📚 Basics of Stack

🔹 Introduction to Stack

- **Stack** = A linear data structure.
- **Order → LIFO (Last In, First Out).**
- Example:
  - 🍽️ Plates stacked in a cafeteria.
  - ↩️ Undo operation in text editor.
  - 🌐 Browser back button.

🔹 Stack Representation

1️⃣ Stack using Arrays

👉 Here stack size is fixed.

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

#define MAX 5  // maximum size of stack

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

    // push operation
    void push(int x) {
        if (top == MAX - 1) {
            cout << "Stack Overflow\n";
            return;
        }
```

```
        arr[++top] = x;
        cout << x << " pushed into stack\n";
    }

    // pop operation
    void pop() {
        if (top == -1) {
            cout << "Stack Underflow\n";
            return;
        }
        cout << arr[top--] << " popped from stack\n";
    }

    // peek operation
    int peek() {
        if (top == -1) {
            cout << "Stack is Empty\n";
            return -1;
        }
        return arr[top];
    }

    // check empty
    bool isEmpty() {
        return (top == -1);
    }

    // check full
    bool isFull() {
        return (top == MAX - 1);
    }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);

    cout << "Top element: " << s.peek() << endl;

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

```
    cout << "Is Empty? " << (s.isEmpty() ? "Yes" : "No") << endl;
    return 0;
}
```

2️⃣ Stack using Linked List

👉 Here stack size is dynamic.

```
#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; }

    // push operation
    void push(int x) {
        Node* newNode = new Node(x);
        newNode->next = top;
        top = newNode;
        cout << x << " pushed into stack\n";
    }

    // pop operation
    void pop() {
        if (top == NULL) {
            cout << "Stack Underflow\n";
            return;
        }
        cout << top->data << " popped from stack\n";
        Node* temp = top;
        top = top->next;
        delete temp;
    }
```

```
    // peek operation
    int peek() {
        if (top == NULL) {
            cout << "Stack is Empty\n";
            return -1;
        }
        return top->data;
    }

    // check empty
    bool isEmpty() {
        return (top == NULL);
    }
};

int main() {
    Stack s;
    s.push(5);
    s.push(15);
    s.push(25);

    cout << "Top element: " << s.peek() << endl;

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

    cout << "Is Empty? " << (s.isEmpty() ? "Yes" : "No") << endl;
    return 0;
}
```

⚙ Stack using STL `stack` (Direct way)

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

int main() {
    stack<int> st;

    st.push(10);   // push()
    st.push(20);
    st.push(30);

    cout << "Top element: " << st.top() << endl; // top()
```

```
    st.pop(); // pop()

    cout << "Stack size: " << st.size() << endl; // size()
    cout << "Is empty? " << (st.empty() ? "Yes" : "No") << endl; // empty()
}
```

⚙ Stack using Array (STL vector)

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

class Stack {
    vector<int> v;
public:
    void push(int x) { v.push_back(x); }
    void pop() { if(!v.empty()) v.pop_back(); }
    int top() { return v.empty() ? -1 : v.back(); }
    bool empty() { return v.empty(); }
    int size() { return v.size(); }
};

int main() {
    Stack st;
    st.push(10);
    st.push(20);
    cout << "Top: " << st.top() << endl;
    st.pop();
    cout << "Size: " << st.size() << endl;
}
```

⚙ Stack using Linked List (STL list)

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

class Stack {
    list<int> l;
public:
    void push(int x) { l.push_back(x); }
    void pop() { if(!l.empty()) l.pop_back(); }
    int top() { return l.empty() ? -1 : l.back(); }
    bool empty() { return l.empty(); }
    int size() { return l.size(); }
};
```

```
int main() {
    Stack st;
    st.push(5);
    st.push(15);
    cout << "Top: " << st.top() << endl;
    st.pop();
    cout << "Size: " << st.size() << endl;
}
```

⚙ Stack using Two Queues (STL queue)

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

class Stack {
    queue<int> q1, q2;
public:
    void push(int x) {
        q2.push(x);
        while(!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
        }
        swap(q1, q2);
    }
    void pop() { if(!q1.empty()) q1.pop(); }
    int top() { return q1.empty() ? -1 : q1.front(); }
    bool empty() { return q1.empty(); }
    int size() { return q1.size(); }
};

int main() {
    Stack st;
    st.push(100);
    st.push(200);
    cout << "Top: " << st.top() << endl;
    st.pop();
    cout << "Size: " << st.size() << endl;
}
```

⚙ Stack using Single Queue (STL queue)

```
#include <iostream>
#include <queue>
```

```cpp
using namespace std;

class Stack {
    queue<int> q;
public:
    void push(int x) {
        q.push(x);
        for(int i = 0; i < (int)q.size()-1; i++) {
            q.push(q.front());
            q.pop();
        }
    }
    void pop() { if(!q.empty()) q.pop(); }
    int top() { return q.empty() ? -1 : q.front(); }
    bool empty() { return q.empty(); }
    int size() { return q.size(); }
};

int main() {
    Stack st;
    st.push(1);
    st.push(2);
    st.push(3);
    cout << "Top: " << st.top() << endl;
    st.pop();
    cout << "Size: " << st.size() << endl;
}
```