Stacks

linear data structure hai jo LIFO principle follow krta hai,

* (Last in → First out)

→ last wala element pehle bhar.

basic operations :

		(TC)
① push(x)	→ add	O(1)
② pop(x)	→ remove top()	O(1)
③ top(x)	→ dekhna top()	O(1)
④ isempty(x)	→ bool T/F	O(1)

Ex → plate Stack
Undo/Redo
browser back button
function calls (Call stack)

Stack implementation

```
stack<int>s;
s.push(10);
s.pop();
int x = s.top();
```

Use cases :

① Expression problems
→ Valid parenthesis
→ infix → postfix
→ Evaluate postfix

② Next greater/Smaller
→ while s.empty() && s.top() <= curr

③ Monotonic Stack
→ increasing } nearest greatest / smallest
→ decreasing }

④ Largest Area/range
→ Maximal Rectangle
→ largest Rectangle in histogram

⇒ Widely used in function calls, recursion, expression evaluation, undo - redo operations and monotonic stack problems like greater/smaller element;
Can be implemented using arrays, linked list or STL containers & they play a crucial role when a problem requires reverse, backtracking or previous state.

Although linear DS, But powers (Monotonicity) reduces TC from $O(n^2)$ to $O(n)$

LC → 20

Valid parenthesis :

→ open bracket must be closed by same type of closing bracket
→ must be in same order
→ closing bracket has same opening bracket as that of closing

Eg → ()[]{}

→ if opening bracket → push. < stack se >
else check karo. same hai ki ni hai

→ Implement back.

→ Eg → I/P → a#b, b

given 2 strings s = a#b, and t = b

→ as we get '#' pop the recent element
→ make the string

and check if both are equal or not

→ Make a string great (LC → 1544)

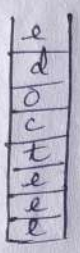
s[i] & s[i+1] where $0 < i < s.length() - 2$

where s[i] == upper(s[i]) (as s[i+1])

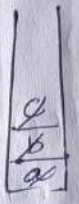
simple get the element push in stack, check top ()

remove the element

"leEetcode"

= Stack →  → "leetcode"

"abABcC"



→ "" empty string

difference b/w a uppercase & lowercase char = (32).

```
stack<char> st;
for(int i=0; i<s.size(); i++)
{
    char ch = s[i];
    if(!st.empty() && abs(st.top() - ch) == 32)
    {
        // bad pair;
        st.pop();
    }
    else{
        st.push(ch);
    }
}
```

[# Since adjacent dependancy check
LIFO is best fit]

} making / filling stack

```
// string creation;
while(!st.empty())
{
    char ch = st.top();
    st.pop();
    ans = ans + ch;
}
reverse(ans.begin(), ans.end());
return ans;
}
```

} string creation

LC → 1021

Remove outter parenthesis

I/p → "(()())(())"

→ decomposition ⇒ "(()())" + "(())"

after removal becomes

"()() + ()"

→ So, answer ⇒ ()()(),

Intution :

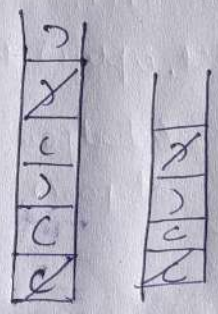
→ Check for the s.top()

if same as (ch) pop the element ;

and push the "ch"

"(()())(())"

```
if(st.top()== ch)
    st.pop();
st.push(ch);
```



→ ()()() Hence O/P

→ Since adjacency is to checked

LIFO is the best used / comfortable to use

"(()(x)(()x)(x)(()x))"

→ ()()()()()

```
stack<char> st;
string ans = "";
for(char ch : s)
{
    if(ch == '(')
    {
        if(!st.empty())          // not outer // pehle se hai
        {
            ans = ans + ch;
            st.push(ch);
        }
    }
    else {   // ab closing aayga
        st.pop();
        if(!st.empty())          // outer
            ans = ans + ch;
    }
}
return ans;
};
```

→ (()())

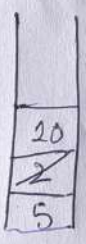
ans = ()(

(

LC → 682 → Baseball Game

["5","2","C","D","+"]

→ (5) (2)



→ 5+2 ⇒ (7)

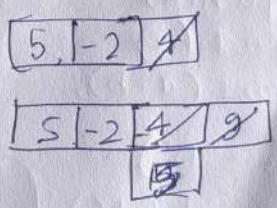
C → (2)

(D)

st.push(2*st.top());

(+)

["5","-2","4","C","D","9","+","+"]



sum = 4+9

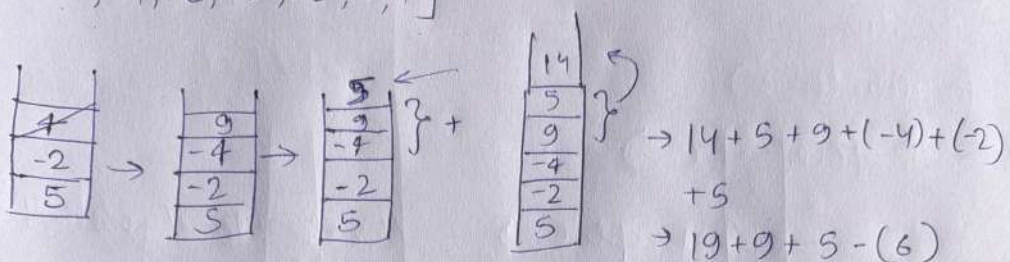
→ ~~13~~

3

8

5 3

[5, -2, 4, C, D, 9, +, +]

→ 

→ 14 + 5 + 9 + (-4) + (-2) + 5

→ 19 + 9 + 5 - (6)

→ 27

```
int sum = 0;
Stack<int> st;
for(auto c : operations)
{
    if (c != "+" && c != "D" && c != "C")
        // matlb integer hai;
        int x = stoi(c);
        st.push(x);
    }
    else if (c == '+')
    {
        a = st.top()
        st.pop();
        b = st.top()
        st.pop()
        st.push(b)
        st.push(a)
        st.push(a+b);
    }
    else if (c == "D")
    {
        int x = st.top();
        st.push(x*2);
    }
    else {
        st.pop();
    }
}
// Stack is ready, sumup karde
while(!st.empty())
{
    sum = sum + st.top()
    st.pop();
}
return sum;
}
```

// phele (b) kyunki no nikla phele