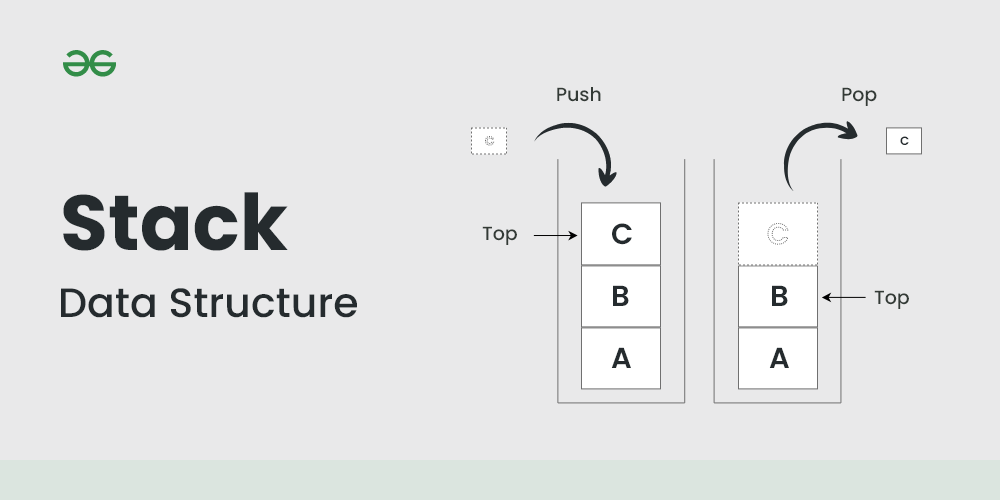
## Stack Data Structure:

A stack is a linear data structure that follows the principle of Last In First Out (LIFO). This means the last element inserted inside the stack is removed first.

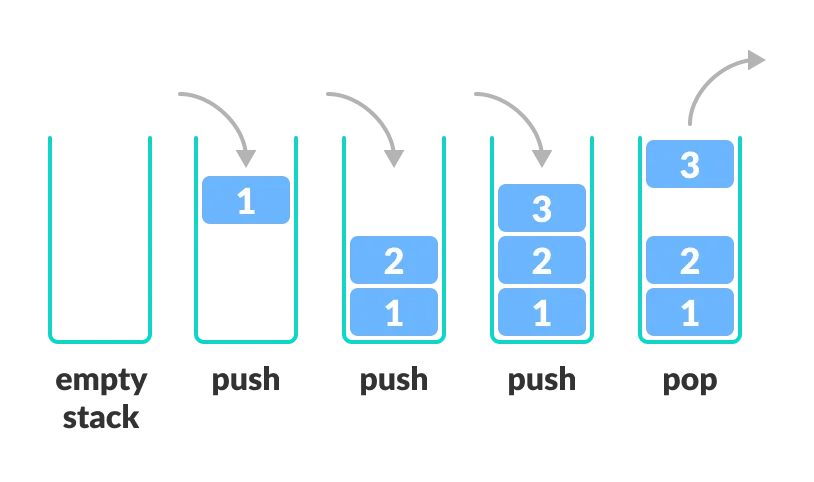
You can think of the stack data structure as the pile of plates on top of another.



## **LIFO Principle of Stack**

In programming terms, putting an item on top of the stack is called push and removing an item is called pop.

In the below image, although item 3 was kept last, it was removed first. This is exactly how the LIFO (Last In First Out) Principle works.



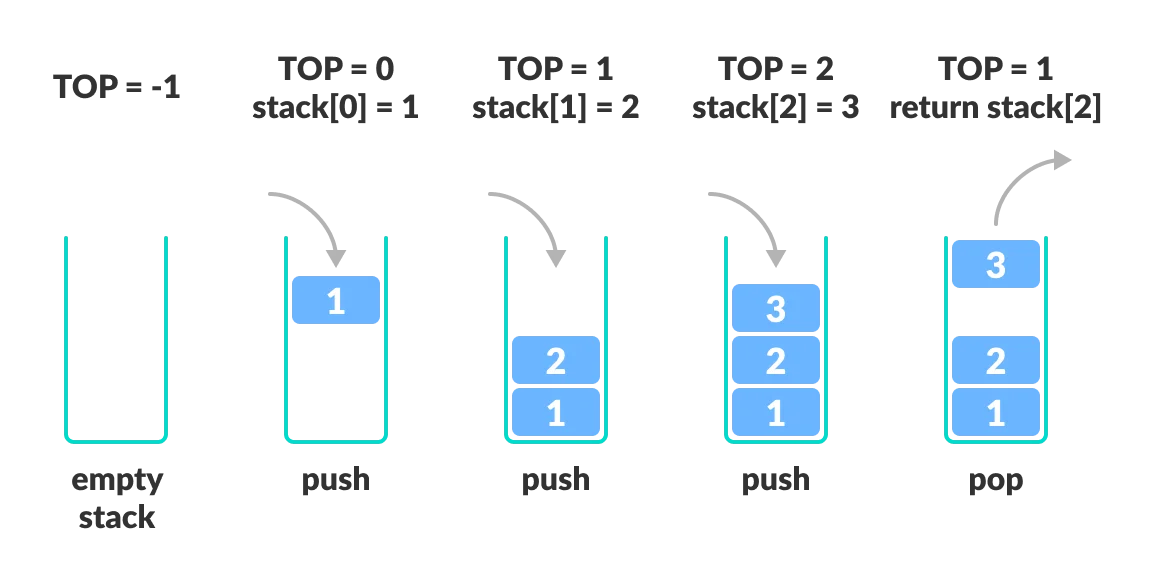
There are some basic operations that allow us to perform different actions on a stack.

* Push: Add an element to the top of a stack
* Pop: Remove an element from the top of a stack
* IsEmpty: Check if the stack is empty
* IsFull: Check if the stack is full
* Peek: Get the value of the top element without removing it

## **Working of Stack Data Structure**

The operations work as follows:

1. A pointer called TOP is used to keep track of the top element in the stack.
2. When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing TOP == -1.
3. On pushing an element, we increase the value of TOP and place the new element in the position pointed to by TOP.
4. On popping an element, we return the element pointed to by TOP and reduce its value.
5. Before pushing, we check if the stack is already full
6. Before popping, we check if the stack is already empty



**Time Complexity**

| **Operations** | **Complexity** |
| --- | --- |
| push() | O(1) |
| pop() | O(1) |
| isEmpty() | O(1) |
| size() | O(1) |

There are two ways to implement a stack –

* Using array
* Using linked list

#### What is #include <stdio.h> ?

In C and C++ programming, #include <stdio.h> is a preprocessor directive that is used to include the Standard Input/Output library in your program. This **library provides functions for performing input and output operations**, such as reading from and writing to the console, files, and other input/output devices.

When you include stdio.h in your code, you gain access to functions like **printf, scanf, fopen, fclose,** and others, as well as predefined constants like stdin, stdout, and stderr. These functions and constants are essential for performing basic input and output operations in C and C++ programs.

## Stack Using array C++ :

#include <iostream>

using namespace std;

int stack[100], n=100, top=-1;

void push(int val) {

if(top>=n-1)

cout<<"Stack Overflow"<<endl;

else {

top++;

stack[top]=val;

}

}

void pop() {

if(top<=-1)

cout<<"Stack Underflow"<<endl;

else {

cout<<"The popped element is "<< stack[top] <<endl;

top--;

}

}

void display() {

if(top>=0) {

cout<<"Stack elements are:";

for(int i=top; i>=0; i--)

cout<<stack[i]<<" ";

cout<<endl;

} else

cout<<"Stack is empty";

}

int main() {

int ch, val;

cout<<"\n1.PUSH 2.POP 3.SHOW\_TOP 4.DISPLAY\_STACK 5.EXIT\n";

do {

cout<<"Enter choice: "<<endl;

cin>>ch;

switch(ch) {

case 1: {

cout<<"Enter value to be pushed:"<<endl;

cin>>val;

push(val);

break;

}

case 2: {

pop();

break;

}

case 3: {

display();

break;

}

case 4: {

cout<<"Exit"<<endl;

break;

}

default: {

cout<<"Invalid Choice"<<endl;

}

}

}while(ch!=4);

return 0;

}

#### Stack Time Complexity

For the array-based implementation of a stack, the push and pop operations take constant time, i.e. O(1).

## **Applications of Stack Data Structure**

Although stack is a simple data structure to implement, it is very powerful. The most common uses of a stack are:

* Expression Evaluation and Conversion
* Backtracking
* Function Call
* Parentheses Checking
* String Reversal
* Syntax Parsing
* Memory Management

## 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 of Stack:

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

### Disadvantages of Stack:

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

### **What is a doubly-linked list used for?**

## It is utilized in navigation systems that require both forward and backward navigation. The browser operates a back and forward button to implement backward and forward navigation of visited web pages. It's also used to represent a standard deck of playing cards.

### **What is the difference between stack and linked list?**

## A stack is an abstract data type representing a collection of components that may be manipulated using push and pop operations. On the other hand, a linked list is a linear collection of data components whose order is not determined by their memory location. This is the primary distinction between a stack and a linked list.

#### Can we create a dynamic grow stack using arrays?

Yes, you can create a dynamically growing stack using an array in C++.

#include <iostream>

using namespace std;

class DynamicStack {

private:

int\* stackArray;

int maxSize;

int top;

public:

DynamicStack() {

maxSize = 10; // Initial capacity of the stack (you can change this as needed).

stackArray = new int[maxSize];

top = -1; // Initialize the top of the stack to -1 (empty stack).

}

~DynamicStack() {

delete[] stackArray;

}

bool isEmpty() {

return (top == -1);

}

void push(int value) {

if (top == maxSize - 1) {

// If the stack is full, double its capacity and copy elements.

int newMaxSize = 2 \* maxSize;

int\* newStackArray = new int[newMaxSize];

for (int i = 0; i <= top; i++) {

newStackArray[i] = stackArray[i];

}

delete[] stackArray;

stackArray = newStackArray;

maxSize = newMaxSize;

}

stackArray[++top] = value;

}

void pop() {

if (isEmpty()) {

cout << "Stack is empty. Cannot pop." << endl;

return;

}

cout << "Popped: " << stackArray[top--] << endl;

}

int peek() {

if (isEmpty()) {

cout << "Stack is empty. Peek failed." << endl;

return -1; // Return a default value for an empty stack.

}

return stackArray[top];

}

};

int main() {

DynamicStack myStack;

myStack.push(1);

myStack.push(2);

myStack.push(3);

cout << "Top element: " << myStack.peek() << endl;

myStack.push(4); // This will trigger a resize of the stack.

cout << "Top element after resize: " << myStack.peek() << endl;

return 0;

}

## Reference:

* <https://www.simplilearn.com/tutorials/data-structure-tutorial/stacks-in-data-structures>
* <https://www.geeksforgeeks.org/stack-data-structure/>
* <https://www.codewhoop.com/stack/stack-using-array.html>
* <https://www.geeksforgeeks.org/implement-a-stack-using-singly-linked-list/>
* <https://www.tutorialspoint.com/cplusplus-program-to-implement-stack-using-linked-list>
* <https://dotnettutorials.net/course/data-structures-and-algorithms-for-beginners-and-professionals/>
* https://www.w3resource.com/cpp-exercises/stack/index.php
* <https://www.javatpoint.com/ds-linked-list-implementation-of-stack>
* Stack using C# -
* https://www.geeksforgeeks.org/c-sharp-stack-with-examples/
* <https://www.programiz.com/csharp-programming/stack>
* <https://www.tutorialsteacher.com/csharp/csharp-stack>
* https://igotanoffer.com/blogs/tech/stacks-and-queues-interview-questions#easy