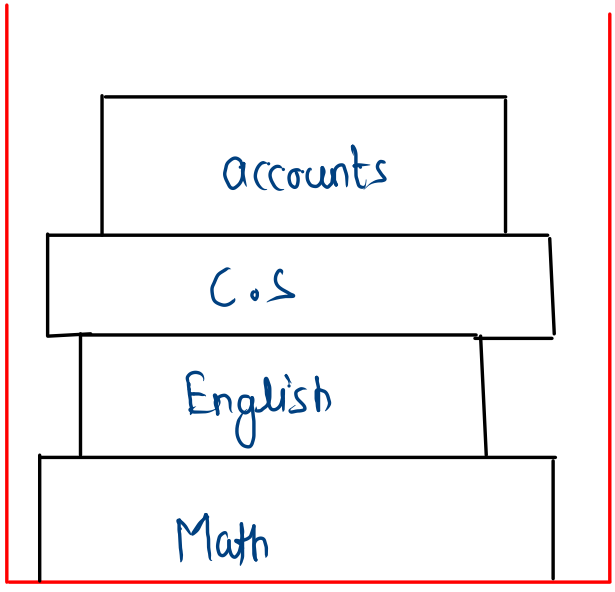
⇒ Stack (LIFO :- last in first out)



chapati box

Note:- stack doesn't have indexing concept

↳ fact :- stack is AL behind the scene.

↳ Stack :- can contain only objects

↳ Stack is dynamic in nature

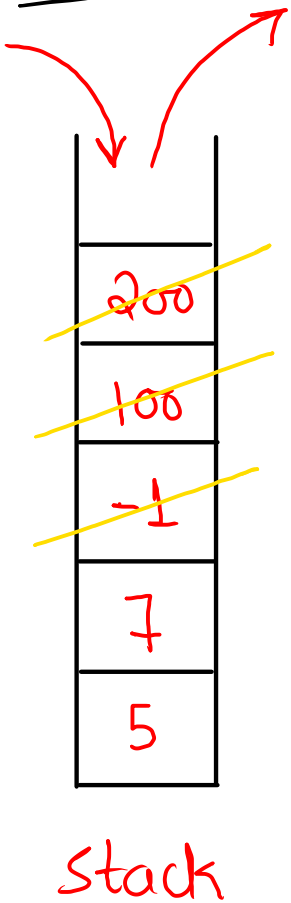
Syntax

triangular brackets

```
Stack<DataType> st = new Stack<>();
```

↳ initial size of st will always be zero

Structure



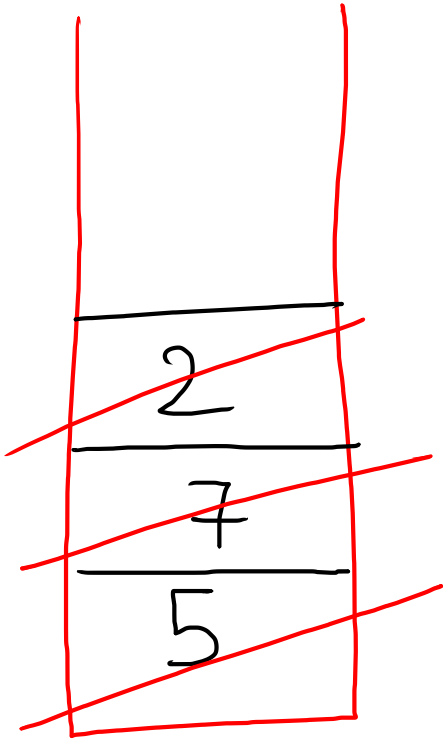
Stack

Inbuilt functions

- To add an element in stack :- `st.push(ele)`
- To remove an element from stack :- `st.pop()`
  (it returns the element which it removes)
- To access top element without removing it :- `st.peek()`
- To access size of stack :- `st.size();`
- To check for stack is empty :- `st.isEmpty()`

ever green

Ex:-



st.push(5)

st.push(7)

st.push(2)

st.pop() ⟶ 2

st.peek() ⟶ 7

st.pop() ⟶ 7

st.pop() ⟶ 5

st.pop() ⟶ Underflow

Stack Syntax Learning

1. **Declare an Empty** $stack\ s$.
2. Take Single Integer T as input.
3. For next T Lines format $(case, x(optional))$

- case 1. $Print$ the $size$ of the $stack$ in a separate line.
- case 2. $Remove$ an element from the stack. If the stack is empty then print -1 in a separate line.
- case 3. Add Integer x to the $stack\ s$.
- case 4. $Print$ an element at the top of the $stack$. If stack is empty print -1 in a seperate line.

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    Stack<Integer> st = new Stack<>();
    int t = scn.nextInt();
    for (int i = 0; i < t; i++) {
        int n = scn.nextInt();
        if (n == 1) {
            printSize(st);
        } else if (n == 2) {
            removeElement(st);
        } else if (n == 3) {
            int x = scn.nextInt();
            addElement(st, x);
        } else if (n == 4) {
            printTopElement(st);
        }
    }
}
```

```java
public static void printSize(Stack<Integer> st) {
    int ans = st.size();
    System.out.println(ans);
}
public static void removeElement(Stack<Integer> st) {
    if ( st.size() == 0 ) {
        System.out.println("-1");
        return;
    }
    st.pop();
}
public static void addElement(Stack<Integer> st, int x) {
    st.push(x);
}
public static void printTopElement(Stack<Integer> st) {
    if ( st.size() == 0 ) {
        System.out.println("-1");
        return;
    }
    int ans = st.peek();
    System.out.println(ans);
}
```

Note:- every inbuilt fn of stack takes O(1) time

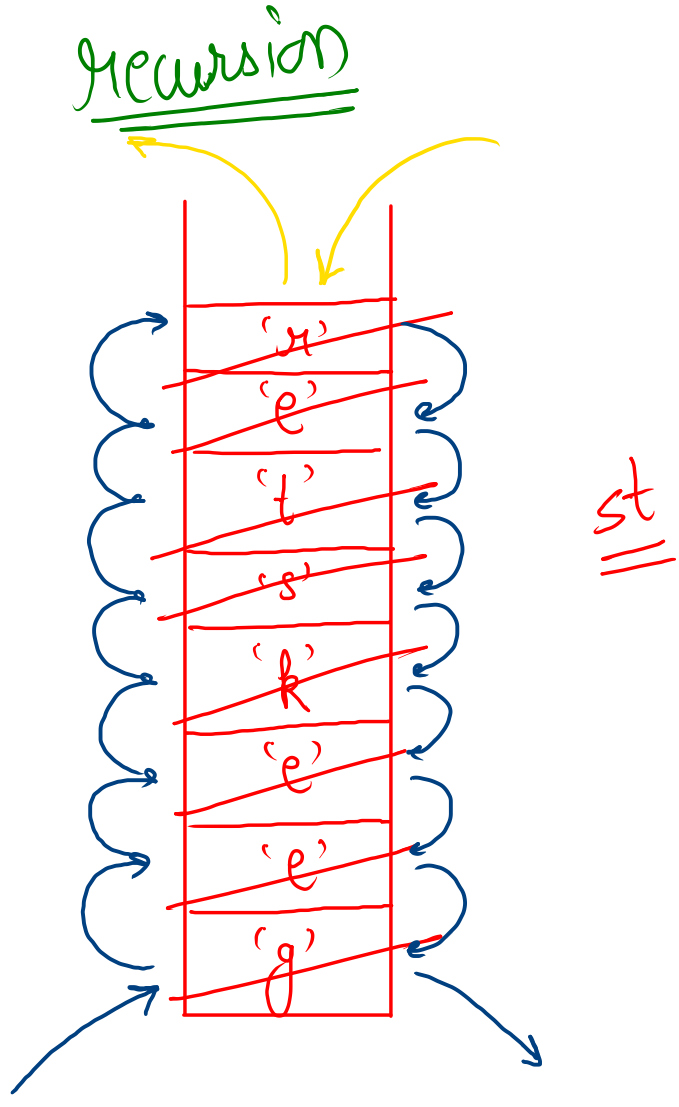
Reverse string

str = "geekster"
0 1 2 3 4 5 6 7
i

ans = ""
ans = "r"
ans = "re"
ans = "ret"
ans = "rets"
ans = "retsk"
ans = "retske"
ans = "retskee"
ans = "retskeeg"

ans = ans + top

Recursion



Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();

    String ans = reverseString(str);
    System.out.println(ans);
}
public static String reverseString(String str) {
    // declare stack
    Stack<Character> st = new Stack<>();

    // fill the stack
    for (char c : str.toCharArray()) {
        st.push(c);
    }

    // empty the stack
    String ans = "";
    while (st.size() > 0) {
        char ch = st.pop();
        ans = ans + ch;
    }
    return ans;
}
```

T.C

↳ O(n)

n = size of stack

oper = 2 * n

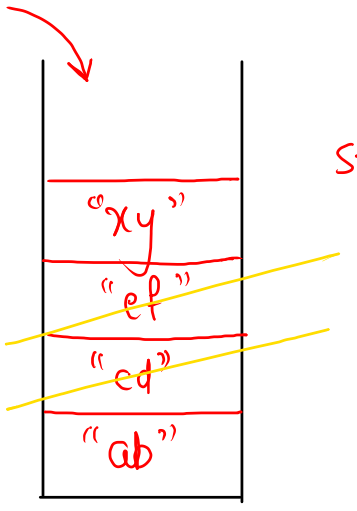
Delete consecutive

arr =

0	1	2	3	4	5
"ab"	"cd"	"ef"	"ef"	"cd"	"xy"

, ans = 2

i

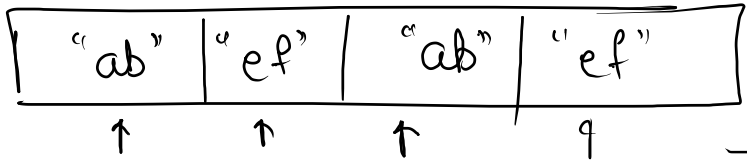


stack

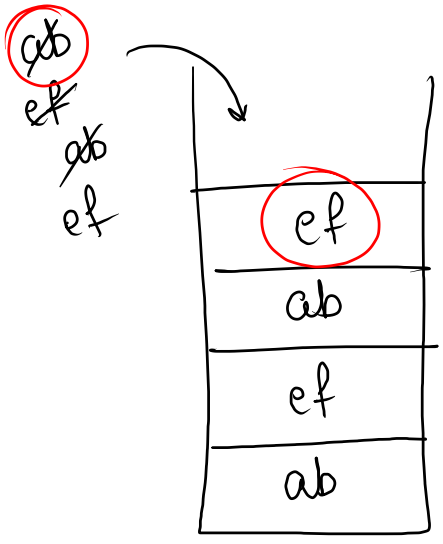
logic :- stack can only store unique elements

ans = st.size()

arr



ans = 4



code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    String[] arr = new String[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.next();
    }
    int ans = deleteConsecutive(arr, n);
    System.out.println(ans);
}
public static int deleteConsecutive(String[] arr, int n) {
    Stack<String> st = new Stack<>();
    for (int i = 0; i < arr.length; i++) {
        if ( st.size() == 0 || arr[i].equals(st.peek()) == false ) {
            st.push(arr[i]);
        } else {
            st.pop();
        }
    }
    return st.size();
}
```

$T.C = O(n)$

n = size of stack