Stacks: Introduction

Let's go over the Stacks pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following

Overview

A **stack** is a linear data structure that follows a last in, first out (LIFO) order to perform operations on its elements. This means that whenever we obtain an element from the stack, it will always return the last inserted element. It's widely used by programmers to solve computational problems.

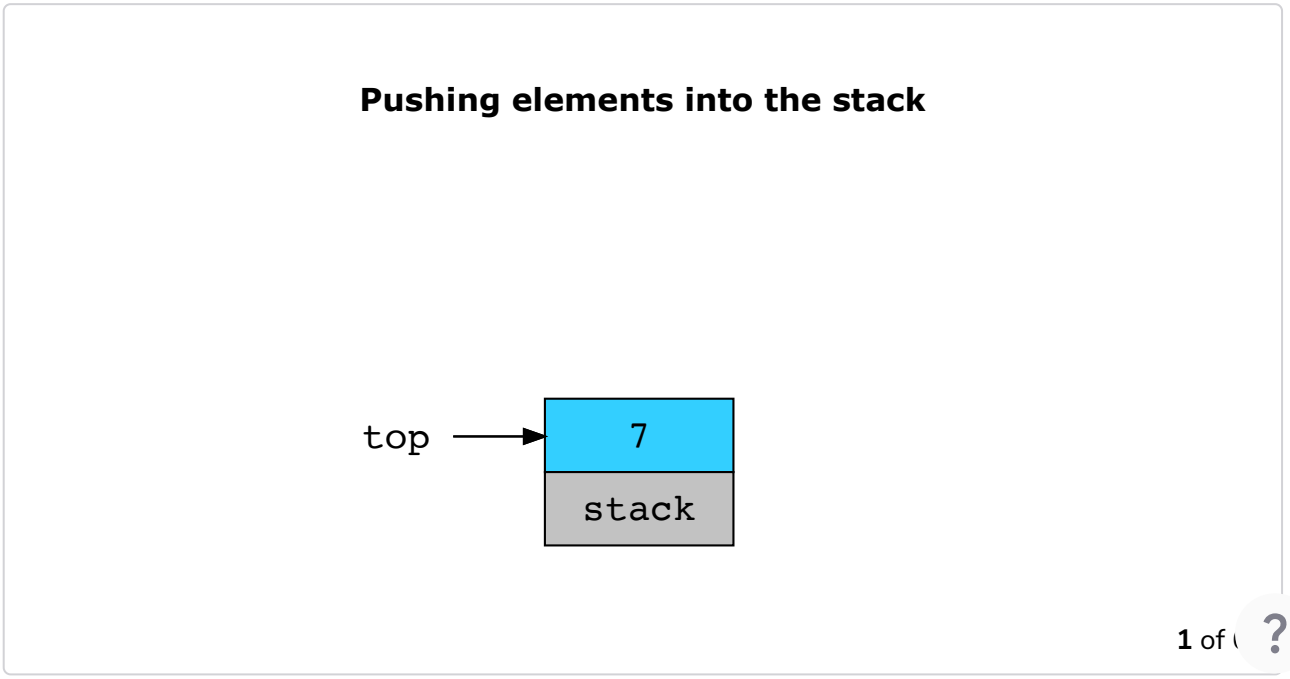
A stack has two primary operations – push and pop. The push operation adds an element to the top of the stack while the pop operation removes an element from the top of the stack. Unlike a list, an element can only be added to the top of the stack, and to add an element at some specified index, all elements at the top of that index must first be removed from th stack and then reinserted.

A stack has many real-world uses, which you might be performing in your daily life without being aware of. For example, consider a pile of plates placed on each other. To get to the bottom plate, all plates on top must be

removed first. The plate at the top is the first to be removed, and the plate at the bottom is the last to be removed. This is just one example of a stack used in the real world.

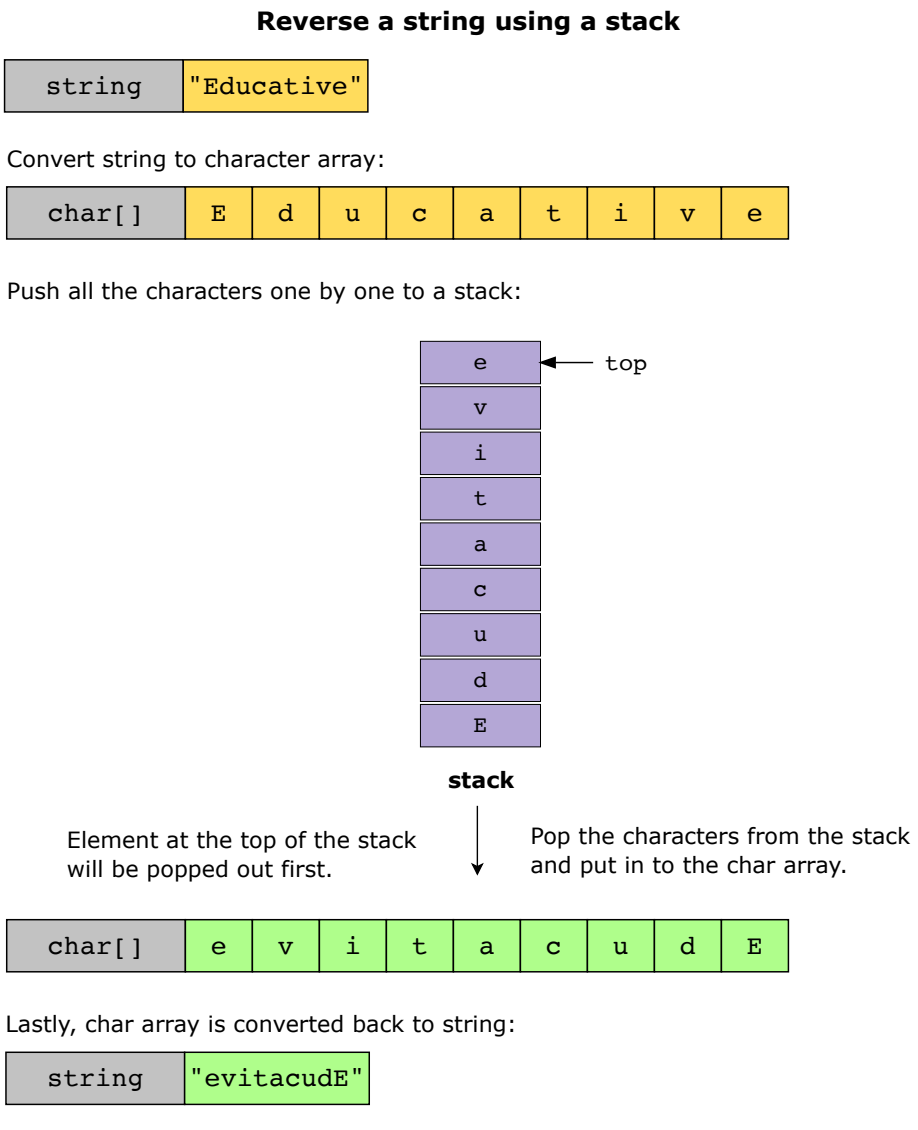
Stacks in programming are used to store elements that are sequentially dependent on each other. When required, the elements can then be popped from the stack while maintaining a fixed order. The nature and correlation of these elements are to be decided by you, as the programmer when deciding how to implement the stack based on the problem at hand. In addition, they are used when elements need to be stored in a safe manner where it is not desirable to modify them from an arbitrary position such as the middle. Repeatedly modifying a stream of elements is another property of the stack where elements are pushed onto it and then popped when certain specified conditions are met between the top of the stack and the next incoming element.

The following illustration demonstrates the push and pop functionality of the stack:



Examples

The following examples illustrate some problems that can be solved with this approach:



1 of 2

Does my problem match this pattern?

Yes, if you want to get elements out in the reverse order in which you inserted them (LIFO).

No, if either of these conditions is fulfilled:

- The problem requires you to get elements out in the same order in which you inserted them. This is known as First-In-First-Out (FIFO) processing.
- The order of the outputs or the order of the processing operations is not significant.

Real-world problems

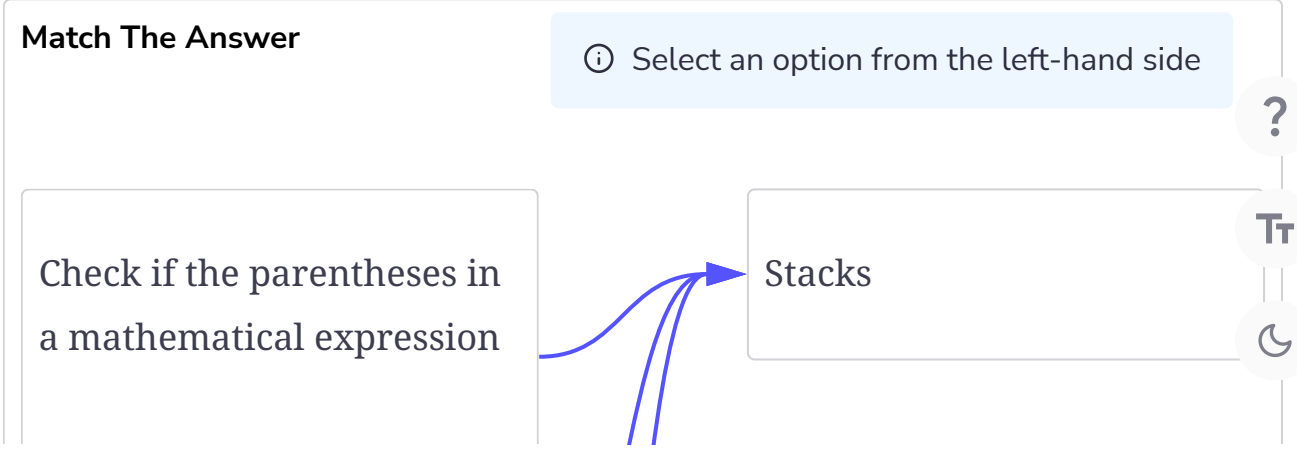
Many problems in the real world share the stack pattern. Let's look at some examples.

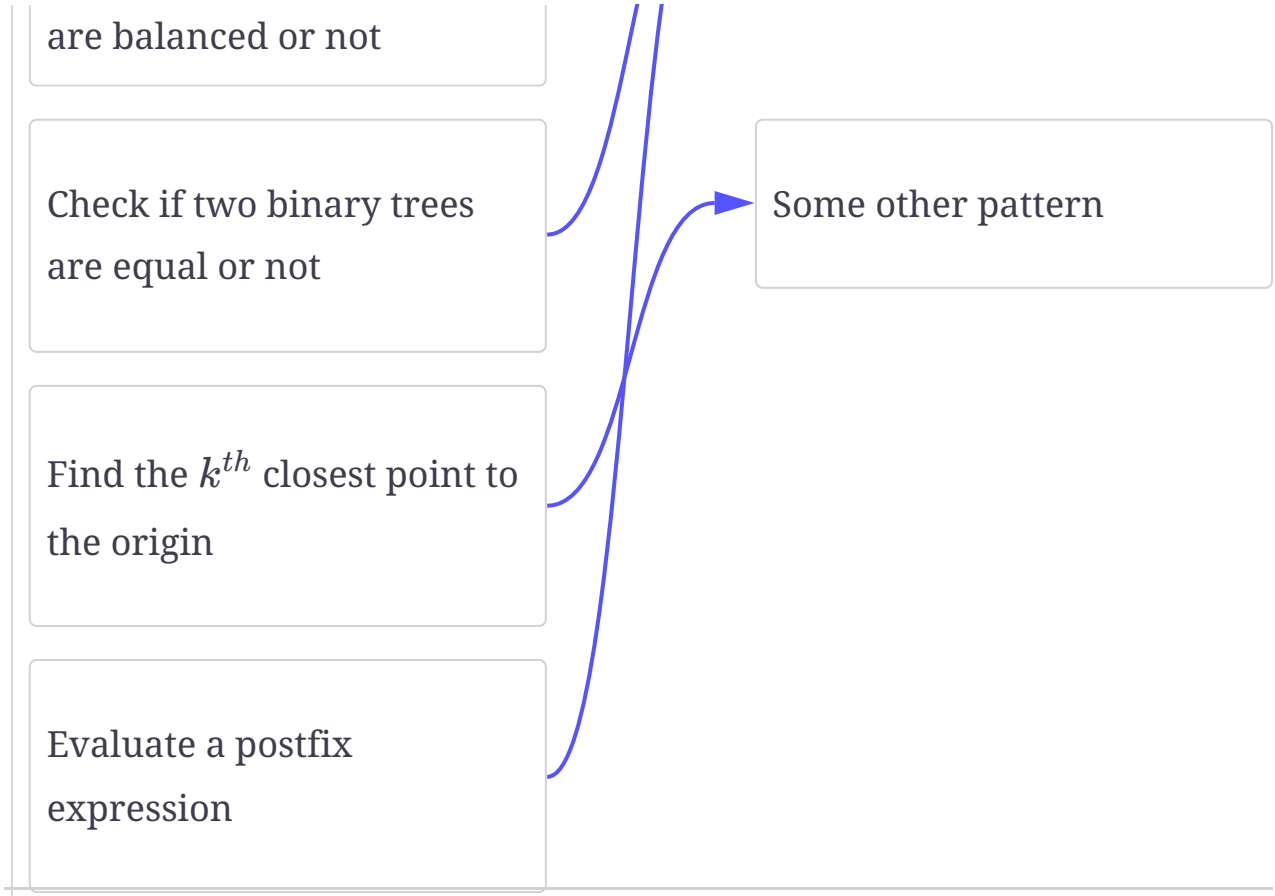
- **Checking code files:** Stacks are used in compilers to check if the parentheses in a code file are balanced.
- **Undo/redo functionality:** Stacks are commonly used to undo/redo things while editing.
- **Recursive call stacks:** The compiler internally calls stack itself when storing information about the recursion calls. Stacks facilitate recursive subroutines where the state of every call is gathered in a stack frame and then put on a stack.

Strategy time!

Match the problems that can be solved using the stacks pattern.

> **Note**: Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.





Reset | Show Solution | Submit

Back
Find All Possible Reci...

Next
Basic Calculator

Mark as Completed