UNIVERSIDAD DE GUADALAJARA

Red Universitaria e Institución Benemérita de Jalisco

# Centro Universitario de Ciencias Exactas e Ingenierias

## Seminario de solución de problemas de traductores II

Profesor: José Juan Meza Espinoza

Sección: D01

Practica 2: Notación posfija


Alumnos:

- Botello Martínez Nadia Noemi
- Ángel Gabriel Mercado Hernández
- Ivo Alberto Ramirez Gaeta
- Uziel Reyes Becerra


15 de octubre 2024

## 1. Introducción

En esta actividad desarrollaremos un algoritmo que nos ayude a poder evaluar una notación postfija, el objetivo es poder simplificar el proceso de cálculo de expresiones aritméticas eliminando la necesidad de gestionar paréntesis y la precedencia de operadores , este de tipo de notación es muy útil en el contexto de lenguajes formales y compiladores ya que nos facilitara la evaluación de expresiones de una manera eficiente.

## 2. Contenido

Esta actividad es un ejemplo práctico de cómo se puede utilizar una estructura de datos (pila) para resolver problemas matemáticos utilizando notación posfija. La implementación en C++ permite un manejo eficiente de la memoria y operaciones, y el uso de funciones matemáticas estándar mejora la funcionalidad del programa al incluir cálculos trigonométricos.

En este programa se utilizará una pila para almacenar los números y declararemos variables para la expresión y los operandos, después leeremos la expresión posfija por el usuario e iniciaremos un bucle el cual recorrerá cada carácter de la expresión.

```
19      for (int i = 0; i < expresion.length(); i++)
20      {
21          // 2. El caracter leido es un operando, simplemente insertarlo en la pila
22          if (isdigit(expresion[i]))
23          {
24              pila.push(string(1, expresion[i]));
25          }
```

En esta parte si el carácter es un numero lo convertiremos a string y quedara en la fila lo que nos permitirá majera los números, en caso de que el carácter leído sea un operador aritmético, dependiendo de la opción.

Extraeremos los últimos dos operandos de la pila ya que son necesarios para realizar la operación y depues realizaremos la operación.

```
// 2. El caracter leido es un operando, simplemente insertar
if (isdigit(expresion[i]))
{
    pila.push(string(1, expresion[i]));
}

// 3. El caracter leido es un operador aritmetico, tomar dos
//    y realizar la operacion indicada por el operando
if (expresion[i] == '+' || expresion[i] == '-' ||
        expresion[i] == '*' || expresion[i] == '/')
{
    operando1 = stof(pila.top());// Tomamos tope de pila
    pila.pop(); // Eliminamos tope de pila

    operando2 = stof(pila.top());
    pila.pop();
```

En la cual utilizaremos un switch para identificar que operación se realizará según el operador, y el resultado se pasará a string y se pondrá de nuevo a la pila. Para los operadores trigonométricos (seno, coseno, tangente), el procedimiento es similar, pero solo se necesita un operando. Se verifica qué función trigonométrica se debe calcular y se empuja el resultado a la pila.

```
    switch (expresion[i])
    {
    case '+':
        pila.push(to_string(operando2 + operando1));
        break;

    case '-':
        pila.push(to_string(operando2 - operando1));
        break;

    case '*':
        pila.push(to_string(operando2 * operando1));
        break;

    case '/':
        pila.push(to_string(operando2 / operando1));
        break;

    default:
        cout << "Error de sintaxis.";
        break;
```

```
//    y realizar la operacion con el.
if (expresion[i] == 'S' || expresion[i] == 'C' || expresion[i] == 'T')
{
    operando1 = stof(pila.top()); // Obtenemos el tope de pila
    pila.pop(); // Eliminamos el tope de pila

    // Verificamos el tipo de funcion trigonometrica y realizamos la ope
    if (expresion[i] == 'S')
    {
        pila.push(to_string(sin(operando1))); // Aplicamos la funcion se
    }

    else if (expresion[i] == 'C')
    {
        pila.push(to_string(cos(operando1))); // Aplicamos la funcion co
    }

    else
    {
        pila.push(to_string(tan(operando1))); // Aplicamos la funcion ta
    }
}
```

## 3. Resultados

```
#include <iostream>
#include <stack>
#include <cmath>

using namespace std;

int main()
{
    stack<string> pila;
    string expresion;
    float operando1;
    float operando2;

    // Leemos expresion
    cin >> expresion;

    /// Algoritmo de eval
    // 1. Leemos toda la
    for (int i = 0; i < e
    {
        // 2. El caracter
        if (isdigit(expre
```

```
Seleccionar "C:\Users\USER\Desktop\8vo semestre\Sem de traductores 2\SEM. TRAD. LENG. 2\
43*42/SC62-/+
12.153575
Process returned 0 (0x0)   execution time : 13.351 s
Press any key to continue.
```

main.cpp

```cpp
#include <iostream>
#include <stack>
#include <cmath>

using namespace std;

int main()
{
    stack<string> pila
    string expresion;
    float operando1;
    float operando2;

    // Leemos expresio
    cin >> expresion;

    /// Algoritmo de e
    // 1. Leemos toda
    for (int i = 0; i
    {
        // 2. El carac
        if (isdigit(ex
        {
            pila.push(
        }
```

```
"C:\Users\USER\Desktop\8vo semestre\Sem de traductores 2\SEM. TRAD. LENG. 2
436*+2-
20.000000
Process returned 0 (0x0)   execution time : 6.907 s
Press any key to continue.
```

main.cpp

```cpp
#include <iostream>
#include <stack>
#include <cmath>

using namespace std;

int main()
{
    stack<string> pila; // De
    string expresion;
    float operando1;
    float operando2;

    // Leemos expresion posfi
    cin >> expresion;

    /// Algoritmo de evaluaci
    // 1. Leemos toda la expr
    for (int i = 0; i < expre
    {
        // 2. El caracter lei
        if (isdigit(expresion
        {
            pila.push(string(
        }

        // 3. El caracter lei
```

```
"C:\Users\USER\Desktop\8vo semestre\Sem de traductores 2\SEM. TRAD. LENG.
82/8*2/
16.000000
Process returned 0 (0x0)   execution time : 11.500 s
Press any key to continue.
```

## 4. Conclusión

La implementación de un algoritmo para evaluar expresiones en notación postfija ha demostrado ser una solución eficaz para simplificar el cálculo de operaciones aritméticas, eliminando la complejidad asociada con los paréntesis y la precedencia de los operadores. Utilizando una estructura de datos como la pila, hemos logrado almacenar y manejar los operandos de manera eficiente, lo que facilita el procesamiento de la expresión ingresada por el usuario.
Además, la capacidad de incorporar funciones matemáticas estándar, como las operaciones trigonométricas, amplía las aplicaciones del programa más allá de las simples operaciones aritméticas, haciendo que este enfoque sea particularmente valioso en el ámbito de los lenguajes formales y los compiladores. En resumen, el desarrollo de este algoritmo no solo refuerza la comprensión de las estructuras de datos y la manipulación de expresiones, sino que también subraya la importancia de la notación postfija en la evaluación eficiente de expresiones matemáticas.

## 5. Bibliografía


- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compiladores: Principios, técnicas y herramientas* (2da ed.). Pearson Education.