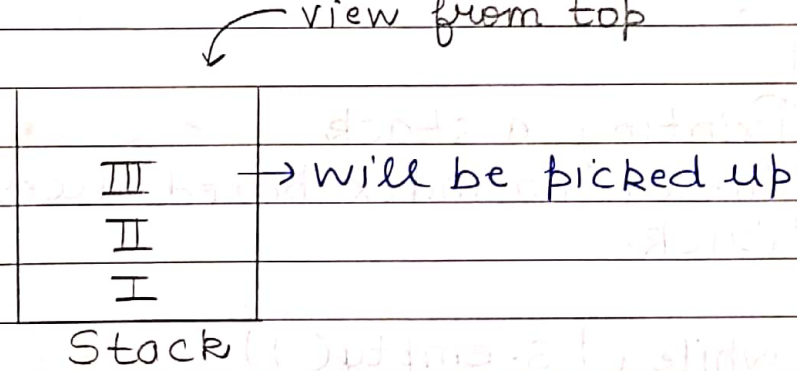
23/04/2023

Stack

Data structure in which data is stored in the form of LIFO i.e Last in First out. The thing inserted at last will be fetched first.

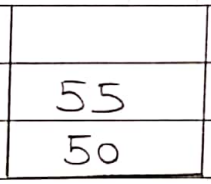
Ex → Plates in marriage can be a real life example of stack. The plate which was inserted at last will be picked first.



Operations in stack

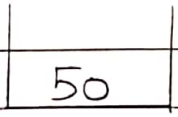
1) push operation → To insert data in the stack.

```
s.push(50);
s.push(55);
```



2) pop operation → To delete data in the stack

```
s.pop();
```



3) peek operation → To see the topmost data

```
s.top(); → 50
```

4) s.empty() → Returns true if stack is empty else return false.

5) S.size() → Returns no. of elements present in the stack.

Creation of stack using STL

```
#include <stack>

int main() {
    stack <int> s; //Stack storing integer
}                  data.
```

Here we have created stack with name as S.

Printing a stack

There is no index based accessing in the stack.

```
while (! s.empty()) { // Run until stack is
                        not empty
    cout << s.top(); //Print top element
    s.pop(); //Remove element
}
```

30
20
10

O/p → 30
20
10

} Reverse order of insertion.

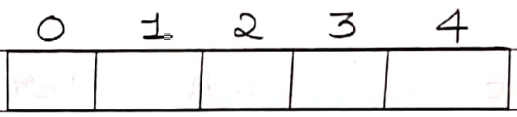
Hence stack can be used for reversal operation.

Implement Stack

We need to implement operations such as

push, pop, top, size and empty.

Push operation



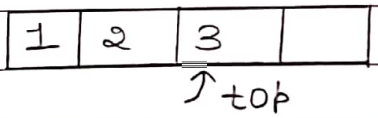
Two scenarios → space available → insert
space not available → Stack overflow

insert ⇒ increment top
arr [top] = element

How to check whether stack has space or not?
If size - top > 1, then space available else not available.

```
void push (int data) {
    if (size - top > 1) {Space available
        top++;
        arr [top] = data;
    }
    else {
        cout << "Stack overflow" << endl;
    }
}
```

Pop operation



Two scenarios → Stack empty → underflow
Stack not empty → top--

top == -1, then stack is empty else not empty.

```
void pop() {
    //Stack empty
    if (top == -1) {

        cout << "Stack underflow";
    }
    else { //Stack not empty
        top--;
    }
}
```

Peek operation

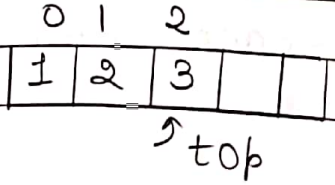
1) If stack is empty, then topmost element does not exist.

2) Else simply cout arr[top].

```
void getTop() {
    //Empty stack
    if (top == -1) {
        cout << "Stack empty";
    }
    else { // Print top element
        cout << arr[top];
    }
}
```

getSize operation

Here no. of elements in the stack should be returned.



No. of elements = 3 i.e top+1

```
int getSize(){
    return top+1;
}
```

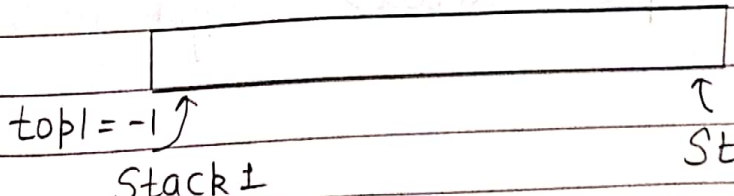
isEmpty operation

```
bool isEmpty(){
    if(top==-1){
        return true; //Stack is empty
    }
    return false; //Stack is not empty
}
```

Q1 2 stacks in an array.
We are required to implement 4 functions i.e. push1, push2, pop1, pop2

Brute force
Dividing the array into 2 half but this will lead to some memory wastage as one stack might get filled but other stack is not full & we are trying to push in full stack & it says overflow but stack has available space.
↳ 2nd stack

Optimal approach



Stack1 and Stack2 are full when top2 - top1 == 1

(1) Push1 operation

Space not available ⇒ top2 - top1 == 1,
then can't insert

```
                ↱ int data
void push1 (  ) {
    //Space not available
    if (top2 - top1 == 1) {
            cout << "Overflow";
    }
    //Space available
    else {
        top1 ++;
        arr [top1] = data;
    }
}
```

(2) Push2 operation

```
void push2 (int data) {
    //Space not available
    if (top2 - top1 == 1) {
            cout << "Overflow";
    }
    //Space available
    else {
        top2 - -  ; //Stack2 to be filled from
        arr [top2] = data;   right to left.
    }
}
```

Time complexity = O(1) → For both push1 & push2

(3) Pop1 operation
top1 == -1, then stack1 is empty

```
void pop1() {
    //Stack1 empty
    if (top1 == -1) {
        cout << "Underflow";
    }
    else { //Stack1 not empty
        top1--;
    }
}
```

(4) Pop2 operation
top2 == size, then stack2 is empty

```
void pop2() {
    //Stack2 empty
    if (top2 == size) {
        cout << "Underflow";
    }
    else { //Stack2 not empty
        top2++; //As stack is filled from
                right to left
    }
}
```

Time complexity = O(1) → For both pop1 & pop2.

Q2 Reverse a string with help of stack.

i/p → "bhavya"
o/p → "ayvahb"

Simply add each character into the stack. After inserting all the characters print the top and pop until stack does not become empty.

Code

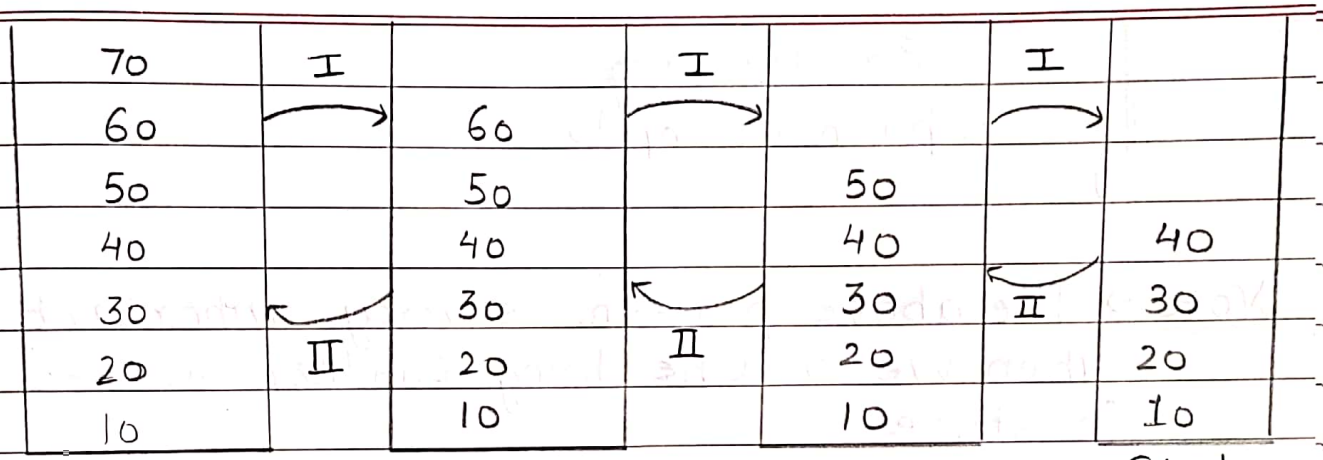
```
void reverse (string str) {

    stack <char> s;
    for (int i=0; i<str.length(); i++){
        //Simply add characters to stack
        s.push(str[i]);
    }
    while (!s.empty()){
        cout <<s.top();
        s.pop();
    }
}
```

Q3 Find the middle element in the stack.

70
60
50
40 → middle
30
20
10

Approach

Here we will be using the concept of stack.



I → temp = s.top();
s.pop();

II → s.push(temp);

Stop

totalSize + 1 / 2 == size

print s.top();

Code

```
void printMiddle (stack<int> &s, int &total Size) {

    // Empty case
    if (s.size() == 0) {
        cout << "No element";
        return;
    }
    // Base case
    if (totalSize/2 + 1 == s.size()) {
        cout << "Middle element = " << s.top();
        return;
    }
    // Operations
    int temp = s.top();
    s.pop();
    // Recursive call
    printMiddle (s, totalSize);
```

```
        // Backtracking
        s.push(temp);
    }
```

Note → The above pattern is very important when we will be doing further questions in stack.