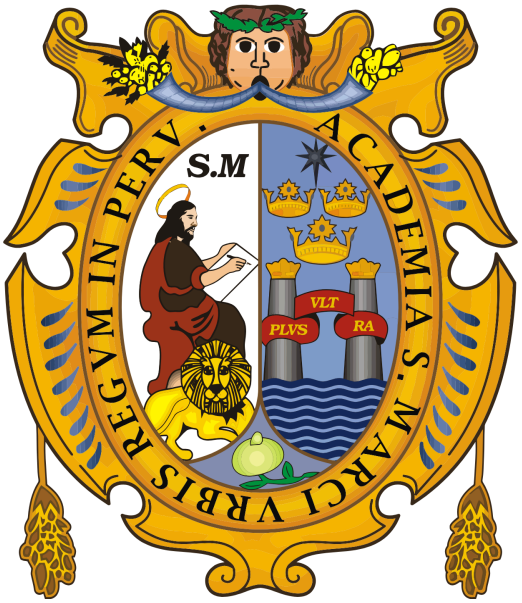
**UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS**
**(Universidad del Perú, Decana de América)**

**FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**




# “Traductor Infija a Postfija con Autómata de Pila”

**Grupo 4**

**Asignatura:**
TEORÍA DE LA COMPUTACIÓN

**Docente:**
VICTOR HUGO BUSTAMANTE OLIVERA

**Integrantes:**

Chamorro Figueroa, Andy Aaron - 24200072
Sánchez Mendoza, Diego Joaquín - 24200206
Verde Jara, Leonel Edwuar - 24200209
Zapata Llaxa, Jorge Andres - 24200211

**2025 - II**

Primer Avance del Proyecto 4
Traductor de Expresiones Infijas a Postfijas (C++)

# INTRODUCCIÓN

**a. Descripción del proyecto**

El proyecto que realizaremos en C++ se encargará de convertir las expresiones infijas (Notación natural, la que usamos comúnmente) a postfijas (Notación donde los operadores van después de los operandos, la que usan las computadoras debido al uso de estructuras de datos simples)

Para esto usaremos el algoritmo “Shunting-Yard”, creado por el científico en computación Edsger Dijkstra. Su nombre significa “Patio de maniobras” de trenes esto es así por su forma en la que hace que los operadores y paréntesis se muevan de manera que respeten la precedencia correcta. El algoritmo funciona de la siguiente manera según la metáfora:

- La expresión infija es como una fila de vagones que llegan.
- La pila de operadores actúa como una vía secundaria.
- Los operadores se mueven de un lado a otro igual que vagones en un patio ferroviario.
- El algoritmo ordena y reacomoda los elementos hasta que la expresión queda lista en orden posfijo.

Lo que buscaremos es que el programa analice la expresión en notación infija que el usuario le dé paso por paso cumpliendo la prioridad de los operadores y el uso de paréntesis pasando de infijas a postfijas.

**b. Necesidad de la traducción**

La principal razón por qué se necesita la traducción del algoritmo Shunting-Yard es porque las computadoras no son capaces de interpretar por sí mismas las expresiones que utilizamos generalmente los humanos. Nosotros escribimos en notación infija.
Por ejemplo: 3 + 4 * (2 - 1)

Las La notación infija requiere de reglas de precedencia como que la multiplicación se hace antes que la suma o que los paréntesis alteran el orden, llegando a ser ambiguas para las computadoras debido a que estas reglas no vienen “incluidas” en la expresión matemática, por lo tanto la computadora tendría que interpretarlas cada vez. Además que las traducciones directas desde infijo a código ejecutable son difíciles de realizar por los lenguajes de programación lo que llega a ser ineficiente y vulnerable a errores

Por esas razones las computadoras utilizan la notación posfija debido a que no necesita reglas de precedencia, el orden de ejecución ya está establecido, no existe ambigüedad y la expresión matemática se evalúa con una pila en una sola pasada lo que genera que la notación sea totalmente clara y determinista siendo ideal para las computadoras en el momento de procesar expresiones matemáticas.

**c. El rol del algoritmo Shunting-Yard como un Autómata de Pila**

El funcionamiento del algoritmo Shunting-Yard se relaciona adecuadamente con las características de un autómata de pila debido a las siguientes causas:

- **Lee una expresión infija símbolo por símbolo**
  Cada token (número, operador o paréntesis) se procesa uno seguido del otro semejante a un autómata de pila que procesa una cadena.
- **Utiliza una pila para controlar la estructura**
  Se guardan en una pila los operadores y paréntesis. Es decir, se almacena información necesaria acerca de la estructura del lenguaje similar a lo que hace un autómata de pila.
- **Responde de acuerdo el símbolo actual + el tope de la pila**
  Sigue ciertas reglas de acuerdo al operador que se encuentra en el tope de la pila y el símbolo actual.
- **La salida se obtiene mientras se utiliza la pila**
  Puede generar cadenas (autómata de pila con salida) como un AP.

**d. Plan de trabajo**

Seguiremos los siguientes pasos para desarrollar el proyecto:

1. Definir los tokens:
   Veremos qué elementos aceptara el programa en este caso serían números, operadores (+ - * /) y paréntesis.
2. Crear las estructuras base:
   Lo haremos con dos listas en C++:
   ➔ Una para el resultado en notaciòn postfija
   ➔ Otra que sería la pila donde se guardarán operadores y paréntesis.
3. Establecer la prioridad de los operadores:
   Esto sería una pequeña tabla para indicar que * y / tienen más prioridad que + y -.

| Operadores | Prioridad |
|---|---|
| / | 2 |
| * | 2 |
| + | 1 |

| | |
|---|---|
| - | 1 |

4. Implementar las reglas principales del método:
   Hacer que el programa siga el método Shunting-Yard. Esto incluye decidir qué hacer cuando aparece un número, un operador o un paréntesis.
5. Hacer pruebas:
   Ver si funciona correctamente con expresiones pequeñas, primero sin paréntesis y luego con ellos.
6. Intentar con expresiones erróneas:
   Detectar expresiones incorrectas, como paréntesis sin cerrar.
7. Preparar el informe final:
   Explicar cómo funciona el programa, cómo se usa y el código.

**e. Primer avance en código**

Mostramos un poco del código donde definimos los tokens iniciales, las estructuras necesarias y la tabla de precedencias.