# Teaching Notes – Stack Implementation in C

## 4. Array Implementation of Stack

Concept:
A stack is a linear data structure that follows the LIFO (Last In First Out) principle.
Operations: PUSH (insert), POP (remove), PEEK (view top).

Using Static Arrays:
1. Define a fixed-size array (say MAX = 5).
2. Use a variable top initialized to -1 (empty stack).
3. PUSH: check overflow, increment top, insert element.
4. POP: check underflow, decrement top.
5. PEEK: return element at stack[top].

C Program – Array Implementation of Stack:

```
#include <stdio.h>
#define MAX 5

int stack[MAX];
int top = -1;

void push(int val) {
    if (top == MAX - 1) {
        printf("Stack Overflow!\n");
    } else {
        stack[++top] = val;
        printf("%d pushed into stack.\n", val);
    }
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow!\n");
    } else {
        printf("%d popped from stack.\n", stack[top--]);
    }
}

void peek() {
    if (top == -1) {
        printf("Stack is Empty!\n");
    } else {
        printf("Top element: %d\n", stack[top]);
    }
}

void display() {
    if (top == -1) {
        printf("Stack is Empty!\n");
    } else {
        printf("Stack elements: ");
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

int main() {
    push(10);
    push(20);
    push(30);
    display();
    peek();
    pop();
    display();
    return 0;
}
```

Drawbacks & Limitations of Array Implementation:
1. Fixed size – cannot grow/shrink dynamically.
2. Wasted memory if stack not full.
3. Overflow occurs if array full even if memory available elsewhere.
4. Less flexible compared to Linked List implementation.

## 5. Multiple Stacks

Concept: Store more than one stack in a single array to optimize memory usage.

Methods:
1. Fixed Division Method – Divide array into equal parts for each stack.
Drawback: Memory waste if one stack is empty and other full.
2. Flexible Method – One stack grows from left, other from right. Stops when they meet.

C Program – Two Stacks in One Array:

```
#include <stdio.h>
#define MAX 10

int arr[MAX];
int top1 = -1;
int top2 = MAX;

void push1(int val) {
    if (top1 + 1 == top2) {
        printf("Stack Overflow!\n");
    } else {
        arr[++top1] = val;
        printf("%d pushed in Stack1.\n", val);
    }
}

void push2(int val) {
    if (top1 + 1 == top2) {
        printf("Stack Overflow!\n");
    } else {
        arr[--top2] = val;
        printf("%d pushed in Stack2.\n", val);
    }
}

void pop1() {
    if (top1 == -1) {
        printf("Stack1 Underflow!\n");
    } else {
        printf("%d popped from Stack1.\n", arr[top1--]);
    }
}

void pop2() {
    if (top2 == MAX) {
        printf("Stack2 Underflow!\n");
    } else {
        printf("%d popped from Stack2.\n", arr[top2++]);
    }
}

int main() {
    push1(10);
    push1(20);
    push2(50);
    push2(60);
    pop1();
    pop2();
    return 0;
}
```

Advantages of Multiple Stacks:
- Saves memory (no need for separate arrays).
- Efficient if both stacks grow towards each other.

Limitations of Multiple Stacks:
- Logic is more complex.
- Still bounded by fixed-size array.