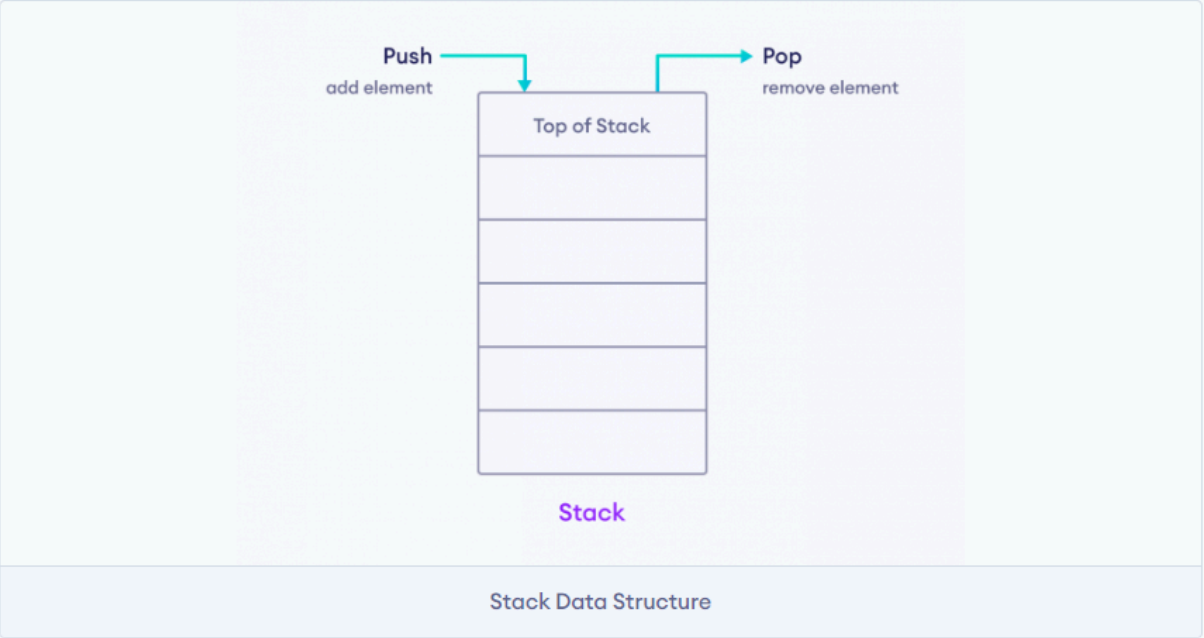
Stack

Overview:

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack is the first one to be removed. It operates on two main operations: push (adds an element to the top) and pop (removes the top element).



Implementing Stack in C++

Using `stack` from `<stack>` header

In C++, the standard library provides a convenient way to implement a stack using the stack container adapter. Here's how you can use it:

```
#include <iostream>
#include <stack> // include stack library
using namespace std;

int main() {
    stack<int> s; // declare a stack of integers

    // Push elements onto the stack
    s.push(10);
    s.push(20);
    s.push(30);
```

```
    // Check if stack is empty before popping
    while (!s.empty()) {
        // Access the top element
        cout << s.top() << " ";
        // Pop the top element
        s.pop();
    }

    return 0;
}
```

Operations on stack

1. `push(val)` : Adds value to the top of the stack.
2. `pop()` : Removes the top element from the stack.
3. `top()` : Accesses the top element without removing it.
4. `empty()` : Checks if the stack is empty.

Operation	Description
`push()`	adds an element into the stack
`pop()`	removes an element from the stack
`top()`	returns the element at the top of the stack
`size()`	returns the number of elements in the stack
`empty()`	returns `true` if the stack is empty

Example Walkthrough

Step-by-Step Execution:

1. Initialization :
   - `stack<int> s;` : Creates an empty stack of integers (s).

2. Push Operations :
   - `s.push(10);` : Pushes 10 onto the stack.
   - `s.push(20);` : Pushes 20 onto the stack.

- `s.push(30);` : Pushes 30 onto the stack.

3. Pop and Top Operations :
   - `s.top()` : Outputs 30 (top element), then removes it with s.pop() .
   - `s.top()` : Outputs 20 (new top element), then removes it with s.pop() .
   - `s.top()` : Outputs 10 (final top element), then removes it with s.pop() .

4. Empty Check :
   - `s.empty()` : Returns true when the stack is empty.

Advantages of Using stack

- Ease of Use : Provides a simple interface with essential stack operations.
- Efficiency : Operations like push, pop, top, and empty are efficient with constant time complexity.
- Safety : Handles memory management and resizing automatically.

Applications of Stack

- Function Call Stack : Used in programming languages to manage function calls and local variables.
- Expression Evaluation : Helps in evaluating postfix expressions (Reverse Polish Notation).
- Backtracking Algorithms : Used in algorithms like Depth-First Search (DFS) to track visited nodes.

Conclusion

Stacks are fundamental in computer science and programming, offering a straightforward way to manage data in a Last In, First Out manner. Using stack in C++ simplifies implementation and provides efficient operations for various applications.

This overview should help you understand and utilize stacks effectively in C++ programming. If you have specific questions or need further clarification, feel free to ask!