**Universidad Nacional de Cajamarca**

**Facultad de Ingeniería**

**Escuela Académico Profesional de Ingeniería de Sistemas**



Conversión de Infija a Prefija, Prefija a Infija y evaluación de expresión infija.

Curso:

Algoritmos y estructura de datos II

Grupo:

A

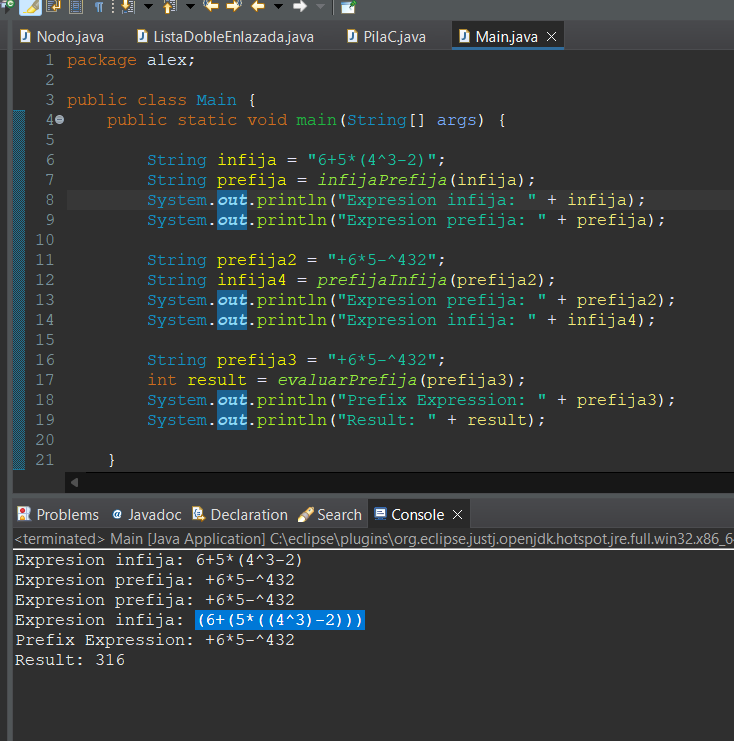
Docente:

Malpica Rodriguez Manuel Enrique

Estudiante:

* Caruajulca Tiglla Alex Eli

Cajamarca – 2023



https://www.youtube.com/watch?v=T4M3OQTX25I

public class Main {

public static void main(String[] args) {

String infija = "6+5\*(4^3-2)";

String prefija = *infijaPrefija*(infija);

System.***out***.println("Expresion infija: " + infija);

System.***out***.println("Expresion prefija: " + prefija);

String prefija2 = "+6\*5-^432";

String infija4 = *prefijaInfija*(prefija2);

System.***out***.println("Expresion prefija: " + prefija2);

System.***out***.println("Expresion infija: " + infija4);

String prefija3 = "+6\*5-^432";

int result = *evaluarPrefija*(prefija3);

System.***out***.println("Prefix Expression: " + prefija3);

System.***out***.println("Result: " + result);

}

public static String infijaPrefija(String infija) {

StringBuilder prefija = new StringBuilder();

PilaC<Character> operadores = new PilaC<>();

String infijaReversa = new StringBuilder(infija).reverse().toString();

for (int i = 0; i < infijaReversa.length(); i++) {

char actualChar = infijaReversa.charAt(i);

if (Character.*isLetterOrDigit*(actualChar)) {

prefija.append(actualChar);

} else if (actualChar == ')') {

operadores.push(actualChar);

} else if (actualChar == '(') {

while (!operadores.vacio() && operadores.peek() != ')') {

prefija.append(operadores.pop());

}

operadores.pop(); // Remove el '('

} else {

while (!operadores.vacio() && *jerarquia*(actualChar) <= *jerarquia*(operadores.peek())) {

prefija.append(operadores.pop());

}

operadores.push(actualChar);

}

}

while (!operadores.vacio()) {

prefija.append(operadores.pop());

}

return prefija.reverse().toString();

}

private static int jerarquia(char operator) {

switch (operator) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return -1;

}

}

public static String prefijaInfija(String prefija) {

PilaC<String> pilaInfija = new PilaC<>();

for (int i = prefija.length() - 1; i >= 0; i--) {

char actualChar = prefija.charAt(i);

if (Character.*isLetterOrDigit*(actualChar)) {

pilaInfija.push(String.*valueOf*(actualChar));

} else {

String operando1 = (String) pilaInfija.pop();

String operando2 = (String) pilaInfija.pop();

String infija = "(" + operando1 + actualChar + operando2 + ")";

pilaInfija.push(infija);

}

}

return (String) pilaInfija.pop();

}

private static int operaciones(char operador, int operando1, int operando2) {

switch (operador) {

case '+':

return operando1 + operando2;

case '-':

return operando1 - operando2;

case '\*':

return operando1 \* operando2;

case '/':

return operando1 / operando2;

case '^':

return (int) Math.*pow*(operando1, operando2);

default:

throw new IllegalArgumentException("Operador inválido: " + operador);

}

}

public static int evaluarPrefija(String prefija) {

PilaC<Integer> stack = new PilaC<>();

for (int i = prefija.length() - 1; i >= 0; i--) {

char actualChar = prefija.charAt(i);

if (Character.*isDigit*(actualChar)) {

stack.push(actualChar - '0'); // Lo convierte a entero

} else {

int operando1 = (int) stack.pop();

int operando2 = (int) stack.pop();

int result = *operaciones*(actualChar, operando1, operando2);

stack.push(result);

}

}

return (int) stack.pop();

}

}

public class PilaC<T> {

ListaDobleEnlazada <T> listaPila;

public PilaC() {

this.listaPila = new ListaDobleEnlazada<T>();

}

public void push(T dato) {

listaPila.agregarInicio(dato);

}

public Object pop() {

return listaPila.eliminarInicio();

}

public int length() {

return listaPila.getLength();

}

public T peek() {

return listaPila.obtenerInicio();

}

public boolean vacio() {

return listaPila.estaVacia();

}

public void limpiar() {

listaPila = new ListaDobleEnlazada<>();

}

*@Override*

public String toString() {

return listaPila.toString();

}

}

public class ListaDobleEnlazada<T> {

private Nodo<T> primero;

private Nodo<T> fin;

private int length;

public ListaDobleEnlazada() {

primero = null;

fin = null;

length = 0;

}

public void agregarInicio(T dato) {

Nodo<T> nuevoNodo = new Nodo<>(dato);

if (primero == null) {

primero = nuevoNodo;

fin = nuevoNodo;

} else {

nuevoNodo.setSiguiente(primero);

primero.setPrevio(nuevoNodo);

primero = nuevoNodo;

}

length++;

}

public void agregarFinal(T dato) {

Nodo<T> nuevoNodo = new Nodo<>(dato);

if (primero == null) {

primero = nuevoNodo;

fin = nuevoNodo;

} else {

nuevoNodo.setPrevio(fin);

fin.setSiguiente(nuevoNodo);

fin = nuevoNodo;

}

length++;

}

public T eliminarInicio() {

if (primero == null) {

return null;

}

T datoEliminado = primero.getDato();

if (primero == fin) {

primero = null;

fin = null;

} else {

primero = primero.getSiguiente();

primero.setPrevio(null);

}

length--;

return datoEliminado;

}

public T eliminarFinal() {

if (fin == null) {

return null;

}

T datoEliminado = fin.getDato();

if (primero == fin) {

primero = null;

fin = null;

} else {

fin = fin.getPrevio();

fin.setSiguiente(null);

}

length--;

return datoEliminado;

}

public int getLength() {

return length;

}

public T obtenerInicio() {

if (primero == null) {

return null;

}

return primero.getDato();

}

public T obtenerFin() {

if (fin == null) {

return null;

}

return fin.getDato();

}

public T eliminar(T dato) {

Nodo<T> actual = primero;

while (actual != null) {

if (actual.getDato().equals(dato)) {

Nodo<T> previo = actual.getPrevio();

Nodo<T> siguiente = actual.getSiguiente();

if (previo != null) {

previo.setSiguiente(siguiente);

} else {

primero = siguiente;

}

if (siguiente != null) {

siguiente.setPrevio(previo);

} else {

fin = previo;

}

length--;

return actual.getDato();

}

actual = actual.getSiguiente();

}

return null;

}

public boolean estaVacia() {

return length == 0;

}

*@Override*

public String toString() {

StringBuilder sb = new StringBuilder();

Nodo<T> actual = primero;

while (actual != null) {

sb.append(actual.getDato()).append(" ");

actual = actual.getSiguiente();

}

return sb.toString();

}

}

public class Nodo<T> {

private T dato;

private Nodo<T> previo;

private Nodo<T> siguiente;

private int prioridad;

public Nodo(T dato) {

this.dato = dato;

this.previo = null;

this.siguiente = null;

}

public T getDato() {

return dato;

}

public void setDato(T dato) {

this.dato = dato;

}

public Nodo<T> getPrevio() {

return previo;

}

public void setPrevio(Nodo<T> previo) {

this.previo = previo;

}

public Nodo<T> getSiguiente() {

return siguiente;

}

public void setSiguiente(Nodo<T> siguiente) {

this.siguiente = siguiente;

}

public Nodo(T dato, int prioridad) {

this.dato = dato;

this.prioridad = prioridad;

}

public int getPrioridad() {

return prioridad;

}

*@Override*

public String toString() {

return dato.toString();

}

}