|  |
| --- |
| FACULTAD DE INGENIERIA |
| PRACTICA 1 |
| LABORATORIO PROGRAMACION DE SISTEMAS |

|  |
| --- |
| Rogelio Daniel Gonzalez Nieto  24-9-2020 |

OBJETIVO

El objetivo de la práctica es crear una aplicación que con la ayuda del analizador ANTLR reconozca gramáticas matemáticas las cuales nos servirán de calculadora.

DESARROLLO

Para el desarrollo de esta practica se llevaron a cabo los siguientes pasos:

1. Creación de gramática y clases lexer y parser mediante el uso de la herramienta ANTLR.
2. Después de generados los archivos estos deben de ser agregados al proyecto.
3. En visual estudio se debe de descargar e instalar las herramientas de ANTLR.
4. Se agregan las llamadas a las funciones leer y parser como se muestra a continuación.

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Antlr;

using Antlr.Runtime;

namespace P1\_Calculadora

{

class Program

{

static void Main(string[] args)

{

string line = "";

//VARIABLE PARA ALMACENAR LA CADENA DE ENTRADA

while (true)

{

line = Console.ReadLine();

//SE ALMACENA LA CADENA DE ENTRADA

if (line.Contains("EXIT") || line.Contains("exit"))

//SI DETECTA EXIT SALE DEL PROGRAMA

break;

var entrada = line + Environment.NewLine;

byte[] byteArray = Encoding.ASCII.GetBytes(entrada);

MemoryStream stream = new MemoryStream(byteArray);

var parametro1 = new ANTLRInputStream(stream);

Calculadora1Lexer lex = new Calculadora1Lexer(parametro1);

//CREAMOS UN LEXER CON LA CADENA QUE ESCRIBIO EL USUARIO

CommonTokenStream tokens = new CommonTokenStream(lex);

//CREAMOS LOS TOKENS SEGUN EL LEXER CREADO

Calculadora1Parser parser = new Calculadora1Parser(tokens);

//CREAMOS EL PARSER CON LOS TOKENS CREADOS

try

{

int iResultado = parser.expresion();

Console.WriteLine(line + " = " + iResultado);

//SE VERIFICA QUE EL ANALIZADOR EMPIECE CON LA EXPRESION

}

catch (RecognitionException e)

{

Console.Error.WriteLine(e.StackTrace);

}

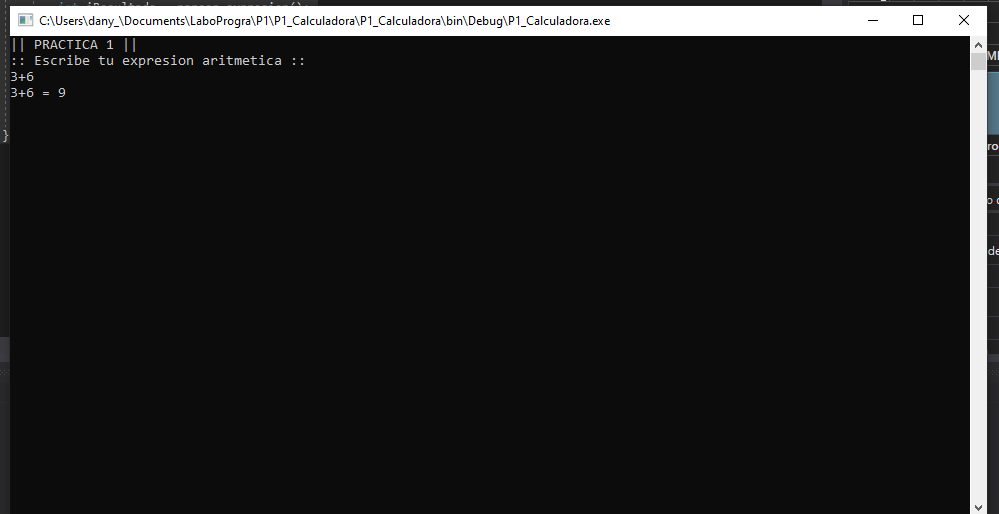
}

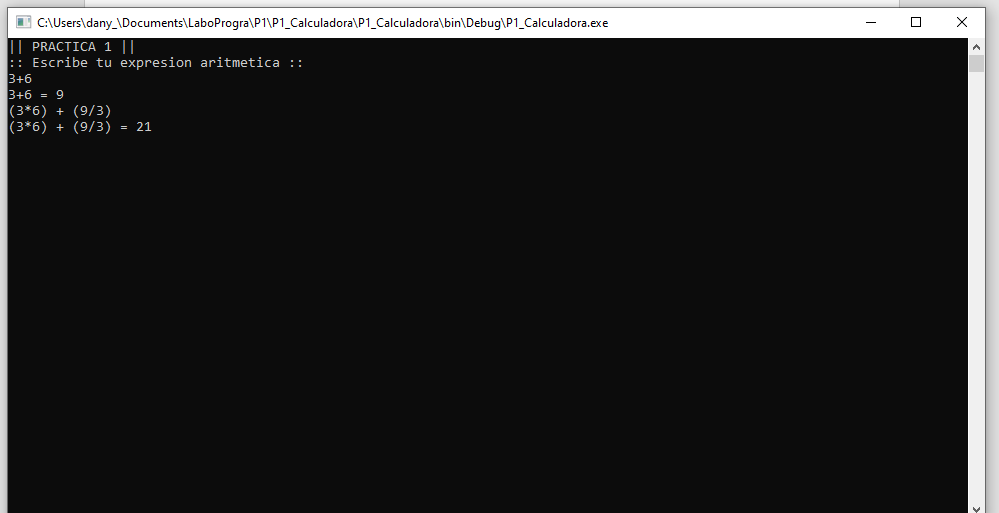
}

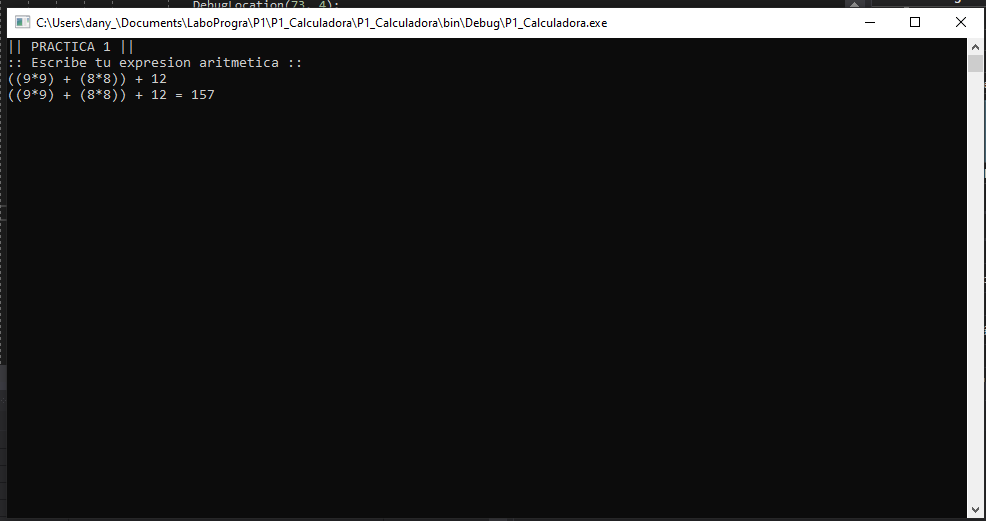
}

}

1. Para ejecutar una operación matemática basta con escribirla en la consola:







1. La gramática usada fue la siguiente:

grammar Calculadora1;

//nombre de la gramatica

/\*

\*opciones de compilacion de la gramatica

\*/

options {

language=CSharp2; //lenguaje objetivo de la gramatica

}

/\*

\* Reglas del Parser

\*/

programa returns[int value] //el programa retornara un valor entero.

: stat{System.Console.WriteLine($stat.value);} //se imprime el valor que calculo el parser.

;

stat returns[int value] //la expresion retornara un valor entero al programa.

:

c = expresion NEWLINE {System.Console.WriteLine($c.value);} //Se imprime el valor adquirido.

|NEWLINE; //no se hace nada.

expresion returns[int value] //El valor calculado por la expresion sera regresado como un entero.

:

a = multiplicacion{$value = $a.value;} ( //Se asina el valor que se retornara en la regla.

MAS b = multiplicacion {$value =$value + $b.value;} //El valor se suma con el actual en la expresion.

|

MENOS b = multiplicacion{$value =$value- $b.value;})\*{System.Console.WriteLine($value);} //El valor se resta con el actual y se imprime el valor.

;

multiplicacion returns[int value] //La regla retorna un entero.

:

a = numero{$value = $a.value;} ( //Se asigna el valor que se regresara.

POR b = numero{$value =$value\* $b.value;} //Se calcula la multiplicacion

|

ENTRE b = numero{$value =$value/ $b.value;})\* //Se calcula la division.

;

numero returns[int value] //La regla retonara un entero.

:

INT {$value = int.Parse($INT.text);} //se convierte a entero la cadena de entrada de la consola.

|

PARENI expresion PAREND {$value = $expresion.value;} //se asigna el valor de la expresion dentro del parentesis.

;

/\*

\* Reglas del Lexer.

\*/

PARENI

: '(' //token de parentesis derecho

;

PAREND

: ')' //token de parentesis izquierdo.

;

MAS

: '+' //token de signo mas

;

MENOS

: '-' //token de signo menos

;

POR

: '\*' //token de signo por

;

INT

:('0'..'9')+ //tokens validos para numeros

;

ENTRE

: '/' //token de signo entre

;

NEWLINE //token para identificar el final de la expresion.

: '\n'

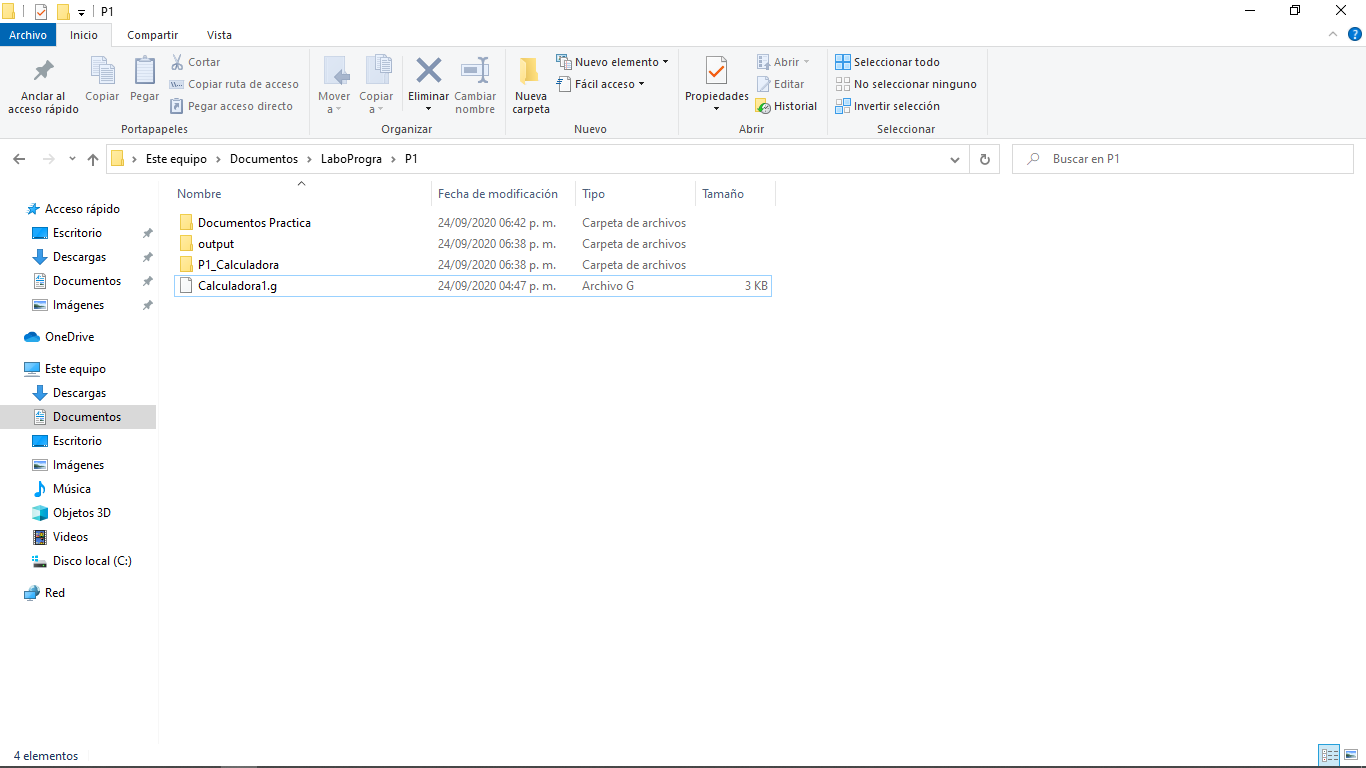
;

WS

: (' '|'\r'|'\n'|'\t')+ {Skip();} //tokens que identifican las secuencas de escape.

;

Las gramáticas deben de estar en archivos con extencion **.g** para que puedan ser reconocidas por el analizador léxico



CONCLUCIONES

Esta practica fue sumamente interesante debido a que tenia tiempo de no programar en C# y sirvió de practica y recordatorio de como funcionan los analizadores léxicos.