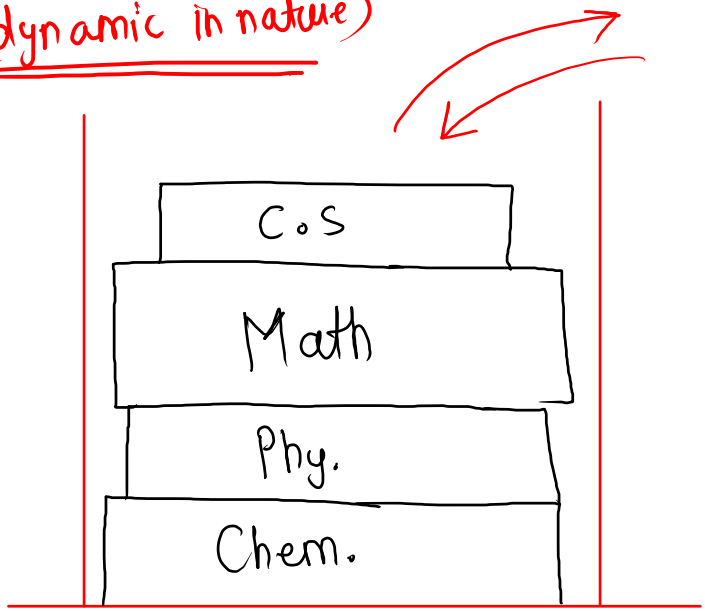
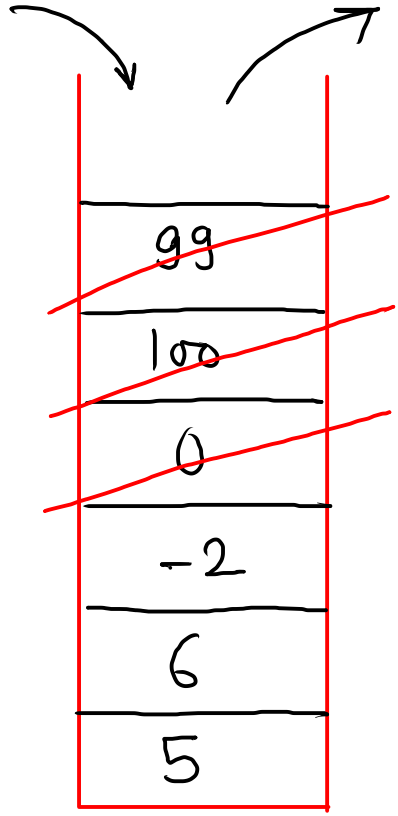
⇒ Stack [LIFO :- last in first out]

(dynamic in nature)



Stack of books



Stack (Integer)

Note:- Stack doesn't contain Indexing

Syntax

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

triangular brackets

→ Inbuilt function

To add an element in Stack :- `st.push(val);`

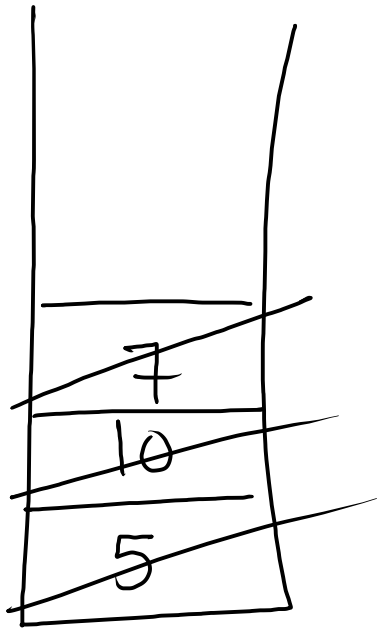
To remove an element in Stack :- `st.pop();`

To access top element from stack :- `st.peek();`
(without removing it)

To get the size of stack :- `st.size();`

To check if stack is empty or not :- `st.isEmpty();`

Ex:-



push(5) ✓

push(10) ✓

push(7) ✓

pop() → 7 ✓

peek() → 10 ✓

peek() → 10 ✓

pop() → 10 ✓

peek() → 5 ✓

pop() → 5 ✓

pop() → error ✓

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.

1)
```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) {
            removeTopElement(st);
        } else if (n == 3) {
            int x = scn.nextInt();
            addElement(st, x);
        } else if (n == 4) {
            printTopElement(st);
        }
    }
}
```

2)
```java
public static void printSize(Stack<Integer> st) {
    System.out.println(st.size());
}
```

3)
```java
public static void removeTopElement(Stack<Integer> st) {
    if ( st.size() == 0 ) {
        System.out.println("-1");
        return;
    }
    st.pop();
}
```

4)
```java
public static void addElement(Stack<Integer> st, int x) {
    st.push(x);
}
```

5)
```java
public static void printTopElement(Stack<Integer> st) {
    if ( st.size() == 0 ) {
        System.out.println("-1");
        return;
    }
    int val = st.peek();
    System.out.println(val);
}
```

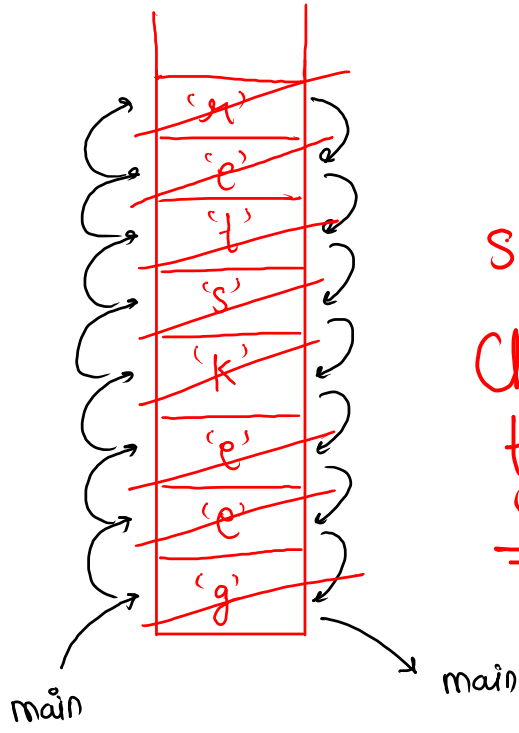
T.C = O(n)

also each inbuilt function will be having T.C of O(1)

Reverse string

str = "geekster";

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



stack of Character type

ans = ans + top element

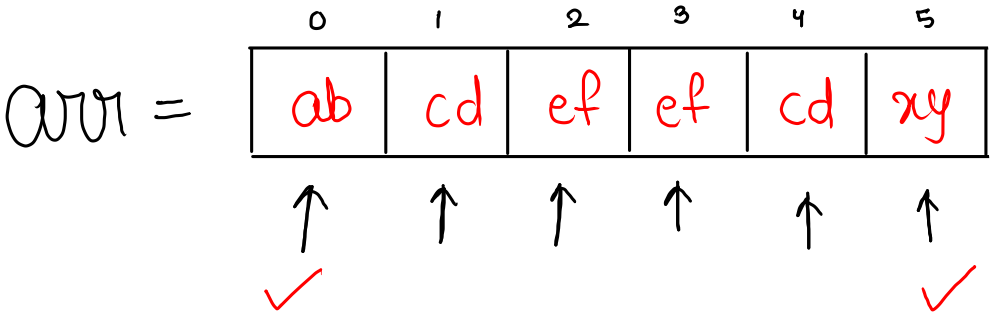
Code

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();
    System.out.println(reverse(str));
}
public static String reverse(String str) {
    Stack<Character> st = new Stack<>();
    for (int i = 0; i < str.length(); i++) {
        char ch = str.charAt(i);
        st.push(ch);
    }

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

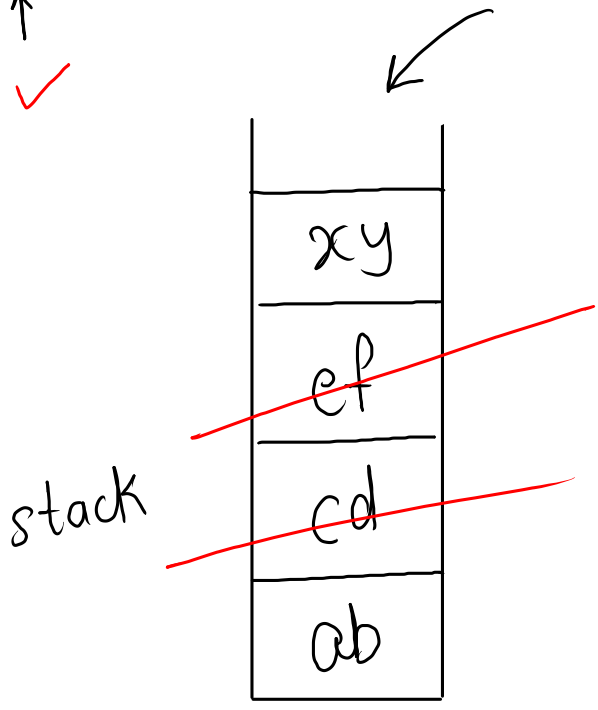
T.C = O(n)
where n is size of stack

Delete consecutive



ans = st.size();

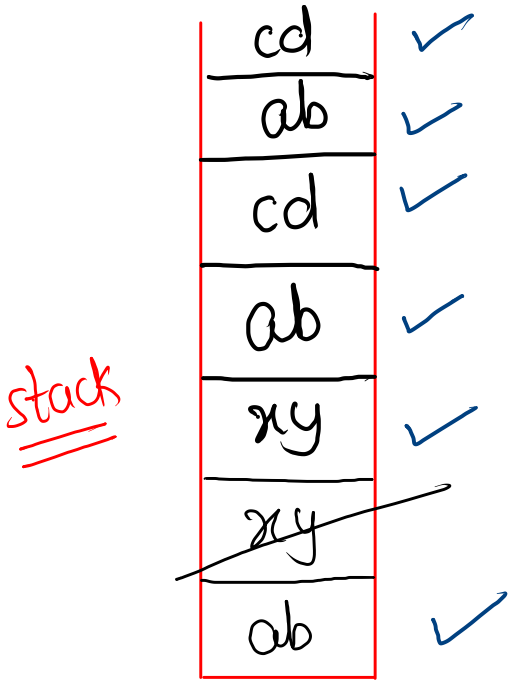
(no. of elements remaining)



fun fact :- stack is arraylist behind to scene

Ex)

arr = [ab , xy , xy , xy , ab , cd , ab , cd]



stack

size()

psud code

1) traverse from 0 to n-1 in array

1.1) if st is empty
st.push(curr.ele);

1.2) else if (curr != peek)
st.push(curr.ele)

1.3) else if (curr == peek())
st.pop()

faith :- stack should contain only non-destroyed element

Code

```
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();
    }
    System.out.println(deleteConsecutive(arr, n));
}
public static int deleteConsecutive(String[] arr, int n) {
    Stack<String> st = new Stack<>();
    for (int i = 0; i < n; i++) {
        if ( st.size() == 0 || st.peek().equals(arr[i]) == false ) {
            st.push( arr[i] );
        } else {
            st.pop();
        }
    }
    return st.size();
}
```

T.C = O(n)