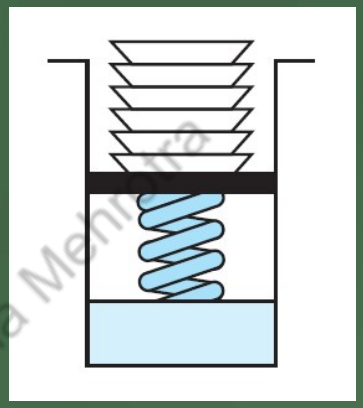
The Stack

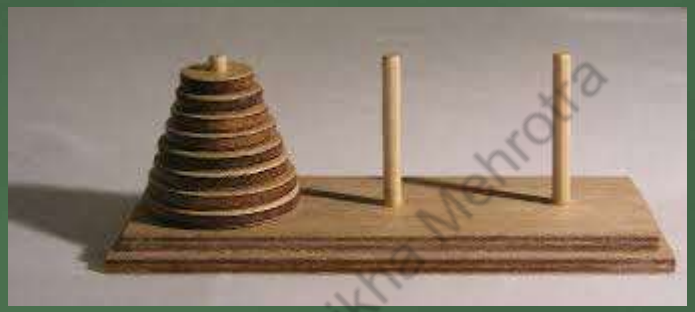
- Linear Data Structure
- Follows Last In, First Out (LIFO)

- Access is allowed only at one point of the structure, normally termed the *top* of the stack
  - access to the most recently added item only
- Operations are limited:
  - push (add item to stack)
  - pop (remove top item from stack)
  - top (get top item without removing it)
  - isEmpty
  - size?
- Described as a "Last In First Out" (LIFO) data structure

Introduction of Stack



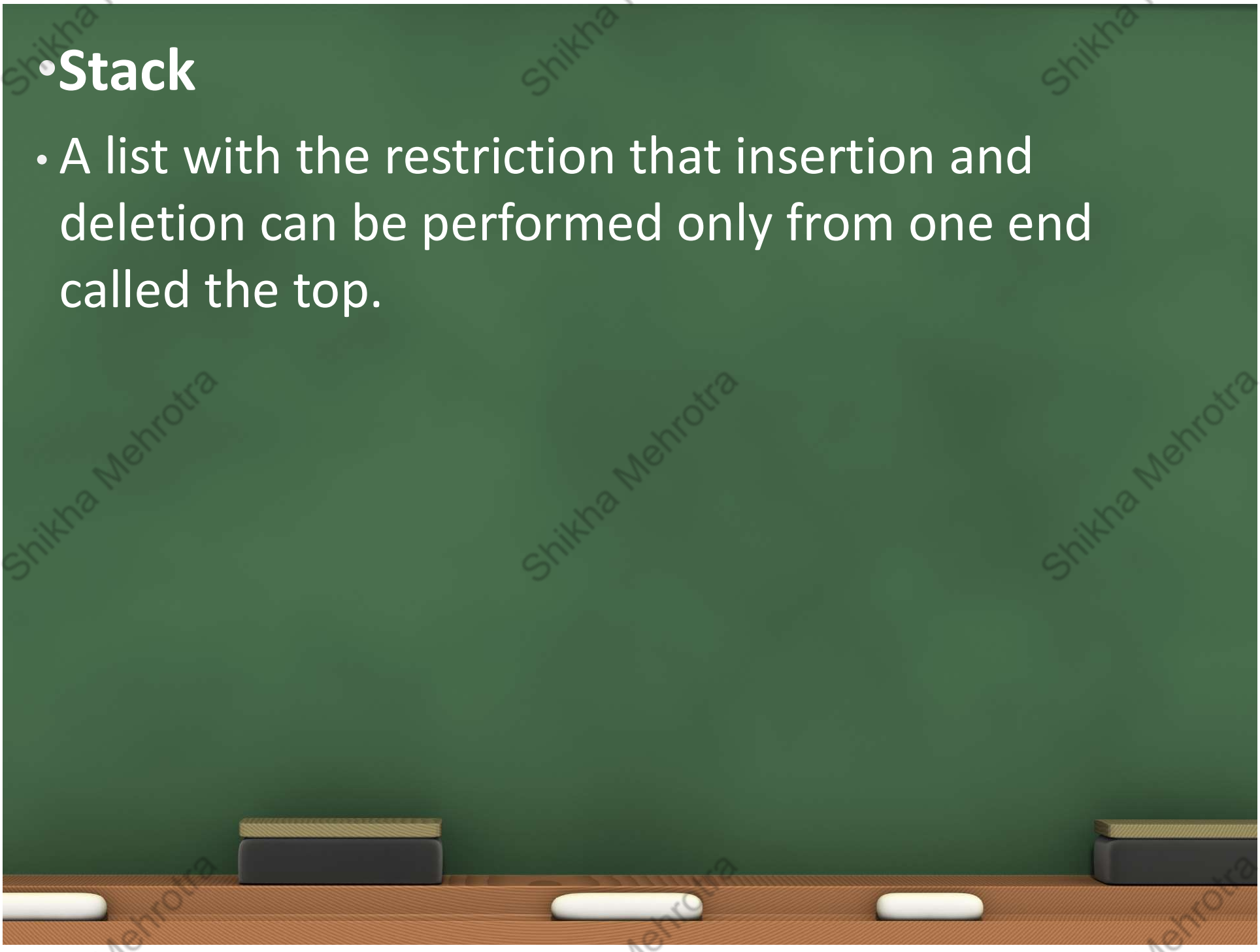
Dinner plates



Tower of Hanoi

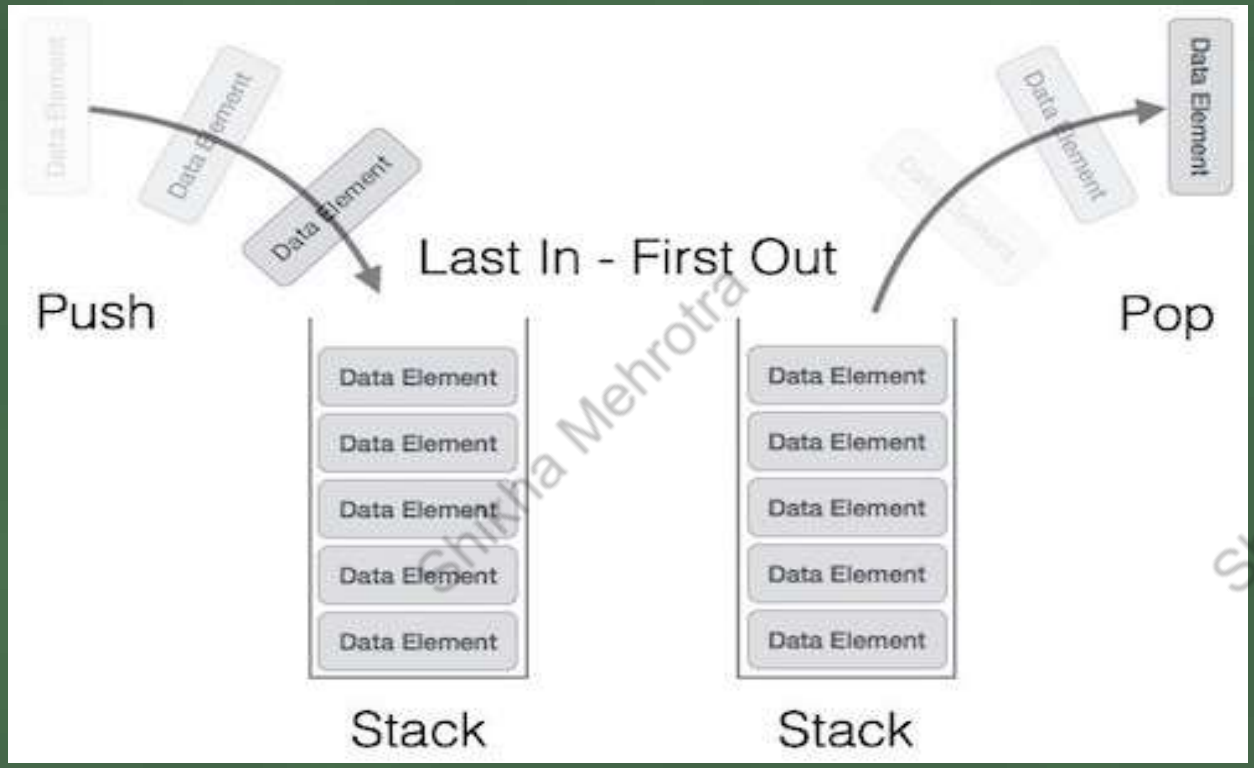
Pack of Tennis balls

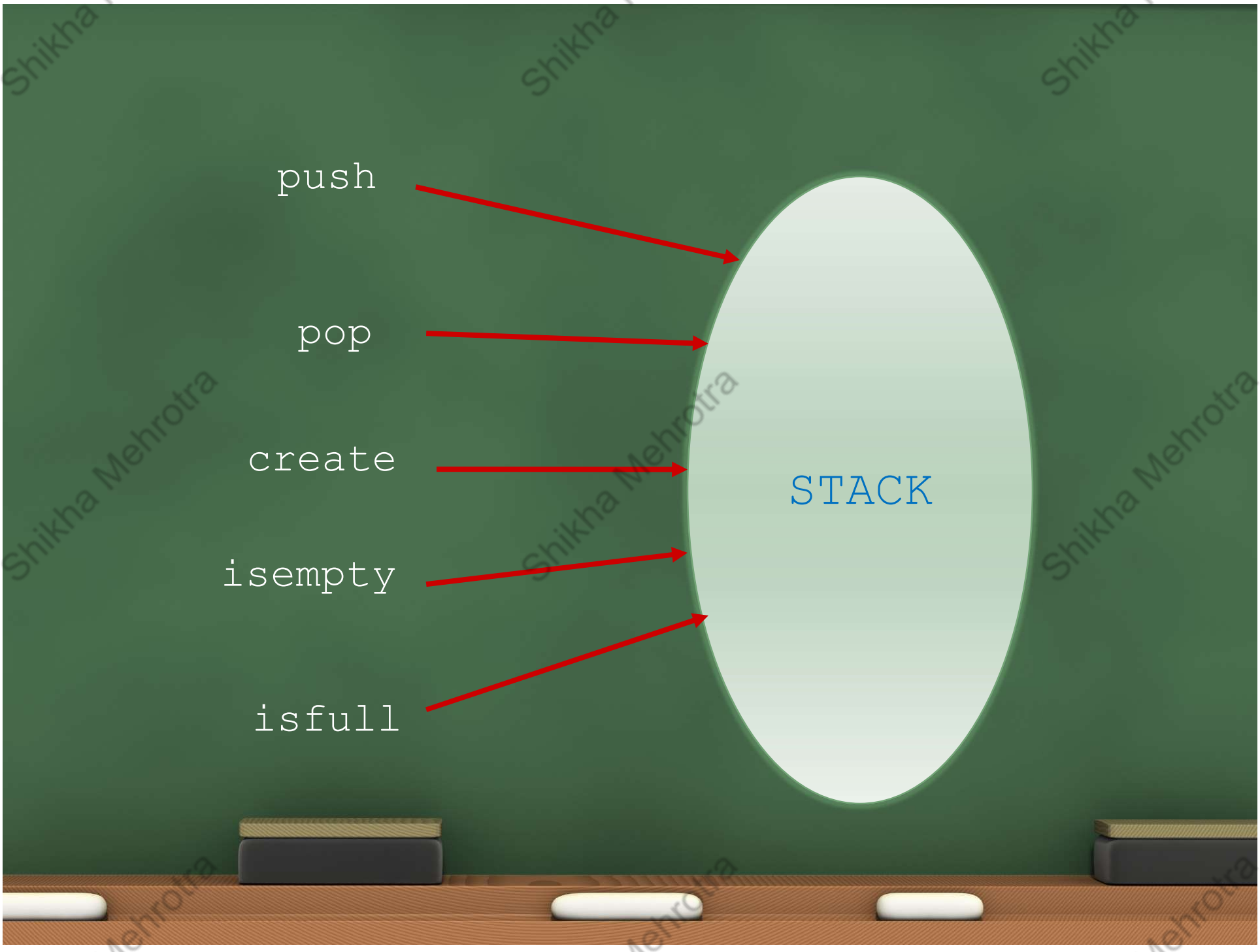
- **Stack**
  - A list with the restriction that insertion and deletion can be performed only from one end called the top.



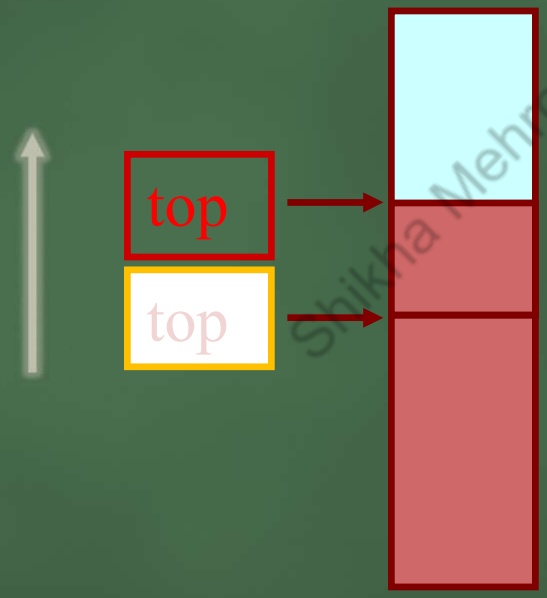
- **Stack**
- The basic implementation of stack is also called LIFO (Last In First Out)
  - It is a list like structure, but elements can be inserted or deleted from only one end. It makes stack less flexible than lists.
  - Many applications need simpler stack rather than lists.
- Only access to the stack is the top element
  - consider trays in a cafeteria
    - to get the bottom tray out, you must first remove all of the elements above

Stack Operations



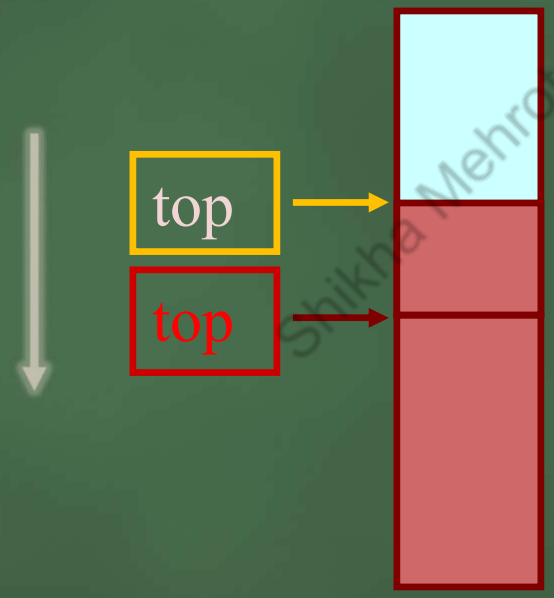

push
pop
create
isempty
isfull
STACK

Push using Stack



PUSH

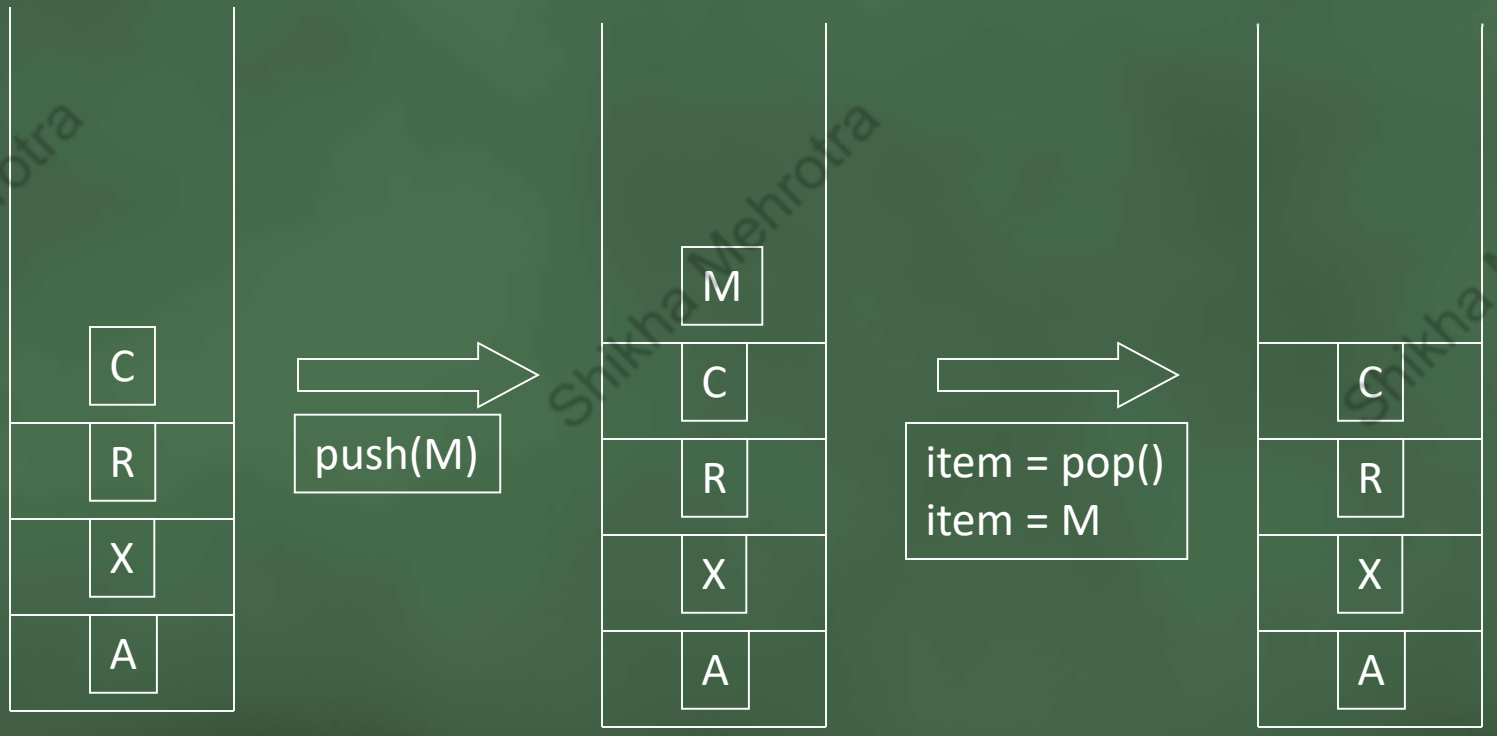
Pop using Stack



POP

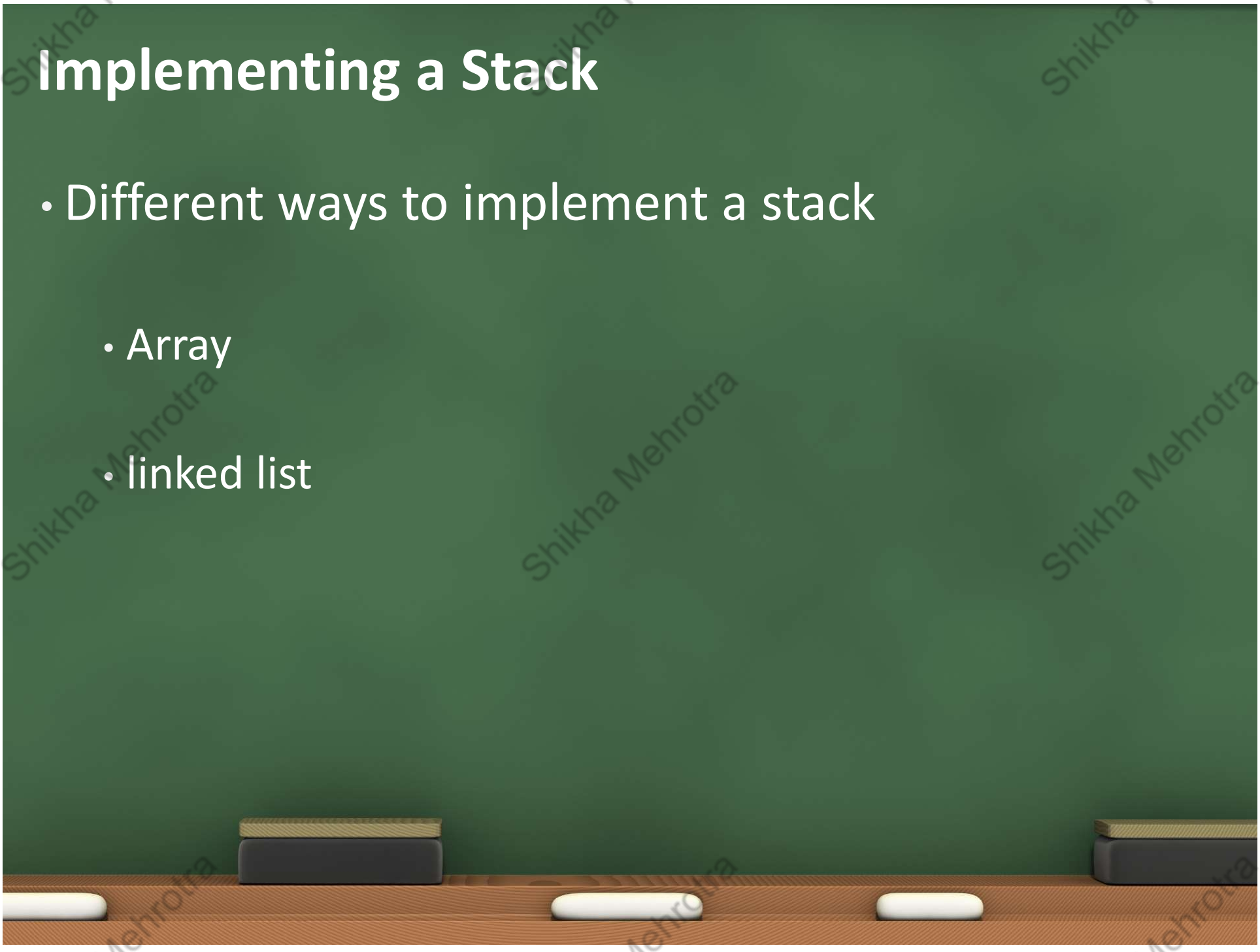
Stack Visualization

Watch Demo



Implementing a Stack

- Different ways to implement a stack
  - Array
  - linked list



Implementation of the Stack using Array

top

0 1 2 3 4 5 6 7 8 9

int A[10]

top == -1 // empty stack

Implementation of the Stack Operations

top

- Push(int x):


```
void push(int x)
{
 if(top==MAX_SIXE -1 )
          return;

 top  = top +1
 A [top] = x
 }
```

0 1 2 3 4 5 6 7 8 9

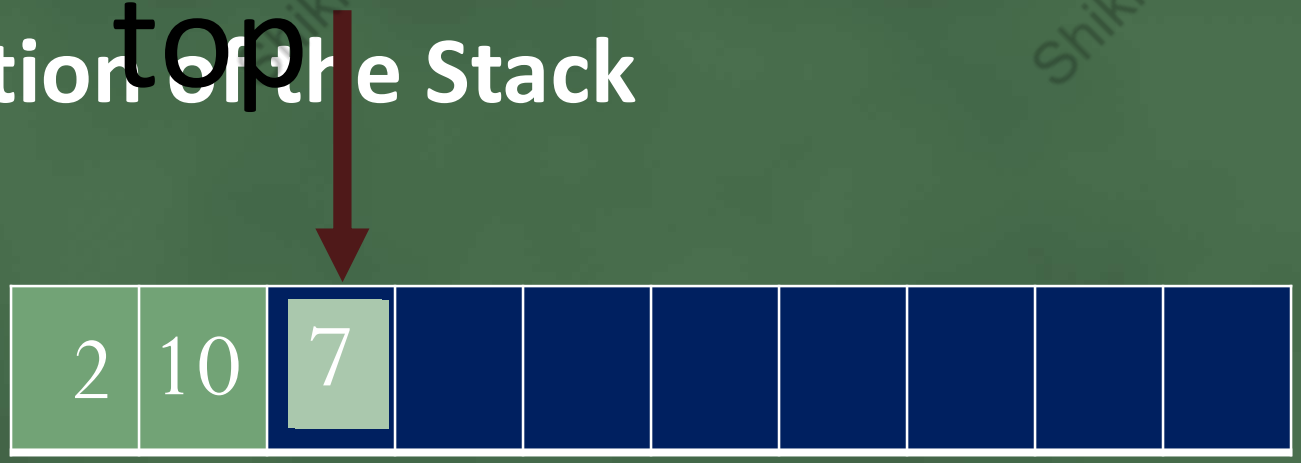
Push(2)

Push(10)

Push(5)

Implementation of the Stack Operations

- Pop():



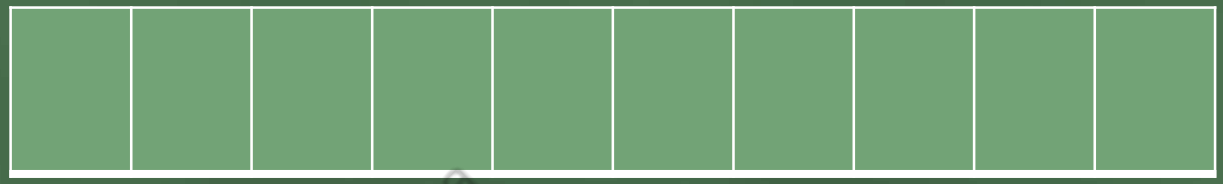

```
int pop()
{
   if(top==-1)
            return -1;
 top = top -1;
}
```

Pop()

Push (7)

Implementation of the Stack Operations

top



- IsEmpty():

```
int IsEmpty()
{
	if(top == -1)
		return 1;
	return 0;
}
```

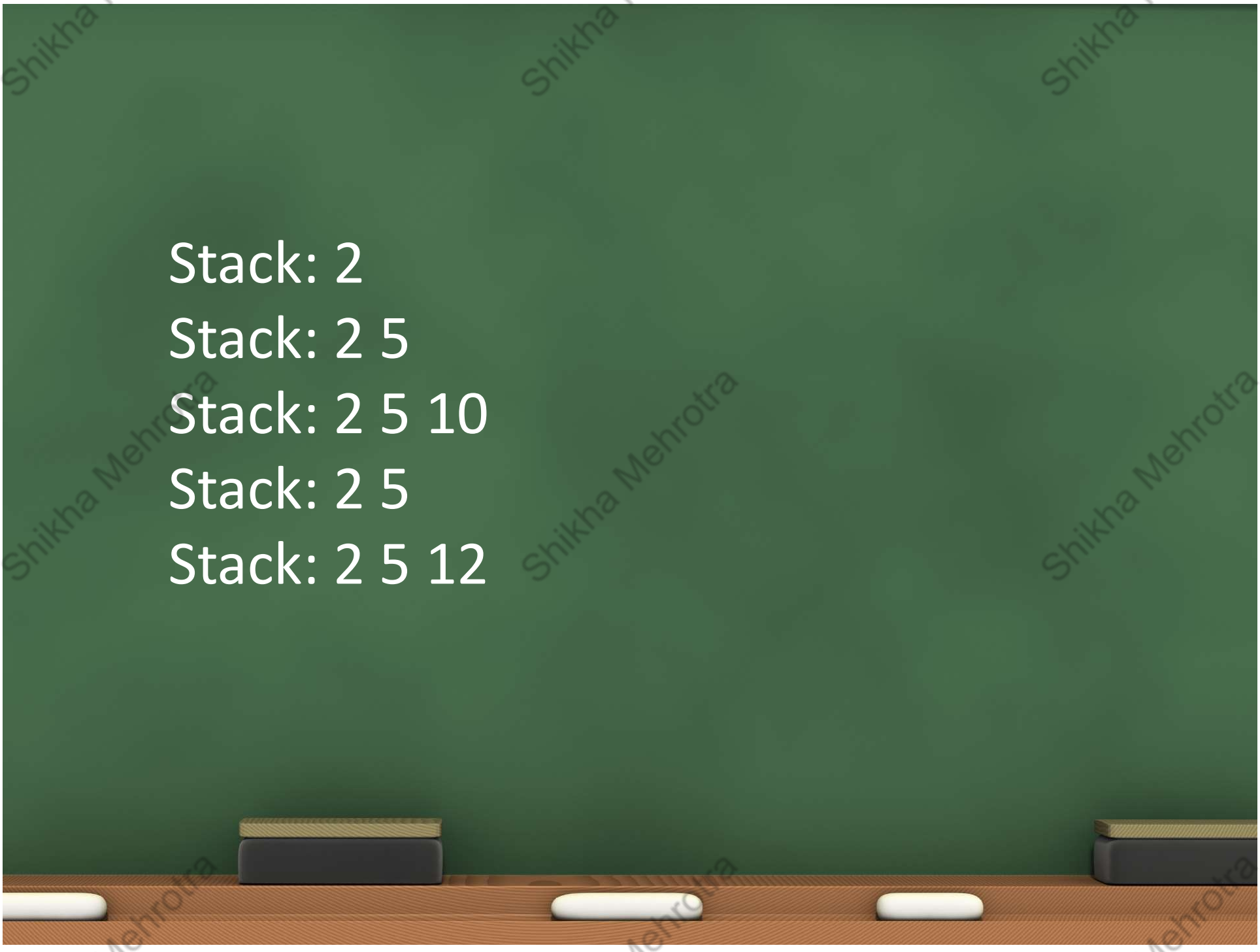
Implementation of the Stack Operations

Top() / peek()

```
int peek()
{
   if(top==-1)
            return -1;
    return A[top];
}
```

```
#include<stdio.h>

#define MAX_SIZE 101

int A[MAX_SIZE];

int top = -1;

void Push(int x)
{
  if(top == MAX_SIZE -1)
    { // overflow case.
          printf("Error: stack overflow\n") ;
          return;
}
A[++top] = x;
}

void Pop()
{
if(top == -1) {
          printf("Error: No element to pop\n");
          return;
}
top--;
}
```

```
int Top()
{
     return A[top];
}

int IsEmpty()
{
    if(top == -1) return 1;
    return 0;
}

void Print() {
int i;
printf("Stack: ");
for(i = 0;i<=top;i++)
printf("%d ",A[i]);
printf("\n");
}


int main() {
  // Code to test the implementation.
  // calling Print() after each push or pop to see the
Push(2);Print();
Push(5);Print();
Push(10);Print();
Pop();Print();
Push(12);Print();
}
```



Further Considerations

- What if static array initially allocated for stack is too small?
  - Terminate execution? 
  - Replace with larger array! 
- Creating a larger array
  - Allocate larger array
  - Use loop to copy elements into new array
  - Delete old array

Designing and Building a Stack class

- The basic functions are:
  - Constructor: construct an empty stack
  - isEmpty(): Examines whether the stack is empty or not
  - Push(): Add a value at the top of the stack
  - Top(): Read the value at the top of the stack
  - Pop(): Remove the value at the top of the stack
  - Display(): Displays all the elements in the stack

Applications of Stacks

- Direct applications:
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Validate XML
  - Function Calls / recursion
  - UNDO in an editor
  - Compilers
  - parsing data between delimiters (Balanced Parentheses)
  - { ( ) }
- Indirect applications:
  - Auxiliary data structure for algorithms
  - Component of other data structures