



Nombre: Dana Carolina Ramírez Velázquez

Código: 220286547

Materia: Traductores de Lenguaje 2

Actividad: Practica 1

Fecha: 25/08/22

Introducción:

Un analizador sintáctico descendente predictivo es un analizador puede empezar con el símbolo inicial e intentar transformarlo en la entrada, intuitivamente esto sería ir dividiendo la entrada progresivamente en partes cada vez más pequeñas. Este intenta encontrar entre las producciones de la gramática la derivación por la izquierda del símbolo inicial para una cadena de entrada.

Es fundamental eliminar la recursividad por la izquierda para poder llevar acabo este analizador.

Para el programa se utilizaron las siguientes reglas de producción y de semántica:

Reglas de Producción sin recursión por la izquierda

$\text{expr} \rightarrow \text{term resto_expr}$

$\text{resto_expr} \rightarrow + \text{term resto_expr}$

$\text{resto_expr} \rightarrow - \text{term resto_expr}$

$\text{term} \rightarrow \text{factor resto_expr}$

$\text{resto_expr} \rightarrow / \text{factor resto_expr}$

$\text{resto_expr} \rightarrow * \text{factor resto_expr}$

$\text{factor} \rightarrow (\text{expr})$

$\text{factor} \rightarrow \text{digito}$

$\text{resto_expr} \rightarrow \epsilon$

$\text{digito} \rightarrow 0 \dots 9$

Reglas Semánticas de no recursión por la izquierda

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

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

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

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

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

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

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

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

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

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

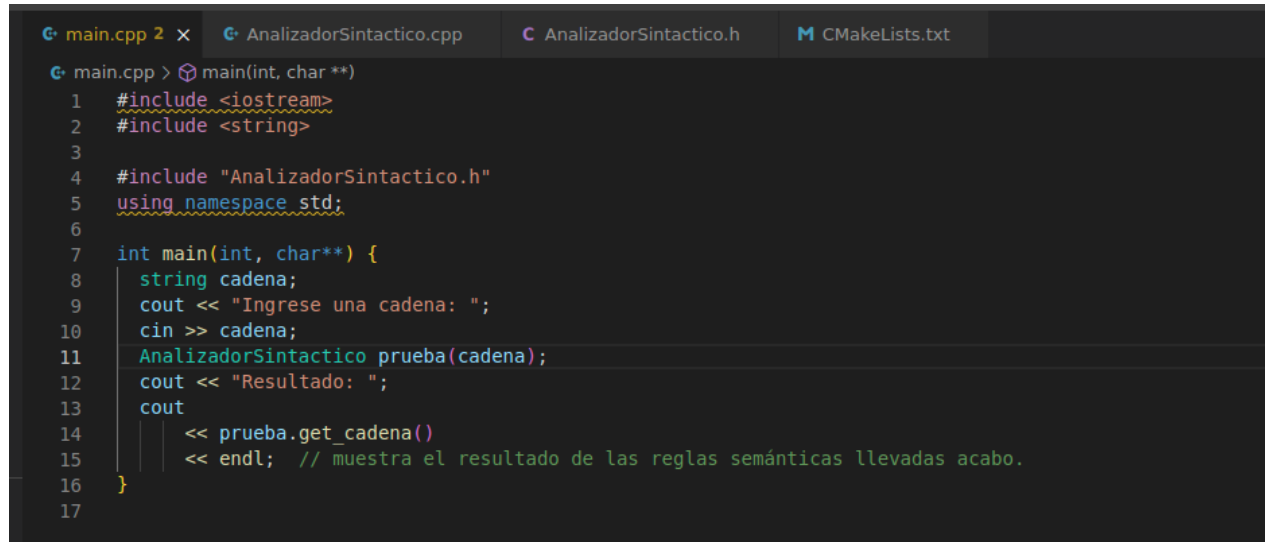
$\text{resto_expr.t} = \text{resto_expr.her}$

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

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

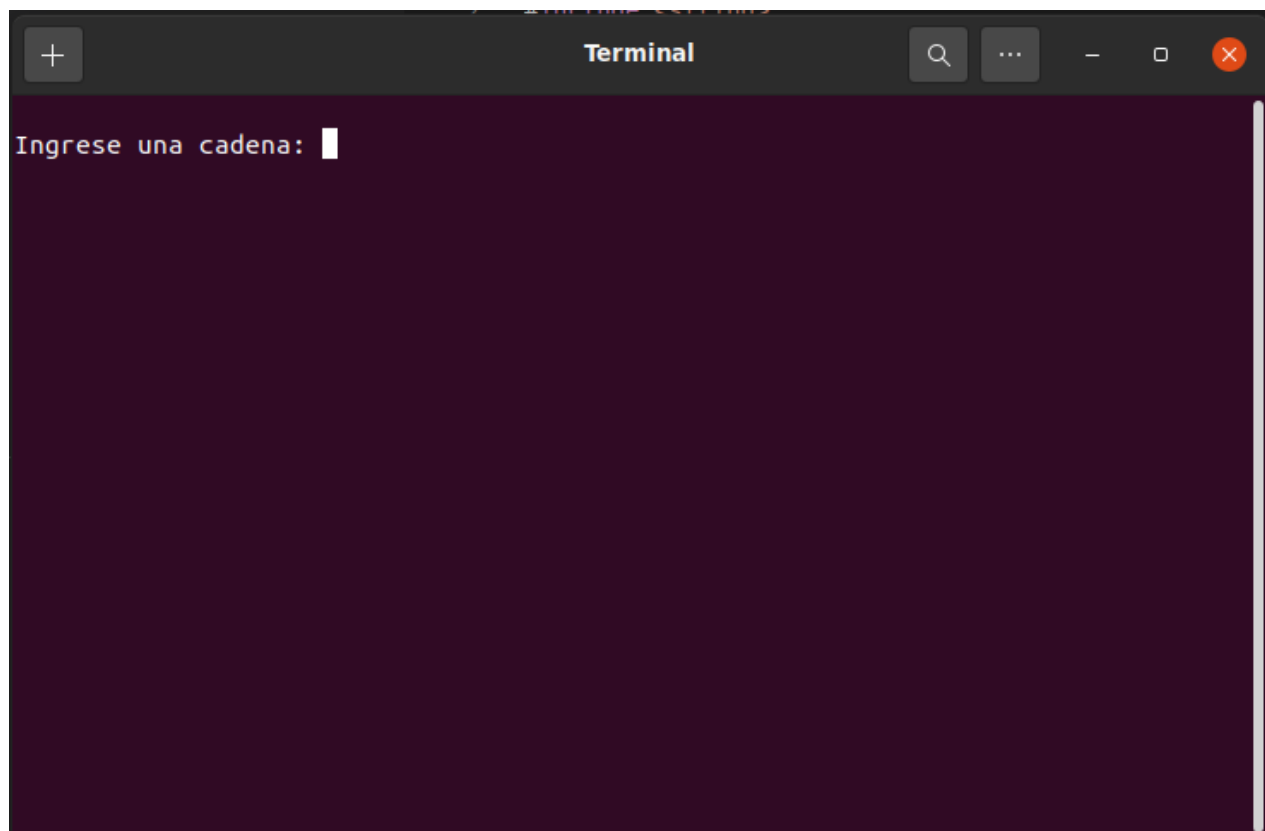
$\text{digito} = 0 \parallel \dots \parallel 9$

Capturas:

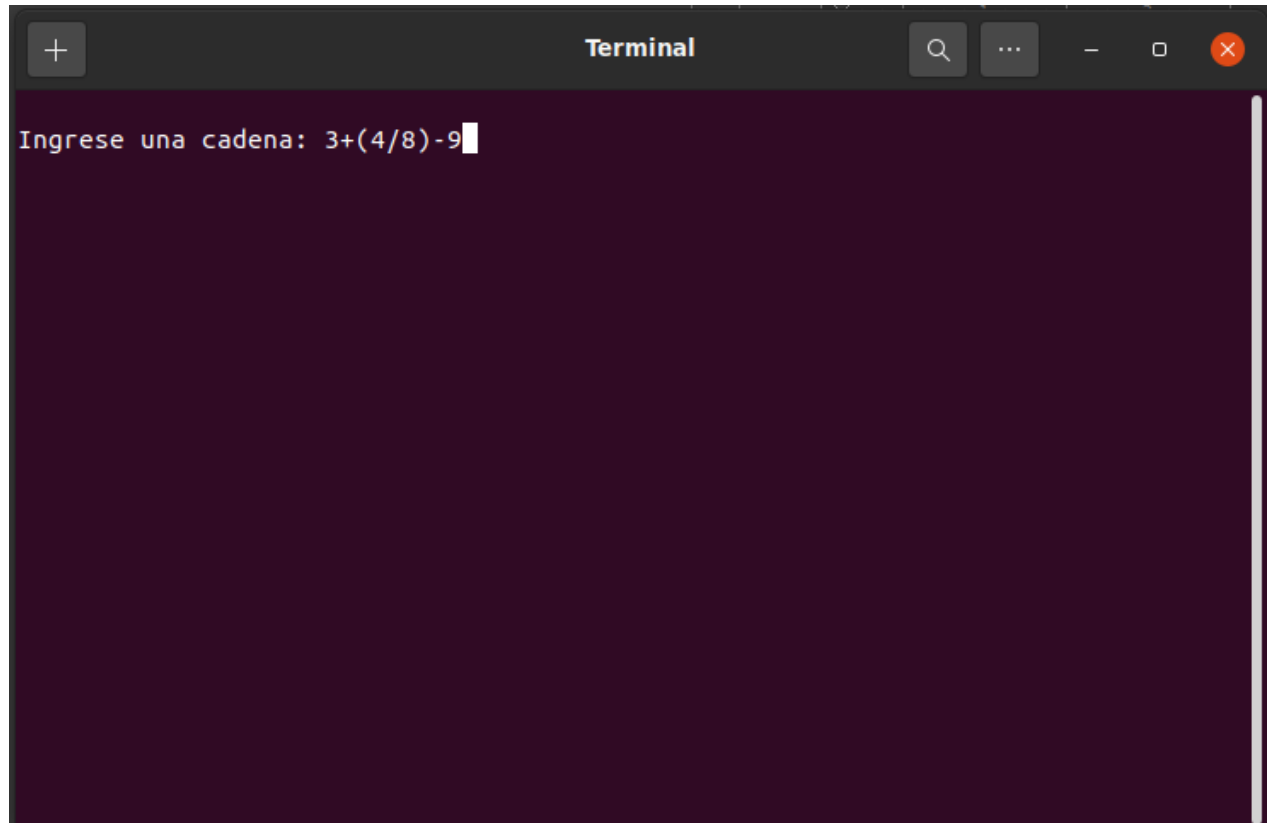


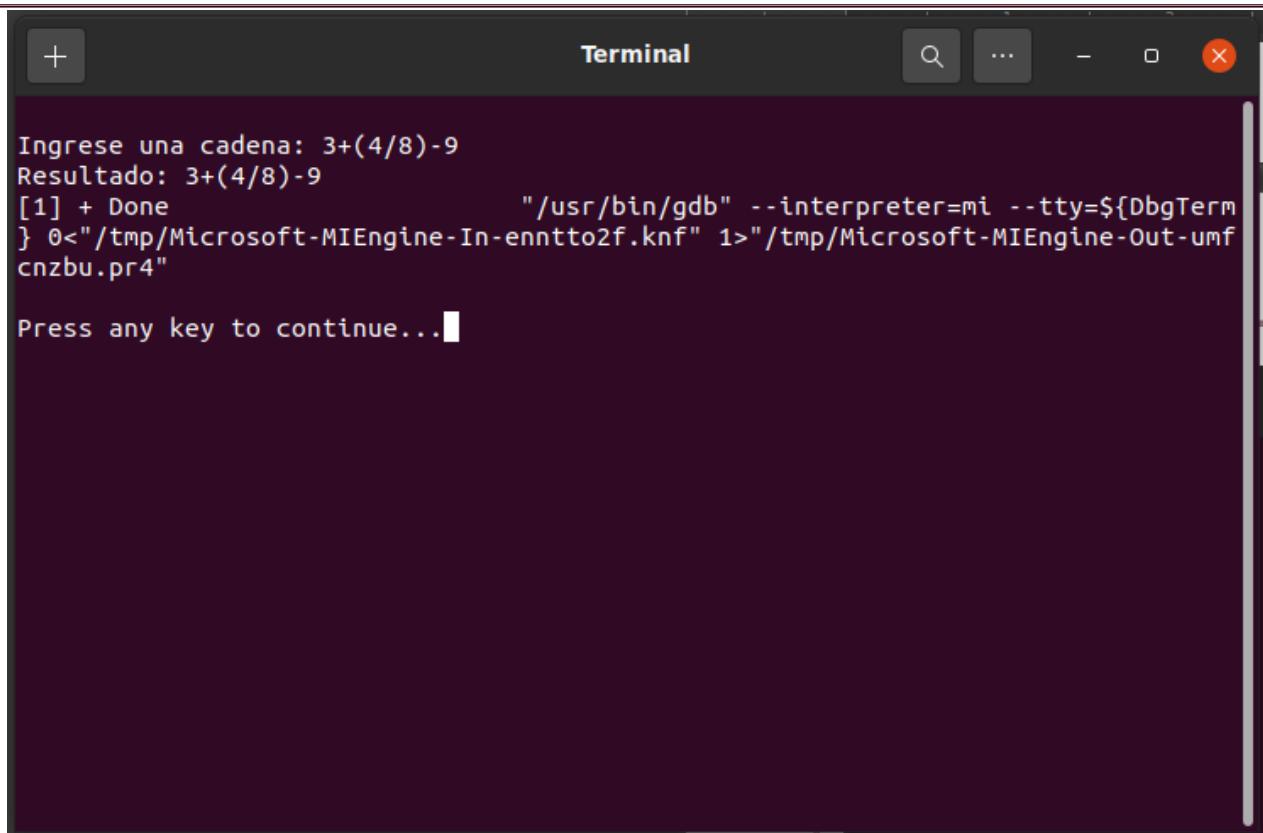
```
main.cpp 2 x  AnalizadorSintactico.cpp  AnalizadorSintactico.h  CMakeLists.txt
main.cpp > main(int, char **)
1  #include <iostream>
2  #include <string>
3
4  #include "AnalizadorSintactico.h"
5  using namespace std;
6
7  int main(int, char**) {
8      string cadena;
9      cout << "Ingrese una cadena: ";
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.



```
Terminal
Ingrese una cadena: 
```



A terminal window titled "Terminal" with a dark background. It shows the execution of a program that prompts for a string, receives "3+(4/8)-9", and displays the result. It then shows the launch of GDB with various flags and file paths, ending with a "Press any key to continue..." prompt.

```
Terminal
+
Ingrese una cadena: 3+(4/8)-9
Resultado: 3+(4/8)-9
[1] + Done
} 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 (*preanalysis) {  
        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.