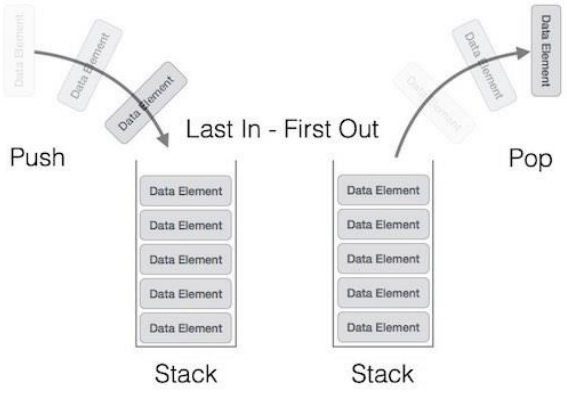
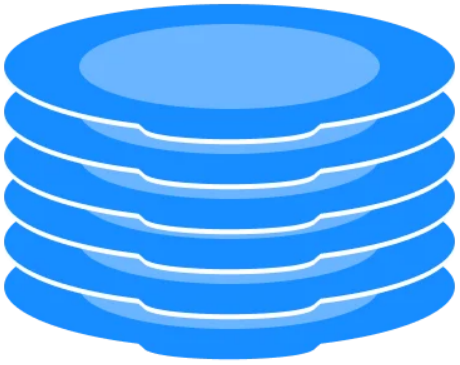
Stack

💡 A stack is a linear data structure that follows the principle of **Last In First Out (LIFO) / First In Last Out (FILO)**. ***Stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.***





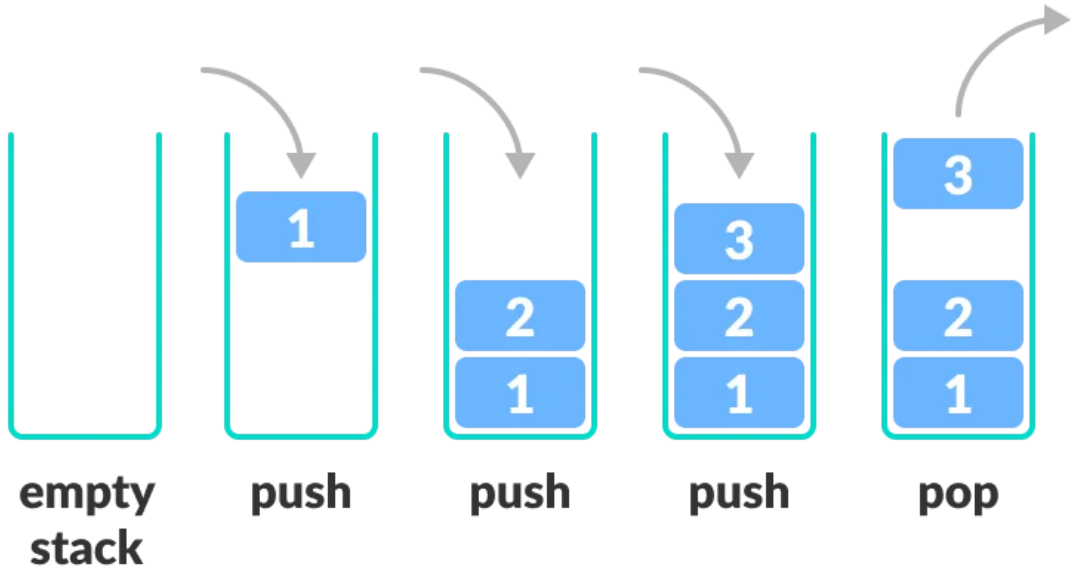
📌 A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing.

Type of Stacks

Fixed Size Stack: Fixed size stack has a fixed size and cannot grow or shrink dynamically. If the stack is full and an attempt is made to add an element to it, an overflow error occurs.

Dynamic Size Stack: A dynamic size stack can grow or shrink dynamically. When the stack is full, it automatically increases its size to accommodate the new element, and when the stack is empty, it decreases its size.

Basic Operation on Stack



push() → Inserting an element into the stack

1 – Checks if the stack is full.
2 – If the stack is full, produces an error and exit.
3 – If the stack is not full, increments top to point next empty space.
4 – Adds data element to the stack location, where top is pointing.
5 – Returns success.

pop() → removing an element from the stack

```
1 – Checks if the stack is empty.
2 – If the stack is empty, produces an error and exit.
3 – If the stack is not empty, accesses the data element at which top is pointing.
4 – Decreases the value of top by 1.
5 – Returns success.
```

peek() → returning the top element of the stack.

```
1. START
2. return the element at the top of the stack
3. END
```

isEmpty() → returning true if stack is empty else false.

```
1. START
2. If the top value is -1, the stack is empty. Return 1.
3. Otherwise, return 0.
4. END
```

size() → returning the size of stack.

isFull() → returning whether stack is full

```
1. START
2. If the size of the stack is equal to the top position of the stack, the stack is full. Return 1.
3. Otherwise, return 0.
4. END
```

Application of Stack Data Structure

Function calls and recursion: When a function is called, the current state of the program is pushed onto the stack. When the function returns, the state is popped from the stack to resume the previous function's execution.

Undo/Redo operations: The undo-redo feature in various applications uses stacks to keep track of the previous actions. Each time an action is performed, it is pushed

onto the stack. To undo the action, the top element of the stack is popped, and the reverse operation is performed.

Expression evaluation: Stack data structure is used to evaluate expressions in infix, postfix, and prefix notations. Operators and operands are pushed onto the stack, and operations are performed based on the stack's top elements.

Browser history: Web browsers use stacks to keep track of the web pages you visit. Each time you visit a new page, the URL is pushed onto the stack, and when you hit the back button, the previous URL is popped from the stack.

Balanced Parentheses: Stack data structure is used to check if parentheses are balanced or not. An opening parenthesis is pushed onto the stack, and a closing parenthesis is popped from the stack. If the stack is empty at the end of the expression, the parentheses are balanced.

Backtracking Algorithms: The backtracking algorithm uses stacks to keep track of the states of the problem-solving process. The current state is pushed onto the stack, and when the algorithm backtracks, the previous state is popped from the stack.

Application of Stack in real life

- CD/DVD stand.
- Stack of books in a book shop.
- Call center systems.
- Undo and Redo mechanism in text editors.
- The history of a web browser is stored in the form of a stack.
- Call logs, E-mails, and Google photos in any gallery are also stored in form of a stack.
- YouTube downloads and Notifications are also shown in LIFO format(the latest appears first).
- Allocation of memory by an operating system while executing a process.

Advantages

Easy implementation: Stack data structure is easy to implement using arrays or linked lists, and its operations

Disadvantage

Limited capacity: Stack data structure has a limited capacity as it can only hold a fixed number of elements. If the

are simple to understand and implement.

Efficient memory utilization: Stack uses a contiguous block of memory, making it more efficient in memory utilization as compared to other data structures.

Fast access time: Stack data structure provides fast access time for adding and removing elements as the elements are added and removed from the top of the stack.

Helps in function calls: Stack data structure is used to store function calls and their states, which helps in the efficient implementation of recursive function calls.

Supports backtracking: Stack data structure supports backtracking algorithms, which are used in problem-solving to explore all possible solutions by storing the previous states.

Used in Compiler Design: Stack data structure is used in compiler design for parsing and syntax analysis of programming languages.

Enables undo/redo operations: Stack data structure is used to enable undo and redo operations in various applications like text editors, graphic design tools, and software development environments.

stack becomes full, adding new elements may result in stack overflow, leading to the loss of data.

No random access: Stack data structure does not allow for random access to its elements, and it only allows for adding and removing elements from the top of the stack. To access an element in the middle of the stack, all the elements above it must be removed.

Memory management: Stack data structure uses a contiguous block of memory, which can result in memory fragmentation if elements are added and removed frequently.

Not suitable for certain applications: Stack data structure is not suitable for applications that require accessing elements in the middle of the stack, like searching or sorting algorithms.

Stack overflow and underflow: Stack data structure can result in stack overflow if too many elements are pushed onto the stack, and it can result in stack underflow if too many elements are popped from the stack.

Recursive function calls limitations: While stack data structure supports recursive function calls, too many recursive function calls can lead to stack overflow, resulting in the termination of the program.

📚 References

GeeksforGeeks | A computer science portal for geeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive

https://www.geeksforgeeks.org/

Online Tutorials, Courses, and eBooks Library | Tutorialspoint

Tutorialspoint is an online learning platform providing free tutorials, paid premium courses, and eBooks. Learn the latest technologies and programming languages C, C++, Java, Python, PHP, Machine Learning, data science, AI, and more.

https://www.tutorialspoint.com

Programiz: Learn to Code for Free

Learn to code in Python, C/C++, Java, and other popular programming languages with our easy to follow tutorials, examples, online compiler and references.

https://www.programiz.com

Tutorials - Javatpoint

Tutorials, Free Online Tutorials, Javatpoint provides tutorials and interview questions of all technology like java tutorial, android, java frameworks, javascript, ajax, core java, sql, python, php, c language etc. for beginners and professionals.

https://www.javatpoint.com/

✍🏻 Author → Serhat Kumas

https://www.linkedin.com/in/serhatkumas/

SerhatKumas - Overview

Computer engineering student who loves coding in different fields instead of focusing on a one spesific area. - SerhatKumas

https://github.com/SerhatKumas