Aplicación de las pilas en el análisis de expresiones aritméticas

Estudiante: Fernando Matamoros Acuña

Estructuras de Datos 1

Profesor: Romario Salas Cerdas

Universidad CENFOTEC

Octubre 2025

Introducción

Las pilas son una de las estructuras de datos fundamentales en la informática, que son utilizadas debido a su naturaleza de acceso LIFO (Last In, First Out). Esta característica permite manejar de manera ordenada y controlada los elementos en contextos donde el último dato insertado debe ser el primero en procesarse, como ocurre en la evaluación de expresiones matemáticas, la gestión de llamadas a funciones o el manejo de deshacer/rehacer en editores (Goodrich, Tamassia & Goldwasser, 2022).

En este estudio de caso se aborda la aplicación de las pilas en el análisis de expresiones aritméticas, la cuál es una funcionalidad esencial tanto en compiladores como en intérpretes de lenguajes de programación. El objetivo es comprender por qué una pila resulta la estructura más adecuada para esta tarea, analizar los beneficios que aporta en términos de seguridad, orden y eficiencia, y revisar un ejemplo concreto dentro de un software de sistema donde este concepto se aplica en la práctica.

¿Por qué una pila es la estructura adecuada?

El análisis y evaluación de expresiones aritméticas requiere procesar los operandos y operadores siguiendo las reglas de prioridad y asociación que determinan el orden de ejecución de las operaciones (Weiss, 2014). Por ejemplo, en una expresión como:

3 + 4 * (2 - 1)

La multiplicación y la operación dentro de los paréntesis deben resolverse antes que la suma. Para manejar este orden de manera estructurada, los algoritmos de análisis emplean dos pilas: una para los operadores y otra para los operandos (Knuth, 1997).

El comportamiento LIFO de la pila permite que el último operador ingresado (el de mayor procedencia activa) sea el primero en aplicarse, reproduciendo exactamente la jerarquía de operaciones matemáticas. Cuando se encuentra un paréntesis de cierre o un operador de menor prioridad, se desapila los operadores previos en el orden correcto, garantizando que la evaluación final siga las reglas sintácticas del lenguaje.

Además las pilas facilitan la conversión entre diferentes notaciones de expresiones:

- Infija: operadores entre operandos (A + B)
- Postfija (RPN): operadores después de los operandos (A B +)
- Prefija: operadores antes de los operandos (+ A B)

El uso de una pila simplifica esta conversión, ya que permite almacenar temporalmente los operadores mientras se reorganizan los términos según su procedencia. En consecuencia, sin una pila, sería muy complejo implementar un algoritmo que respete las reglas del lenguaje y mantenga la consistencia del orden de evaluación (Cormen, Leiserson, Rivest & Stein, 2022).

Seguridad, orden y eficiencia que aporta una pila

Seguridad

El uso de una pila aporta seguridad estructural durante la evaluación de expresiones. Al validar la correspondencia entre paréntesis de apertura y cierre, la pila impide errores sintácticos como ((3 + 2) o 2 + * 4. Cada apertura (se apila, y solo cuando se encuentra un cierre) se desapila, verificando que el orden sea correcto (Goodrich et al., 2022). Si al finalizar el análisis quedan elementos en la pila, el algoritmo detecta automáticamente un error de balanceo.

Asimismo, la pila permite manejar errores en tiempo de ejecución (por ejemplo, división entre cero o expresiones incompletas) sin comprometer la estabilidad del programa. Dado que cada operación se realiza de forma controlada sobre el elemento superior de la pila, el estado del sistema puede restaurarse fácilmente en caso de excepción.

Orden

La pila garantiza el orden lógico y jerárquico en el procesamiento de la expresión. Gracias a su naturaleza LIFO, los operadores de mayor prioridad se aplican primero, y los operandos se consumen en el orden inverso al que fueron insertados, tal como lo requiere la evaluación de expresiones.
Esto mantiene la coherencia entre las reglas de precedencia matemática y la secuencia de ejecución en el código generado (Knuth, 1997).

Por ejemplo, durante la conversión a notación postfija, el orden de aparición de los operadores se gestiona íntegramente mediante operaciones de push y pop, eliminando la necesidad de estructuras de control más complejas como listas doblemente enlazadas o árboles temporales.

Eficiencia

Desde el punto de vista computacional, la pila ofrece operaciones con complejidad O(1) para las funciones push(), pop() y top(). Esto significa que el tiempo de ejecución no depende del tamaño de la expresión, sino únicamente del número de elementos procesados en un momento dado (Cormen et al., 2022).
El análisis completo de una expresión aritmética tiene, por tanto, complejidad O(n), siendo n el número de tokens (operandos y operadores).

Además, el uso de una pila minimiza el consumo de memoria, pues únicamente mantiene en memoria los símbolos necesarios para la operación actual. Esta eficiencia convierte a la pila en la estructura ideal para compiladores, intérpretes y analizadores léxicos que requieren evaluar miles de expresiones por segundo.

Ejemplo en software de un sistema

Un ejemplo representativo del uso de pilas en software de sistema se encuentra en el compilador de C/C++. Durante la etapa de análisis sintáctico y generación de código intermedio, el compilador debe interpretar expresiones como:

```
resultado = (a + b) * (c - d / e);
```

El compilador convierte la expresión infija a notación postfija (RPN) mediante una pila de operadores. Posteriormente, otra pila se utiliza para evaluar la expresión y generar instrucciones equivalentes en código máquina o bytecode (Aho, Lam, Sethi & Ullman, 2007).

Por ejemplo, la expresión anterior se traduce a:

```
a b + c d e / - *
```

Y su evaluación ocurre paso a paso utilizando una pila de operandos:

Paso	Token	Acción	Pila (de abajo hacia arriba)
1	a	push(a)	a
2	b	push(b)	a, b
3	+	pop b, a → push(a+b)	(a+b)
4	c	push(c)	(a+b), c
5	d	push(d)	(a+b), c, d
6	e	push(e)	(a+b), c, d, e
7	/	pop e, d → push(d/e)	(a+b), c, (d/e)
8	-	pop (d/e), c → push(c−d/e)	(a+b), (c−d/e)
9	*	pop (c−d/e), (a+b) → push((a+b)*(c−d/e))	resultado

Este mismo principio se aplica internamente en los intérpretes de lenguajes de scripting (Python, JavaScript, Lua), en los cuales la pila de operandos forma parte del stock de ejecución del intérprete. Cada operación aritmética genera instrucciones que manipulan la pila virtual del proceso, reflejando de forma directa el modelo conceptual descrito (Goodrich et al., 2022).

Conclusión

El análisis de expresiones aritméticas representa una de las aplicaciones más clásicas y pedagógicas de las pilas dentro de la programación. Su naturaleza LIFO se ajusta de manera perfecta a las necesidades de evaluación de expresiones, garantizando orden lógico, seguridad sintáctica y eficiencia temporal.
Tanto los compiladores modernos como los intérpretes de lenguajes de alto nivel dependen de este principio para procesar instrucciones complejas de forma rápida y fiable.

La implementación de esta funcionalidad en C++ mediante clases Nodo y Pila no solo permite afianzar el dominio de las estructuras dinámicas, sino también comprender los fundamentos de cómo un software de sistema traduce el código fuente a operaciones ejecutables, destacando la relevancia práctica de las pilas en la ingeniería del software contemporánea.

Referencias

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). Compilers: Principles, Techniques, and Tools (2nd ed.). Pearson.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (4th ed.). MIT Press.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2022). Data Structures and Algorithms in C++ (2nd ed.). Wiley.

Knuth, D. E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.). Addison-Wesley.

Weiss, M. A. (2014). Data Structures and Algorithm Analysis in C++ (4th ed.). Pearson.