CLOS: 01, 03

Stack implementation in C++

Stack is the fundamental data structure used in computer science to store collections of objects. It can operate on the **Last In, First Out (LIFO)** principle, where the most recently added object is the first one to be removed. It can make the stacks incredibly useful in situations where you reverse a series of the operations or repeatedly undo operations.

Stack Data Structure in C++

A stack can be visualized as the vertical stack of the elements, like the stack of the plates. We can only add or remove the top plate. Similarly, in the stack data structures, elements can add to and removed from the top of the stack.

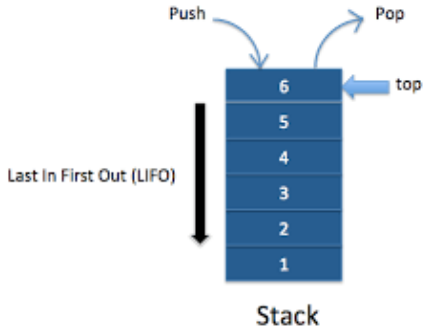

Figure 1: Stack

Stacks can be implemented using the arrays or linked lists:

1. **Array-based implementation**: It can use a simple array to store the elements of the stack. The pointer is used to keep the top of the stack.

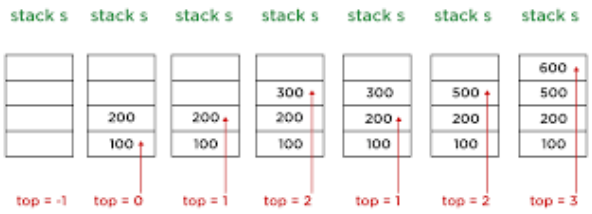

Figure 2: Stack as Array

Operation	Description
push(x)	Add element x to the top of the stack
pop()	Removes the top element
peek() / top()	Returns the top element without removing it
isEmpty()	Returns true if the stack is empty
isFull()	Returns true if the stack is full (array-based)
size()	Returns the number of elements in the stack

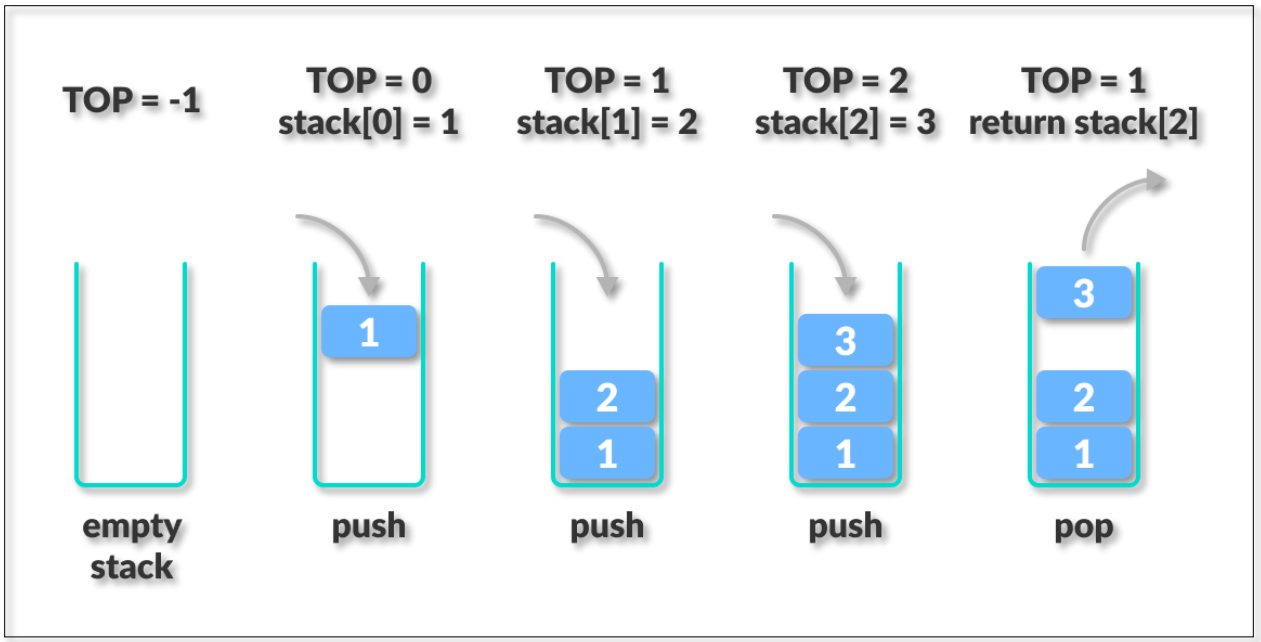

Figure 3: Stack Operation

A. Stack Using Array

```
#include <iostream>
using namespace std;

#define MAX 5
class Stack {
    int arr[MAX];
    int top;
public:
    Stack() { top = -1; }
```

```
    bool isEmpty() { return top == -1; }
    bool isFull() { return top == MAX - 1; }

    void push(int val) {
        if (isFull()) cout << "Stack Overflow!\n";
        else arr[++top] = val;
    }

    void pop() {
        if (isEmpty()) cout << "Stack Underflow!\n";
        else top--;
    }

    int peek() {
        if (!isEmpty()) return arr[top];
        cout << "Stack Empty!\n";
        return -1;
    }

    void display() {
        cout << "Stack: ";
        for (int i = top; i >= 0; i--) cout << arr[i] << " ";
        cout << endl;
    }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
```

```
    s.push(30);
    s.display();
    s.pop();
    s.display();
}
```

B. Stack Using Linked List

Linked List based implementation: Each element in the stack is the node in a linked list. The top of the stack is simply the head of the linked list.

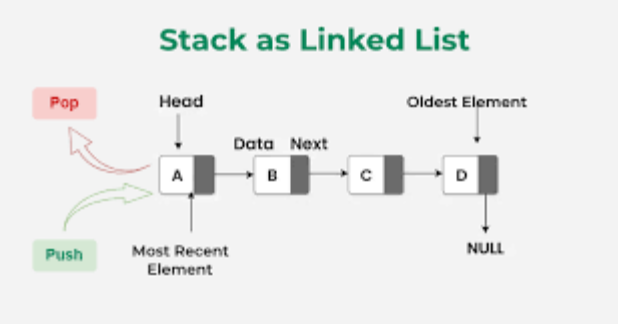

Figure 4 : Stack as Linked list

```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) { data = val; next = NULL; }
};

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

```
    bool isEmpty() { return top == NULL; }

    void push(int val) {
        Node* newNode = new Node(val);
        newNode->next = top;
        top = newNode;
    }

    void pop() {
        if (isEmpty()) {
            cout << "Stack Underflow!\n";
            return;
        }
        Node* temp = top;
        top = top->next;
        delete temp;
    }

    int peek() {
        if (!isEmpty()) return top->data;
        cout << "Stack Empty!\n";
        return -1;
    }

    void display() {
        cout << "Stack: ";
        for (Node* t = top; t != NULL; t = t->next)
            cout << t->data << " ";
        cout << endl;
    }
};
```

```
int main() {
    Stack s;
    s.push(5);
    s.push(10);
    s.push(15);
    s.display();
    s.pop();
    s.display();
}
```

Applications of Stack

Application	Description
Function Calls	Manages return addresses and local variables
Undo/Redo	Keeps history of user actions
Browser Navigation	Back/forward buttons use stacks
Balanced Parentheses	Validates brackets in expressions
Expression Conversion	Converts infix ↔ postfix/prefix

Practice Questions

1. **Balanced Parentheses:** Write a function that checks if every opening bracket **{[(** has a matching closing bracket **)]}**.

Input	Output
{{{()}}}{}(()){}	Accepted: Sequence is right

2. **Infix → Postfix Conversion:** Convert infix expression **(A+B)*C** into postfix form **A B + C ***.

   Hint: Use a stack for operators.

   Pop from the stack when precedence is higher or equal before pushing a new operator

3. **Infix → Prefix Conversion:** Convert **(A+B)*(C+D)** into prefix *** + A B + C D**.

   Steps:

   1. Reverse the infix expression
   2. Swap (with)
   3. Convert to postfix

4. Reverse the result → Prefix

4. **Evaluate Postfix Expression:** Example: **2 3 + 5 * = 25**

   Use a stack to push operands and pop two elements when encountering an operator.

5. **Complex Expression Conversion:**

Convert ((A+B)*C-(D-E))^(F+G)	**Output**: A B + C * D E - - F G + ^