**PRÁCTICA 2**

**ALEJANDRO PEREZ**

**FELIPE CADAVID RINCON**

**TEORÍA DE LENGUAJES Y LABORATORIO**

**INGENIERÍA DE SISTEMAS**

**UNIVERSIDAD DE ANTIOQUIA**

**JUNIO 2018**

**Descripción de la Actividad**

Realizar una aplicación de software que sea una calculadora que reconozca expresiones en infijo y si la expresión es válida que muestre el resultado de la operación, en caso contrario que muestre los errores que contiene y realizar un manejo de ellos.

**Debe reconocer:**

Operando (Enteros) = {…, -3, -2, -1, 0, 1, 2, 3, …}

Operadores = {+, -, \*, /}

El Signo paréntesis: = {(, )}

**Ejemplos de expresiones válidas:**

1+3\*5+2=18

2(1+5\*2)+3=25

**Ejemplo de expresión inválida:**

3+((2)

Se debe mostrar mensaje de error porque: Falta un paréntesis derecho, tiene más paréntesis izquierdos que derecho, esperaba un paréntesis derecho, etc. y especificar en qué posición detectó el error.

**Desarrollo de la Actividad**

1. **Análisis del problema.**
2. Para resolver ese ejercicio primero debemos definir la gramática y sus producciones.
3. <S> → <S> + <T>
4. <S> → <S> - <T>
5. <S> → <T>
6. <T> → <T> \* <U>
7. <T> → <T> / <U>
8. <T> → <U>
9. <U> → (<S>)
10. <U> → I
11. Agregamos los atributos y el símbolo de acción:

<S>p → <S>q + <T>r {Sume}s, t, u

s = q; t = r; (p, u) = new

<S>p → <S>q - <T>r {Reste} s, t, u

s = q; t = r; (p, u) = new

<S>p → <T>r

p = q

<T>r → <T>q \* <U>r {Multiplique} s, t, u

s = q; t = r; (p, u) = new

<T>p → <T>q / <U>r {Divida} s, t, u

s = q; t = r; (p, u) = new

<T>p → <U>q

p = q

<U>p → (<S>q)

p = q

<U>p → Iq

p = q

1. Definimos los conjuntos de selección:
2. **Construcción del Autómata de Pila.**

Símbolos de entrada = {+, -, \*, /, I, ┤}

Estados = {<S>, <T>, <U>, {Sume}, {Reste}, {Multiplique}, {Divida}}

Estado Inicial = {▼, <S> }

Tabla de transiciones =

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | + | - | | | \* | | / | | I | | ┤ |
| <S> | 2 | 4 | | | 1 | |  | |  | |  |
| <T> | 2 | 4 | | | 4 | |  | |  | |  |
| <U> | 4 | 3 | | | 3 | |  | |  | |  |
| {Sume} |  | | | | | | | | | | |
| {Reste} |  | | | | | | | | | | |
| {Multiplique} |  | | | | | | | | | | |
| {Divida} |  | | | | | | | | | | |
| ▼ |  | |  |  | |  | |  | | A | |

1. **Boceto del algoritmo:**

Con base lo anterior, se puede elaborar el siguiente algoritmo base que usaremos como referencia para el desarrollo del programa.

estado = 0

leer(símbolo)

while símbolo != ┤ do

casos de estado

“0”: casos de símbolo

“S”: estado = 2

“O”: estado = 4

“N”: estado = 1

“1”: casos de símbolo

“S”: estado = 2

“O”: estado = 4

“N”: estado = 4

“2”: casos de símbolo

“S”: estado = 4

“O”: estado = 3

“N”: estado = 3

“3”: casos de símbolo

“S”: estado = 4

“O”: estado = 0

“N”: estado = 4

“4”: casos de símbolo

“S”: estado = 4

“O”: estado = 4

“N”: estado = 4

1. **Resultado tras Producción y Desarrollo**

Para desarrollar el programa se tuvieron las siguientes consideraciones:

* Para implementar este reconocedor se utilizó JAVA como lenguaje de programación.
* Se utilizó NETBEANS como IDE de programación
* Se utilizó el Paradigma Objetual.
* Se desarrolló el programa usando un modelo de cascada (Analizar Requisitos, Crear Diseño, Implementar, Verificar y hacer Pruebas)

**Link del proyecto:** https://github.com/pipecadav/ReconocedorExpresiones