

DECLARACION VARIABLE:

--- AST (Árbol de Sintaxis Abstracta) ---

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$miNumero",
        "text": "$miNumero",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "numero",
      "valor": {
        "type": "LiteralNumero",
        "value": 123
      }
    }
  ]
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. ✓

DECLARACION constante:

--- Código LibreScript ---

```
$$miConstante: numero = 456;
```

--- AST (Árbol de Sintaxis Abstracta) ---

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionConstante",
      "mutable": false,
      "nombre": {
        "type": "IDENTIFICADOR_CONST",
        "value": "$$miConstante",
        "text": "$$miConstante",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "numero",
      "valor": {
        "type": "LiteralNumero",
        "value": 456
      }
    }
  ]
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. ✓

ASIGNACION a variable existente

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js


--- Código LibreScript ---

```
$miNumero: numero = 10;
$miNumero = 20;
imprimir ($miNumero);
```

--- AST (Árbol de Sintaxis Abstracta) ---

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$miNumero",
        "text": "$miNumero",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "numero",
      "valor": {
        "type": "LiteralNumero",
        "value": 10
      }
    },
    {
      "type": "Asignacion",
      "designable": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$miNumero",
        "text": "$miNumero",
        "offset": 24,
        "lineBreaks": 0,
        "line": 2,
        "col": 1
      },
      "operador": "=",
      "valor": {
        "type": "LiteralNumero",
        "value": 20
      }
    },
    {
      "type": "Imprimir",
      "argumentos": [
        {
          "type": "Variable",
          "nombre": "$miNumero"
        }
      ]
    }
  ]
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

ASIGNACION a una constante (debería fallar semánticamente)

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---

```
$$miConstante: numero = 100;
$$miConstante = 200;
```

--- AST (Árbol de Sintaxis Abstracta) ---


```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionConstante",
      "mutable": false,
```

```

"nombre": {
  "type": "IDENTIFICADOR_CONST",
  "value": "$$miConstante",
  "text": "$$miConstante",
  "offset": 0,
  "lineBreaks": 0,
  "line": 1,
  "col": 1
},
"tipo": "numero",
"valor": {
  "type": "LiteralNumero",
  "value": 100
}
},
{
  "type": "Asignacion",
  "designable": {
    "type": "IDENTIFICADOR_CONST",
    "value": "$$miConstante",
    "text": "$$miConstante",
    "offset": 29,
    "lineBreaks": 0,
    "line": 2,
    "col": 1
  },
  "operador": "=",
  "valor": {
    "type": "LiteralNumero",
    "value": 200
  }
}
]
}

```

--- Análisis Semántico ---

Error Semántico : Lado izquierdo de asignación inválido:

```
{
  "type": "IDENTIFICADOR_CONST",
  "value": "$$miConstante",
  "text": "$$miConstante",
  "offset": 29,
  "lineBreaks": 0,
  "line": 2,
  "col": 1
}
```

Suma de números y asignación

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---

```
$resultado: numero = 10 + 5;
```

```
imprimir($resultado);
```

--- AST (Árbol de Sintaxis Abstracta) ---

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$resultado",
        "text": "$resultado",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "numero",
      "valor": {
        "type": "OpBinaria",
        "operador": "+",
        "izquierda": {
          "type": "LiteralNumero",


```

```

      "value": 10
    },
    "derecha": {
      "type": "LiteralNumero",
      "value": 5
    }
  }
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "Variable",
      "nombre": "$resultado"
    }
  ]
}
]
}
}

```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

Suma de número y texto (debería fallar semánticamente)

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---

```
$miValor: numero = 10 + "hola";
```


--- AST (Árbol de Sintaxis Abstracta) ---

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$miValor",
        "text": "$miValor",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "numero",
      "valor": {
        "type": "OpBinaria",
        "operador": "+",
        "izquierda": {
          "type": "LiteralNumero",
          "value": 10
        },
        "derecha": {
          "type": "LiteralTexto",
          "value": "hola"
        }
      }
    }
  ]
}

```

--- Análisis Semántico ---

Error Semántico : Tipo incompatible para variable '\$miValor'. Se esperaba 'numero' pero se obtuvo 'texto'.

PS C:\Users\LC-16\Desktop\Librescript-main-1>

Comparación de igualdad

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---


```
$esIguar: booleano = 5 == 5;
```

```
imprimir($esIguar);
```

--- AST (Árbol de Sintaxis Abstracta) ---

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$esIguar",
        "text": "$esIguar",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "booleano",
      "valor": {
        "type": "OpBinaria",
        "operador": "==",
        "izquierda": {
          "type": "LiteralNumero",
          "value": 5
        },
        "derecha": {
          "type": "LiteralNumero",
          "value": 5
        }
      }
    },
    {
      "type": "Imprimir",
      "argumentos": [
        {
          "type": "Variable",
          "nombre": "$esIguar"
        }
      ]
    }
  ]
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

Condicional **si**

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---

```
si (verdadero) {
  imprimir("Esto es verdadero");
}
```


--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "CondicionalSi",
      "condicion": {
        "type": "LiteralBooleano",
        "value": true
      },
      "bloqueSi": null,
    }
  ]
}
```


```
    "bloquesSiNoSi": [],
    "bloqueSiNo": null
  }
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

SiNo si

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---


```
si (falso) {
  imprimir("Esto no se imprime");
} siNo si (verdadero) {
  imprimir("Esto si se imprime");
}
```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "CondicionalSi",
      "condicion": {
        "type": "LiteralBooleano",
        "value": false
      },
      "bloqueSi": null,
      "bloquesSiNoSi": [],
      "bloqueSiNo": null
    }
  ]
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

Si — SIno

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---

```
si (falso) {
  imprimir("Esto no se imprime");
} siNo {
  imprimir("Esto si se imprime");
}
```


--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "CondicionalSi",
      "condicion": {
        "type": "LiteralBooleano",
        "value": false
      },
      "bloqueSi": null,
      "bloquesSiNoSi": [],
      "bloqueSiNo": null
    }
  ]
}
```

```
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

PS C:\Users\LC-16\Desktop\Librescript-main-1>

si Sino si siNo

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---


```
si (falso) {  
    imprimir("Esto no se imprime");  
} siNo si (falso) {  
    imprimir("Esto tampoco se imprime");  
} siNo {  
    imprimir("Esto si se imprime");  
}
```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```
{  
  "type": "Programa",  
  "sentencias": [  
    {  
      "type": "CondicionalSi",  
      "condicion": {  
        "type": "LiteralBooleano",  
        "value": false  
      },  
      "bloqueSi": null,  
      "bloquesSiNoSi": [],  
      "bloqueSiNo": null  
    }  
  ]  
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

MIENTRAS

PS C:\Users\LC-16\Desktop\Librescript-main-1> node main.js

--- Código LibreScript ---

```
$contador: numero = 0;  
mientras ($contador < 5) {  
    imprimir($contador);  
    $contador = $contador + 1;  
}
```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.


```
{  
  "type": "Programa",  
  "sentencias": [  
    {  
      "type": "DeclaracionVariable",  
      "mutable": true,  
      "nombre": {  
        "type": "IDENTIFICADOR_VAR",  
        "value": "$contador",  
        "text": "$contador",  
        "offset": 0,  
        "lineBreaks": 0,  
        "line": 1,  
        "col": 1  
      },  
      "tipo": "numero",  
      "valor": {  
        "type": "LiteralNumero",  
        "value": 0  
      }  
    }  
  ]  
}
```

```

    }
  },
  {
    "type": "BucleMientras",
    "condicion": {
      "type": "OpBinaria",
      "operador": "<",
      "izquierda": {
        "type": "Variable",
        "nombre": "$contador"
      },
      "derecha": {
        "type": "LiteralNumero",
        "value": 5
      }
    },
    "bloque": null
  }
]
}

```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

PARA

--- Código LibreScript ---

```

para ($i: numero = 0; $i < 10; $i = $i + 1) {
  imprimir($i);
}

```

--- AST (Árbol de Sintaxis Abstracta) ---

Error de parseo: Syntax error at line 1 col 35:

```

1 para ($i: numero = 0; $i < 10; $i = $i + 1) {
    ^
2   imprimir($i);
3 }

```

Unexpected OP_ASIGNACION token: "=". Instead, I was expecting to see one of the following:

A ws token based on:

```

_ $ebnf$1 → _ $ebnf$1 • %ws
_ → • _ $ebnf$1
AccesoArreglo → LiteralPrimario • _ %LBRACKET _ Expresion _ %RPAREN _ %LBRACKET _ Expresion _ %RPAREN
LiteralPrimario → • AccesoArreglo
ExpresionPostfija → • LiteralPrimario
ExpresionUnaria → • ExpresionPostfija
ExpresionPotencia → • ExpresionUnaria
ExpresionMultiplicativa → • ExpresionPotencia
ExpresionAditiva → • ExpresionMultiplicativa
ExpresionRelacional → • ExpresionAditiva
ExpresionIgualdad → • ExpresionRelacional
ExpresionLogicaAnd → • ExpresionIgualdad
ExpresionLogicaOr → • ExpresionLogicaAnd
Expresion → • ExpresionLogicaOr
IncrementoPara → • Expresion
BuclePara$ebnf$3 → • IncrementoPara
BuclePara → %PR_PARA _ %LPAREN _ BuclePara$ebnf$1 _ %PUNTO_Y_COMA _ BuclePara$ebnf$2 _ %PUNTO_Y_COMA _ •
BuclePara$ebnf$3 _ %RPAREN _ BloqueCodigo
EstructuraControl → • BuclePara
Sentencia → • EstructuraControl
Sentencias → • Sentencia _nl
Programa → _nl • Sentencias _nl

```

A RPAREN token based on:

```

BuclePara → %PR_PARA _ %LPAREN _ BuclePara$ebnf$1 _ %PUNTO_Y_COMA _ BuclePara$ebnf$2 _ %PUNTO_Y_COMA _
BuclePara$ebnf$3 _ • %RPAREN _ BloqueCodigo
EstructuraControl → • BuclePara
Sentencia → • EstructuraControl
Sentencias → • Sentencia _nl
Programa → _nl • Sentencias _nl

```

A OP_OR token based on:

ExpresionLogicaOr → ExpresionLogicaOr _ • %OP_OR _ ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl
 A OP_AND token based on:
 ExpresionLogicaAnd → ExpresionLogicaAnd _ • %OP_AND _ ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl
 A OP_EQ token based on:
 ExpresionIgualdad → ExpresionIgualdad _ • %OP_EQ _ ExpresionRelacional
 ExpresionLogicaAnd → • ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl
 A OP_NEQ token based on:
 ExpresionIgualdad → ExpresionIgualdad _ • %OP_NEQ _ ExpresionRelacional
 ExpresionLogicaAnd → • ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl
 A OP_LT token based on:
 ExpresionRelacional → ExpresionRelacional _ • %OP_LT _ ExpresionAditiva
 ExpresionIgualdad → • ExpresionRelacional
 ExpresionLogicaAnd → • ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl
 A OP_GT token based on:
 ExpresionRelacional → ExpresionRelacional _ • %OP_GT _ ExpresionAditiva
 ExpresionIgualdad → • ExpresionRelacional
 ExpresionLogicaAnd → • ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl

Programa → _nl • Sentencias _nl

A OP_LTE token based on:

- ExpresionRelacional → ExpresionRelacional _ • %OP_LTE _ ExpresionAditiva
- ExpresionIgualdad → • ExpresionRelacional
- ExpresionLogicaAnd → • ExpresionIgualdad
- ExpresionLogicaOr → • ExpresionLogicaAnd
- Expresion → • ExpresionLogicaOr
- IncrementoPara → • Expresion
- BuclePara\$sebnf\$3 → • IncrementoPara
- BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
- BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
- EstructuraControl → • BuclePara
- Sentencia → • EstructuraControl
- Sentencias → • Sentencia _nl
- Programa → _nl • Sentencias _nl

A OP_GTE token based on:

- ExpresionRelacional → ExpresionRelacional _ • %OP_GTE _ ExpresionAditiva
- ExpresionIgualdad → • ExpresionRelacional
- ExpresionLogicaAnd → • ExpresionIgualdad
- ExpresionLogicaOr → • ExpresionLogicaAnd
- Expresion → • ExpresionLogicaOr
- IncrementoPara → • Expresion
- BuclePara\$sebnf\$3 → • IncrementoPara
- BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
- BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
- EstructuraControl → • BuclePara
- Sentencia → • EstructuraControl
- Sentencias → • Sentencia _nl
- Programa → _nl • Sentencias _nl

A OP_SUMA token based on:

- ExpresionAditiva → ExpresionAditiva _ • %OP_SUMA _ ExpresionMultiplicativa
- ExpresionRelacional → • ExpresionAditiva
- ExpresionIgualdad → • ExpresionRelacional
- ExpresionLogicaAnd → • ExpresionIgualdad
- ExpresionLogicaOr → • ExpresionLogicaAnd
- Expresion → • ExpresionLogicaOr
- IncrementoPara → • Expresion
- BuclePara\$sebnf\$3 → • IncrementoPara
- BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
- BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
- EstructuraControl → • BuclePara
- Sentencia → • EstructuraControl
- Sentencias → • Sentencia _nl
- Programa → _nl • Sentencias _nl

A OP_RESTA token based on:

- ExpresionAditiva → ExpresionAditiva _ • %OP_RESTA _ ExpresionMultiplicativa
- ExpresionRelacional → • ExpresionAditiva
- ExpresionIgualdad → • ExpresionRelacional
- ExpresionLogicaAnd → • ExpresionIgualdad
- ExpresionLogicaOr → • ExpresionLogicaAnd
- Expresion → • ExpresionLogicaOr
- IncrementoPara → • Expresion
- BuclePara\$sebnf\$3 → • IncrementoPara
- BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
- BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
- EstructuraControl → • BuclePara
- Sentencia → • EstructuraControl
- Sentencias → • Sentencia _nl
- Programa → _nl • Sentencias _nl

A OP_MULT token based on:

- ExpresionMultiplicativa → ExpresionMultiplicativa _ • %OP_MULT _ ExpresionPotencia
- ExpresionAditiva → • ExpresionMultiplicativa
- ExpresionRelacional → • ExpresionAditiva
- ExpresionIgualdad → • ExpresionRelacional
- ExpresionLogicaAnd → • ExpresionIgualdad
- ExpresionLogicaOr → • ExpresionLogicaAnd
- Expresion → • ExpresionLogicaOr
- IncrementoPara → • Expresion
- BuclePara\$sebnf\$3 → • IncrementoPara
- BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
- BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
- EstructuraControl → • BuclePara
- Sentencia → • EstructuraControl
- Sentencias → • Sentencia _nl
- Programa → _nl • Sentencias _nl

A OP_DIV token based on:

ExpresionMultiplicativa → ExpresionMultiplicativa _ • %OP_DIV _ ExpresionPotencia
ExpresionAditiva → • ExpresionMultiplicativa
ExpresionRelacional → • ExpresionAditiva
ExpresionIgualdad → • ExpresionRelacional
ExpresionLogicaAnd → • ExpresionIgualdad
ExpresionLogicaOr → • ExpresionLogicaAnd
Expresion → • ExpresionLogicaOr
IncrementoPara → • Expresion
BuclePara\$sebnf\$3 → • IncrementoPara
BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
EstructuraControl → • BuclePara
Sentencia → • EstructuraControl
Sentencias → • Sentencia _nl
Programa → _nl • Sentencias _nl

A OP_MODULO token based on:

ExpresionMultiplicativa → ExpresionMultiplicativa _ • %OP_MODULO _ ExpresionPotencia
ExpresionAditiva → • ExpresionMultiplicativa
ExpresionRelacional → • ExpresionAditiva
ExpresionIgualdad → • ExpresionRelacional
ExpresionLogicaAnd → • ExpresionIgualdad
ExpresionLogicaOr → • ExpresionLogicaAnd
Expresion → • ExpresionLogicaOr
IncrementoPara → • Expresion
BuclePara\$sebnf\$3 → • IncrementoPara
BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
EstructuraControl → • BuclePara
Sentencia → • EstructuraControl
Sentencias → • Sentencia _nl
Programa → _nl • Sentencias _nl

A OP_POTENCIA token based on:

ExpresionPotencia → ExpresionUnaria _ • %OP_POTENCIA _ ExpresionPotencia
ExpresionMultiplicativa → • ExpresionPotencia
ExpresionAditiva → • ExpresionMultiplicativa
ExpresionRelacional → • ExpresionAditiva
ExpresionIgualdad → • ExpresionRelacional
ExpresionLogicaAnd → • ExpresionIgualdad
ExpresionLogicaOr → • ExpresionLogicaAnd
Expresion → • ExpresionLogicaOr
IncrementoPara → • Expresion
BuclePara\$sebnf\$3 → • IncrementoPara
BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
EstructuraControl → • BuclePara
Sentencia → • EstructuraControl
Sentencias → • Sentencia _nl
Programa → _nl • Sentencias _nl

A OP_INCREMENTO token based on:

ExpresionPostfija → ExpresionPostfija _ • %OP_INCREMENTO
ExpresionUnaria → • ExpresionPostfija
ExpresionPotencia → • ExpresionUnaria
ExpresionMultiplicativa → • ExpresionPotencia
ExpresionAditiva → • ExpresionMultiplicativa
ExpresionRelacional → • ExpresionAditiva
ExpresionIgualdad → • ExpresionRelacional
ExpresionLogicaAnd → • ExpresionIgualdad
ExpresionLogicaOr → • ExpresionLogicaAnd
Expresion → • ExpresionLogicaOr
IncrementoPara → • Expresion
BuclePara\$sebnf\$3 → • IncrementoPara
BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
EstructuraControl → • BuclePara
Sentencia → • EstructuraControl
Sentencias → • Sentencia _nl
Programa → _nl • Sentencias _nl

A OP_DECREMENTO token based on:

ExpresionPostfija → ExpresionPostfija _ • %OP_DECREMENTO
ExpresionUnaria → • ExpresionPostfija
ExpresionPotencia → • ExpresionUnaria
ExpresionMultiplicativa → • ExpresionPotencia
ExpresionAditiva → • ExpresionMultiplicativa
ExpresionRelacional → • ExpresionAditiva

ExpresionIgualdad → • ExpresionRelacional
 ExpresionLogicaAnd → • ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl

A PUNTO token based on:

AccesoMiembro → LiteralPrimario _ • %PUNTO _ %IDENTIFICADOR_GRAL
 LiteralPrimario → • AccesoMiembro
 ExpresionPostfija → • LiteralPrimario
 ExpresionUnaria → • ExpresionPostfija
 ExpresionPotencia → • ExpresionUnaria
 ExpresionMultiplicativa → • ExpresionPotencia
 ExpresionAditiva → • ExpresionMultiplicativa
 ExpresionRelacional → • ExpresionAditiva
 ExpresionIgualdad → • ExpresionRelacional
 ExpresionLogicaAnd → • ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl

A PUNTO token based on:

AccesoMiembro → LiteralPrimario _ • %PUNTO _ %ALMOHADILLA _ %IDENTIFICADOR_GRAL
 LiteralPrimario → • AccesoMiembro
 ExpresionPostfija → • LiteralPrimario
 ExpresionUnaria → • ExpresionPostfija
 ExpresionPotencia → • ExpresionUnaria
 ExpresionMultiplicativa → • ExpresionPotencia
 ExpresionAditiva → • ExpresionMultiplicativa
 ExpresionRelacional → • ExpresionAditiva
 ExpresionIgualdad → • ExpresionRelacional
 ExpresionLogicaAnd → • ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl

A LBRACKET token based on:

AccesoArreglo → LiteralPrimario _ • %LBRACKET _ Expresion _ %RBRACKET
 LiteralPrimario → • AccesoArreglo
 ExpresionPostfija → • LiteralPrimario
 ExpresionUnaria → • ExpresionPostfija
 ExpresionPotencia → • ExpresionUnaria
 ExpresionMultiplicativa → • ExpresionPotencia
 ExpresionAditiva → • ExpresionMultiplicativa
 ExpresionRelacional → • ExpresionAditiva
 ExpresionIgualdad → • ExpresionRelacional
 ExpresionLogicaAnd → • ExpresionIgualdad
 ExpresionLogicaOr → • ExpresionLogicaAnd
 Expresion → • ExpresionLogicaOr
 IncrementoPara → • Expresion
 BuclePara\$sebnf\$3 → • IncrementoPara
 BuclePara → %PR_PARA _ %LPAREN _ BuclePara\$sebnf\$1 _ %PUNTO_Y_COMA _ BuclePara\$sebnf\$2 _ %PUNTO_Y_COMA _ •
 BuclePara\$sebnf\$3 _ %RPAREN _ BloqueCodigo
 EstructuraControl → • BuclePara
 Sentencia → • EstructuraControl
 Sentencias → • Sentencia _nl
 Programa → _nl • Sentencias _nl

A LBRACKET token based on:

```
AccesoArreglo → LiteralPrimario _ • %LBRACKET _ Expresion _ %RPAREN _ %LBRACKET _ Expresion _ %RPAREN
LiteralPrimario → • AccesoArreglo
ExpresionPostfija → • LiteralPrimario
ExpresionUnaria → • ExpresionPostfija
ExpresionPotencia → • ExpresionUnaria
ExpresionMultiplicativa → • ExpresionPotencia
ExpresionAditiva → • ExpresionMultiplicativa
ExpresionRelacional → • ExpresionAditiva
ExpresionIgualdad → • ExpresionRelacional
ExpresionLogicaAnd → • ExpresionIgualdad
ExpresionLogicaOr → • ExpresionLogicaAnd
Expresion → • ExpresionLogicaOr
IncrementoPara → • Expresion
BuclePara$sebnf$3 → • IncrementoPara
BuclePara → %PR_PARA _ %LPAREN _ BuclePara$sebnf$1 _ %PUNTO_Y_COMA _ BuclePara$sebnf$2 _ %PUNTO_Y_COMA _ •
BuclePara$sebnf$3 _ %RPAREN _ BloqueCodigo
EstructuraControl → • BuclePara
Sentencia → • EstructuraControl
Sentencias → • Sentencia _nl
Programa → _nl • Sentencias _nl
```

Error cerca de la línea 1, columna 35 (offset 34). Token: '=' (tipo: OP_ASIGNACION)
No se pudo generar el AST debido a errores de parseo.

AL USAR “==” el analisis sale correcto aunque no deberia, == normalmente es un operador de comparación, no de asignación. Esto no cambia el valor de \$i, así que sería un bucle infinito (porque \$i siempre será 0 si no se modifica).


```
iMac-de-YO-2:Librescript-main-1 Rockman$ node main.js
--- Código LibreScript ---
para ($i: numero = 0; $i < 10; $i == $i + 1) {
  imprimir($i);
}
--- AST (Árbol de Sintaxis Abstracta) ---
¡Gramática ambigua! Múltiples resultados de parseo encontrados.
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "BuclePara",
      "inicializacion": {
        "type": "DeclaracionVariable",
        "mutable": true,
        "nombre": {
          "type": "IDENTIFICADOR_VAR",
          "value": "$i",
          "text": "$i",
          "offset": 6,
          "lineBreaks": 0,
          "line": 1,
          "col": 7
        },
        "tipo": "numero",
        "valor": {
          "type": "LiteralNumero",
          "value": 0
        }
      },
      "condicion": {
        "type": "OpBinaria",
        "operador": "<",
        "izquierda": {
          "type": "Variable",
          "nombre": "$i"
        },
        "derecha": {
          "type": "LiteralNumero",
          "value": 10
        }
      }
    }
  ]
}
```

```

    },
    "incremento": {
      "type": "OpBinaria",
      "operador": "==",
      "izquierda": {
        "type": "Variable",
        "nombre": "$i"
      },
    },
    "derecha": {
      "type": "OpBinaria",
      "operador": "+",
      "izquierda": {
        "type": "Variable",
        "nombre": "$i"
      },
    },
    "derecha": {
      "type": "LiteralNumero",
      "value": 1
    }
  }
},
"bloque": null
}
]
}

```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

SEGUN

iMac-de-YO-2:Librescript-main-1 Rockman\$ node main.js

--- Código LibreScript ---

\$dia: texto = "Lunes";

```

segun ($dia) {
  caso "Lunes": {
    imprimir("Hoy es lunes de trabajo.");
  } romper;
  caso "Viernes": {
    imprimir("¡Casi fin de semana!");
  } romper;
  pordefecto: {
    imprimir("Otro día de la semana.");
  }
}

```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$dia",
        "text": "$dia",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
    },
    "tipo": "texto",
    "valor": {
      "type": "LiteralTexto",
      "value": "Lunes"
    }
  ],
  {
    "type": "EstructuraSegun",
    "expresionEvaluar": {
      "type": "Variable",


```

```

    "nombre": "$dia"
  },
  "casos": [],
  "pordefecto": null
}
]
}

```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

SWITCH COMPLEJO

iMac-de-YO-2:Librescript-main-1 Rockman\$ node main.js

--- Código LibreScript ---

```
$numero: numero = 5;
```

```

segun ($numero) {
  caso 1: {
    imprimir("Uno");
  } romper;
  caso 2: { // Este caso "caería" al siguiente si no tuviera romper, pero tu gramática lo exige
    imprimir("Dos");
  } romper;
  caso 3: {
    // Bloque vacío
  } romper;
  caso 4: {
    imprimir("Cuatro");
  } romper;
  pordefecto: {
    // Bloque por defecto vacío
  }
}

```

--- Tokens ---

```

[
  {
    type: 'IDENTIFICADOR_VAR',
    value: '$numero',
    text: '$numero',
    line: 1,
    col: 1
  },
  { type: 'DOS_PUNTOS', value: ':', text: ':', line: 1, col: 8 },
  {
    type: 'TIPO_NUMERO',
    value: 'numero',
    text: 'numero',
    line: 1,
    col: 10
  },
  { type: 'OP_ASIGNACION', value: '=', text: '=', line: 1, col: 17 },
  { type: 'numero', value: '5', text: '5', line: 1, col: 19 },
  { type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 1, col: 20 },
  { type: 'PR_SEGUN', value: 'segun', text: 'segun', line: 2, col: 1 },
  { type: 'LPAREN', value: '(', text: '(', line: 2, col: 7 },
  {
    type: 'IDENTIFICADOR_VAR',
    value: '$numero',
    text: '$numero',
    line: 2,
    col: 8
  },
  { type: 'RPAREN', value: ')', text: ')', line: 2, col: 15 },
  { type: 'LBRACE', value: '{', text: '{', line: 2, col: 17 },
  { type: 'PR_CASO', value: 'caso', text: 'caso', line: 3, col: 5 },
  { type: 'numero', value: '1', text: '1', line: 3, col: 10 },
  { type: 'DOS_PUNTOS', value: ':', text: ':', line: 3, col: 11 },
  { type: 'LBRACE', value: '{', text: '{', line: 3, col: 13 },
  {
    type: 'PR_IMPRIMIR',
    value: 'imprimir',

```

```

text: 'imprimir',
line: 4,
col: 9
},
{ type: 'LPAREN', value: '(', text: '(', line: 4, col: 17 },
{ type: 'texto', value: '"Uno"', text: '"Uno"', line: 4, col: 18 },
{ type: 'RPAREN', value: ')', text: ')', line: 4, col: 23 },
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 4, col: 24 },
{ type: 'RBRACE', value: '}', text: '}', line: 5, col: 5 },
{
  type: 'PR_ROMPER',
  value: 'romper',
  text: 'romper',
  line: 5,
  col: 7
},
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 5, col: 13 },
{ type: 'PR_CASO', value: 'caso', text: 'caso', line: 6, col: 5 },
{ type: 'numero', value: '2', text: '2', line: 6, col: 10 },
{ type: 'DOS_PUNTOS', value: ':', text: ':', line: 6, col: 11 },
{ type: 'LBRACE', value: '{', text: '{', line: 6, col: 13 },
{
  type: 'comentario_linea',
  value: '// Este caso "caería" al siguiente si no tuviera romper, pero tu gramática lo exige',
  text: '// Este caso "caería" al siguiente si no tuviera romper, pero tu gramática lo exige',
  line: 6,
  col: 15
},
{
  type: 'PR_IMPRIMIR',
  value: 'imprimir',
  text: 'imprimir',
  line: 7,
  col: 9
},
{ type: 'LPAREN', value: '(', text: '(', line: 7, col: 17 },
{ type: 'texto', value: '"Dos"', text: '"Dos"', line: 7, col: 18 },
{ type: 'RPAREN', value: ')', text: ')', line: 7, col: 23 },
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 7, col: 24 },
{ type: 'RBRACE', value: '}', text: '}', line: 8, col: 5 },
{
  type: 'PR_ROMPER',
  value: 'romper',
  text: 'romper',
  line: 8,
  col: 7
},
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 8, col: 13 },
{ type: 'PR_CASO', value: 'caso', text: 'caso', line: 9, col: 5 },
{ type: 'numero', value: '3', text: '3', line: 9, col: 10 },
{ type: 'DOS_PUNTOS', value: ':', text: ':', line: 9, col: 11 },
{ type: 'LBRACE', value: '{', text: '{', line: 9, col: 13 },
{
  type: 'comentario_linea',
  value: '// Bloque vacío',
  text: '// Bloque vacío',
  line: 10,
  col: 9
},
{ type: 'RBRACE', value: '}', text: '}', line: 11, col: 5 },
{
  type: 'PR_ROMPER',
  value: 'romper',
  text: 'romper',
  line: 11,
  col: 7
},
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 11, col: 13 },
{ type: 'PR_CASO', value: 'caso', text: 'caso', line: 12, col: 5 },
{ type: 'numero', value: '4', text: '4', line: 12, col: 10 },
{ type: 'DOS_PUNTOS', value: ':', text: ':', line: 12, col: 11 },
{ type: 'LBRACE', value: '{', text: '{', line: 12, col: 13 },
{
  type: 'PR_IMPRIMIR',
  value: 'imprimir',

```



```

text: 'imprimir',
line: 13,
col: 9
},
{ type: 'LPAREN', value: '(', text: '(', line: 13, col: 17 },
{
  type: 'texto',
  value: "'Cuatro'",
  text: "'Cuatro'",
  line: 13,
  col: 18
},
{ type: 'RPAREN', value: ')', text: ')', line: 13, col: 26 },
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 13, col: 27 },
{ type: 'RBRACE', value: '}', text: '}', line: 14, col: 5 },
{
  type: 'PR_ROMPER',
  value: 'romper',
  text: 'romper',
  line: 14,
  col: 7
},
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 14, col: 13 },
{
  type: 'PR_PORDEFECTO',
  value: 'pordefecto',
  text: 'pordefecto',
  line: 15,
  col: 5
},
{ type: 'DOS_PUNTOS', value: ':', text: ':', line: 15, col: 15 },
{ type: 'LBRACE', value: '{', text: '{', line: 15, col: 17 },
{
  type: 'comentario_linea',
  value: '// Bloque por defecto vacío',
  text: '// Bloque por defecto vacío',
  line: 16,
  col: 9
},
{ type: 'RBRACE', value: '}', text: '}', line: 17, col: 5 },
{ type: 'RBRACE', value: '}', text: '}', line: 18, col: 1 }
]

```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.


```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$numero",
        "text": "$numero",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "numero",
      "valor": {
        "type": "LiteralNumero",
        "value": 5
      }
    },
    {
      "type": "EstructuraSegun",
      "expresionEvaluar": {
        "type": "Variable",
        "nombre": "$numero"
      },
      "casos": [],
      "pordefecto": null
    }
  ]
}

```

```
]
}
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

Llamadas a funciones

--- Código LibreScript ---

```
funcion sumar($a: numero, $b: numero): numero {
    devolver $a + $b;
}
```

```
$resultado: numero = sumar(5, 3);
imprimir($resultado);
```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionFuncion",
      "nombre": {
        "type": "IDENTIFICADOR_GRAL",
        "value": "sumar",
        "text": "sumar",
        "offset": 8,
        "lineBreaks": 0,
        "line": 1,
        "col": 9
      },
      "parametros": [
        {
          "type": "Parametro",
          "nombre": {
            "type": "IDENTIFICADOR_VAR",
            "value": "$a",
            "text": "$a",
            "offset": 14,
            "lineBreaks": 0,
            "line": 1,
            "col": 15
          },
          "tipo": "numero"
        },
        {
          "type": "Parametro",
          "nombre": {
            "type": "IDENTIFICADOR_VAR",
            "value": "$b",
            "text": "$b",
            "offset": 26,
            "lineBreaks": 0,
            "line": 1,
            "col": 27
          },
          "tipo": "numero"
        }
      ],
      "tipoRetorno": {
        "type": "DOS_PUNTOS",
        "value": ":",
        "text": ":",
        "offset": 37,
        "lineBreaks": 0,
        "line": 1,
        "col": 38
      },
      "bloque": "numero"
    },
    {
      "type": "DeclaracionVariable",
      "mutable": true,
```

```

"nombre": {
  "type": "IDENTIFICADOR_VAR",
  "value": "$resultado",
  "text": "$resultado",
  "offset": 74,
  "lineBreaks": 0,
  "line": 5,
  "col": 1
},
"tipo": "numero",
"valor": {
  "type": "LlamadaFuncion",
  "callee": {
    "type": "IdentificadorGral",
    "nombre": "sumar"
  },
  "argumentos": [
    {
      "type": "LiteralNumero",
      "value": 5
    },
    {
      "type": "LiteralNumero",
      "value": 3
    }
  ]
}
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "Variable",
      "nombre": "$resultado"
    }
  ]
}
]
}

```

--- Análisis Semántico ---

Error Semántico ●: Nodo de tipo AST no reconocido o malformado:

```
{"type": "DOS_PUNTOS", "value": ":", "text": ":", "offset": 37, "lineBreaks": 0, "line": 1, "col": 38}
```

--- Código LibreScript ---

```

funcion mostrarMensaje($msg: texto): vacio {
  imprimir("Mensaje:", $msg);
}

```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionFuncion",
      "nombre": {
        "type": "IDENTIFICADOR_GRAL",
        "value": "mostrarMensaje",
        "text": "mostrarMensaje",
        "offset": 8,
        "lineBreaks": 0,
        "line": 1,
        "col": 9
      },
      "parametros": [
        {
          "type": "Parametro",
          "nombre": {
            "type": "IDENTIFICADOR_VAR",
            "value": "$msg",
            "text": "$msg",
            "offset": 23,
            "lineBreaks": 0,
            "line": 1,

```

```

        "col": 24
      },
      "tipo": "texto"
    }
  ],
  "tipoRetorno": {
    "type": "DOS_PUNTOS",
    "value": ":",
    "text": ":",
    "offset": 35,
    "lineBreaks": 0,
    "line": 1,
    "col": 36
  },
  "bloque": "vacio"
}
]
}

```

--- Análisis Semántico ---

Error Semántico ●: Nodo de tipo AST no reconocido o malformado:

```
{ "type": "DOS_PUNTOS", "value": ":", "text": ":", "offset": 35, "lineBreaks": 0, "line": 1, "col": 36 }
```

iMac-de-YO-2:Librescript-main-1 Rockman\$ node main.js

--- Código LibreScript ---

```

funcion calcularArea($ancho: numero, $alto: numero): numero {
  $area: numero = $ancho * $alto;
  devolver $area;
}

```

```
$miAncho: numero = 10;
```

```
$miAlto: numero = 5;
```

```

$resultadoArea: numero = calcularArea($miAncho, $miAlto); // Llamada con variables
imprimir("El área es: " + $resultadoArea);

```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionFuncion",
      "nombre": {
        "type": "IDENTIFICADOR_GRAL",
        "value": "calcularArea",
        "text": "calcularArea",
        "offset": 8,
        "lineBreaks": 0,
        "line": 1,
        "col": 9
      },
      "parametros": [
        {
          "type": "Parametro",
          "nombre": {
            "type": "IDENTIFICADOR_VAR",
            "value": "$ancho",
            "text": "$ancho",
            "offset": 21,
            "lineBreaks": 0,
            "line": 1,
            "col": 22
          },
          "tipo": "numero"
        },
        {
          "type": "Parametro",
          "nombre": {
            "type": "IDENTIFICADOR_VAR",
            "value": "$alto",
            "text": "$alto",
            "offset": 37,
            "lineBreaks": 0,
            "line": 1,

```

```

        "col": 38
    },
    "tipo": "numero"
}
],
"tipoRetorno": {
    "type": "DOS_PUNTOS",
    "value": ":",
    "text": ":",
    "offset": 51,
    "lineBreaks": 0,
    "line": 1,
    "col": 52
},
"bloque": "numero"
},
{
    "type": "DeclaracionVariable",
    "mutable": true,
    "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$miAncho",
        "text": "$miAncho",
        "offset": 121,
        "lineBreaks": 0,
        "line": 6,
        "col": 1
    },
    "tipo": "numero",
    "valor": {
        "type": "LiteralNumero",
        "value": 10
    }
},
{
    "type": "DeclaracionVariable",
    "mutable": true,
    "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$miAlto",
        "text": "$miAlto",
        "offset": 144,
        "lineBreaks": 0,
        "line": 7,
        "col": 1
    },
    "tipo": "numero",
    "valor": {
        "type": "LiteralNumero",
        "value": 5
    }
},
{
    "type": "DeclaracionVariable",
    "mutable": true,
    "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$resultadoArea",
        "text": "$resultadoArea",
        "offset": 165,
        "lineBreaks": 0,
        "line": 8,
        "col": 1
    },
    "tipo": "numero",
    "valor": {
        "type": "LlamadaFuncion",
        "callee": {
            "type": "IdentificadorGral",
            "nombre": "calcularArea"
        },
        "argumentos": [
            {
                "type": "Variable",
                "nombre": "$miAncho"
            }
        ]
    }
}

```

```

    },
    {
      "type": "Variable",
      "nombre": "$miAlto"
    }
  ]
}
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "OpBinaria",
      "operador": "+",
      "izquierda": {
        "type": "LiteralTexto",
        "value": "El área es: "
      },
      "derecha": {
        "type": "Variable",
        "nombre": "$resultadoArea"
      }
    }
  ]
}
}
}
}

```

--- Análisis Semántico ---

Error Semántico ●: Nodo de tipo AST no reconocido o malformado:

```
{"type":"DOS_PUNTOS","value":".", "text":".", "offset":51, "lineBreaks":0, "line":1, "col":52}
```

iMac-de-YO-2:Librescript-main-1 Rockman\$ node main.js

--- Código LibreScript ---

```

funcion procesarDatos($nombre: texto, $edad: numero, $activo: booleano): vacio {
  imprimir("Procesando a: " + $nombre);
  si ($activo && $edad >= 18) {
    imprimir($nombre + " es un usuario activo y mayor de edad.");
  } siNo {
    imprimir($nombre + " no cumple los criterios completos.");
  }
}

```

```

procesarDatos("Juan", 25, verdadero);
procesarDatos("Maria", 17, verdadero);
procesarDatos("Pedro", 30, falso);

```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionFuncion",
      "nombre": {
        "type": "IDENTIFICADOR_GRAL",
        "value": "procesarDatos",
        "text": "procesarDatos",
        "offset": 8,
        "lineBreaks": 0,
        "line": 1,
        "col": 9
      },
      "parametros": [
        {
          "type": "Parametro",
          "nombre": {
            "type": "IDENTIFICADOR_VAR",
            "value": "$nombre",
            "text": "$nombre",
            "offset": 22,
            "lineBreaks": 0,
            "line": 1,
            "col": 23
          }
        }
      ]
    }
  ]
}

```

```

        "tipo": "texto"
    },
    {
        "type": "Parametro",
        "nombre": {
            "type": "IDENTIFICADOR_VAR",
            "value": "$edad",
            "text": "$edad",
            "offset": 38,
            "lineBreaks": 0,
            "line": 1,
            "col": 39
        },
        "tipo": "numero"
    },
    {
        "type": "Parametro",
        "nombre": {
            "type": "IDENTIFICADOR_VAR",
            "value": "$activo",
            "text": "$activo",
            "offset": 53,
            "lineBreaks": 0,
            "line": 1,
            "col": 54
        },
        "tipo": "booleano"
    }
],
"tipoRetorno": {
    "type": "DOS_PUNTOS",
    "value": ":",
    "text": ":",
    "offset": 71,
    "lineBreaks": 0,
    "line": 1,
    "col": 72
},
"bloque": "vacio"
},
{
    "type": "ExpresionSentencia",
    "expresion": {
        "type": "LlamadaFuncion",
        "callee": {
            "type": "IdentificadorGral",
            "nombre": "procesarDatos"
        },
        "argumentos": [
            {
                "type": "LiteralTexto",
                "value": "Juan"
            },
            {
                "type": "LiteralNumero",
                "value": 25
            },
            {
                "type": "LiteralBooleano",
                "value": true
            }
        ]
    }
},
{
    "type": "ExpresionSentencia",
    "expresion": {
        "type": "LlamadaFuncion",
        "callee": {
            "type": "IdentificadorGral",
            "nombre": "procesarDatos"
        },
        "argumentos": [
            {
                "type": "LiteralTexto",

```

```
{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$numeros",
        "text": "$numeros",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,

```



```

    "col": 1
  },
  "tipo": {
    "type": "TipoArreglo",
    "tipoElemento": "numero"
  },
  "valor": {
    "type": "CreacionArreglo",
    "elementos": [
      {
        "type": "LiteralNumero",
        "value": 10
      },
      {
        "type": "LiteralNumero",
        "value": 20
      },
      {
        "type": "LiteralNumero",
        "value": 30
      },
      {
        "type": "LiteralNumero",
        "value": 40
      }
    ]
  }
},
{
  "type": "DeclaracionVariable",
  "mutable": true,
  "nombre": {
    "type": "IDENTIFICADOR_VAR",
    "value": "$nombres",
    "text": "$nombres",
    "offset": 39,
    "lineBreaks": 0,
    "line": 2,
    "col": 1
  },
  "tipo": {
    "type": "TipoArreglo",
    "tipoElemento": "texto"
  },
  "valor": {
    "type": "CreacionArreglo",
    "elementos": [
      {
        "type": "LiteralTexto",
        "value": "Ana"
      },
      {
        "type": "LiteralTexto",
        "value": "Luis"
      },
      {
        "type": "LiteralTexto",
        "value": "Carlos"
      }
    ]
  }
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "OpBinaria",
      "operador": "+",
      "izquierda": {
        "type": "LiteralTexto",
        "value": "Primer numero: "
      },
      "derecha": {
        "type": "AccesoArreglo",
        "arreglo": {


```

```

      "type": "Variable",
      "nombre": "$numeros"
    },
    "indice": {
      "type": "LiteralNumero",
      "value": 0
    }
  }
}
],
},
{
  "type": "Asignacion",
  "designable": {
    "type": "AccesoArreglo",
    "arreglo": {
      "type": "Variable",
      "nombre": "$numeros"
    },
    "indice": {
      "type": "LiteralNumero",
      "value": 1
    }
  },
  "operador": "=",
  "valor": {
    "type": "LiteralNumero",
    "value": 25
  }
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "OpBinaria",
      "operador": "+",
      "izquierda": {
        "type": "LiteralTexto",
        "value": "Segundo numero actualizado: "
      },
      "derecha": {
        "type": "AccesoArreglo",
        "arreglo": {
          "type": "Variable",
          "nombre": "$numeros"
        },
        "indice": {
          "type": "LiteralNumero",
          "value": 1
        }
      }
    }
  ]
}
]
}

```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

OBJETO

--- Código LibreScript ---

```
$persona: Objeto = {nombre: "Alicia", edad:30};
```

```
imprimir($persona.nombre);
```

```
imprimir($persona.edad);
```

--- Tokens ---

```
[
  {
    type: 'IDENTIFICADOR_VAR',
```

```

value: '$persona',
text: '$persona',
line: 1,
col: 1
},
{ type: 'DOS_PUNTOS', value: ':', text: ':', line: 1, col: 9 },
{
  type: 'TIPO_OBJETO',
  value: 'Objeto',
  text: 'Objeto',
  line: 1,
  col: 11
},
{ type: 'OP_ASIGNACION', value: '=', text: '=', line: 1, col: 18 },
{ type: 'LBRACE', value: '{', text: '{', line: 1, col: 20 },
{
  type: 'IDENTIFICADOR_GRAL',
  value: 'nombre',
  text: 'nombre',
  line: 1,
  col: 21
},
{ type: 'DOS_PUNTOS', value: ':', text: ':', line: 1, col: 27 },
{
  type: 'texto',
  value: '"Alicia"',
  text: '"Alicia"',
  line: 1,
  col: 29
},
{ type: 'COMA', value: ',', text: ',', line: 1, col: 37 },
{
  type: 'IDENTIFICADOR_GRAL',
  value: 'edad',
  text: 'edad',
  line: 1,
  col: 39
},
{ type: 'DOS_PUNTOS', value: ':', text: ':', line: 1, col: 43 },
{ type: 'numero', value: '30', text: '30', line: 1, col: 44 },
{ type: 'RBRACE', value: '}', text: '}', line: 1, col: 46 },
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 1, col: 47 },
{
  type: 'PR_IMPRIMIR',
  value: 'imprimir',
  text: 'imprimir',
  line: 3,
  col: 1
},
{ type: 'LPAREN', value: '(', text: '(', line: 3, col: 9 },
{
  type: 'IDENTIFICADOR_VAR',
  value: '$persona',
  text: '$persona',
  line: 3,
  col: 10
},
{ type: 'PUNTO', value: '.', text: '.', line: 3, col: 18 },
{
  type: 'IDENTIFICADOR_GRAL',
  value: 'nombre',
  text: 'nombre',
  line: 3,
  col: 19
},
{ type: 'RPAREN', value: ')', text: ')', line: 3, col: 25 },
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 3, col: 26 },
{
  type: 'PR_IMPRIMIR',
  value: 'imprimir',
  text: 'imprimir',
  line: 4,
  col: 1
},
{ type: 'LPAREN', value: '(', text: '(', line: 4, col: 9 },

```

```

{
  type: 'IDENTIFICADOR_VAR',
  value: '$persona',
  text: '$persona',
  line: 4,
  col: 10
},
{ type: 'PUNTO', value: '.', text: '.', line: 4, col: 18 },
{
  type: 'IDENTIFICADOR_GRAL',
  value: 'edad',
  text: 'edad',
  line: 4,
  col: 19
},
{ type: 'RPAREN', value: ')', text: ')', line: 4, col: 23 },
{ type: 'PUNTO_Y_COMA', value: ';', text: ';', line: 4, col: 24 }
]

```

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$persona",
        "text": "$persona",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "Objeto",
      "valor": {
        "type": "CreacionObjetoLiteral",
        "propiedades": [
          {
            "type": "ParClaveValor",
            "clave": "nombre",
            "valor": {
              "type": "LiteralTexto",
              "value": "Alicia"
            }
          },
          {
            "type": "ParClaveValor",
            "clave": "edad",
            "valor": {
              "type": "LiteralNumero",
              "value": 30
            }
          }
        ]
      }
    },
    {
      "type": "Imprimir",
      "argumentos": [
        {
          "type": "AccesoMiembro",
          "objeto": {
            "type": "Variable",
            "nombre": "$persona"
          },
          "propiedad": "nombre",
          "accesoConAlmohadilla": false
        }
      ]
    },
    {
      "type": "Imprimir",


```

```

"argumentos": [
  {
    "type": "AccesoMiembro",
    "objeto": {
      "type": "Variable",
      "nombre": "$persona"
    },
    "propiedad": "edad",
    "accesoConAlmohadilla": false
  }
]
}
}
}

```

--- Análisis Semántico ---

Error Semántico : El operando izquierdo de '.' debe ser una instancia de clase (o 'este'). Se obtuvo 'Objeto' para el objeto que precede a '.'.nombre'.

OTRAS OPERACIONES:

iMac-de-YO-2:Librescript-main-1 Rockman\$ node main.js

--- Código LibreScript ---

```
$textoNumero: texto = "123";
```

```
$otroNumero: numero = 45;
```

```
$numeroConvertido: numero = aNum($textoNumero); // Convertir texto a numero
```

```
$resultadoSuma: numero = $numeroConvertido + $otroNumero;
```

```
imprimir("El texto convertido es: " + $numeroConvertido);
```

```
imprimir("El resultado de la suma es: " + $resultadoSuma);
```

--- AST (Árbol de Sintaxis Abstracta) ---

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$textoNumero",
        "text": "$textoNumero",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "texto",
      "valor": {
        "type": "LiteralTexto",
        "value": "123"
      }
    },
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$otroNumero",
        "text": "$otroNumero",
        "offset": 29,
        "lineBreaks": 0,
        "line": 2,
        "col": 1
      },
      "tipo": "numero",
      "valor": {
        "type": "LiteralNumero",
        "value": 45
      }
    }
  ],
}

```

```

{
  "type": "DeclaracionVariable",
  "mutable": true,
  "nombre": {
    "type": "IDENTIFICADOR_VAR",
    "value": "$numeroConvertido",
    "text": "$numeroConvertido",
    "offset": 56,
    "lineBreaks": 0,
    "line": 4,
    "col": 1
  },
  "tipo": "numero",
  "valor": {
    "type": "LlamadaFuncion",
    "callee": {
      "type": "IdentificadorGral",
      "nombre": "aNum"
    },
    "argumentos": [
      {
        "type": "Variable",
        "nombre": "$textoNumero"
      }
    ]
  }
},
{
  "type": "DeclaracionVariable",
  "mutable": true,
  "nombre": {
    "type": "IDENTIFICADOR_VAR",
    "value": "$resultadoSuma",
    "text": "$resultadoSuma",
    "offset": 133,
    "lineBreaks": 0,
    "line": 6,
    "col": 1
  },
  "tipo": "numero",
  "valor": {
    "type": "OpBinaria",
    "operador": "+",
    "izquierda": {
      "type": "Variable",
      "nombre": "$numeroConvertido"
    },
    "derecha": {
      "type": "Variable",
      "nombre": "$otroNumero"
    }
  }
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "OpBinaria",
      "operador": "+",
      "izquierda": {
        "type": "LiteralTexto",
        "value": "El texto convertido es: "
      },
      "derecha": {
        "type": "Variable",
        "nombre": "$numeroConvertido"
      }
    }
  ]
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "OpBinaria",


```

```

      "operador": "+",
      "izquierda": {
        "type": "LiteralTexto",
        "value": "El resultado de la suma es: "
      },
      "derecha": {
        "type": "Variable",
        "nombre": "$resultadoSuma"
      }
    }
  ]
}
}
}

```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 

iMac-de-YO-2:Librescript-main-1 Rockman\$ node main.js

--- Código LibreScript ---

\$a: numero = 10;

\$b: numero = 20;

\$c: numero = 5;

\$res1: numero = \$a + \$b * \$c; // Precedencia: $10 + (20 * 5) = 110$

\$res2: numero = (\$a + \$b) * \$c; // Paréntesis: $(10 + 20) * 5 = 150$

\$res3: numero = \$a ** 2; // Potencia: $10 * 10 = 100$

\$res4: numero = \$b / \$c + \$a % 3; // División ($20/5=4$), Módulo ($10\%3=1$), Suma ($4+1=5$)

\$cond1: booleano = (\$a > \$b) && (\$b < \$c); // (falso) && (falso) = falso

\$cond2: booleano = (\$a < \$b) || (\$b < \$c); // (verdadero) || (falso) = verdadero

\$cond3: booleano = !(\$a == \$b); // !(falso) = verdadero

imprimir("Resultados de expresiones:");

imprimir(\$res1); // Salida: 110

imprimir(\$res2); // Salida: 150

imprimir(\$res3); // Salida: 100

imprimir(\$res4); // Salida: 5

imprimir(\$cond1); // Salida: falso

imprimir(\$cond2); // Salida: verdadero

imprimir(\$cond3); // Salida: verdadero

\$negativo: numero = -\$a;

imprimir(\$negativo); // Salida: -10

--- AST (Árbol de Sintaxis Abstracta) ---

¡Gramática ambigua! Múltiples resultados de parseo encontrados.

```

{
  "type": "Programa",
  "sentencias": [
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$a",
        "text": "$a",
        "offset": 0,
        "lineBreaks": 0,
        "line": 1,
        "col": 1
      },
      "tipo": "numero",
      "valor": {
        "type": "LiteralNumero",
        "value": 10
      }
    },
    {
      "type": "DeclaracionVariable",
      "mutable": true,
      "nombre": {
        "type": "IDENTIFICADOR_VAR",
        "value": "$b",
        "text": "$b",

```

```

    "offset": 17,
    "lineBreaks": 0,
    "line": 2,
    "col": 1
  },
  "tipo": "numero",
  "valor": {
    "type": "LiteralNumero",
    "value": 20
  }
},
{
  "type": "DeclaracionVariable",
  "mutable": true,
  "nombre": {
    "type": "IDENTIFICADOR_VAR",
    "value": "$c",
    "text": "$c",
    "offset": 34,
    "lineBreaks": 0,
    "line": 3,
    "col": 1
  },
  "tipo": "numero",
  "valor": {
    "type": "LiteralNumero",
    "value": 5
  }
},
{
  "type": "DeclaracionVariable",
  "mutable": true,
  "nombre": {
    "type": "IDENTIFICADOR_VAR",
    "value": "$res1",
    "text": "$res1",
    "offset": 51,
    "lineBreaks": 0,
    "line": 5,
    "col": 1
  },
  "tipo": "numero",
  "valor": {
    "type": "OpBinaria",
    "operador": "+",
    "izquierda": {
      "type": "Variable",
      "nombre": "$a"
    },
    "derecha": {
      "type": "OpBinaria",
      "operador": "**",
      "izquierda": {
        "type": "Variable",
        "nombre": "$b"
      },
      "derecha": {
        "type": "Variable",
        "nombre": "$c"
      }
    }
  }
},
{
  "type": "DeclaracionVariable",
  "mutable": true,
  "nombre": {
    "type": "IDENTIFICADOR_VAR",
    "value": "$res2",
    "text": "$res2",
    "offset": 123,
    "lineBreaks": 0,
    "line": 6,
    "col": 1
  },

```



```

"tipo": "numero",
"valor": {
  "type": "OpBinaria",
  "operador": "**",
  "izquierda": {
    "type": "OpBinaria",
    "operador": "+",
    "izquierda": {
      "type": "Variable",
      "nombre": "$a"
    },
    "derecha": {
      "type": "Variable",
      "nombre": "$b"
    }
  },
  "derecha": {
    "type": "Variable",
    "nombre": "$c"
  }
},
{
  "type": "DeclaracionVariable",
  "mutable": true,
  "nombre": {
    "type": "IDENTIFICADOR_VAR",
    "value": "$res3",
    "text": "$res3",
    "offset": 194,
    "lineBreaks": 0,
    "line": 7,
    "col": 1
  },
  "tipo": "numero",
  "valor": {
    "type": "OpBinaria",
    "operador": "***",
    "izquierda": {
      "type": "Variable",
      "nombre": "$a"
    },
    "derecha": {
      "type": "LiteralNumero",
      "value": 2
    }
  },
  "tipo": "numero",
  "valor": {
    "type": "OpBinaria",
    "operador": "+",
    "izquierda": {
      "type": "OpBinaria",
      "operador": "/",
      "izquierda": {
        "type": "Variable",
        "nombre": "$b"
      },
      "derecha": {
        "type": "Variable",
        "nombre": "$c"
      }
    }
  }
}

```

```

    },
    "derecha": {
      "type": "OpBinaria",
      "operador": "%",
      "izquierda": {
        "type": "Variable",
        "nombre": "$a"
      },
      "derecha": {
        "type": "LiteralNumero",
        "value": 3
      }
    }
  },
  {
    "type": "DeclaracionVariable",
    "mutable": true,
    "nombre": {
      "type": "IDENTIFICADOR_VAR",
      "value": "$cond1",
      "text": "$cond1",
      "offset": 346,
      "lineBreaks": 0,
      "line": 10,
      "col": 1
    },
    "tipo": "booleano",
    "valor": {
      "type": "OpBinaria",
      "operador": "&&",
      "izquierda": {
        "type": "OpBinaria",
        "operador": ">",
        "izquierda": {
          "type": "Variable",
          "nombre": "$a"
        },
        "derecha": {
          "type": "Variable",
          "nombre": "$b"
        }
      },
      "derecha": {
        "type": "Variable",
        "nombre": "$c"
      }
    }
  },
  {
    "type": "DeclaracionVariable",
    "mutable": true,
    "nombre": {
      "type": "IDENTIFICADOR_VAR",
      "value": "$cond2",
      "text": "$cond2",
      "offset": 420,
      "lineBreaks": 0,
      "line": 11,
      "col": 1
    },
    "tipo": "booleano",
    "valor": {
      "type": "OpBinaria",
      "operador": "||",
      "izquierda": {
        "type": "OpBinaria",

```

```

      "operador": "<",
      "izquierda": {
        "type": "Variable",
        "nombre": "$a"
      },
      "derecha": {
        "type": "Variable",
        "nombre": "$b"
      }
    },
    "derecha": {
      "type": "OpBinaria",
      "operador": "<",
      "izquierda": {
        "type": "Variable",
        "nombre": "$b"
      },
      "derecha": {
        "type": "Variable",
        "nombre": "$c"
      }
    }
  },
  {
    "type": "DeclaracionVariable",
    "mutable": true,
    "nombre": {
      "type": "IDENTIFICADOR_VAR",
      "value": "$cond3",
      "text": "$cond3",
      "offset": 502,
      "lineBreaks": 0,
      "line": 12,
      "col": 1
    },
    "tipo": "booleano",
    "valor": {
      "type": "OpUnaria",
      "operador": "!",
      "operando": {
        "type": "OpBinaria",
        "operador": "==",
        "izquierda": {
          "type": "Variable",
          "nombre": "$a"
        },
        "derecha": {
          "type": "Variable",
          "nombre": "$b"
        }
      }
    }
  },
  {
    "type": "Imprimir",
    "argumentos": [
      {
        "type": "LiteralTexto",
        "value": "Resultados de expresiones:"
      }
    ]
  },
  {
    "type": "Imprimir",
    "argumentos": [
      {
        "type": "Variable",
        "nombre": "$res1"
      }
    ]
  },
  {
    "type": "Imprimir",
    "argumentos": [

```

```

    {
      "type": "Variable",
      "nombre": "$res2"
    }
  ],
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "Variable",
      "nombre": "$res3"
    }
  ]
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "Variable",
      "nombre": "$res4"
    }
  ]
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "Variable",
      "nombre": "$cond1"
    }
  ]
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "Variable",
      "nombre": "$cond2"
    }
  ]
},
{
  "type": "Imprimir",
  "argumentos": [
    {
      "type": "Variable",
      "nombre": "$cond3"
    }
  ]
},
{
  "type": "DeclaracionVariable",
  "mutable": true,
  "nombre": {
    "type": "IDENTIFICADOR_VAR",
    "value": "$negativo",
    "text": "$negativo",
    "offset": 851,
    "lineBreaks": 0,
    "line": 23,
    "col": 1
  },
  "tipo": "numero",
  "valor": {
    "type": "OpUnaria",
    "operador": "-",
    "operando": {
      "type": "Variable",
      "nombre": "$a"
    }
  }
},
{
  "type": "Imprimir",

```

```
"argumentos": [  
  {  
    "type": "Variable",  
    "nombre": "$negativo"  
  }  
]  
}  
]
```

--- Análisis Semántico ---

Análisis semántico completado sin errores. 