
1. Conversión de Expresión Infija a Notación Posfija (Postfijo)

Para evaluar expresiones matemáticas correctamente, primero se **convierte la expresión infija** (notación común con paréntesis y operadores) a **notación posfija** (postfijo). Esto se logra mediante el uso de **una pila de operadores**.

Pasos clave del proceso:

1. **Leer la expresión de izquierda a derecha.**
2. **Números** → Se envían directamente a la lista de salida (expresión posfija).
3. **Operadores** → Se comparan sus prioridades con los operadores en la pila:
 - Si el operador de fuera (nuevo) tiene mayor prioridad, **entra a la pila**.
 - Si el operador de dentro (tope de la pila) tiene **igual o mayor prioridad**, **se saca** y se envía a la expresión posfija antes de insertar el nuevo.
4. **Paréntesis de apertura (** → Siempre entra a la pila.
5. **Paréntesis de cierre)** → Se sacan operadores hasta encontrar el paréntesis de apertura (, el cual se descarta.
6. **Al final de la expresión**, se vacía la pila, enviando los operadores restantes a la expresión posfija.

📌 **Ejemplo de conversión:**

Expresión infija:

$((17 + 3) * (21 - 2) / 5) ^ 3$

Expresión posfija resultante:

17 → 3 → + → 21 → 2 → - → * → 5 → / → 3 → ^

2. Evaluación de la Expresión Posfija

Una vez convertida la expresión a **postfijo**, se evalúa utilizando **una pila dinámica de números**.

Pasos clave:

1. **Recorrer la expresión posfija de izquierda a derecha.**
2. **Números** → Se apilan.
3. **Operadores** → Se desapilan **dos números**, se realiza la operación y el resultado se vuelve a apilar.
4. **El último número en la pila es el resultado final.**

📌 **Ejemplo de evaluación:**

1. $17 + 3 = 20 \rightarrow$ Se apila 20.
2. $21 - 2 = 19 \rightarrow$ Se apila 19.
3. $20 * 19 = 380 \rightarrow$ Se apila 380.
4. $380 / 5 = 76 \rightarrow$ Se apila 76.
5. $76 ^ 3 = 438976 \rightarrow$ Resultado final.

- El resultado final de la evaluación es 438976.
 - Validado con calculadora para asegurar exactitud.
-

3.Material de Apoyo y Entrega de Tarea

📌 Se proporciona:

- Documento con la explicación completa.
- Cinco archivos de prueba.
- El enunciado de la tarea.
- Otro documento de apoyo con ejemplos resueltos.
- Video explicativo con todo el proceso.

📌 Entrega:

- Fecha límite: sábado.
- Se recomienda trabajar en parejas, pero es opcional.

implementar una **cola estática de tamaño cinco**, donde cada entrada en la cola es un **nodo** que representa una expresión matemática leída desde un archivo. La cola funcionará como un sistema FIFO (First In, First Out), atendiendo las expresiones en el orden en que ingresaron.

Aquí está el desglose de lo que necesitas hacer:

1. **Leer los cinco archivos** y extraer las expresiones matemáticas.
2. **Crear una cola estática de tamaño cinco**, donde cada posición almacena un nodo.
3. **Cada nodo contiene una expresión matemática** y posiblemente una referencia al siguiente nodo.
4. **Atender la cola en orden**, evaluando o procesando las expresiones conforme sean extraídas.

Plan de Implementación

1. **Lectura y Almacenamiento en la Cola (FIFO)**
 - Leer expresiones matemáticas desde archivos.
 - Almacenar cada expresión en una cola estática (de tamaño 5).
2. **Conversión de Notación Infija a Posfija con una Pila Dinámica**
 - Usar una pila dinámica para manejar la conversión utilizando **la tabla de prioridades** de operadores.
3. **Evaluación de la Expresión Posfija con otra Pila**
 - Usar una pila de evaluación (último en entrar, primero en salir - LIFO).
 - Extraer números y operadores en el orden correcto.

manejear **números de cualquier tamaño**, lo que significa que podría ser necesario trabajar con **strings** en lugar de tipos numéricos fijos como int o double. También, la implementación debe hacerse en **Dev-C++ o Visual Studio**, sin el uso de **templates** ni listas para la cola, solo pilas dinámicas.

- **Lectura de archivos y almacenamiento en una cola estática** (arreglo de tamaño 5).
- **Conversión de notación infija a posfija** usando una **pila dinámica**.
- **Evaluación de la expresión posfija** usando otra **pila dinámica**.
- **Manejo de números largos** almacenándolos como **strings**.

Explicación del Código

1. **Cola Estática (ColaEstatica)**
 - Se usa un **arreglo de tamaño 5** para almacenar las expresiones matemáticas.
 - Se mantiene un índice **frente** y **final** para controlar la inserción y extracción de expresiones en **orden FIFO**.
2. **Pilas Dinámicas (Pila)**

- Se implementan con una lista enlazada (`NodoPila`).
 - Se usan para la conversión de infijo a posfijo y para evaluar la expresión.
3. **Conversión de Infijo a Posfijo** (`infijoAPosfijo`)
- Se usa una **pila dinámica** para manejar operadores con base en su **prioridad**.
 - Se manejan paréntesis y operadores correctamente.
4. **Evaluación de Expresión Posfija** (`evaluarPosfijo`)
- Se usa otra **pila dinámica** para calcular el resultado en orden correcto.
 - Soporta operadores `+, -, *, /, ^`.
5. **Lectura de archivos** (`leerArchivos`)
- Lee hasta 5 archivos (`expresion1.txt`, `expresion2.txt`, etc.).
 - Encola cada expresión en la **cola estática**.
6. **Ejecución en `main()`**
- Se leen expresiones de archivos y se procesan en **orden FIFO**.
 - Se imprime la versión **infija, posfija y su resultado**.
-

Ejemplo de Entrada y Salida

Archivos

- `expresion1.txt`: "3 + 5 * 2"
- `expresion2.txt`: "(8 - 3) / 2"
- `expresion3.txt`: "2^3 + 4"
- `expresion4.txt`: "10 * (5 + 2)"
- `expresion5.txt`: "6 / 3 + 7"

Infija: 3 + 5 * 2

Posfija: 3 5 2 * +

Resultado: 13

Infija: (8 - 3) / 2

Posfija: 8 3 - 2 /

Resultado: 2.5

...

Explicación del Código

1. **Cola Estática (colaEstatica)**
 - Implementada con un **arreglo de nodos** (sin listas ni estructuras automáticas).
 - Almacena hasta 5 expresiones matemáticas.
 - Permite recorrer los elementos sin eliminarlos.
 2. **Pilas Dinámicas (pila)**
 - Implementadas con listas enlazadas.
 - Se usan para la conversión de **infijo a posfijo** y para la **evaluación de la expresión**.
 3. **Conversión de Notación Infija a Posfija (infijoAPosfijo)**
 - Utiliza una **pila dinámica** para manejar operadores.
 - Ordena los operadores según su prioridad.
 4. **Evaluación de Expresión Posfija (evaluarPosfijo)**
 - Utiliza otra **pila dinámica** para evaluar la expresión.
 - Maneja operadores +, -, *, /, ^.
 5. **Lectura de Archivos (leerArchivos)**
 - Lee hasta 5 archivos de texto (expresion1.txt, expresion2.txt, etc.).
 - Encola cada expresión en la **cola estática**.
 6. **Recorrido e Impresión en main()**
 - **Se recorre la cola sin eliminar elementos**.
 - Se imprime la **expresión original, la posfija y su evaluación**.
-

Salida Esperada

```
makefile
Infija: 3 + 5 * 2
Posfija: 3 5 2 * +
Resultado: 13

Infija: (8 - 3) / 2
Posfija: 8 3 - 2 /
Resultado: 2.5
```

...

manualmente sin usar estructuras automáticas del lenguaje.

- **Cola estática** → Implementada con un **arreglo de nodos**.
- **Pilas dinámicas** → Implementadas con **listas enlazadas**.
- **Conversión infijo a posfijo** → Usando nuestra propia pila.
- **Evaluación de la expresión posfija** → Usando nuestra propia pila.
- **Nada de librerías estándar para estructuras de datos** (como vector, stack, queue, etc.).

el proceso de lectura y almacenamiento de la expresión matemática en una **cola estática** y luego estructurando la forma en que los operadores y números se almacenan en **nodos individuales**.

Aquí hay algunas claves de lo que debes asegurarte en la implementación en C++:

1. **Leer cada archivo línea por línea** → Cada archivo contiene **una única expresión**.
2. **Crear la cola estática (arreglo de nodos) de tamaño 5** → No es una lista enlazada.
3. **Cada nodo en la cola almacena una referencia a una lista enlazada** → En esa lista enlazada se desglosa la expresión en **números y operadores como nodos individuales**.
4. **Construcción de la expresión en la cola:**
 - Si es un **número**, se almacena en un nodo.
 - Si es un **operador o paréntesis**, se almacena en otro nodo.
5. **Uso de la tabla de prioridades** → Esto se aplicará en la conversión de **infijo a posfijo** más adelante.

- **Cola estática de tamaño 5**

- Cada posición almacena una referencia a una lista enlazada.

- **Cada lista enlazada representa una expresión matemática**

- Cada nodo en la lista almacena un **número** o un **operador** individualmente.
 - Se sigue el orden de lectura del archivo.

Explicación del Código

1. **Cola Estática (ColaEstatica)**
 - Implementada con un **arreglo de nodos** de tamaño **5**.
 - Cada nodo contiene una **lista enlazada** que representa la expresión.
2. **Lista Enlazada (NodoLista)**
 - Cada nodo almacena un **número** o un **operador** como un `string`.
3. **Procesamiento de la Expresión (procesarExpresion)**
 - Se divide la expresión en **números y operadores**.
 - Se almacena en una **lista enlazada**.
4. **Lectura de Archivos (leerArchivos)**
 - Se leen **5 archivos** (`expresion1.txt`, `expresion2.txt`, ...).
 - Cada expresión se **almacena en la cola estática**.
5. **Impresión de la Cola (imprimirCola)**
 - Se recorre la cola y se muestra cada expresión separada en **nodos enlazados**.

Ejemplo de Entrada (Archivos)

Archivo expresion1.txt

(4+3)^2*5/6

Archivo expresion2.txt

((17+3)*(21-2)/5)^3

Salida Esperada

```
Expresion 1: ( -> 4 -> + -> 3 -> ) -> ^ -> 2 -> * -> 5 -> / -> 6 -> NULL  
Expresion 2: ( -> ( -> 17 -> + -> 3 -> ) -> * -> ( -> 21 -> - -> 2 -> ) -> /  
-> 5 -> ) -> ^ -> 3 -> NULL  
...
```

el **primer paso** de leer los archivos y construir la estructura de datos:

1. Leer la expresión desde el archivo línea por línea.
2. Dividir la expresión en números y operadores.
3. Crear una lista enlazada, donde cada nodo contiene un número u operador.
4. Almacenar cada lista en una posición de la cola estática.
5. Terminar la lista con NULL cuando la expresión se haya leído por completo.

Paso siguiente: Conversión de Infijo a Posfijo

- Usaremos una **pila dinámica** para manejar la prioridad de operadores.
- Aplicaremos la **tabla de prioridades** que tienes en el documento.
- Se generará una **nueva lista enlazada** con la expresión en **notación posfija**.

verificando la correcta representación de las expresiones matemáticas en la cola estática con listas enlazadas. Te aseguras de que los operadores y números se almacenen en el orden correcto antes de proceder a la conversión de infijo a posfijo.

Si ya tienes la estructura de almacenamiento validada, el siguiente paso sería **convertir la expresión de infijo a posfijo** usando una **pila dinámica**. Esto implica:

1. Recorrer la lista enlazada de cada expresión.
2. Aplicar la tabla de prioridades para manejar operadores correctamente.
3. Generar una nueva lista enlazada con la expresión en notación posfija.

Pasos para la conversión de infijo a posfijo

1. Recorrer la lista enlazada de la expresión infija (almacenada en la cola).
 2. Usar una pila para manejar operadores con base en la tabla de prioridades.
 3. Construir una nueva lista enlazada que almacenará la expresión en notación posfija.
-

Estructura en C++

Para esto, implementaremos:

- **Pila Dinámica** → Almacena operadores (+, -, *, /, ^).
- **Lista Enlazada de salida** → Almacena la expresión en notación posfija.

Explicación del Código

1. **Pila Dinámica (Pila)**
 - Se usa para manejar **operadores** con base en su prioridad.
2. **Lista Enlazada (NodoLista)**
 - Almacena la **expresión infija** y luego la **expresión posfija**.
3. **Conversión Infijo a Posfijo (infijoAPosfijo)**
 - Se recorren los nodos de la **expresión infija**.
 - Se **usan las prioridades** para organizar operadores.
 - Se genera una **nueva lista enlazada** con la expresión en notación posfija.
4. **Ejemplo en main()**
 - Se crea la **expresión infija** en lista enlazada.
 - Se **convierte a posfijo**.
 - Se imprimen ambas expresiones.

Ejemplo de Salida

```
Expresión Infija: ( -> 17 -> + -> 3 -> ) -> * -> ( -> 21 -> - -> 2 -> ) -> / -> 5 -> ) -> ^ -> 3 -> NULL
Expresión Posfija: 17 -> 3 -> + -> 21 -> 2 -> - -> * -> 5 -> / -> 3 -> ^ -> NULL
```

Flujo de Conversión Infijo → Posfijo

1. Leer un token de la lista de la expresión infija (que está en la cola estática).
2. Si es un número, va directamente a la lista de salida (expresión posfija).
3. Si es un operador, se compara su prioridad con el tope de la pila de operadores:
 - Si su **prioridad fuera de la pila** es mayor que la **prioridad dentro de la pila** del tope, entra a la pila.

- Si es menor o igual, **se desapila el tope y se pasa a la lista posfija**, repitiendo la comparación.
4. **Si es un paréntesis de apertura "(",** siempre entra a la pila.
 5. **Si es un paréntesis de cierre ")",** **se desapilan operadores** hasta encontrar su paréntesis de apertura correspondiente.
 - **El paréntesis no se pasa a la salida,** solo los operadores entre ellos.
 6. **Al final de la expresión,** se desapilan todos los operadores restantes y se pasan a la salida.

Explicación del Código

1. **Pila Dinámica (Pila)**
 - Se usa para manejar **operadores** con base en su prioridad.
2. **Lista Enlazada (NodoLista)**
 - Almacena la **expresión infija** y luego la **expresión posfija**.
3. **Conversión Infijo a Posfijo (infijoAPosfijo)**
 - Se recorren los nodos de la **expresión infija**.
 - **Se usan las prioridades** para organizar operadores.
 - Se genera una **nueva lista enlazada** con la expresión en notación posfija.
4. **Ejemplo en main()**
 - Se crea la **expresión infija** en lista enlazada.
 - **Se convierte a posfijo.**
 - Se imprimen ambas expresiones.

Ejemplo de Salida

```
Expresión Infija: ( -> 17 -> + -> 3 -> ) -> * -> ( -> 21 -> - -> 2 -> ) -> /
-> 5 -> ) -> ^ -> 3 -> NULL
Expresión Posfija: 17 -> 3 -> + -> 21 -> 2 -> - -> * -> 5 -> / -> 3 -> ^ ->
NULL
```

el proceso de conversión **infijo a posfijo**, manejando correctamente **la pila de operadores y la lista de salida**. Ya cubriste los puntos clave:

1. **Números van directo a la lista posfija.**
2. **Operadores se manejan con la tabla de prioridades.**
3. **Si un operador entrante tiene menor prioridad que el tope de la pila,**
 - Se desapila el tope y se mueve a la **lista posfija**.
 - Luego, el nuevo operador entra en la pila.
4. **Paréntesis de cierre ")",**
 - Se desapilan operadores hasta encontrar el paréntesis de apertura,
 - Pasando los operadores a la **lista posfija**.

la **conversión de infijo a posfijo** y que ahora tienes bien estructurada la **lista posfija**. También confirmaste que el orden de los operadores y números en la **lista dinámica de salida** es correcto.

Siguientes Pasos

1. **Validar la salida:** Asegurar que la lista en **notación posfija** realmente refleja la conversión correcta de la expresión infija.
2. **Evaluuar la expresión posfija:** Ahora que ya tenemos la expresión en **posfijo**, toca usar otra **pila dinámica** para calcular el resultado final.
3. **Implementar la evaluación en C++:**
 - **Recorrer la lista posfija.**
 - **Si es un número,** se apila.
 - **Si es un operador,** se sacan dos elementos de la pila, se evalúa y el resultado se vuelve a apilar.
 - **Al final, el único valor en la pila es el resultado.**

dentro del **arreglo de la cola estática (tamaño 5)** no se almacenan directamente las expresiones, sino **referencias al primer nodo de una lista enlazada**.

Cada posición en la cola **apunta al primer nodo de una lista**, que representa una **expresión matemática desglosada en operadores y números**. Es lo mismo que sucede con una lista enlazada normal, donde el nodo inicial da acceso a todos los demás.

Resumen de la Estructura

1. **Cola estática (arreglo de 5 nodos)**
 - Cada nodo en la cola contiene una **referencia al primer nodo de una lista**.
 2. **Lista enlazada en cada nodo de la cola**
 - Almacena la expresión en **números y operadores como nodos individuales**.
 3. **Proceso**
 - **Se lee un archivo, se convierte en una lista enlazada.**
 - **Se almacena la referencia en la cola estática.**
 - **Cada posición de la cola apunta al inicio de su lista correspondiente.**
-
- **Cola estática:** `NodoCola cola[5];`
 - **Cada NodoCola tiene un puntero a NodoLista** (inicio de la expresión).
 - **Se accede a la expresión recorriendo la lista enlazada desde la cola.**

Proceso de Evaluación

1. Recorrer la lista de la expresión posfija.
2. Si es un número, se apila.
3. Si es un operador, se sacan dos números de la pila:
 - o num2 = desapilar()
 - o num1 = desapilar()
 - o Se evalúa la operación correspondiente.
 - o El resultado se vuelve a apilar.
4. Al final de la evaluación, el resultado estará en el tope de la pila.

Explicación del Código

1. Pila Dinámica (**Pila**)
 - o Almacena solo números.
 - o Se usa para evaluar la expresión en notación posfija.
2. Lista Enlazada (**NodoLista**)
 - o Contiene la expresión en notación posfija.
3. Evaluación (**evaluarPosfijo**)
 - o Se recorre la lista posfija.
 - o Se apilan números.
 - o Si es un operador, se desapilan dos números, se calcula el resultado y se vuelve a apilar.
 - o Al final, el resultado queda en la pila.
4. Prueba en **main()**
 - o Se crea la lista posfija correspondiente a:

4 3 + 2 ^ 5 * 6 /

- o Se imprime la lista.
- o Se evalúa y muestra el resultado final.

Salida Esperada

Expresión Posfija: 4 -> 3 -> + -> 2 -> ^ -> 5 -> * -> 6 -> / -> NULL
Resultado: 17.5

la evaluación de la expresión posfija paso a paso, asegurándose de que:

1. Los números se apilan correctamente.
2. Los operadores desapilan dos valores, aplican la operación y reapilan el resultado.
3. El orden de evaluación es correcto, cumpliendo con la notación posfija.

Ya verificaste el primer cálculo:

- $4 + 3 \rightarrow 7$ (reapilado en la pila).
- $7 \wedge 2 \rightarrow 49$ (reapilado en la pila).

Siguiendo este mismo proceso, los siguientes pasos serán:

- Multiplicar $49 * 5$
- Dividir el resultado entre 6
- Obtener el valor final.

la evaluación de la expresión **posfija**, asegurando que:

1. **Cada número se apila correctamente.**
2. **Cada operador toma los dos valores superiores de la pila**, aplica la operación y **reapila el resultado**.
3. **El último resultado en la pila es el valor final de la expresión.**

El último cálculo que hiciste:

- $49 \times 5 = 245$ (reapilado en la pila).
- $245 \div 6 = 40.83$ (resultado final en la pila).

el resultado final es **40.8666**, y ahora estás validando el cálculo con una calculadora para asegurarte de que la evaluación posfija fue correcta.

Si la calculadora da el mismo resultado que tu proceso manual, significa que:

- **La conversión infijo → posfijo está correcta.**
- **La evaluación posfija con la pila dinámica está bien implementada.**
- **El algoritmo respeta las reglas de prioridad de operadores.**

la **pila de operadores** contiene **un paréntesis y una multiplicación (*)**.

Como los paréntesis no se almacenan en la notación posfija, **se eliminan**.

Luego, se encuentra la **división (/)**, que tiene **prioridad 2**.

- En la pila, el operador en el tope es una **multiplicación (*)**, que también tiene **prioridad 2**.
- **Como las prioridades son iguales**, se saca el operador ***** de la pila y se pasa a la expresión posfija.
- Luego, **se inserta el operador / en la pila**.

Esto sigue las **reglas de precedencia de operadores**, asegurando que * se evalúe antes que / si aparecen en el mismo nivel.

La **conversión a notación posfija** ha finalizado correctamente.

Últimos pasos del proceso:

1. **Se encontró un ^ (potencia)**
 - o Como la pila estaba vacía, **entra directo a la pila**.
 2. **Se encontró un 3**
 - o Como es un número, **va directo a la expresión posfija**.
 3. **Se llegó al final de la lista**
 - o Se sacan **todos los operadores restantes en orden inverso**.
 - o En este caso, **solo quedaba el ^**, que se mueve a la **expresión posfija**.
-

Expresión Posfija Final

17 -> 3 -> + -> 21 -> 2 -> - -> * -> 5 -> / -> 3 -> ^

 **La pila de operadores quedó vacía, lo que indica que el proceso se completó correctamente.**

Próximo paso: Evaluación de la expresión posfija

Ahora toca recorrer la expresión posfija y calcular el resultado final usando una **pila dinámica de números**.

evaluación de la expresión posfija.

Se está siguiendo correctamente el algoritmo de **pila dinámica para evaluar expresiones**.

Pasos realizados hasta ahora:

1. **Apilamos los números** conforme aparecen en la expresión posfija:
 - o 17 → **Se apila**.
 - o 3 → **Se apila**.
2. **Encontramos un +, por lo que se desapilan dos valores y se suman:**
 - o num2 = 3, num1 = 17
 - o resultado = num1 + num2 = 17 + 3 = 20
 - o 20 **se vuelve a apilar**.
3. **Continuamos leyendo y apilando números:**
 - o 21 → **Se apila**.
 - o 2 → **Se apila**.

4. Encontramos un **-**, por lo que se desapilan dos valores y se restan:

- o num2 = 2, num1 = 21
- o resultado = num1 - num2 = 21 - 2 = 19
- o 19 se vuelve a apilar.

5. Seguimos apilando más números:

- o 5 → Se apila.
- o 3 → Se apila.

la fase de **evaluación de la división (/)**, pero notaste que **no la insertaste correctamente en la expresión posfija** en el paso anterior.

Corrección del Proceso

1. Revisar si la división (/) está correctamente ubicada en la expresión posfija.
2. Si la omitiste en la conversión infijo → posfijo, corregirla y asegurarte de que la jerarquía de operadores se respete.
3. Verificar que la evaluación siga el orden correcto:
 - o Después de 19 y 5, debería aplicarse la división antes de continuar con 3^{\wedge} .

Corrección del Proceso

1. Verificación de operaciones completadas:
 - o ✓ 17 + 3 = 20
 - o ✓ 21 - 2 = 19
 - o ! Faltaba la multiplicación (*) entre 20 * 19.
2. Corrección aplicada:
 - o num2 = 19, num1 = 20
 - o resultado = num1 * num2 = 20 * 19 = 380
 - o 380 se vuelve a apilar.
3. Continuamos con la evaluación de la división (/):
 - o num2 = 5, num1 = 380
 - o resultado = num1 / num2 = 380 / 5 = 76
 - o 76 se vuelve a apilar.
4. Último número leído:
 - o 3 → Se apila.

la conversión y evaluación de la expresión matemática utilizando **notación posfija (postfijo)** y una **pila dinámica**.

Último paso evaluado:

✓ Cálculo de la potencia (\wedge)

- num2 = 3, num1 = 76
- resultado = num1 \wedge num2 = 76 \wedge 3 = 438976
- Este valor es el resultado final, almacenado en la pila.

 **Verificación con la calculadora**

- Confirmaste que el resultado **coincide exactamente** con la operación matemática original.