Data Sturctures

Stack

Agenda

- What is Stack?
- Stack ADT
- Stack Tracing
- Stack Implementation
- Applications on Stack
- Expressions Conversion
- Simple Calculator Project Hands on

What is the Stack?

Stack : logical linear data structure that follows the Last-In-First-Out (***LIFO***) or First-In-Last-Out (***FILO***) order principle.

- *Logical* ⟶ Last-In-First-Out **(LIFO)** or First-In-Last-Out **(FILO)** logic
- *Linear* ⟶ Sequential Order

Ex: A pile of plates in a restaurant

- **The first** plate is placed, **the last** plate is washed

In C++: Main Function

- **The first** function called, **the last** function executed

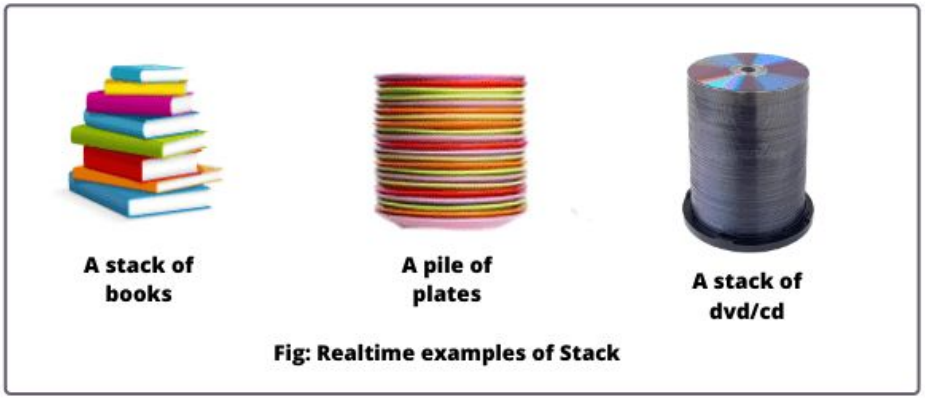


Fig: Realtime examples of Stack

src

Stack ADT

Main functionalities of Stack:

- **push**(*val*) → Inserts *val* at the top.
- **pop**() → Removes and returns the top element.

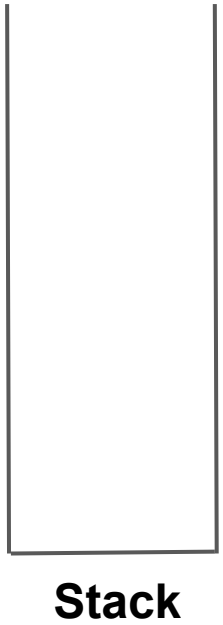
Another Useful functionalities:

- **peek**() → Returns the top element without removing it.
- **isEmpty**() → Checks if the stack is empty.
- **isFull**() → As a helper function in array-based.
- **size**() → Return size of stack.

```
template<class _Tp>
class Stack{
public:
//Main Operations
    void push(_Tp val);
    _Tp pop();

//Auxiliary Operations
    _Tp peek(); //top in STLs
    bool isEmpty();
    bool isFull();
    void clear();
    int size();
};
```

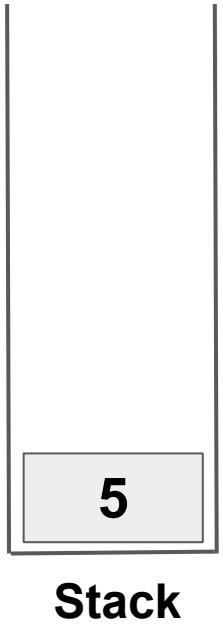
Stack Tracing

- **isEmpty**() → *True*
- **size**() → *0*



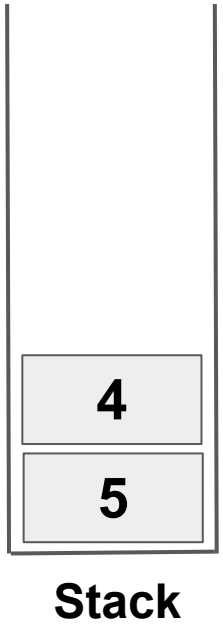
Stack Tracing

- **push**(**5**)
- **peek**() →*5*
- **isEmpty**() →*False*
- **size**() →*1*



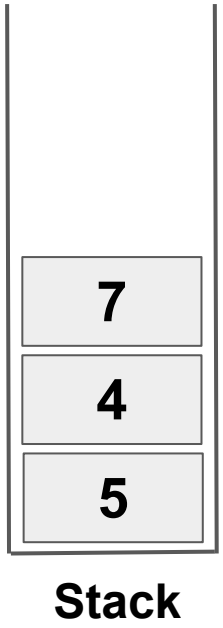
Stack Tracing

- **push**(**4**)
- **peek**() →*4*
- **isEmpty**() →*False*
- **size**() →*2*



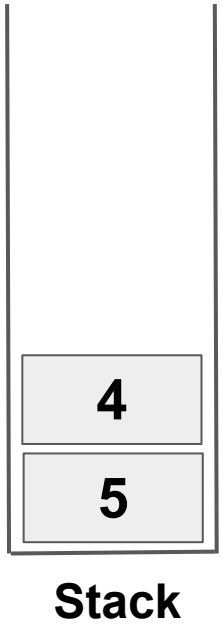
Stack Tracing

- **push**(7)
- **peek**() → *7*
- **isEmpty**() → *False*
- **size**() → *3*



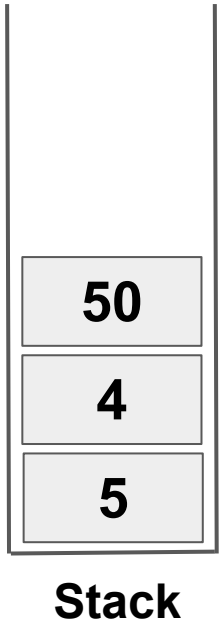
Stack Tracing

- **pop**() → *7*
- **peek**() → *4*
- **isEmpty**() → *False*
- **size**() → *2*



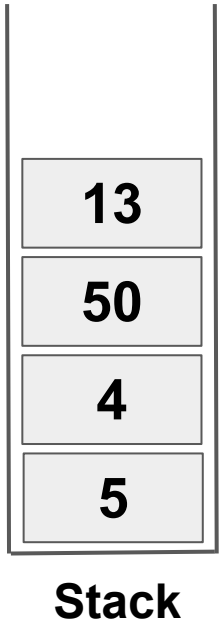
Stack Tracing

- **push**(**50**)
- **peek**() →*50*
- **isEmpty**() →*False*
- **size**() →*3*



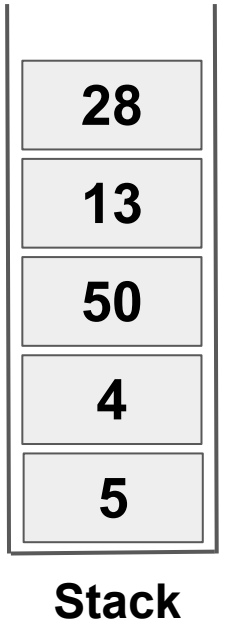
Stack Tracing

- **push**(**13**)
- **peek**() →*13*
- **isEmpty**() →*False*
- **size**() →*4*



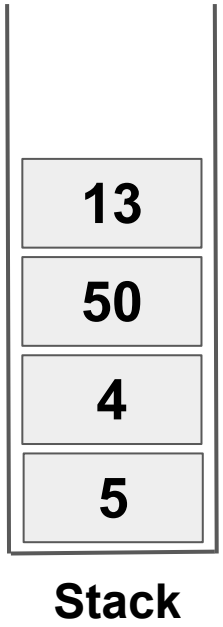
Stack Tracing

- **push**(**28**)
- **peek**() →*28*
- **isEmpty**() →*False*
- **size**() →*5*



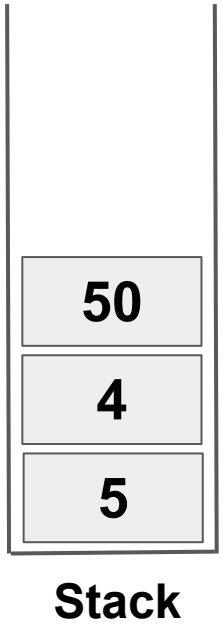
Stack Tracing

- **pop**() → *28*
- **peek**() → *13*
- **isEmpty**() → *False*
- **size**() → *4*



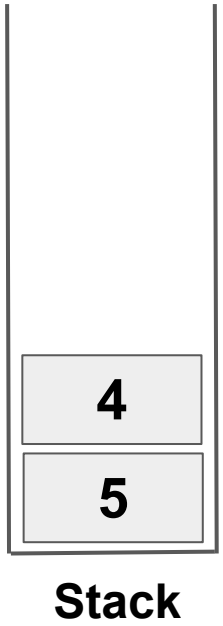
Stack Tracing

- **pop**() → *13*
- **peek**() → *50*
- **isEmpty**() → *False*
- **size**() → *3*



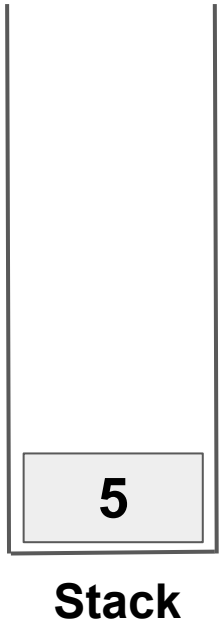
Stack Tracing

- **pop**() →*50*
- **peek**() →*4*
- **isEmpty**() →*False*
- **size**() →*2*



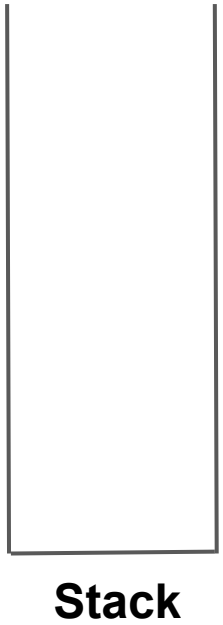
Stack Tracing

- **pop**() →*4*
- **peek**() →*5*
- **isEmpty**() →*False*
- **size**() →*1*



Stack Tracing

- **isEmpty**() → *True*
- **size**() → *0*



Stack Implementation

- Challenge yourself to implement it , take 10 minutes for thinking

  implement any code to satisfy LIFO

Stack Applications

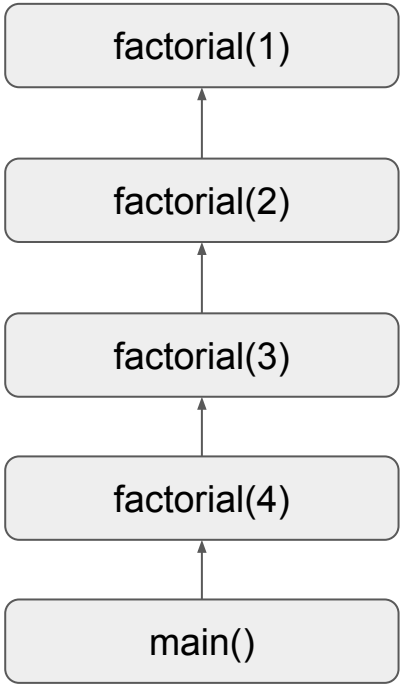
```
int factorial(int n){
    return n == 1 ? 1 : n * factorial(n - 1);
}
```

```
static void F3()
{
    StackTrace stackTrace = new StackTrace();
    StackFrame[] frames = stackTrace.GetFrames();
    for (int i = 0; i < frames?.Length; i++)
        Console.WriteLine(frames[i].GetMethod());
}
1 reference
static void F2() { F3(); }
1 reference
static void F1() { F2(); }

0 references
static void Main()
{
    F1();
```

```
Microsoft Visual Studio Debu
Void F3()
Void F2()
Void F1()
Void Main()
```

```
static void F3()
{
    int zero = 0;
    Console.WriteLine(2/zero);
}
1 reference
static void F2() { F3(); }
1 reference
static void F1() { F2(); }

0 references
static void Main()
{
    F1();
```

```
Microsoft Visual Studio Debu
Unhandled exception. System.Divide
   at Learning.Program.F3() in D:\?
   at Learning.Program.F2() in D:\?
   at Learning.Program.F1() in D:\?
   at Learning.Program.Main() in D:
```



Functions Call Stack

Stack Applications

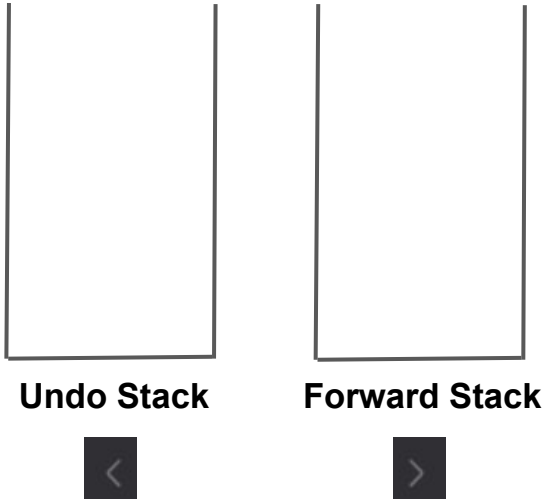
```cpp
//This function to print vectors of any type
template<class _Tp>
void print(const vector<_Tp>& container){
    for(const _Tp& element : container)
        cout << element << ' ';
    cout << endl;
}

//This is a general function to reverse vectors of any type
template<class _Tp>
void reverseVector(vector<_Tp>& v){
    //Put the content of vector in stack to apply (LIFO) principle
    stack<_Tp> stk;

    //Fill the stack
    for(const _Tp& i : v)
        stk.push(i);

    //Again refill the vector
    for(int i = 0; stk.size() and i < (int) v.size(); ++i , stk.pop())
        v[i] = stk.top();
}

void solve(){
    vector<int> v = {1, 2, 3, 4, 5};
    reverseVector(v);
    print(v);
}
```

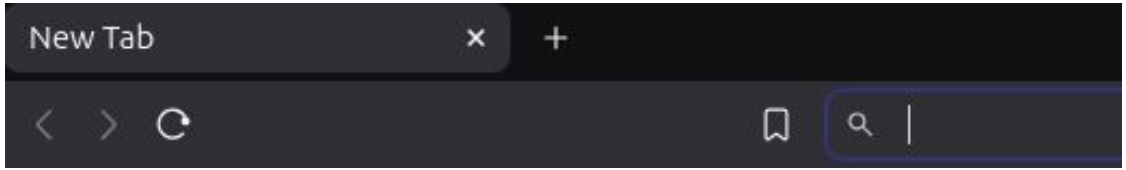
Reversing Processes

```
anaselwkel@ana
5 4 3 2 1
```

Stack Applications

- **Open New Tab in Browser**
  - **Go to new page ?**
    Search about it
    put this in stack
    show the top of undo stack

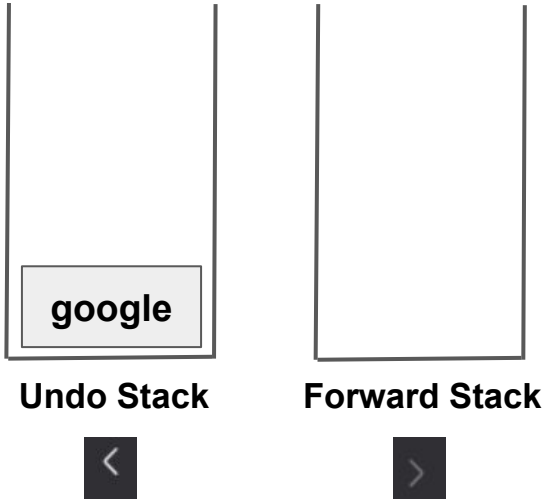


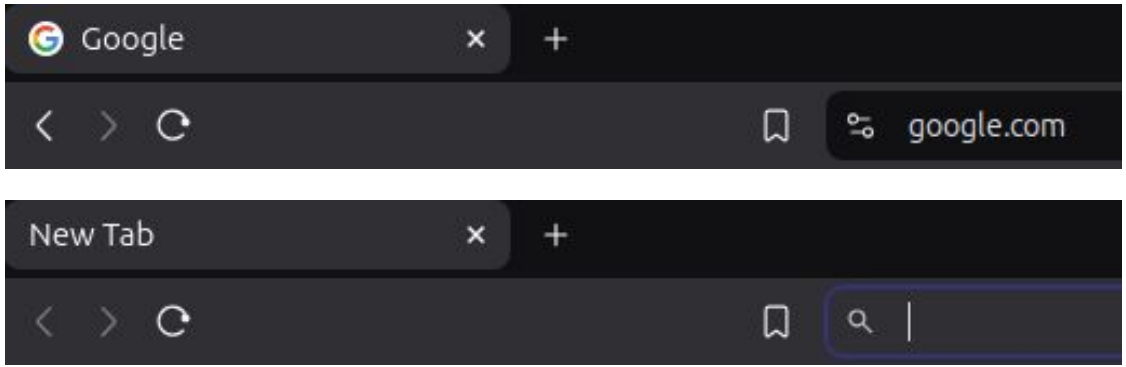


Undo Operations

Stack Applications

- Go to Google

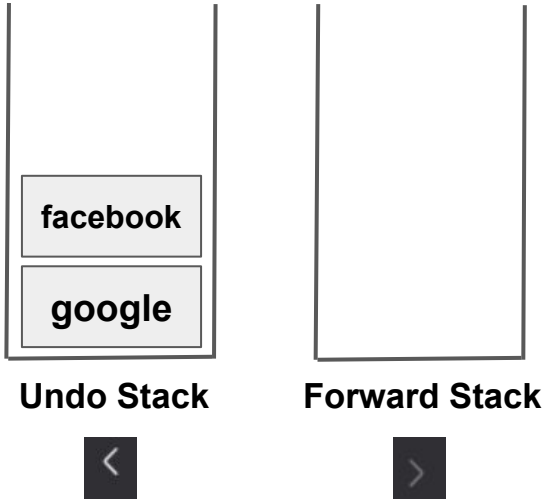




Undo Operations

Stack Applications

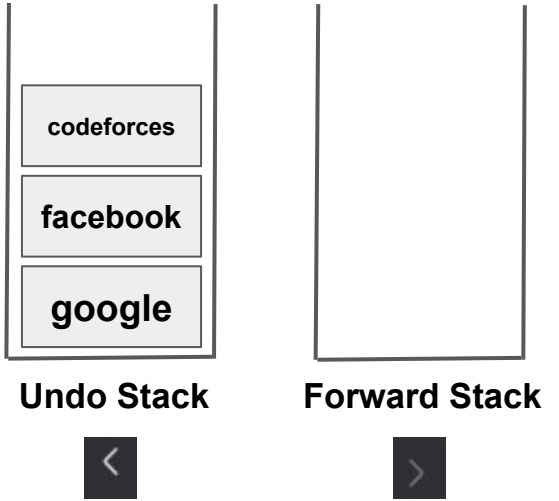
- Go to Facebook

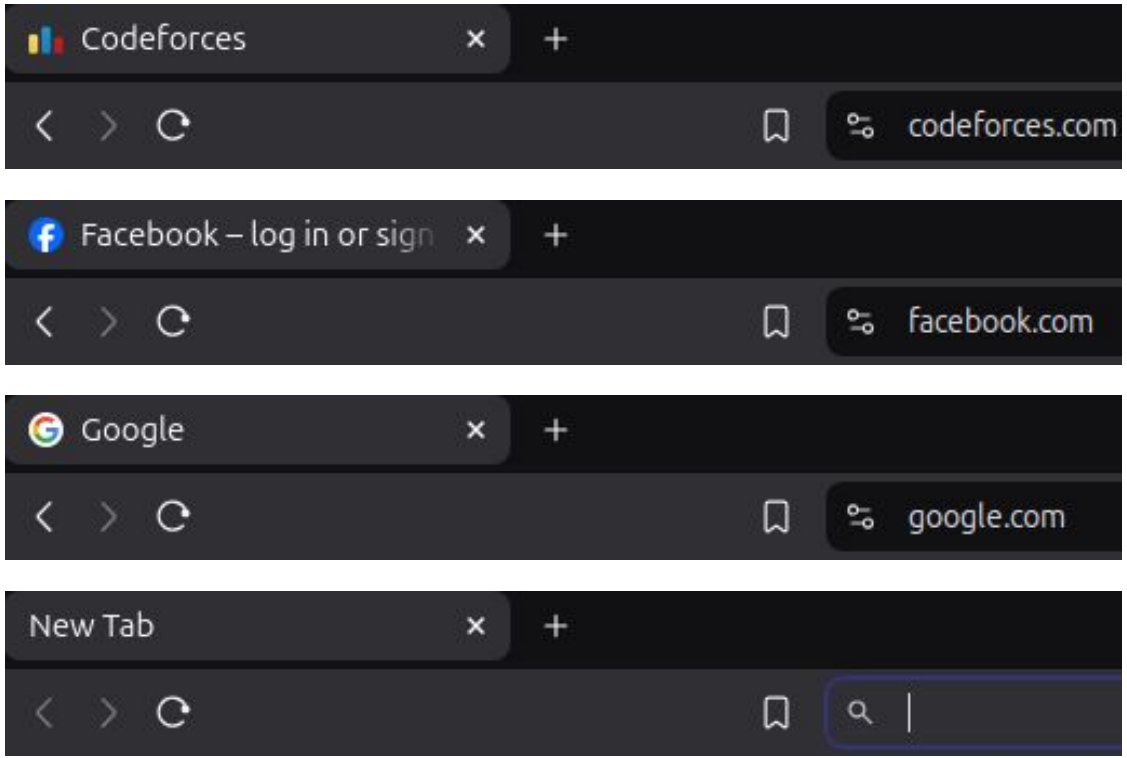


Undo Operations

Stack Applications

- Go to Codeforces
  - Do you want to return to previous?
    Press on this <
    Take the top of Undo Stack
    Put this in Forward Stack

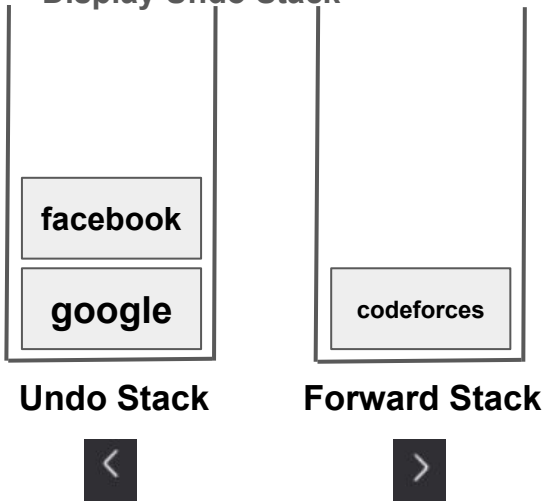




Undo Operations

Stack Applications

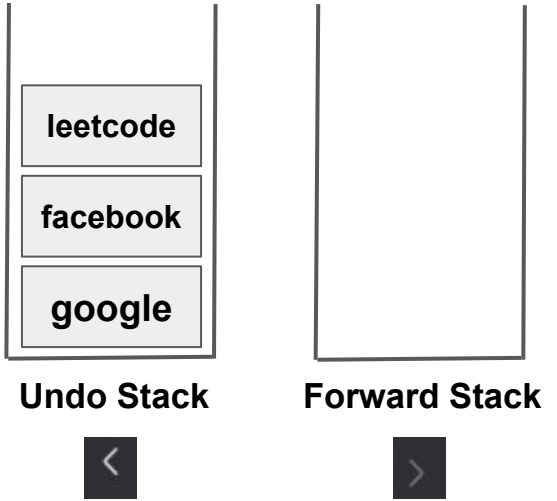
- **Return to Facebook**
  - **Do you want to go to new page?**
    Search about it
    Put it in Undo stack
    Clear Forward Stack
    Display Undo Stack

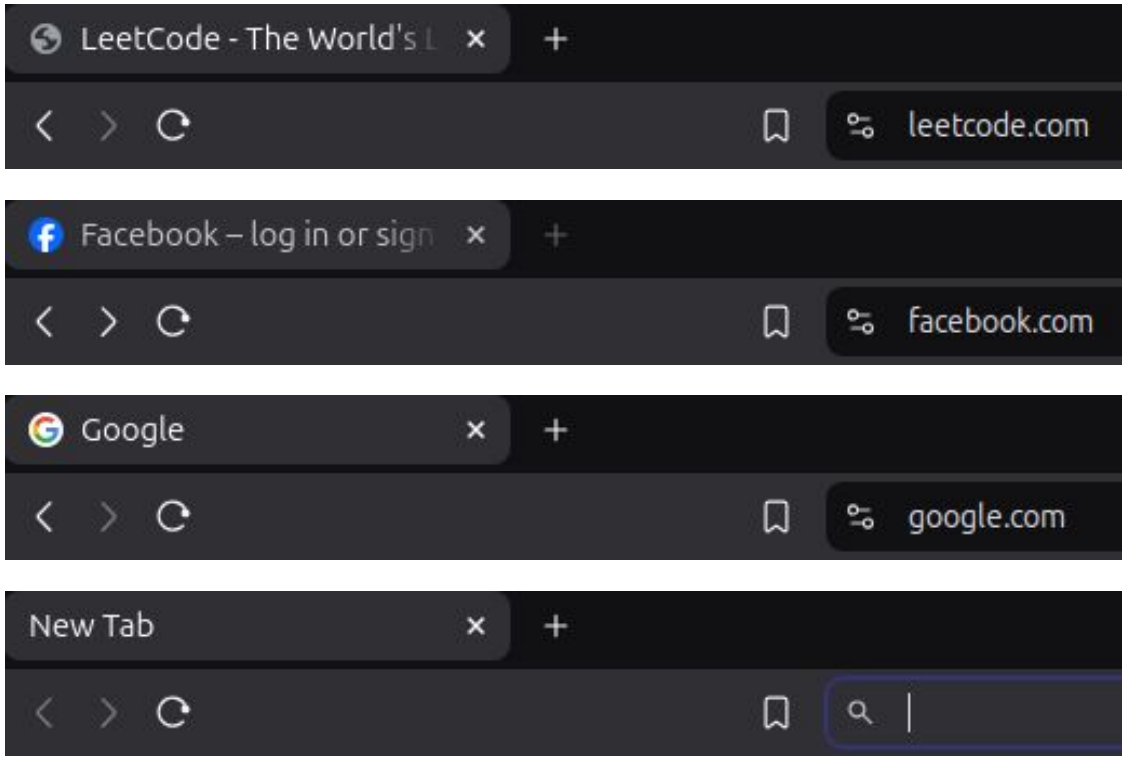


Undo Operations

Stack Applications

- Go to LeetCode

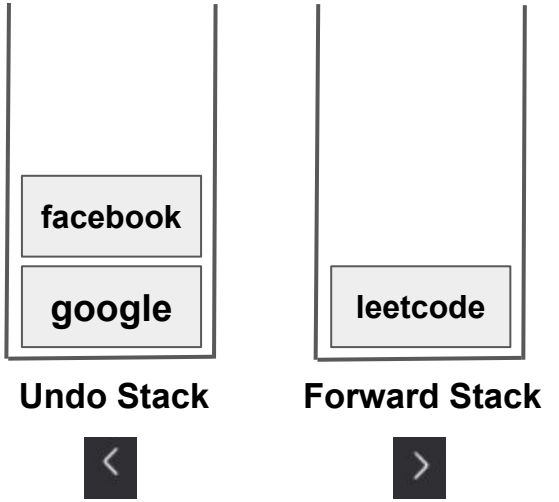




Undo Operations

Stack Applications

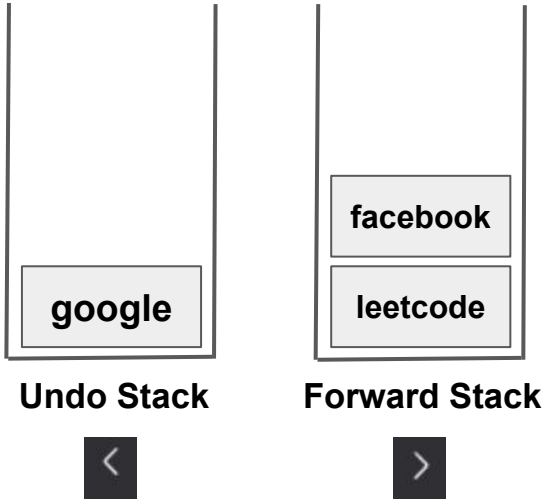
- **Return to Facebook**

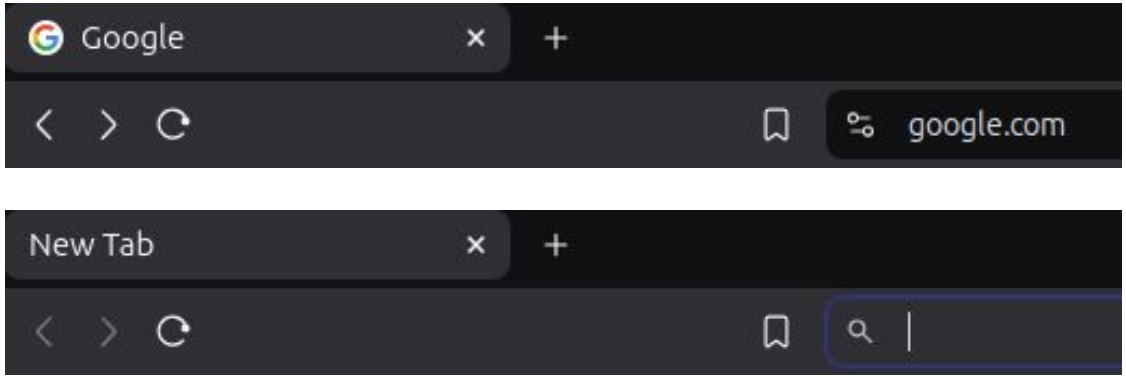


Undo Operations

Stack Applications

- **Return to Google**

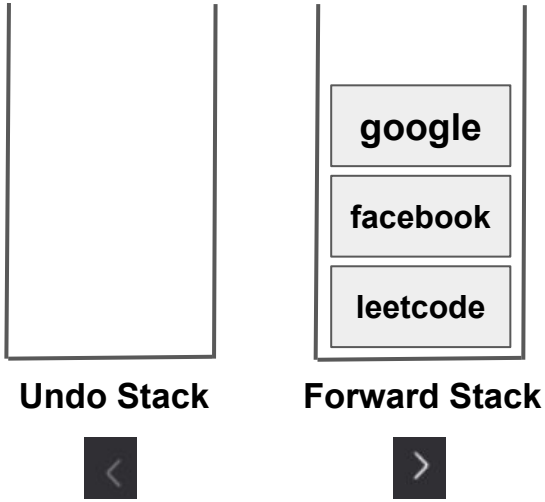


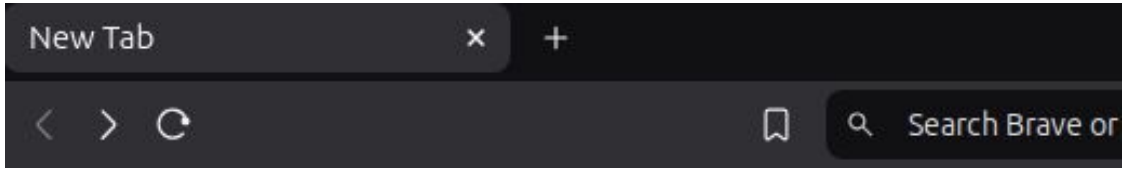


Undo Operations

Stack Applications

- **Return to New Tab**

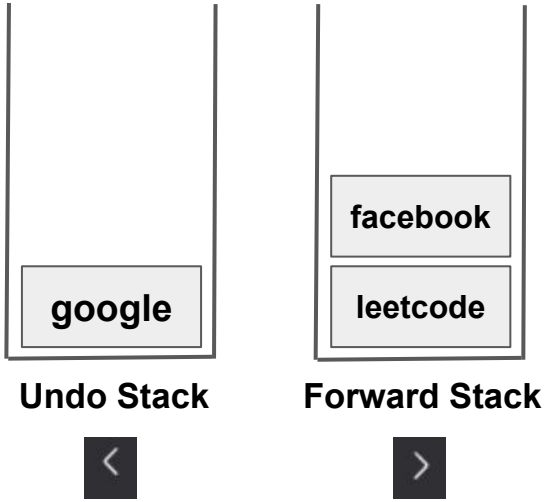


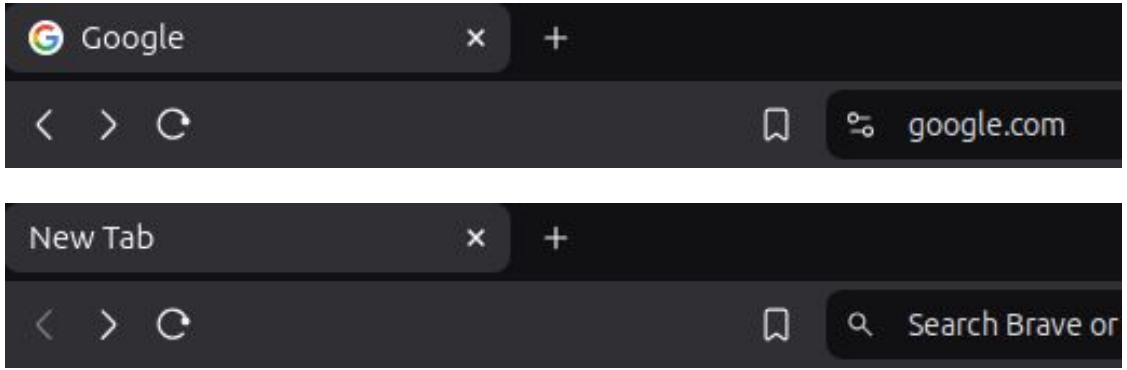


Undo Operations

Stack Applications

- **Go to previous Google page**

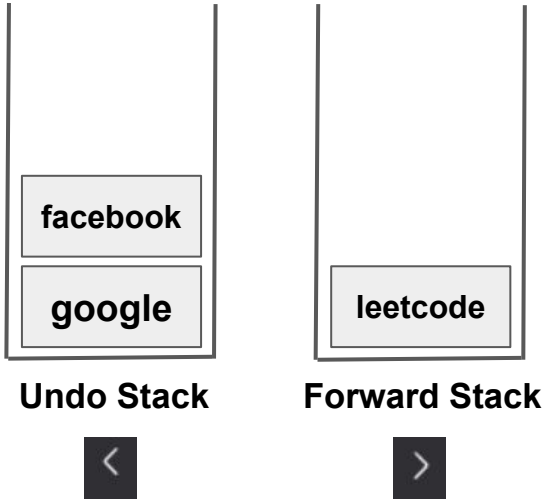




Undo Operations

Stack Applications

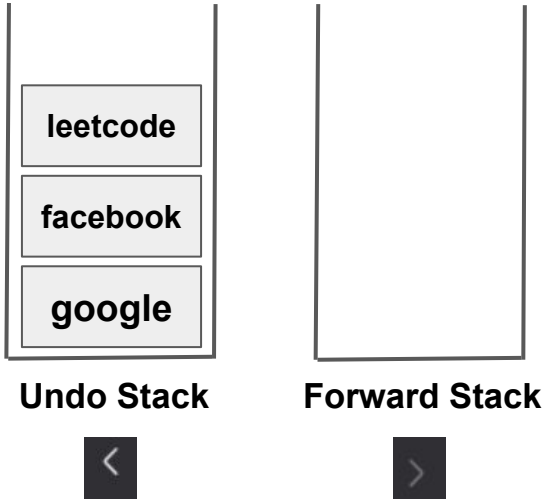
- Go to previous Facbook page

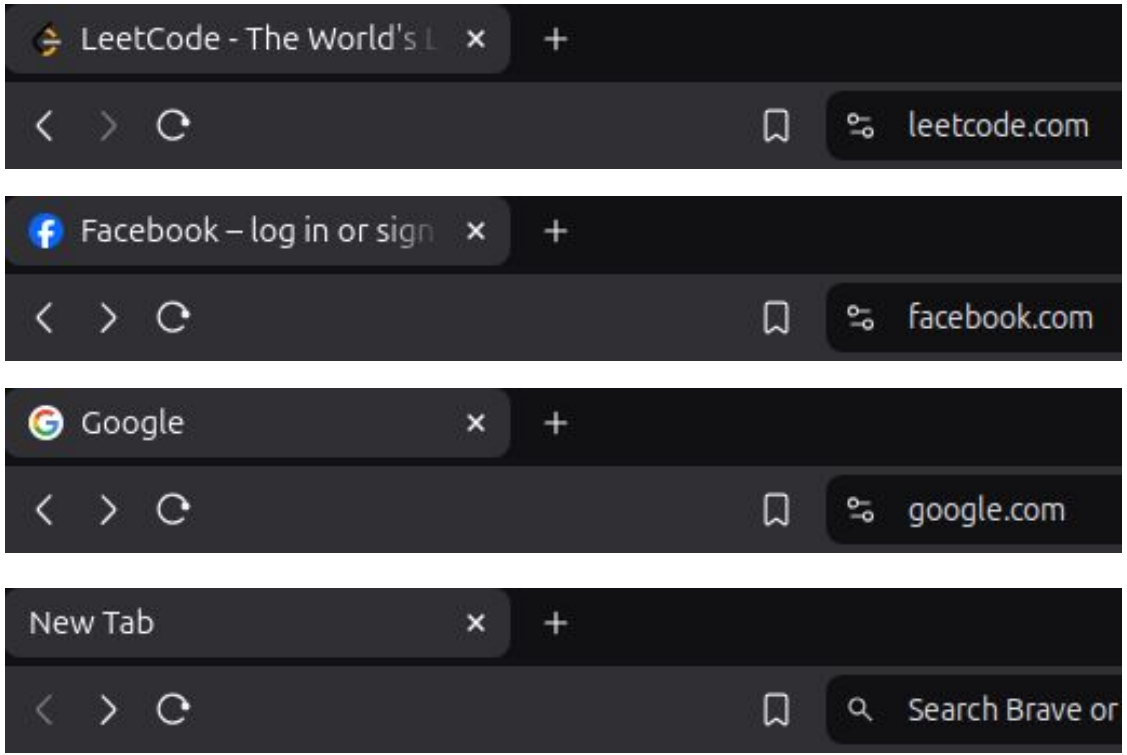


Undo Operations

Stack Applications

- Go to previous LeetCode page

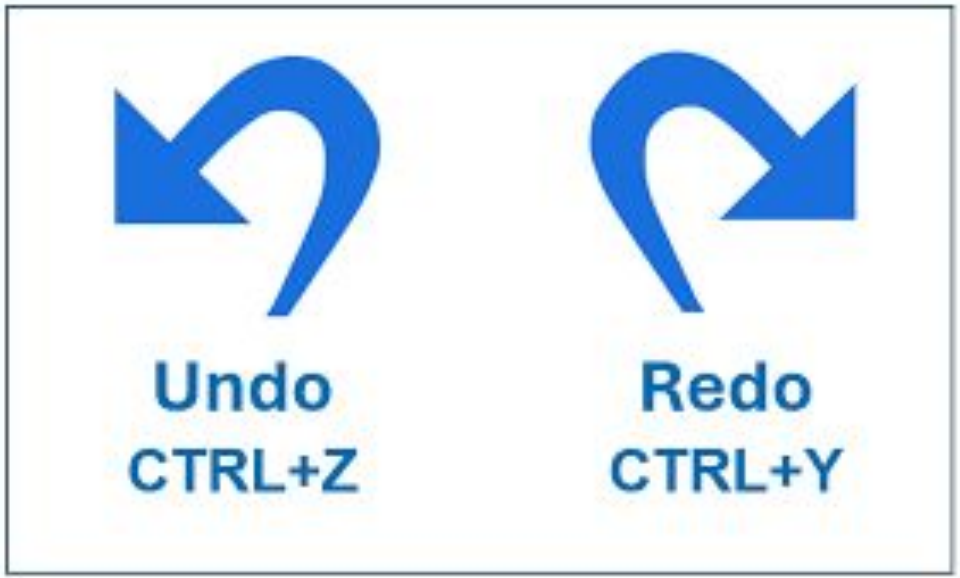




Undo Operations

Stack Applications

- You can apply pervious application on several another
- In other words: **Any application satisfies (Ctrl + Z), (Ctrly + Y)**



src

Stack Applications

- **Syntax Validation** (matching parentheses, brackets, braces).
- **Managing scopes** (e.g., variable declarations, nested blocks).
- **Matching Tags** in HTML.

How it works ?

Try to Solve this problem (leetcode 20)
And See this video (link)

```
for( int i = 0;i < 10; ++i
int main() {
    cout << "Code Salad" << endl;
}
}
```

Balancing and Matching Symbols

Stack Applications

- Some Traversal Algorithms (e.g. DFS, Tree traversals (inorder, preorder, postorder),..) and Backtracking Techniques.

We will explain this in Graphs.

Expression Conversions

In **math,** this expression 2 + 4 + 5 called **infix expression.**

- We know that any binary operators have two operands (e.g. 2 + 4).
- 2 and 4 →**Operands** of this **operator +** and above order called **infix order**.
- If the **operator** before has **operands** (+ 2 4) →called (**prefix order**)
- If the **operator** after has **operands** (2 4 +)→ called (**postfix order)**

In **Computers**, Evaluation of math expression in infix order not easy.

- **Shunting Yard Algorithm** (**Dijkstra's Algorithm**)
  - One of **stack-based** algorithm that uses in **parsing techniques** before evaluating expressions.
  - This algorithm converts an **infix** expression (2 + 4 * 5) into **postfix** expression (2 4 5 * +), which is **easier to evaluate** using a stack.

Precedence and Associativity Rules

Token	Operator	Precedence	Associativity
( ) [ ] → .	function call array element struct or union member	17	left-to-right
-- ++	increment, decrement	16	left-to-right
-- ++ ! ~ - + & * sizeof	decrement, increment logical not one's complement unary minus or plus address or indirection size (in bytes)	15	right-to-left
(type)	type cast	14	right-to-left
* / %	multiplicative	13	Left-to-right
+ -	binary add or subtract	12	left-to-right
<< >>	shift	11	left-to-right
> >= < <=	relational	10	left-to-right
== !=	equality	9	left-to-right
&	bitwise and	8	left-to-right
^	bitwise exclusive or	7	left-to-right
|	bitwise or	6	left-to-right
&&	logical and	5	left-to-right
||	logical or	4	left-to-right
?:	conditional	3	right-to-left
= += -= /= *= %= <<= >>= &= ^=	assignment	2	right-to-left
,	comma	1	left-to-right

Expression Conversions

Convert the following expressions to prefix and postfix notations:

- 2 + 4 * 5
  - 2 4 5 * + (**Postfix**)
  - + 2 * 4 5 (**Prefix**)

- (4 + 8) * (6 - 5) / ((3 - 2) * (2 + 2))
  - 4 8 + 6 5 - * 3 2 - 2 2 + * / (**Postfix**)
  - / * + 4 8 - 6 5 * - 3 2 + 2 2 (**Prefix**)

- Try to Solve this Problem(**link**) first then, see this video (**link**)

Simple Calculator Project

- Make the simple calculator
  - Supports these operators: () , + , - , / , * , ^
  - Implement it in **C++**
  - Ignore input validation (Suppose that input in true format always)
  - **Think** well in all scenarios! (improve **testing and debugging skills**)
  - Challenge yourself to improve your **problem-solving skills**
  - Don’t ask any chatbots

To participate in activity, go to Discord Channel to see more details

Sample Run:

```
(4+8)*(6-5)/((3-2)*(2+2))
3
```

Sheet link: (Link)
Problems solution playlist: (Link)

"Practice Makes Perfect"

Thank You!

Anas Elwkel