4 (válido: está en <n>).
  2. El segundo carácter (<m>) es 2, pero esto viola las reglas, ya que el segundo carácter debe ser una letra en <m>.
  3. **Resultado**: No válido.

a4: 1B

* 1. El primer carácter (<c>) es 1 (válido: está en <n>).
  2. El segundo carácter (<m>) es B (válido: está en <m>).
  3. No hay resto adicional (<resto>), lo cual es permitido.
  4. **Resultado**: Válido.

### ****Parte b) Reglas en EBNF****

Para convertir las reglas a **EBNF** (Extended Backus-Naur Form), usamos las siguientes mejoras:

* Los corchetes [] indican opcionalidad.
* Los paréntesis () agrupan elementos.
* Los operadores \* indican repetición de 0 o más veces.

<identificador> ::= <c> <m> [<resto>]

<resto> ::= <c> {<c>}

<c> ::= <n> | <m>

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

<m> ::= "A" | "B" | "C"

#### Explicación:

1. [<resto>]: El **resto** es opcional.
2. <c> <c>: El **resto** puede ser uno o más caracteres, representados por {<c>}
3. "...": Los terminales se ponen entre comillas dobles para diferenciarlos.

### Resumen de resultados

* **Identificadores válidos**: 1AB,1B
* **Identificadores no válidos**: ABAc,4278

EJERCICIO 2:

### Reglas de la gramática

**Expresión**:

1. Expresion::=Termino{["+"∣"−"]Termino}

Una expresión puede ser un término, seguido opcionalmente por uno o más términos separados por operadores + o −

**Término**:

1. Termino::=Factor{["∗"∣"div"]Factor}

Un término puede ser un factor, seguido opcionalmente por uno o más factores separados por \* o div

**Factor**:

1. Factor::=′(′Expresion′)′∣Variable∣Constante

Un factor puede ser:

* 1. Una expresión entre paréntesis.
  2. Una variable.
  3. Una constante.

### Análisis de la expresión (9 div 3) – 4

Descomponemos la expresión de acuerdo con las reglas.

**(9 div 3):**

* 1. Es un **factor** porque está entre paréntesis. Según la regla:

Factor::=′(′Expresion′)′

* 1. Dentro de los paréntesis:
     1. 9 y 3 son **constantes** (válido por la regla de **Factor**).
     2. Div es un operador permitido en la regla de **Término**: Termino::=Factor{["∗"∣"div"]Factor}
     3. Por lo tanto, 9 div 3 es un término válido.

Así, (9 div 3)es un **factor válido**.

**(9 div 3) – 4:**

* 1. El operador − conecta (9 div 3) (un factor válido) con 4.
  2. Según la regla de **Expresión**:

Expresion::=Termino{["+"∣"−"]Termino}

* 1. (9 div 3) es un término válido (porque un término puede ser un solo factor).
  2. 4 es un término válido (porque una constante es un factor, y un factor es un término).

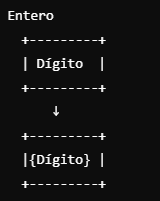
Por lo tanto, (9 div 3) − 4 es una **expresión válida**.

### Conclusión

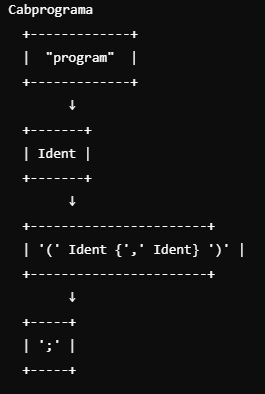
La expresión (9 div 3) – 4 es **sintácticamente correcta** según las reglas dadas en la gramática EBNF.

EJERCICIO 3:

**Entero ::= Dígito{Dígito}**



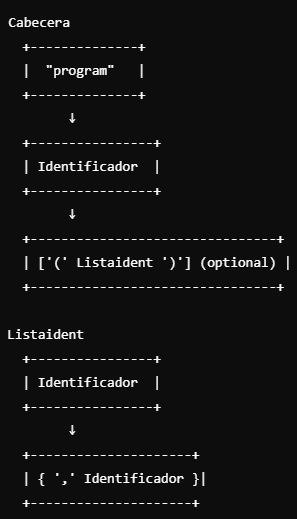
**Cabprograma ::= “program” Ident ‘(’Ident {“,” Ident} ‘)’ ”;”**



**Programa ::= Cabecera “;” Bloque “.”**

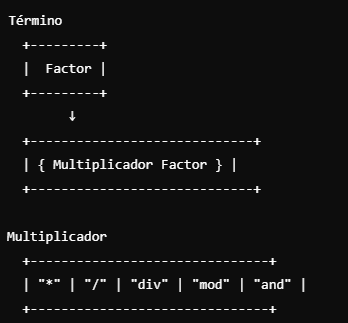
**Cabecera ::= “program” Identificador [‘(’Listaident‘)’]**

**Listaident : := Identificador {“,”Identificador}**



**Término ::= Factor {Multiplicador Factor}**

**Multiplicador ::= “\*” | “/” | “div” | “mod” | “and”**



EJERCICIO 4:

4.1

Programa ::= Cabecera Bloque "."

4.2

Cabecera ::= "PROGRAM" Identificador [ '(' Identificador { ',' Identificador } ')' ] ';'

4.3

SentenciaCompuesta ::= "BEGIN" Proposición { ';' Proposición } "END"

4.4

ExpresionSimple ::= [ '+' | '-' ] Término { ('+' | '-' | "or") Término }

4.5

Expresion ::= ExpresionSimple [ ('=' | '<' | '>' | "<>" | "<=" | ">=" | "IN") ExpresionSimple ]