



Nombre: Dana Carolina Ramírez Velázquez

Código: 220286547

Materia: Traductores de Lenguaje 2

Actividad: Tarea 8

Fecha: 25/08/22

Gramáticas regulares:

Como toda gramática está formada por la cuadrupla (V, T, P, S) ; diremos que es una gramática regular si sus reglas de producción P son de la siguiente forma:

$$A \rightarrow uB \text{ o } A \rightarrow u$$

En este caso se llama Gramática Regular Lineal por la derecha, ya que el símbolo no terminal del consecuente (parte derecha de la flecha) es el más a la derecha de los símbolos.

O bien: $A \rightarrow Bu$ o $A \rightarrow u$ se llama Gramática Regular Lineal por la izquierda porque el símbolo más a la izquierda del consecuente es el símbolo no terminal.

Autómatas finitos:

Un autómata finito (AF) o máquina de estado finito es un modelo computacional que realiza cálculos en forma automática sobre una entrada para producir una salida.

Este modelo está conformado por un alfabeto, un conjunto de estados finito, una función de transición, un estado inicial y un conjunto de estados finales. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

La finalidad de los autómatas finitos es la de reconocer lenguajes regulares, que corresponden a los lenguajes formales más simples según la Jerarquía de Chomsky.

Máquinas de estado:

Una Máquina de Estado Finito (Finite State Machine), llamada también Autómata Finito es una abstracción computacional que describe el comportamiento de un

sistema reactivo mediante un número determinado de Estados y un número determinado de Transiciones entre dicho Estados.

Las Transiciones de un estado a otro se generan en respuesta a eventos de entrada externos e internos; a su vez estas transiciones y/o subsecuentes estados pueden generar otros eventos de salida. Esta dependencia de las acciones (respuesta) del sistema a los eventos de entrada hace que las Máquinas de Estado Finito (MEF) sean una herramienta adecuada para el diseño de Sistemas Reactivos y la Programación Conducida por Eventos (Event Driven Programming), cual es el caso de la mayoría de los sistemas embebidos basados en microcontroladores o microprocesadores.

Las MEF se describen gráficamente mediante los llamados Diagramas de Estado Finito (DEF), llamados también Diagramas de Transición de Estados.

Expresión regular:

En cómputo teórico y teoría de lenguajes formales, una expresión regular, o expresión racional, también son conocidas como regex o regexp, por su contracción de las palabras inglesas regular expression, es una secuencia de caracteres que conforma un patrón de búsqueda. Se utilizan principalmente para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones.

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto. Las expresiones regulares proporcionan una manera muy flexible de buscar o reconocer cadenas de texto. Por ejemplo, el grupo formado por las cadenas Handel, Händel y Haendel se describe con el patrón "H(a|ä|ae)ndel".

La mayoría de las formalizaciones proporcionan los siguientes constructores: una expresión regular es una forma de representar los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje.

Reglas Semánticas de no recursión por la izquierda

$\text{resto_expr.hier} = \text{term.t}$

$\text{expr.t} = \text{resto_expr.t}$

$\text{resto_expr.hier}_1 = \text{resto_expr.hier} \parallel \text{term.t} \parallel '+'$

$\text{resto_expr.t} = \text{resto_expr.t}_1$

$\text{resto_expr.hier}_1 = \text{resto_expr.hier} \parallel \text{term.t} \parallel '-'$

$\text{resto_expr.t} = \text{resto_expr.t}_1$

$\text{resto_expr.hier}_1 = \text{resto_expr.hier} \parallel \text{factor.t} \parallel '/'$

$\text{resto_expr.t} = \text{resto_expr.t}_1$

$\text{resto_expr.hier}_1 = \text{resto_expr.hier} \parallel \text{factor.t} \parallel '*'$

$\text{resto_expr.t} = \text{resto_expr.t}_1$

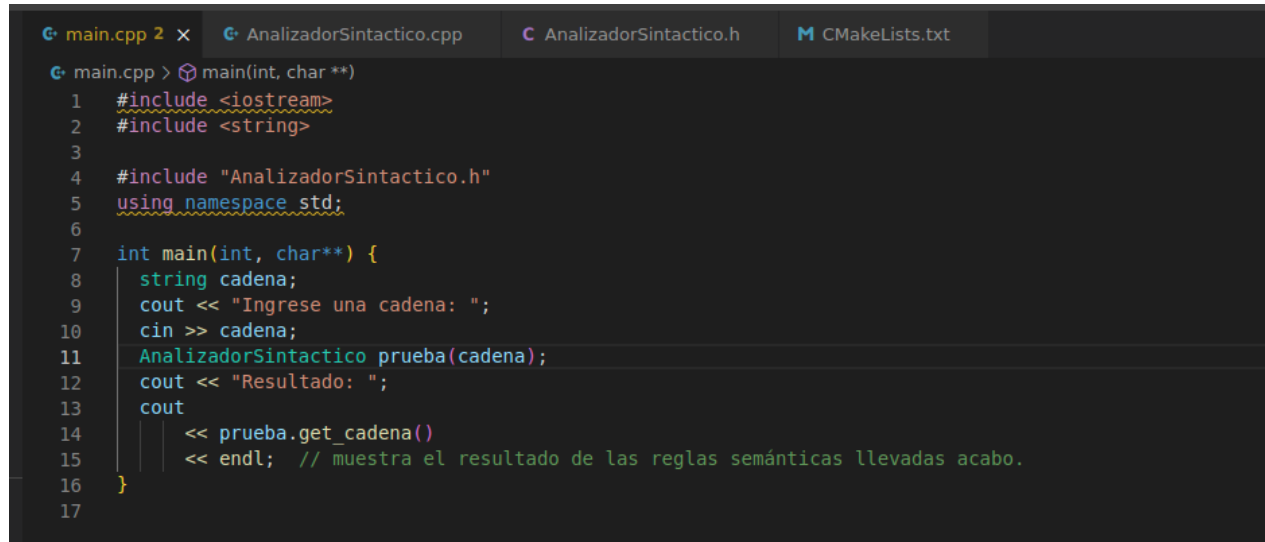
$\text{resto_expr.t} = \text{resto_expr.hier}$

$\text{factor.t} = \text{digito.t}$

$\text{factor.t} = (\text{expr.t})$

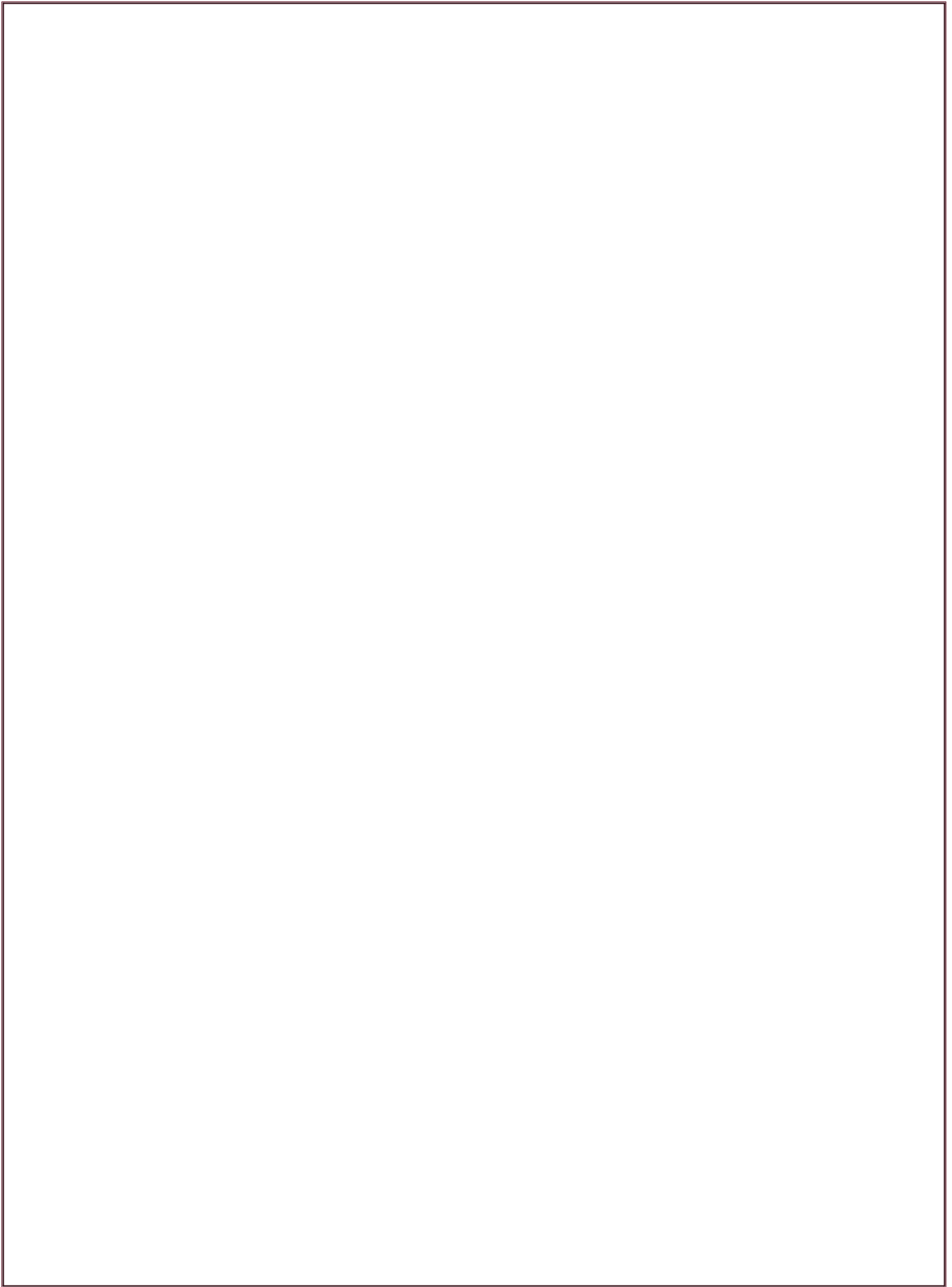
$\text{digito} = 0 \parallel \dots \parallel 9$

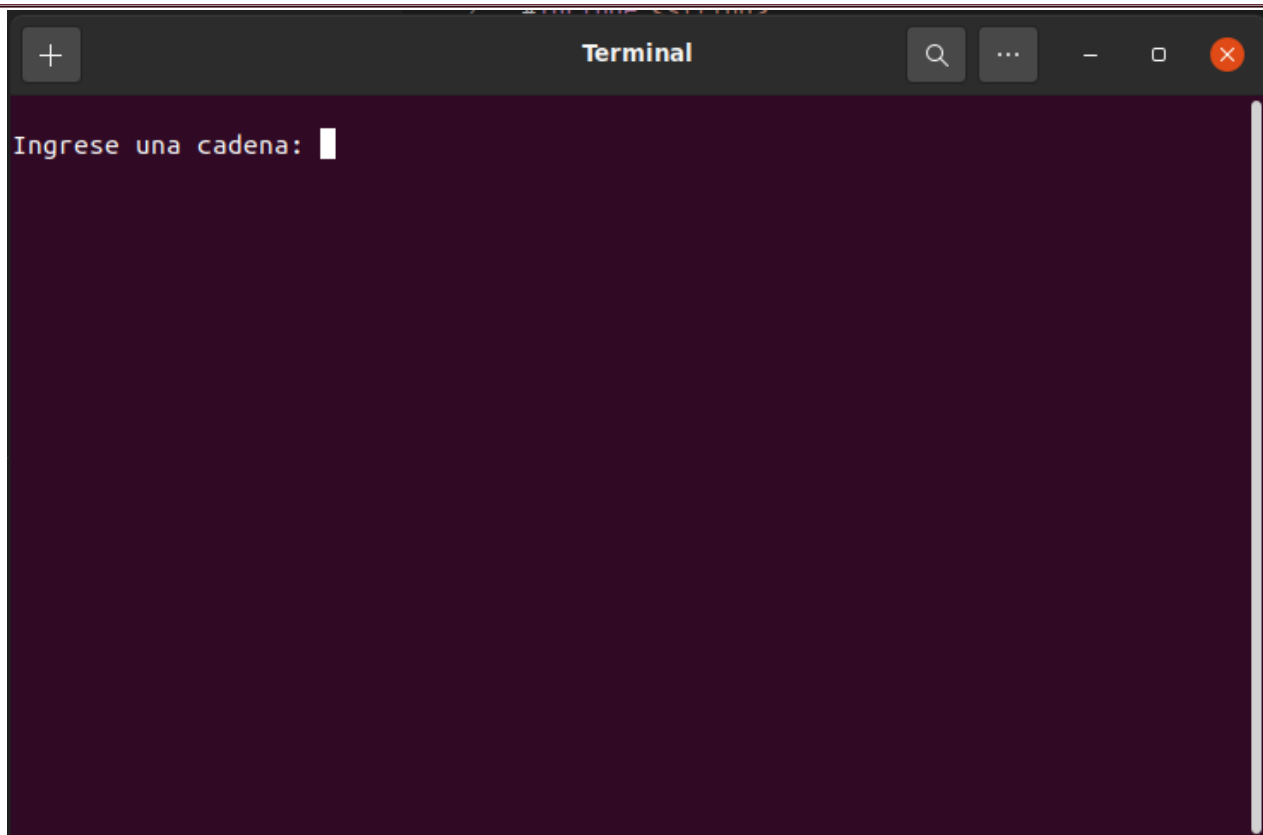
Capturas:

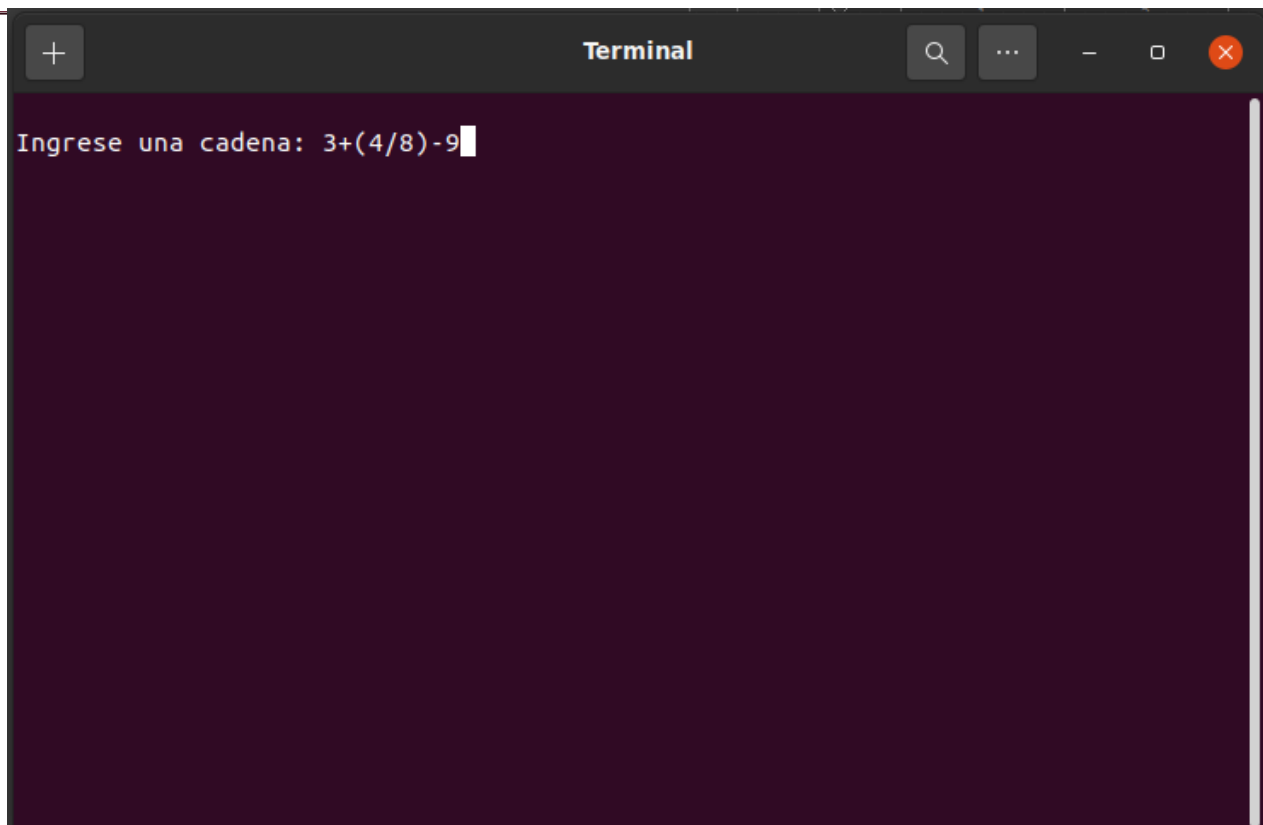


```
main.cpp 2 x  AnalizadorSintactico.cpp  AnalizadorSintactico.h  CMakeLists.txt
main.cpp > main(int, char **)
1  #include <iostream>
2  #include <string>
3
4  #include "AnalizadorSintactico.h"
5  using namespace std;
6
7  int main(int, char**) {
8      string cadena;
9      cout << "Ingresa una cadena: ";
10     cin >> cadena;
11     AnalizadorSintactico prueba(cadena);
12     cout << "Resultado: ";
13     cout
14     |   << prueba.get_cadena()
15     |   << endl; // muestra el resultado de las reglas semánticas llevadas acabo.
16 }
17
```

En el main el usuario ingresa una cadena y creo un objeto de la clase analizador sintactico con esa cadena.

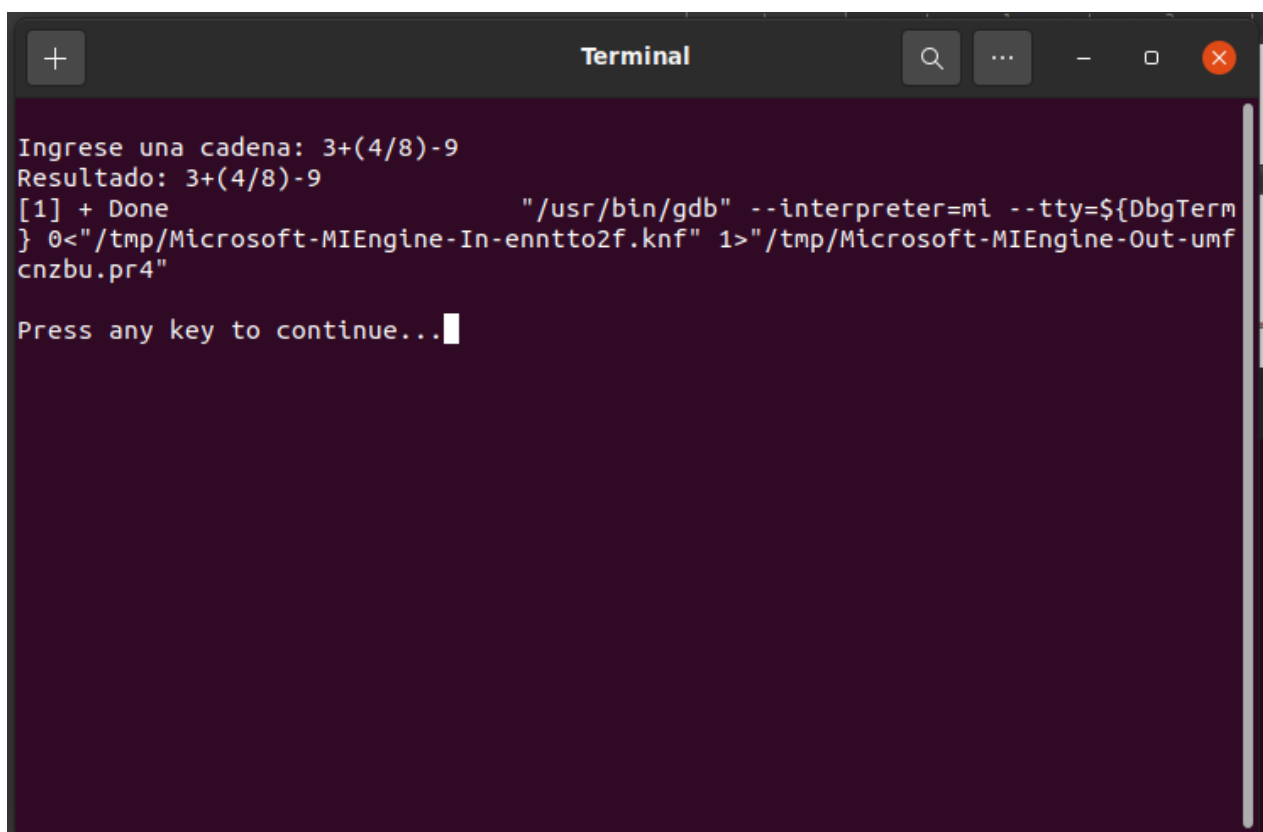






A terminal window titled "Terminal" with a dark background. The prompt "Ingrese una cadena: " is followed by the input string "3+(4/8)-9".

```
Ingrese una cadena: 3+(4/8)-9
```



A terminal window titled "Terminal" with a dark background. It shows the execution of a command and its output. The prompt "Ingrese una cadena: " is followed by the input string "3+(4/8)-9". The output "Resultado: 3+(4/8)-9" is displayed. Below this, the command "[1] + Done" is shown, followed by a long string of characters. The prompt "Press any key to continue..." is displayed at the bottom.

```
Ingrese una cadena: 3+(4/8)-9
Resultado: 3+(4/8)-9
[1] + Done
"/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm
} 0<"/tmp/Microsoft-MIEngine-In-enntto2f.knf" 1>"/tmp/Microsoft-MIEngine-Out-umf
cnzbu.pr4"
Press any key to continue...
```

Código:

Main

```
#include <iostream>
#include <string>

#include "AnalizadorSintactico.h"
using namespace std;

int main(int, char**) {
    string cadena;
    cout << "Ingrese una cadena: ";
    cin >> cadena;
    AnalizadorSintactico prueba(cadena);
    cout << "Resultado: ";
    cout
        << prueba.get_cadena()
        << endl; // muestra el resultado de las reglas semánticas llevadas acabo.
}
```

Header AnalizadorSintactico

```
#ifndef ANALIZADORSINTACTICO_H
#define ANALIZADORSINTACTICO_H

#include <iostream>
#include <string>
#pragma once

using namespace std;

class AnalizadorSintactico {
public:
    explicit AnalizadorSintactico(const string &cadena);
    string get_cadena() { return sem_cadena; }
```

```
private:
    string m_cadena;
    string sem_cadena;
    char *preanalysis;

    void coincidir(char dato);
    void term();
    void fact();
    void Resto_expr();
    void digito();
    void expr();
};
```

```
#endif
```

CPP AnalizadorSintactico

```
#include "AnalizadorSintactico.h"
```

```
AnalizadorSintactico::AnalizadorSintactico(const string &cadena) {
    m_cadena = cadena;
    preanalysis = &m_cadena[0];
    expr();
}
```

```
void AnalizadorSintactico::coincidir(char dato) {
    sem_cadena += dato;
    preanalysis++;
}
```

```
void AnalizadorSintactico::term() {
    fact();
    Resto_expr();
}
```

```
}
```

```
void AnalizadorSintactico::Resto_expr() {
```

```
    switch (*preanalysis) {
```

```
        case '+':
```

```
            coincidir('+');
```

```
            term();
```

```
            Resto_expr();
```

```
            break;
```

```
        case '-':
```

```
            coincidir('-');
```

```
            term();
```

```
            Resto_expr();
```

```
            break;
```

```
        case '*':
```

```
            coincidir('*');
```

```
            fact();
```

```
            Resto_expr();
```

```
            break;
```

```
        case '/':
```

```
            coincidir('/');
```

```
            fact();
```

```
            Resto_expr();
```

```
            break;
```

```
    }
```

```
}
```

```
void AnalizadorSintactico::fact() {
```

```
    if (*preanalysis == '(') {
```

```
        coincidir('(');
```

```
        expr();
```

```
        coincidir(')');
```

```
} else {  
    digito();  
}  
}
```

```
void AnalizadorSintactico::digito() {  
    switch (*preanalisis) {  
        case '0':  
            coincidir('0');  
            break;  
        case '1':  
            coincidir('1');  
            break;  
        case '2':  
            coincidir('2');  
            break;  
        case '3':  
            coincidir('3');  
            break;  
        case '4':  
            coincidir('4');  
            break;  
        case '5':  
            coincidir('5');  
            break;  
        case '6':  
            coincidir('6');  
            break;  
        case '7':  
            coincidir('7');  
            break;  
        case '8':
```

```
        coincidir('8');
        break;
    case '9':
        coincidir('9');
        break;
    default:
        cout << "No coincide con la gramatica" << endl;
        break;
    }
}

void AnalizadorSintactico::expr() {
    term();
    Resto_expr();
}
```

Conclusiones:

Es necesario eliminar la recursion a la izquierda y mantener la cadena dentro de la gramática para evitar un error en el análisis.