Day 20: Implement Stack

Gati Goyal

”Think like a stack: handle the last task first.”
— Anonymous

1 Introduction

A **Stack** is a linear data structure that follows the **Last In First Out (LIFO)** principle. The most recently added element is the first to be removed. It supports the following operations:

- **Push:** Add an element to the top of the stack.
- **Pop:** Remove the top element of the stack.
- **Peek/Top:** View the top element without removing it.

2 Applications of Stack

- Expression evaluation and conversion (e.g., infix to postfix).
- Backtracking algorithms (e.g., navigating a maze).
- Function call management in recursion.

3 Code

```c
#include <stdio.h>

#define MAX 100

// Define the Stack structure
typedef struct {
    int top;
    int data[MAX];
} Stack;

```

```c
// Push operation (pass by reference to modify the original stack)
void push(Stack *stack, int value) {
    if (stack->top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack->top++;
    stack->data[stack->top] = value;

    // Display the stack after the push
    printf("Stack after push: ");
    for (int i = 0; i <= stack->top; i++) {
        printf("%d ", stack->data[i]);
    }
    printf("\n");
}

// Pop operation (pass by reference to modify the original stack)
int pop(Stack *stack) {
    if (stack->top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    int poppedValue = stack->data[stack->top];
    stack->top--;

    // Display the stack after the pop
    printf("Stack after pop: ");
    for (int i = 0; i <= stack->top; i++) {
        printf("%d ", stack->data[i]);
    }
    printf("\n");

    return poppedValue;
}

// Main function to test stack operations
int main() {
    Stack stack;
    stack.top = -1; // Initialize the stack to be empty

    // Push elements onto the stack
    push(&stack, 5);
    push(&stack, 10);
    push(&stack, 15);

    // Pop elements from the stack
    printf("Popped: %d\n", pop(&stack)); // Should print 15
    printf("Popped: %d\n", pop(&stack)); // Should print 10
    printf("Popped: %d\n", pop(&stack)); // Should print 5

    // Try popping from an empty stack
    printf("Popped: %d\n", pop(&stack)); // Should print "Stack Underflow"

    return 0;
}
```

4 Stack operations: Visual Representation and Output

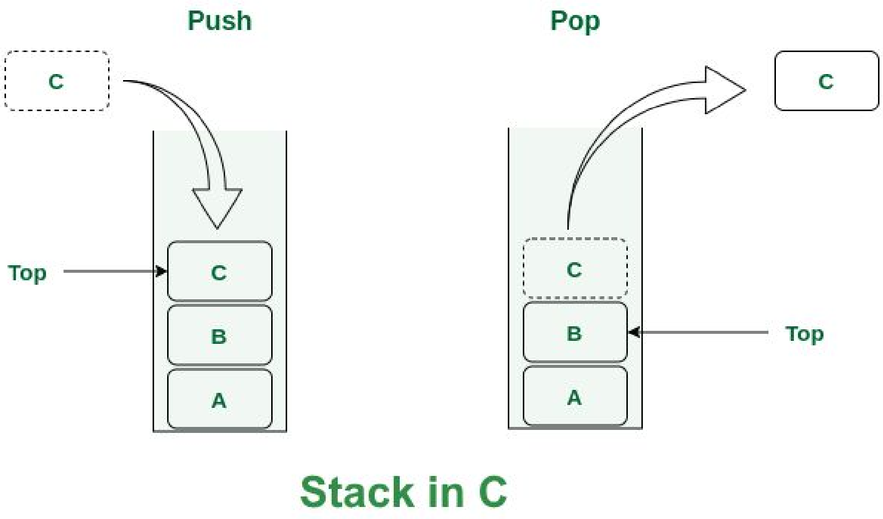
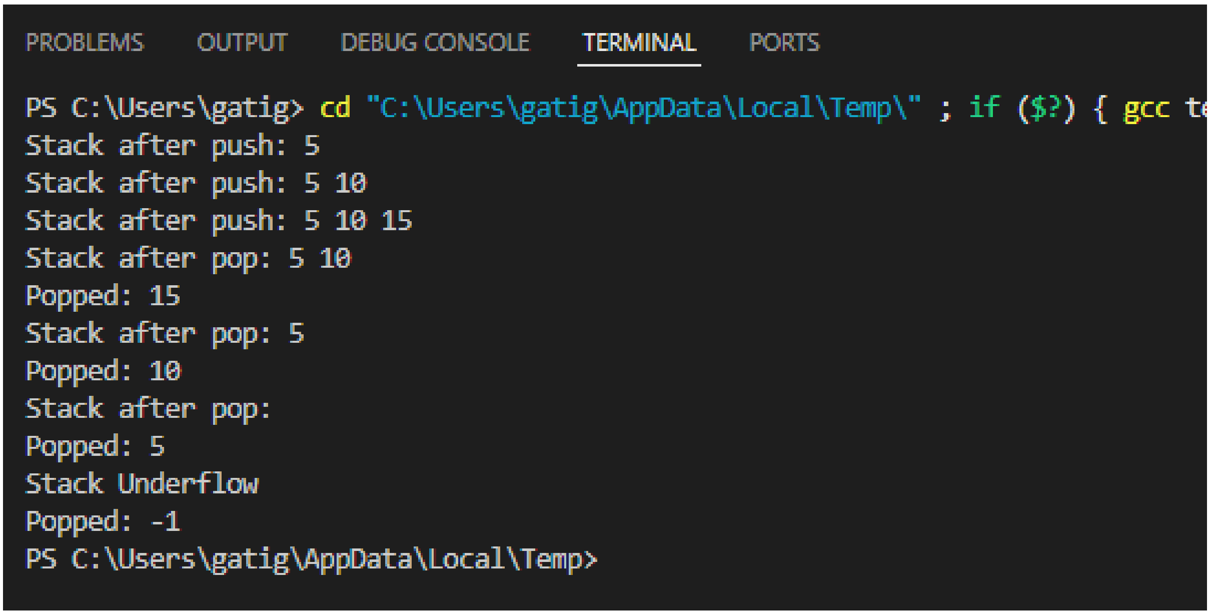


Figure 1: Stack Operations: Push and Pop

5 Conclusion

The stack data structure is an essential tool for implementing algorithms involving recursion, backtracking, and expression evaluation. Its simple LIFO approach is intuitive and effective for a wide range of use cases.



```
PS C:\Users\gatig> cd "C:\Users\gatig\AppData\Local\Temp\" ; if ($?) { gcc te
Stack after push: 5
Stack after push: 5 10
Stack after push: 5 10 15
Stack after pop: 5 10
Popped: 15
Stack after pop: 5
Popped: 10
Stack after pop:
Popped: 5
Stack Underflow
Popped: -1
PS C:\Users\gatig\AppData\Local\Temp>
```

Figure 2: Program Output for Stack