12-01-2026 - Explore data structures such as Stacks, learn about their operations, and use them to solve problems in a variety of domains.

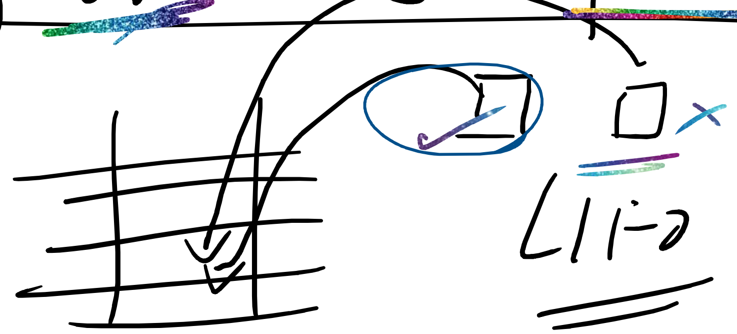
12 January 2026   09:10

=> **Stack:**

ADT : Abstract Data Type

2 Things: Data & Operations

=>



Top = -1

Data:

1) Space for storing elements

2) Top Position

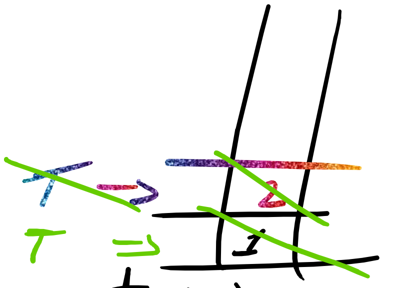
Operations:

1) Push() 3) Peek()

2) Pop() 4) stackTop()

5) isEmpty() 6) isFull()

1, 2, 3 ✓ ✓ ✓



size = ~~2~~ ~~1~~ ~~0~~
2 ~~1~~

top = ~~-1~~ ~~1~~ ~~2~~
-1 ~~1~~

Stack overflow = isFull()

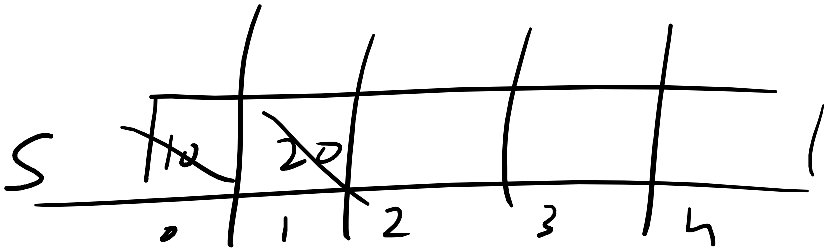
Stack overflow = isFull()

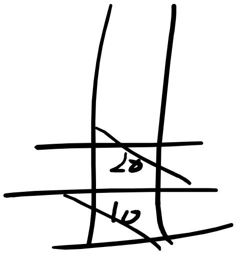
Stack underflow = isEmpty()

[C - Dennis Ritchie & DSA - Cormen]

⇒ **Stack Implementation :**

Arrays & Linked List :

⇒ 



Note:

```
struct Stack {
    int size;
    int top;
    int *S;
}

void main() {
    struct Stack *st;
    print(" Enter size of stack");
    printf("Enter size of stack ");
    scanf("%d", &st.size)
    st.S = new int[size];  st.top=-1;
  getch();
}
```

Prerequisite: Dot & Arrow Operator

```
struct ABC {
  int size
  int length
}
```

* ABC - logical

& ABC - Physical

=> No pointer:

abc -> ABC[size] [Arrow]

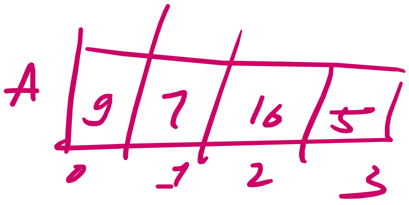
Use Pointer:

&abc.size [Dot]

=> Operations:

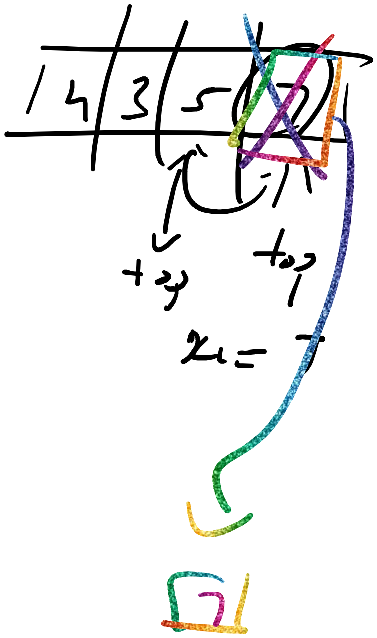
```
void push (struct stack *st, int x) {
```

```
void push (struct stack *st, int x) (
        if (st->top == st->size-1)
            print (Stack overflow);
        else {
            st->top++
            st->s[st->top] = x;
```



A[1] = 7

```
=>    int pop (stack *st) {
          int x = -1;
          if (st->top == -1) {
              printf (Stack underflow);
          else {
            x = st->s[st->top];
             st->top--;
          }
          return x;
```


4
3
5
7
top
top
x = 7
7