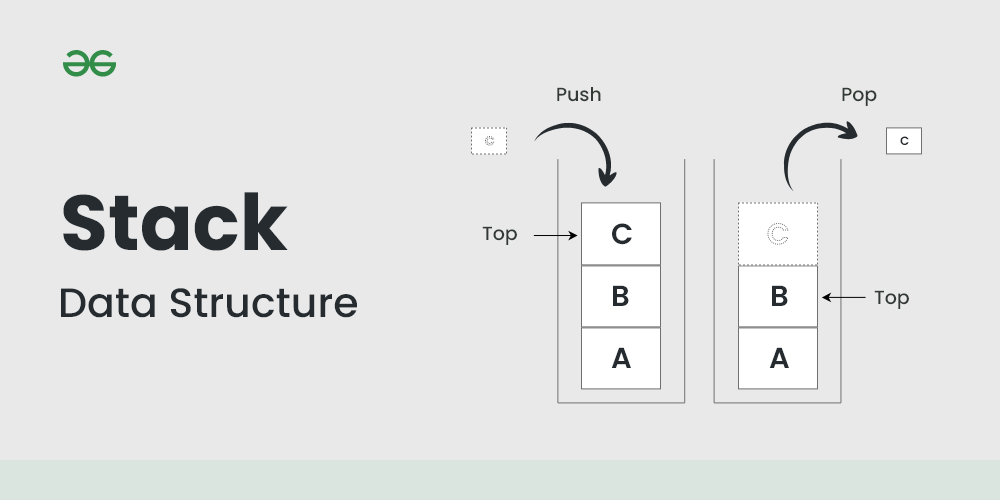
**Структура данных стека**

Стек — **это** линейная структура данных, которая следует определенному порядку выполнения операций. Порядок может быть **LIFO (Last In First Out)** или **FILO (First In Last Out)**. **LIFO** подразумевает, что элемент, вставленный последним, выводится первым, а **FILO** подразумевает, что элемент, вставленный первым, выводится последним.

Это похоже на стопку тарелок, где последняя добавленная тарелка — первая, которую нужно убрать. **Подумайте об этом так:**

* Добавление элемента в стек похоже на добавление новой тарелки сверху.
* При извлечении элемента верхняя пластина удаляется из стопки.



## Ключевые операции над структурами данных стека

* **Push**: добавляет элемент наверх стека.
* **Pop**: Удаляет верхний элемент из стека.
* **Peek**: возвращает верхний элемент, не удаляя его.
* **IsEmpty**: проверяет, пуст ли стек.
* **IsFull**: проверяет, заполнен ли стек (в случае массивов фиксированного размера).

## Применение стековых структур данных

* Рекурсия
* Оценка и анализ выражений
* Поиск в глубину (DFS)
* Отменить/Повторить операции
* История браузера
* Вызовы функций

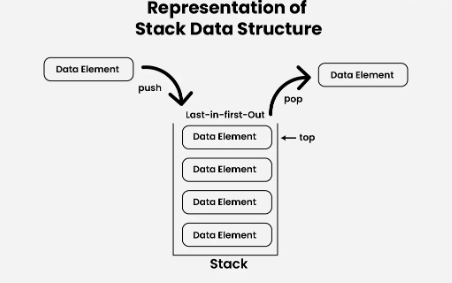
Для реализации стека необходимо поддерживать **указатель на вершину стека**, которая является последним вставляемым элементом, поскольку **мы можем получить доступ только к элементам, находящимся на вершине стека.**

### **Принцип LIFO (последним пришел — первым ушел) в структуре данных стека:**

Эта стратегия гласит, что элемент, который был вставлен последним, выйдет первым. В качестве примера из реальной жизни можно взять стопку тарелок, поставленных друг на друга. Тарелка, которую мы ставим последней, находится наверху, и поскольку мы убираем тарелку, которая находится сверху, мы можем сказать, что тарелка, которая была поставлена ​​последней, выйдет первой.

## Представление структуры данных стека:

Стек следует принципу LIFO (последним пришел — первым ушел), то есть элемент, помещенный последним, извлекается первым.



## **Типы структуры данных стека:**

* **Стек фиксированного размера**: как следует из названия, стек фиксированного размера имеет фиксированный размер и не может динамически увеличиваться или уменьшаться. Если стек полон и делается попытка добавить в него элемент, возникает ошибка переполнения. Если стек пуст и делается попытка удалить из него элемент, возникает ошибка потери емкости.
* **Динамический размер стека**: динамический размер стека может увеличиваться или уменьшаться динамически. Когда стек полон, он автоматически увеличивает свой размер, чтобы вместить новый элемент, а когда стек пуст, он уменьшает свой размер. Этот тип стека реализован с использованием связанного списка, так как он позволяет легко изменять размер стека.

## **Основные операции со структурой данных** стека:

Для того чтобы производить манипуляции в стеке, нам предоставлены определенные операции.

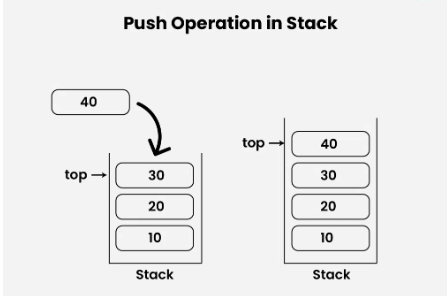
* **push()** для вставки элемента в стек
* **pop()** для удаления элемента из стека
* **top()** Возвращает верхний элемент стека.
* **isEmpty()** возвращает true, если стек пуст, в противном случае — false.
* **isFull()** возвращает true, если стек полон, в противном случае — false.

### **Операция Push в структуре данных стека:**

Добавляет элемент в стек. Если стек полон, то говорят, что произошло **состояние переполнения.**

**Алгоритм операции Push:**

* Перед помещением элемента в стек мы проверяем, **заполнен** ли стек.
* Если стек заполнен **(top == capacity-1)**, то **происходит переполнение стека**, и мы не можем вставить элемент в стек.
* В противном случае мы увеличиваем значение top на 1 **(top = top + 1)** и новое значение вставляется в **верхнюю позицию**.
* Элементы можно помещать в стек до тех пор, пока не будет достигнута его **вместимость.**

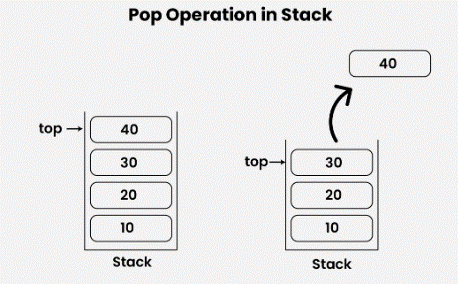


### **Операция извлечения в структуре данных стека:**

Удаляет элемент из стека. Элементы выталкиваются в обратном порядке, в котором они были вставлены. Если стек пуст, то говорят, что это **состояние Underflow.**

**Алгоритм операции «Поп»:**

* Перед извлечением элемента из стека мы проверяем, **пуст** ли стек.
* Если стек пуст (top == -1), то **стек переполняется**, и мы не можем удалить ни один элемент из стека.
* В противном случае мы сохраняем значение top, уменьшаем значение top на 1 **(top = top – 1)** и возвращаем сохраненное значение top.

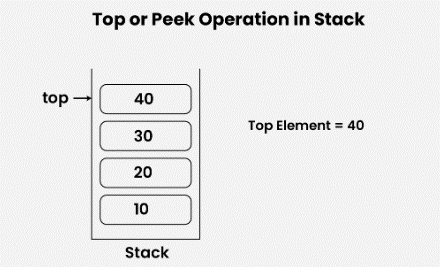


### **Операция «Top» или «Peek» в структуре данных стека:**

Возвращает верхний элемент стека.

**Алгоритм для верхней операции:**

* Прежде чем вернуть верхний элемент из стека, мы проверяем, пуст ли стек.
* Если стек пуст (вершина == -1), мы просто выводим «Стек пуст».
* В противном случае мы возвращаем элемент, хранящийся по **индексу = top**.

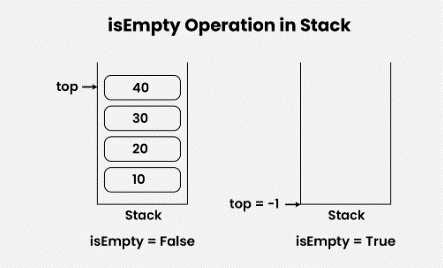


### **Операция isEmpty в структуре данных стека:**

Возвращает true, если стек пуст, в противном случае — false.

**Алгоритм для операции isEmpty**:

* Проверьте значение **top** в стеке.
* Если **(top == -1)**, то стек **пуст**, поэтому возвращаем **true**.
* В противном случае стек не пуст, поэтому возвращаем **false**.



### Операция isFull в **структуре данных стека**:

Возвращает true, если стек полон, в противном случае false.

**Алгоритм для операции isFull:**

* Проверьте значение **top** в стеке.
* Если **(top == capacity-1),** то стек **заполнен**, поэтому возвращаем **true**.
* В противном случае стек не заполнен, поэтому возвращаем **false**.

## 

## Реализация **структуры данных** стека:

Базовые операции, которые можно выполнять в стеке, включают push, pop и peek. Существует два способа реализовать стек –

* Использование массива
* Использование связанного списка

В реализации на основе массива операция push реализуется путем увеличения индекса верхнего элемента и сохранения нового элемента по этому индексу. Операция pop реализуется путем возврата значения, сохраненного по верхнему индексу, и последующего уменьшения индекса верхнего элемента.

В реализации на основе связанного списка операция push реализуется путем создания нового узла с новым элементом и установки следующего указателя текущего верхнего узла на новый узел. Операция pop реализуется путем установки следующего указателя текущего верхнего узла на следующий узел и возврата значения текущего верхнего узла.

### **Реализация структуры данных стека с использованием массива:**

Рекомендуемая проблема

[Реализовать стек с использованием массива](https://www.geeksforgeeks.org/problems/implement-stack-using-array/1/?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card)

*// C# program to implement basic stack*

*// operations*

**using** **System**;

**namespace** **ImplementStack** {

**class** **Stack** {

**private** int[] ele;

**private** int top;

**private** int max;

**public** Stack(int size)

{

ele = **new** int[size]; *// Maximum size of Stack*

top = -1;

max = size;

}

**public** **void** push(int item)

{

**if** (top == max - 1) {

Console.WriteLine("Stack Overflow");

**return**;

}

**else** {

ele[++top] = item;

}

}

**public** int pop()

{

**if** (top == -1) {

Console.WriteLine("Stack is Empty");

**return** -1;

}

**else** {

Console.WriteLine("{0} popped from stack ",

ele[top]);

**return** ele[top--];

}

}

**public** int peek()

{

**if** (top == -1) {

Console.WriteLine("Stack is Empty");

**return** -1;

}

**else** {

Console.WriteLine("{0} popped from stack ",

ele[top]);

**return** ele[top];

}

}

**public** **void** printStack()

{

**if** (top == -1) {

Console.WriteLine("Stack is Empty");

**return**;

}

**else** {

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

Console.WriteLine("{0} pushed into stack",

ele[i]);

}

}

}

}

*// Driver program to test above functions*

**class** **Program** {

**static** **void** Main()

{

Stack p = **new** Stack(5);

p.push(10);

p.push(20);

p.push(30);

p.printStack();

p.pop();

}

}

}

**Выход**

10 помещено в стек

20 помещено в стек

30 помещено в стек

30 Вытащено из стека

Верхний элемент: 20

Элементов в стеке: 20 10

**Преимущества реализации массива:**

* Легко реализовать.
* Память экономится, так как указатели не используются.

**Недостатки реализации массива:**

* Он не является динамическим, то есть не увеличивается и не уменьшается в зависимости от потребностей во время выполнения. [Но в случае динамических массивов, таких как vector в C++, list в Python, ArrayList в Java, стеки также могут увеличиваться и уменьшаться при реализации массива].
* Общий размер стека должен быть определен заранее.

### **Реализация структуры данных стека с использованием связанного списка:**

*// C# Code for Linked List Implementation*

**using** **System**;

**public** **class** **StackAsLinkedList** {

StackNode root;

**public** **class** **StackNode** {

**public** int data;

**public** StackNode next;

**public** StackNode(int data) { **this**.data = data; }

}

**public** bool isEmpty()

{

**if** (root == **null**) {

**return** **true**;

}

**else**

**return** **false**;

}

**public** **void** push(int data)

{

StackNode newNode = **new** StackNode(data);

**if** (root == **null**) {

root = newNode;

}

**else** {

StackNode temp = root;

root = newNode;

newNode.next = temp;

}

Console.WriteLine(data + " pushed to stack");

}

**public** int pop()

{

int popped = int.MinValue;

**if** (root == **null**) {

Console.WriteLine("Stack is Empty");

}

**else** {

popped = root.data;

root = root.next;

}

**return** popped;

}

**public** int peek()

{

**if** (root == **null**) {

Console.WriteLine("Stack is empty");

**return** int.MinValue;

}

**else** {

**return** root.data;

}

}

*// Driver code*

**public** **static** **void** Main(String[] args)

{

StackAsLinkedList sll = **new** StackAsLinkedList();

sll.push(10);

sll.push(20);

sll.push(30);

Console.WriteLine(sll.pop() + " popped from stack");

Console.WriteLine("Top element is " + sll.peek());

}

}

*/\* This code contributed by PrinciRaj1992 \*/*

**Выход**

10 помещено в стек

20 помещено в стек

30 помещено в стек

30 выскочило из стека

Верхний элемент — 20

Элементов в стеке: 20 10

**Преимущества внедрения связанного списка:**

* Реализация стека в виде связанного списка может увеличиваться и уменьшаться в зависимости от потребностей во время выполнения.
* Он используется во многих виртуальных машинах, таких как JVM.

**Недостатки реализации связанного списка:**

* Требует дополнительной памяти из-за использования указателей.
* Произвольный доступ в стеке невозможен.

## **Анализ сложности операций над структурой данных стека:**

| **Операции** | **Сложность времени** | **Сложность пространства** |
| --- | --- | --- |
| **push()** | О(1) | О(1) |
| **pop()** | О(1) | О(1) |
| **top() or peek()** | О(1) | О(1) |
| **isEmpty()** | О(1) | О(1) |
| **isFull()** | О(1) | О(1) |

## Преимущества стековой **структуры данных**:

* **Простота:** Стеки представляют собой простую и понятную структуру данных, что делает их пригодными для широкого спектра приложений.
* **Эффективность:** операции занесения и извлечения данных из стека могут выполняться за постоянное время **(O(1))**, обеспечивая эффективный доступ к данным.
* **Last-in, First-out (LIFO):** Стеки следуют принципу LIFO, гарантируя, что последний элемент, добавленный в стек, будет первым удаленным. Такое поведение полезно во многих сценариях, таких как вызовы функций и оценка выражений.
* **Ограниченное использование памяти:** стеки должны хранить только элементы, которые были в них помещены, что делает их более эффективными с точки зрения использования памяти по сравнению с другими структурами данных.

## Недостатки стековой **структуры данных**:

* **Ограниченный доступ:** доступ к элементам в стеке возможен только сверху, что затрудняет извлечение или изменение элементов в середине стека.
* **Возможность переполнения:** если в стек помещается больше элементов, чем он может вместить, возникнет ошибка переполнения, что приведет к потере данных.
* **Не подходит для произвольного доступа:** стеки не допускают произвольного доступа к элементам, что делает их непригодными для приложений, где доступ к элементам должен осуществляться в определенном порядке.
* **Ограниченная емкость:** стеки имеют фиксированную емкость, что может стать ограничением, если количество элементов, которые необходимо сохранить, неизвестно или сильно варьируется.

## **Применение структуры данных стека:**

* Преобразование инфикса в постфикс /префикс
* Функции возврата и отмены действий имеются во многих редакторах, например, в Photoshop.
* Функции прямого и обратного перехода в веб-браузерах
* В управлении памятью любой современный компьютер использует стек в качестве основного управления для выполнения задачи. Каждая программа, которая выполняется в компьютерной системе, имеет свои собственные распределения памяти.
* Стек также помогает в реализации вызова функций в компьютерах. Последняя вызванная функция всегда завершается первой.