Saddam Tech DSA Web Series – Episode 01

Stack

1. What is a Stack? (Introduction to Stack)

A **stack** is a **linear data structure** that follows the **LIFO (Last In, First Out)** principle, meaning the last element added is the first to be removed.

- **Key Characteristics of Stack:**
  - **Linear Structure** – Stores data sequentially.
  - **LIFO Principle** – Last element inserted is removed first.
  - **Restricted Access** – Insertion and deletion happen only at the top.
  - **Efficient Operations** – Push, Pop, and Peek operations take **O(1) time**.
  - **Memory Allocation** – Can be implemented using **arrays** (fixed size) or **linked lists** (dynamic size).
- **Real-World Analogy of Stack:**
  - **Stack of Plates** – The last plate placed is the first to be removed.
  - **Undo/Redo in Text Editors** – The most recent action is undone first.
  - **Browser History (Back Button)** – The last visited page is revisited first.

---

2. Stack Operations in C

Stacks support five primary operations:

1. Push(x) – Insert an Element

- Adds an element x to the **top of the stack**.
- Updates the **top pointer**.
- **Time Complexity: O(1)**

2. Pop() – Remove the Top Element

- Removes the **top element** from the stack.
- If the stack is empty, it triggers an **underflow condition**.
- **Time Complexity: O(1)**

3. Peek() – Get the Top Element

- Returns the **top element** without removing it.
- Used when checking the last inserted element.
- **Time Complexity: O(1)**

4. isEmpty() – Check if Stack is Empty

- Returns 1 (true) if the stack is **empty**, otherwise 0 (false).
- **Time Complexity: O(1)**

5. isFull() – Check if Stack is Full

- Returns 1 (true) if the stack is **at maximum capacity**, otherwise 0 (false).
- **Time Complexity: O(1)**

---

3. Stack Implementation in C

🔹 **1. Stack Using Array (Static Implementation)**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5  // Maximum size of stack
int stack[MAX], top = -1;

// Push Operation
void push(int value) {
    if (top == MAX - 1) {
        printf("Stack Overflow! Cannot push %d\n", value);
        return;
    }
    stack[++top] = value;
    printf("%d pushed to stack\n", value);
}

// Pop Operation
int pop() {
    if (top == -1) {
        printf("Stack Underflow! Cannot pop\n");
        return -1;
    }
    return stack[top--];
}

// Peek Operation
int peek() {
    if (top == -1) {
        printf("Stack is empty!\n");
        return -1;
    }
    return stack[top];
}
```

```
33 |// Check if Stack is Empty
34 |int isEmpty() {
35 |    return (top == -1);
36 |}
37 |
38 |// Display Stack
39 |void display() {
40 |    if (top == -1) {
41 |        printf("Stack is empty!\n");
42 |        return;
43 |    }
44 |    printf("Stack elements: ");
45 |    for (int i = 0; i <= top; i++) {
46 |        printf("%d ", stack[i]);
47 |    }
48 |    printf("\n");
49 |}
50 |
51 |int main() {
52 |    push(10);
53 |    push(20);
54 |    push(30);
55 |    display();
56 |    printf("Popped element: %d\n", pop());
57 |    printf("Top element: %d\n", peek());
58 |    display();
59 |    return 0;
60 |}
```

OUTPUT :

```
10 PUSHED TO STACK
20 PUSHED TO STACK
30 PUSHED TO STACK
STACK ELEMENTS: 10 20 30
POPPED ELEMENT: 30
TOP ELEMENT: 20
STACK ELEMENTS: 10 20
```

🔹 2. Stack Using Pointers (Dynamic Implementation)

```
1 | #include <stdio.h>
2 |
3 | #define Size 6
4 |
5 | struct Stack
6 | {
7 |     int arr[Size];
8 |     int top;
9 | };
10 |
11 |void push(struct Stack * ptr, int Value){
12 |    if(ptr->top<Size){
13 |        ptr->top++;
14 |        ptr->arr[(ptr->top)]=Value;
15 |    }
```

```
16 |     else{
17 |         printf("Overflow");
18 |     }
19 |}
20 |
21 |void pop(struct Stack * ptr){
22 |     if((ptr->top)>=0){
23 |         ptr->top--;
24 |     }
25 |     else{
26 |         printf("Stack Underflow");
27 |     }
28 |}
29 |
30 |void peek(struct Stack * ptr){
31 |     printf("Top Value is :%d",  ptr->arr[ptr->top]);
32 |}
33 |
34 |void display(struct Stack * ptr){
35 |     if((ptr->top)>=0){
36 |         for(int i=0; i<=(ptr->top); i++){
37 |             printf("\n%d", ptr->arr[i]);
38 |         }
39 |     }
40 |}
41 |
42 |void main(){
43 |     struct Stack Stk;
44 |     Stk.top=-1;
45 |
46 |     // struct Stack Stk={ .top=-1};
47 |
48 |     push(&Stk , 10);
49 |     push(&Stk , 80);
50 |     push(&Stk , 90);
51 |     push(&Stk , 100);
52 |     push(&Stk , 110);
53 |     display(&Stk);
54 |     printf("\n----------");
55 |     pop(&Stk);
56 |     pop(&Stk);
57 |     pop(&Stk);
58 |     display(&Stk);
59 |     // peek(&Stk);
60 |}
```

OUTPUT :

```
10
80
90
100
110
----------
10
80
```

🔹 3. Stack Using Linked List (Dynamic Implementation)

```c
#include <stdio.h>
#include <stdlib.h>

struct Node {   // Node structure for Stack
    int data;
    struct Node* next;
};

// Pointer to top node
struct Node* top = NULL;

// Push Operation
void push(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Heap Overflow! Cannot push %d\n", value);
        return;
    }

    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("%d pushed to stack\n", value);
}

// Pop Operation
int pop() {
    if (top == NULL) {
        printf("Stack Underflow! Cannot pop\n");
        return -1;
    }
    struct Node* temp = top;
    int poppedValue = temp->data;
    top = top->next;
    free(temp);
    return poppedValue;
}

// Peek Operation
int peek() {
    if (top == NULL) {
        printf("Stack is empty!\n");
        return -1;
    }
    return top->data;
}

// Display Stack
void display() {
    struct Node* temp = top;
    if (temp == NULL) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
```

```
59 |     printf("\n");
60 | }
61 |
62 | int main() {
63 |     push(5);
64 |     push(15);
65 |     push(25);
66 |     display();
67 |     printf("Popped element: %d\n", pop());
68 |     printf("Top element: %d\n", peek());
69 |     display();
70 |     return 0;
71 | }
```

OUTPUT:

```
5 PUSHED TO STACK
15 PUSHED TO STACK
25 PUSHED TO STACK
STACK ELEMENTS: 25 15 5
POPPED ELEMENT: 25
TOP ELEMENT: 15
STACK ELEMENTS: 15 5
```

---

4 Stack Applications in C

- **Function Call Stack**
  - **Recursive functions use stacks to store function calls.**

```
1 | int factorial(int n) {
2 |     if (n == 0) return 1;
3 |     return n * factorial(n - 1);
4 | }
```

- **Next Greater Element Problem**
  - **Finds the next greater element for each number in an array.**

```
1 | void nextGreaterElement(int arr[], int n) {
2 |     int stack[n], top = -1, result[n];
3 |
4 |     for (int i = n - 1; i >= 0; i--) {
5 |         while (top != -1 && arr[i] >= stack[top]) {
6 |             top--;
7 |         }
8 |         result[i] = (top == -1) ? -1 : stack[top];
9 |         stack[++top] = arr[i];
10 |    }
11 |
12 |    for (int i = 0; i < n; i++) {
13 |        printf("%d -> %d\n", arr[i], result[i]);
14 |    }
15 | }
```

✅ **Example:**

```
int arr[] = {4, 5, 2, 25};
nextGreaterElement(arr, 4);
```

✅ **Output:**

```
4 -> 5
5 -> 25
2 -> 25
25 -> -1
```

---

5. Summary

- **Stack follows LIFO (Last In, First Out).**
- **Operations:** Push, Pop, Peek, isEmpty, isFull.
- **Implementation:** Arrays, Linked Lists.
- **Applications:** Function Calls, Undo/Redo, DFS, Expression Evaluation.
- **Advanced:** Next Greater Element, Bracket Matching.

🚀 **Mastering stacks in C is essential for coding interviews and DSA preparation!**

🔗 **Follow Me on Social Media:**

💼 LinkedIn: https://www.linkedin.com/in/saddamskst/

💬 Telegram: t.me/SaddamTechs

💻 GitHub: https://github.com/saddamskst/

🌐 Website: https://skst.in/

📸 Instagram: https://www.instagram.com/saddamskst/

🐦 Twitter/X: https://twitter.com/saddamskst