UNIVERSIDAD DEL VALLE DE GUATEMALA

Laboratorio – Construcción con Lex y Yacc

Andre Marroquin Tarot- 22266

Sergio Orellana- 221122

Brandon Reyes- 22992

Andy Fuentes - 22944

Davis Roldán - 22672

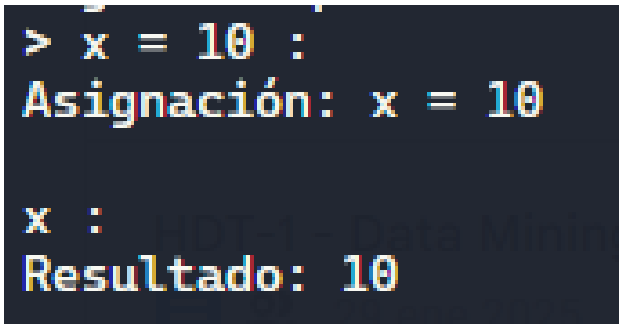
Diseño de lenguajes de programación

Guatemala, 2025

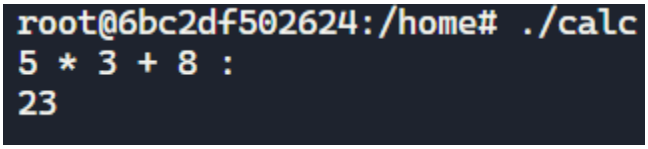
Link del repositorio:
https://github.com/SergioAle210/Laboratorio2-Compiladores

Link del video:
https://youtu.be/Q1HL5_A2nyQ

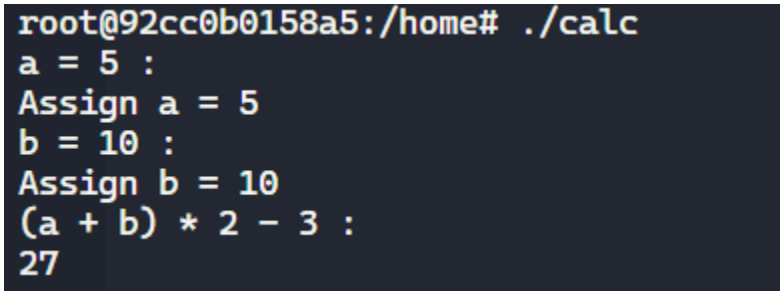
1. Cree un programa que asigne un valor a una variable.



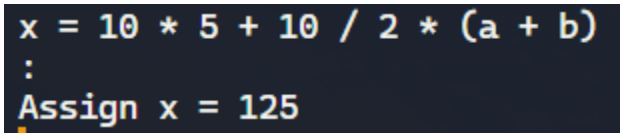
2. Cree un programa que realice una operación aritmética simple.



3. Experimente con expresiones más complejas y verifique que el compilador las procese correctamente.



4. Modifique el lenguaje para incluir la asignación de variables con expresiones aritméticas.



5. Agregue manejo de errores al compilador para detectar tokens inválidos en el programa fuente.

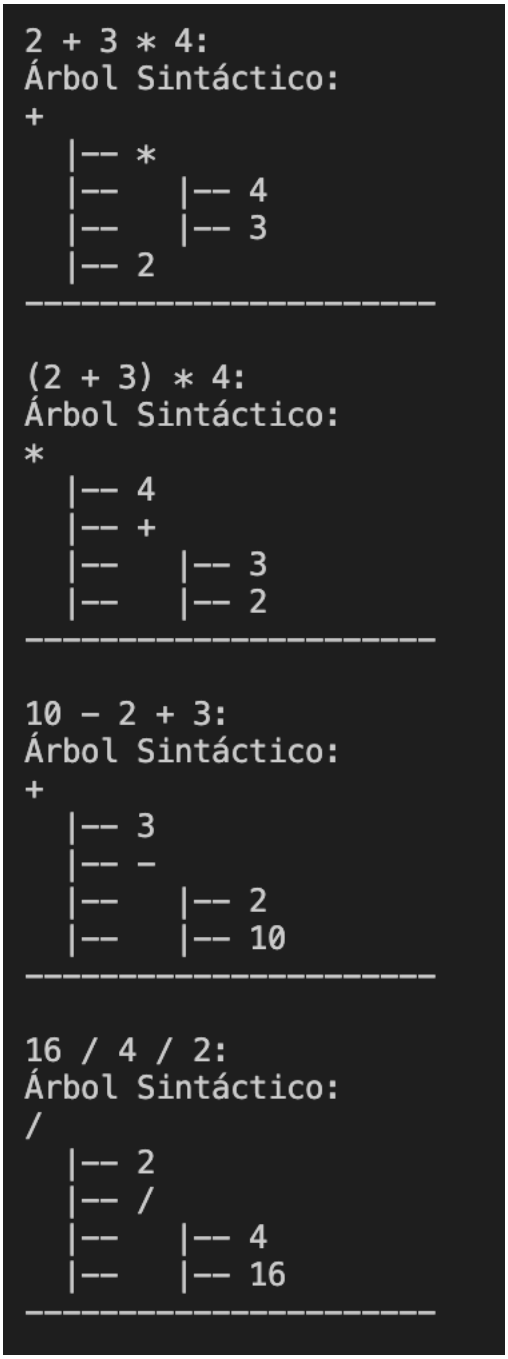
```
root@2802939526ee:/home# ./calc
Ingrese expresiones o asignaciones ('exit:' para salir):
> @ 6 / 3:
Error léxico: Token inválido '@'
Error de sintaxis: syntax error
Error: Expresión inválida.

4 # + 4:
Error léxico: Token inválido '#'
Error de sintaxis: syntax error
Error: Expresión inválida.
```

6. Experimente con la precedencia de operadores en el lenguaje y observe cómo afecta la generación del árbol sintáctico.

Precedencia original:

%right ASSIGN
%left '+' '-'
%left '*' '/'
%left '(' ')'



Otra configuración:

%right ASSIGN
%left '*' '/'
%left '+' '-'

%left '(' ')'

```
> 2 + 3 * 4:
Árbol Sintáctico:
*
  |-- 4
  |-- +
  |--   |-- 3
  |--   |-- 2
----------------------

(2 + 3) * 4:
Árbol Sintáctico:
*
  |-- 4
  |-- +
  |--   |-- 3
  |--   |-- 2
----------------------

10 - 2 + 3:
Árbol Sintáctico:
+
  |-- 3
  |-- -
  |--   |-- 2
  |--   |-- 10
----------------------

16 / 4 / 2:
Árbol Sintáctico:
/
  |-- 2
  |-- /
  |--   |-- 4
  |--   |-- 16
----------------------
```

Conclusiones:

Al hacer pruebas con la precedencia de operadores en mi lenguaje, noté cómo cambia la forma en que se construye el árbol sintáctico y se evalúan las expresiones. Con la configuración estándar, donde * y / tienen más prioridad que + y -, las expresiones se resolvieron correctamente, respetando el orden matemático esperado. Por ejemplo, 2 + 3 * 4 dio 14 porque primero se evaluó 3 * 4 y luego se sumó 2, generando un árbol sintáctico bien estructurado.
Cuando invertí la precedencia, haciendo que + y - tuvieran más prioridad, la evaluación cambió por completo. Ahora, 2 + 3 * 4 se interpretó como (2 + 3) * 4, dando un resultado incorrecto de 20. El árbol sintáctico reflejó este cambio al ubicar la suma antes que la multiplicación, lo que claramente afectó la ejecución. Algo similar ocurrió al modificar la asociatividad de * y /. En la configuración normal, 16 / 4 / 2 se resolvía de izquierda a derecha ((16 / 4) / 2 = 2), pero al cambiar la asociatividad, el cálculo pasó a 16 / (4 / 2) = 8, alterando completamente el resultado.
Esto me dejó claro que la forma en que defina la precedencia y asociatividad en Yacc tiene un impacto directo en la interpretación de las expresiones. Un pequeño ajuste en estas reglas puede cambiar por completo los cálculos y la estructura del árbol sintáctico. Es clave configurar correctamente estas reglas para evitar errores en la evaluación y asegurar que el lenguaje funcione como se espera.