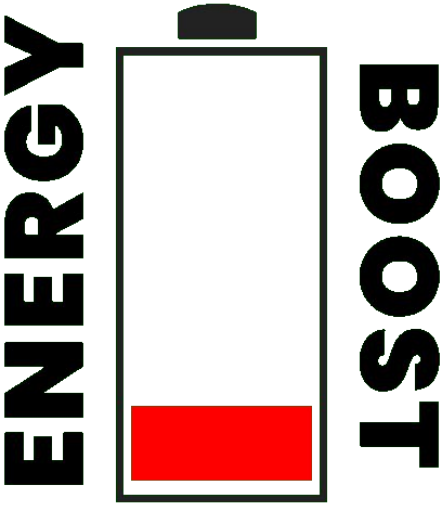
Stack
Data Structure

Menus with Backward option

You have to implement a menu in which you have the option to go to next menu and then come back to the previous menu as well.



Menus with Backward option

```
-----------------Main Menu-----------------
-1. Exit
1. Go to First Sub Menu
Enter your Option:
```

```
-----------------1st Sub Menu-----------------
0. Go Back
2. Go to Second Sub Menu
Enter your Option:
```

```
-----------------2nd Sub Menu-----------------
0. Go Back
3. Go to Third Sub Menu
Enter your Option:
```

Menus with Backward option

How do you think you can implement that?

Menus with Backward option

In order to implement that we have to store the information of the previous menu so that whenever user presses 0 we can go to that previous menu.

Menus with Backward option

In order to implement that we have to store the information of the previous menu so that whenever user presses 0 we can go to that previous menu.

Can we store that information in a Single Variable?

Menus with Backward option

But there are more than one sub menus. So how can we track or store the information of all the previous menus so that user can keep going back.

Menus with Backward option

But there are more than one sub menus. So how can we track or store the information of all the previous menus so that user can keep going back.

Can we Store this information in an Array?

Menus with Backward option

But there are more than one sub menus. So how can we track or store the information of all the previous menus so that user can keep going back.

Can we Store this information in an Array?

0	1	2	3	4	5	6	7
Main Menu							

Menus with Backward option

But there are more than one sub menus. So how can we track or store the information of all the previous menus so that user can keep going back.

Can we Store this information in an Array?

0	1	2	3	4	5	6	7
Main Menu	1st Sub						

Menus with Backward option

But there are more than one sub menus. So how can we track or store the information of all the previous menus so that user can keep going back.

Can we Store this information in an Array?

0	1	2	3	4	5	6	7
Main Menu	1st Sub	2nd Sub					

Menus with Backward option

Yes we can !!
But we have to put some constraints on that array.

What are those Constraints?

0	1	2	3	4	5	6	7
Main Menu	1st Sub	2nd Sub					

Menus with Backward option

We have to insert new element at the end of the Array and we also have to delete the element from the end of the Array.

0	1	2	3	4	5	6	7
Main Menu	1st Sub	2nd Sub					

Menus with Backward option

We have to insert new element at the end of the Array and we also have to delete the element from the end of the Array.

2	2nd Menu
1	1st Menu
0	Main Menu

Stack

We consider this a new Data Structure and call it Stack and the end of the Array is called Top of the Stack

2	2nd Menu
1	1st Menu
0	Main Menu

Stack: Real life Examples

We consider this a new Data Structure and call it Stack and the end of the Array is called Top of the Stack

2	2nd Menu
1	1st Menu
0	Main Menu

Stack: Real life Examples

We consider this a new Data Structure and call it Stack and the end of the Array is called Top of the Stack

2	2nd Menu
1	1st Menu
0	Main Menu

Stack: LIFO

Stack follows the LIFO (Last-In-First-Out) principle.

2	2nd Menu
1	1st Menu
0	Main Menu

Stack: Operations on a Stack

We have to add or delete elements from the top of the stack.

In Technical Terms

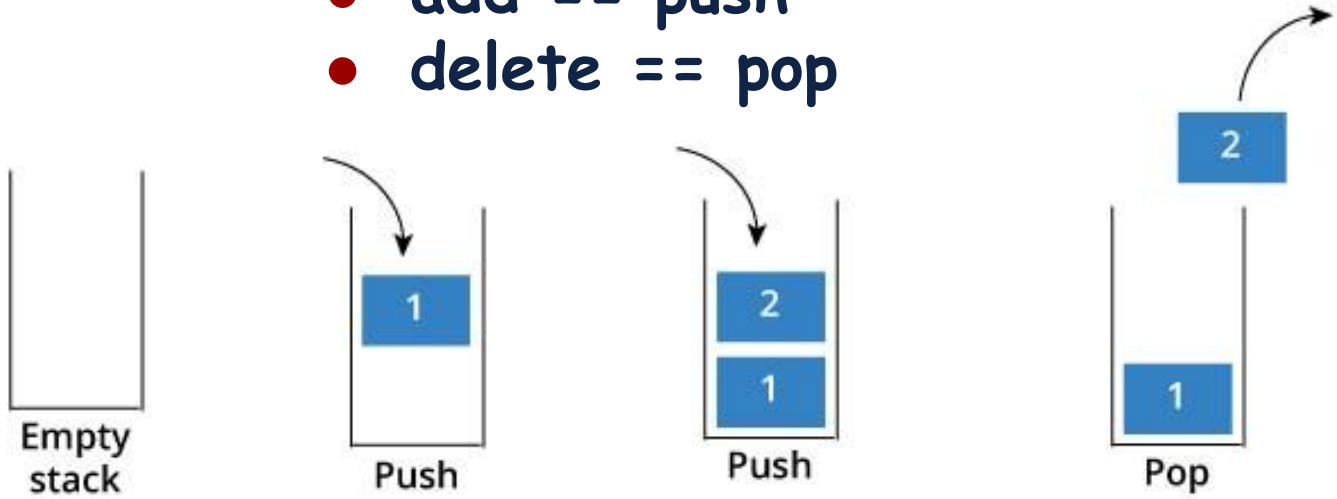
- add == push
- delete == pop

Stack: Operations on a Stack

We have to add or delete elements from the top of the stack.

In Technical Terms

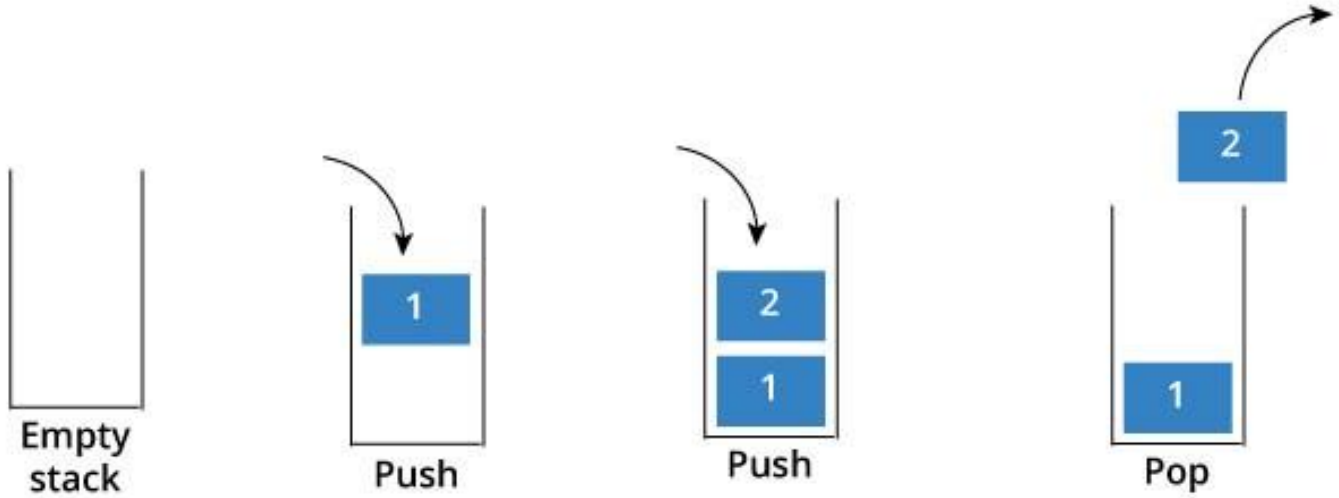
- add == push
- delete == pop



Stack: Operations on a Stack

We can now apply other operations on the Stack as well.

- View whether the Stack is empty or not
- View the top most element of the Stack



Stack: Implementation using Array

Make a Class named Stack and add push, pop, top and isEmpty operations.

Stack: Implementation using Array

```
const int MAX = 20;
class Stack
{
    int top;
    int myStack[MAX]; // stack array

public:
    Stack()
    {
        top = -1;
    }
```

```
// pushes element on to the stack
    bool push(int item)
    {
        if (top >= (MAX - 1))
        {
            cout << "Stack Overflow!!!";
            return false;
        }
        else
        {
            top = top + 1;
            myStack[top] = item;
            return true;
        }
    }
```

Stack: Implementation using Array

```
// removes or pops elements out of the stack
    int pop()
    {
        if (top < 0)
        {
            cout << "Stack Underflow!!";
            return 0;
        }
        else
        {
            int item = myStack[top];
            top = top - 1;
            return item;
        }
    }
```

```
    // check if stack is empty
    bool isEmpty()
    {
        return (top < 0);
    }

    void clear()
    {
        top = -1;
    }
};
```

Activity: Drawback of Stack with Array

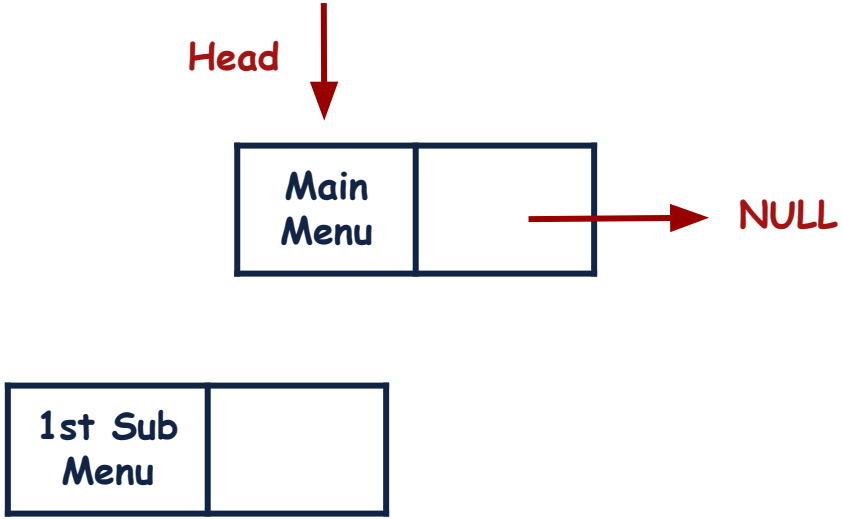
Can you see any limitation of Stack when implemented using Arrays?

Activity: Implementation using Linked List

Now can you implement the Stack using Linked List?

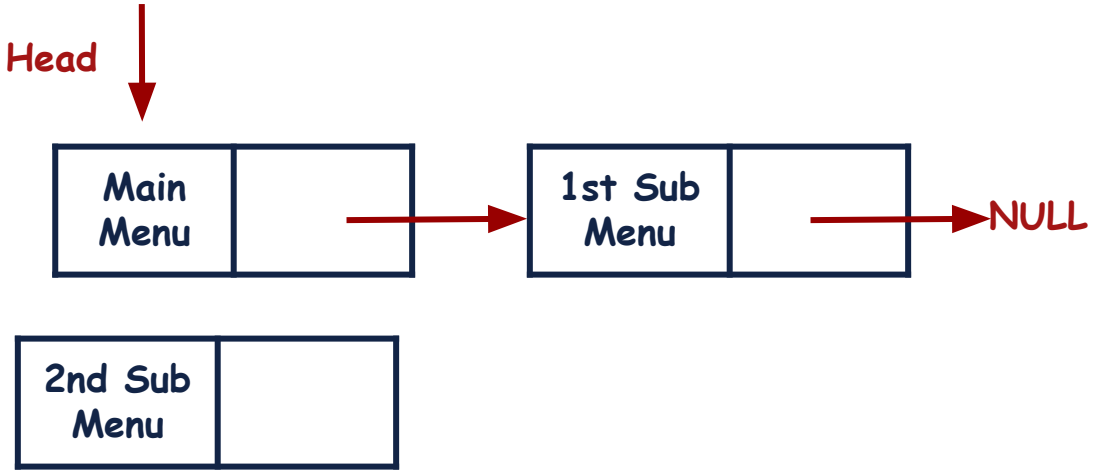
Stack: Implementation using Linked List

Should we insert the new node at the start or end of the linklist?



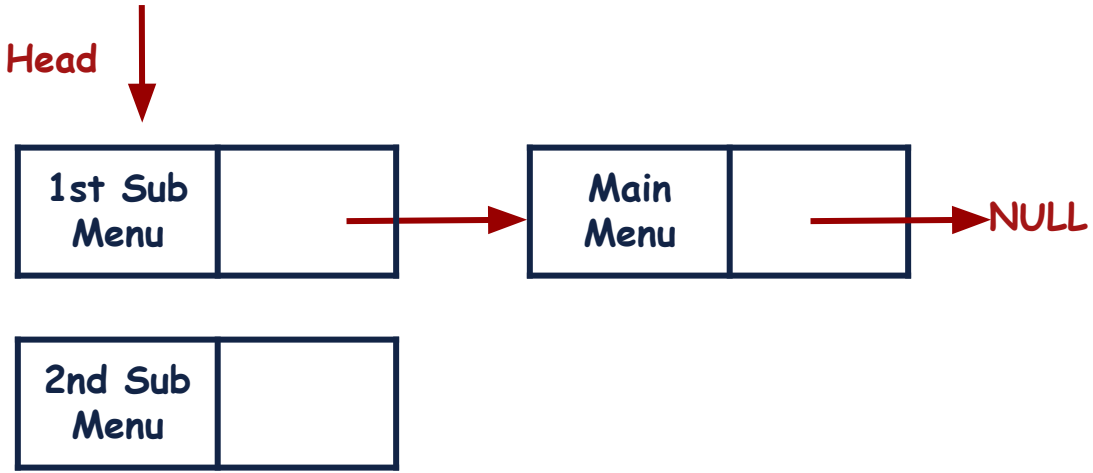
Stack: Implementation using Linked List

If we add the new node at the **end of the linked list** then we have to **move the head pointer** to the end of the linked list before adding the new record.



Stack: Implementation using Linked List

Therefore, we will add the new node at the start of the linked list.



Stack: Implementation using Linked List

```
struct node
{
    int data;
    struct node* next;
};
```

```
class Stack
{
    struct node* top;
public:
    Stack()
    {
        top = NULL;
    }
```

```
bool push(int item)
    {
        struct node* record = new node();
        record->data = item;
        record->next = top;
        top = record;
        return true;
    }
```

Stack: Implementation using Linked List

```
struct node
{
    int data;
    struct node* next;
};
```

```
class Stack
{
    struct node* top;
public:
    Stack()
    {
        top = NULL;
    }
```

```
int pop()
    {
        if(top == NULL)
        {
            cout << "Stack Underflow!!";
            return 0;
        }
        else
        {

            struct node* temp;
            temp = top;
            top = top->next;

            int item = temp->data;
            delete temp;
            return item;
        }
    }
};
```

Learning Objective

Students should be able to **recognize** real life problems where **stacks** data structure are appropriate to solve the problem efficiently.

Self Assessment

1. Implement the Menus System with back option.

2. Solve the Bracket Balancing Problem on leetcode.
   https://leetcode.com/problems/valid-parentheses/