understand and demonstrate Push and pop

Certainly, let's explore the implementation and demonstration of the push and pop operations in a stack.

Aim:

- To understand the concept of push and pop operations in a stack data structure and demonstrate their implementation using a C program.

Theory:

- A stack is an abstract data type (ADT) that follows the Last-In-First-Out (LIFO) principle. This means that the last element added to the stack will be the first one to be removed.

- **Push Operation**: The push operation adds an element to the top of the stack.
- **Pop Operation**: The pop operation removes the top element from the stack and returns it.

Procedure:

1. **Stack Representation:**
   - We'll use an array to represent the stack.
   - We'll need a `top` variable to keep track of the index of the topmost element in the stack.
   - Initially, `top` is set to -1, indicating an empty stack.
2. **Push Implementation:**
   - Check if the stack is full (i.e., `top` is at the maximum array index). If full, indicate a stack overflow error.
   - If not full, increment `top` and store the new element at `stack[top]`.
3. **Pop Implementation:**
   - Check if the stack is empty (i.e., `top` is -1). If empty, indicate a stack underflow error.
   - If not empty, retrieve the element at `stack[top]`, decrement `top`, and return the retrieved element.

Code:

C

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

```
#define MAX_SIZE 100

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

// Function to add an element to the stack
void push(int item) {
    if (top >= MAX_SIZE - 1) {
        printf("Error: Stack Overflow\n");
    } else {
        stack[++top] = item; // Pre-increment top, then assign value
        printf("%d pushed to the stack\n", item);
    }
}

// Function to remove and return the top element from the stack
int pop() {
    if (top < 0) {
        printf("Error: Stack Underflow\n");
        return -1; // Or handle the error in another appropriate way
    } else {
        int item = stack[top--]; // Retrieve value, then decrement top
        printf("%d popped from the stack\n", item);
        return item;
    }
}

int main() {
    push(10);
    push(20);
    push(30);

    printf("Popped element: %d\n", pop());
    printf("Popped element: %d\n", pop());

    return 0;
}
```

Output:

```
10 pushed to the stack
20 pushed to the stack
30 pushed to the stack
```

```
30 popped from the stack
Popped element: 30
20 popped from the stack
Popped element: 20
```

This demonstration illustrates the fundamental push and pop operations on a stack. The code provides error handling for overflow and underflow scenarios.

Feel free to ask if you'd like to explore more advanced stack operations or different implementations!