INSTITUTO TECNOLÓGICO DE NUEVO LEÓN

Ingeniería En Sistemas Computacionales

Lenguajes y Autómatas II

U1

Tema: Análisis semántico

Proyecto 1

Profesor.
Juan Pablo Rosas Baldazo

Presenta.

Luis Fernando Dominguez Perez

14480482

Índice

Introducción

Este proyecto tiene como fin codificar un programa que pueda realizar la evaluación de la expresión que incluyera operandos numéricos y operadores matemáticos (+, -, *, /, ^).

Así mismo que pudiera mostrar los diversos ordenes prefija y posfija para poder apreciar que este correcto el orden de la notación

Descripción

Para realizar el proyecto se tuvo que realizar una investigación para poder tener la lógica correcta para los diversos órdenes y así poder codificarlo ya que la mayoría de los videos tutoriales que se podían encontrar solo mostraban el funcionamiento del programa, por lo cual se tuvo que recurrir a foros de programación, así como un video-clase que sirvió de ayuda para la realización del programa.

Pseudocodigo

```
package evaluadorexpresiones;

import java.util.Scanner;

public class Evaluador {
    public static void main(String[] args) {
        print ("Digite expresión que desea evaluar");
        String infija, infija1 = Expresión ingresada;
        print ("Resultado y ordenes de la expresión");
    }
    public static double evaluar (String infija, String infija1){
        String prefija= conversión expresión a prefija;
        String posfija = conversión expresión a posfija;
        print ("La expresión posfija es: " + posfija);
        print ("La expresión infija es: " + infija);
        print ("La expresión prefija es: " + prefija);


        return Resultado de la expresión;
    }
      private static String convertirpre(String infija) {
      pila = caracteres de la expresión;
```

```
    for (recorrido de la pila){
        char letra = separar carácter por carácter;
        if (si el carácter no es operador) {
            if (si la pila está vacía) {
         (entonces) se apila carácter en la pila;
            } en caso contrario {
                int pe = prioridad en la expresión del operando;
                int pp = prioridad en pila del siguiente operando;
            if (pe > pp){
                 apilar operando en pila;
            } en caso contrario {
                prefija += se desapila el operando de la pila;
                se apila el operando en la pila e ingresa a prefija;
            }
         }
        } en caso contrario {
            prefija += el operador se ingresa a prefija;
        }
      }
    mientras (la pila no esté vacía) {
        prefija += se desapilan los operandos y se ingresan a prefija;
    }
    devuelve el orden de prefija;
}
```

```
 private convertirpos (String infija) {
pila = número de caracteres de la expresión;
    for (recorrido de la pila) {
     char letra = separar carácter por carácter;
     if (si el carácter es operador) {
         if (si la pila está vacía) {
      (entonces) se apila carácter en la pila;
         } en caso contrario {
             int pe = prioridad en la expresión del operador;
             int pp = prioridad en pila del operador;
         if (pe > pp){
              apilar operador en pila;
         } en caso contrario {
             posfija = se desopila el operador de la pila;
             se apila el operador en la pila e ingresa a posfija;
         }
      }
    } en caso contrario {
         posfija = el operando se ingresa a posfija;
    }
  }
mientras (la pila no esté vacía) {
    posfija = se desapilan los operadores y se ingresan a posfija;
}
```

```
  return devuelve el orden de posfija;
}
   private static int prioridadEnExpresion (char operador) {
     if (operador == '^') devuelve ID# 4;
     if (operador == '*' || operador == '/') devuelve ID# 2;
     if (operador == '+' || operador == '-') devuelve ID# 1;
     if (operador == '(') devuelve ID# 5;
     return 0;
 }
     private static int prioridadEnPila (char operador) {
     if (operador == '^') devuelve ID# 3;
     if (operador == '*' || operador == '/') devuelve ID# 2;
     if (operador == '+' || operador == '-') devuelve ID# 1;
     if (operador == '(') return 0;
     return 0;
 }
   private static double evaluar (String posfija) {
   tamaño pila = cantidad de caracteres de la expresión;
   for (recorrido de la pila) {
     char letra = separar carácter por carácter;
     if (si el carácter no es operador) {
        double num = se apila operando en la pila;
        se apila numero en la pila;
     } en caso contrario {
         double num2 = se desapila el operando;
```

```
            double num1 = se desapila el operando;
            double num3 = resultado de la operación;
            apila resultado de la operación en la pila;
          }
        }
        return devuelve resultado de la expresión;
    }
    private static boolean esOperador (char letra) {
      Si es alguno de los operadores (letra == '*'|| letra == '/'|| letra == '+'||
        letra == '-'||letra == '('||letra == ')' ||letra == '^') {
        responde verdadero;
    }
      En caso contrario responde falso;
    }
    private static double operacion (char letra, double num1, double num2) {
        si el operando (letra == '*') regresa una multiplicación;
        si el operando (letra == '/') regresa una división;
        si el operando (letra == '+') regresa una suma;
        si el operando (letra == '-') regresa una resta;
        si el operando (letra == '^') regresa una potencia;
        return 0;
    }
}
```

```
package evaluadorexpresiones;
public class Pila {

    se crea variable int n, tope;
    se crea objeto pila [];
    public Pila (int n) {
        esta n es = tamaño de la pila;
        tope = 0;
        pila = tiene un tamaño = n;
    }
    public boolean estaVacia(){
        si esta vacía regresa un 0;
    }
    public boolean estaLlena(){
        si está llena regresa el número de elementos en la pila;
    }
    public boolean apilar(recibe un dato){
        if(la pila esta llena){
            regresa un falso;
        }
          en caso contrario
        posicion de la pila actual = dato;
        posicion de la pila avanza 1;
        regresa un verdadero;
```

```
    }
    public Object desapilar(){
        if(la pila esta vacía) {
            regresa un valor nulo;
        }
        posición de la pila retrocede 1;
        regresa posición de la pila;
    }
    public Object elementoTope(){
        regresa posición anterior de la pila;
    }
}
```

Experimentación

Primero que nada, los errores que nos marcaba al codificar, se estuvo investigando en foros de programación para poder resolver los inconvenientes, así como la utilización de try…catch para corregir excepciones que surgieron.

Resultados

Se logró llegar al objetivo del proyecto, al ingresar alguna operación matemática.

Ingresando los siguientes ejemplos

1+2*3/2

La exprecion posfija es: 123*2/+

La exprecion infija es: 1+2*3/2

La exprecion prefija es: +1*2/32

El resultado es: 4.0

2-5/3*2^2

La exprecion posfija es: 253/22^*-

La exprecion infija es: 2-5/3*2^2

La expresión prefija es: -2/5*3^22

El resultado es: -4.666666666666667

Conclusión

El programa es fácil de realizar, pero se debe tener cuidado con la lógica que fue la mayor razón del cual resultaban errores que se arreglaron con try..catch, pero se tuvo que ingresar a varios foros y videos para poder tener claro la lógica y así empezar a modificar la codificación para que pudiera realizar lo que requeríamos para este proyecto, lo cual al final si se pudo lograr.

Referencias (bibliografía)

Juan Carlos Zuluaga, 17 de abril 2015
https://www.youtube.com/watch?v=xKhA9W1fUZU&t=2016s

Juan Carlos Zuluaga, 20 de marzo 2014
https://www.youtube.com/watch?v=d7UZdz_yGXQ

Quinston Pimenta, 25 de junio 2014
https://www.youtube.com/watch?v=TCPCjuXAszQ&t=319s