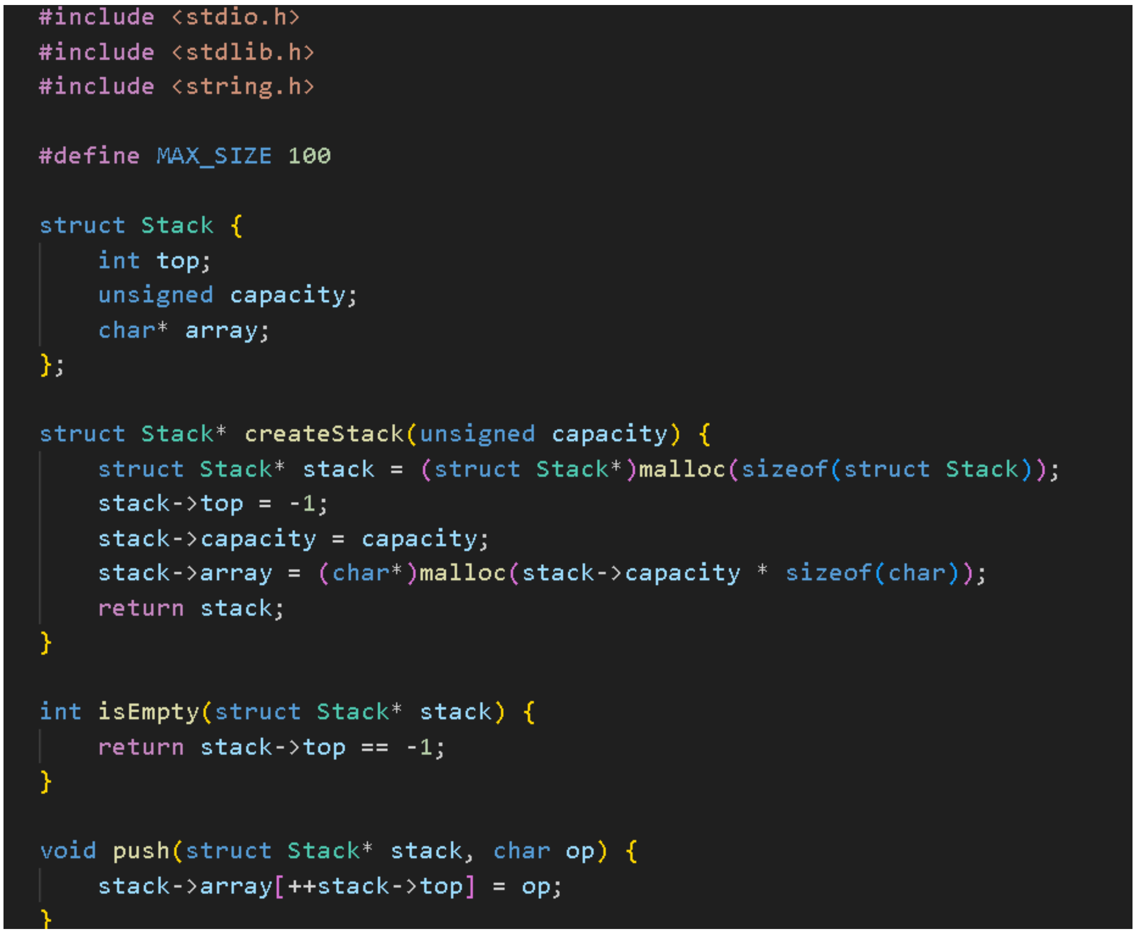
**DS – LAB 3 (28th Dec 2023)**

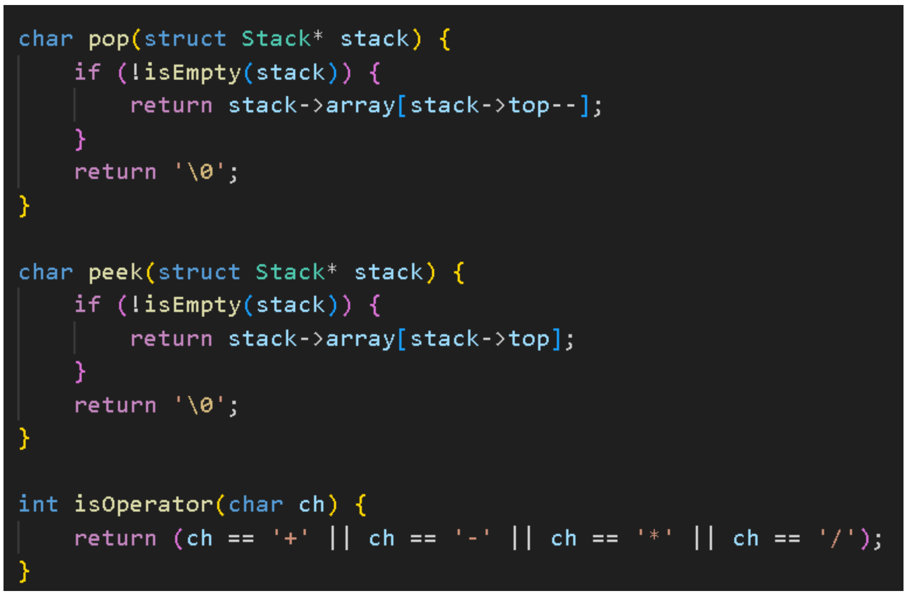
**Rishabh Kumar (1BM22CS221)**

1) Infix to postfix conversion using stacks

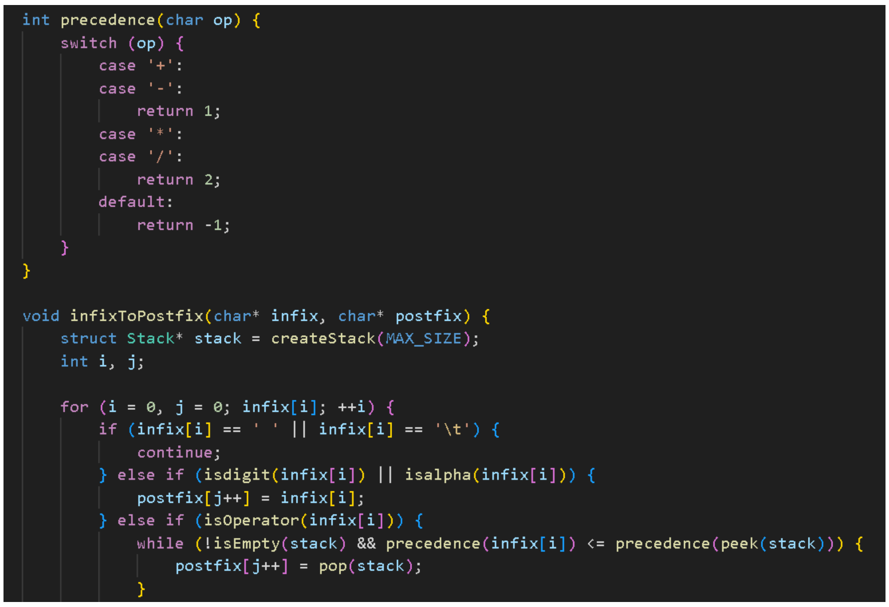
Code:



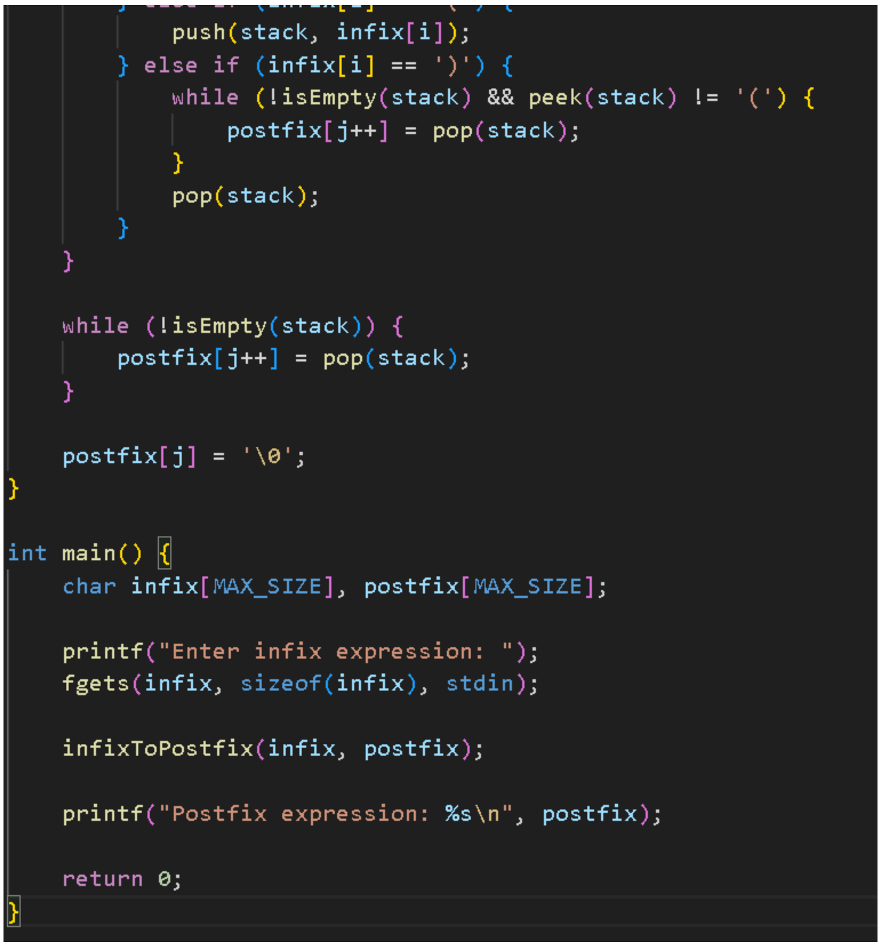
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 100

struct Stack {
    int top;
    unsigned capacity;
    char* array;
};

struct Stack* createStack(unsigned capacity) {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (char*)malloc(stack->capacity * sizeof(char));
    return stack;
}

int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

void push(struct Stack* stack, char op) {
    stack->array[++stack->top] = op;
}
```



```c
char pop(struct Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->array[stack->top--];
    }
    return '\0';
}

char peek(struct Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->array[stack->top];
    }
    return '\0';
}

int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}
```

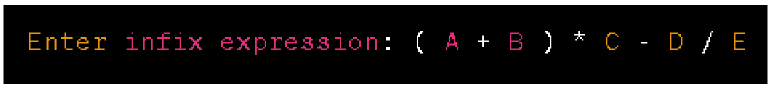


```c
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        default:
            return -1;
    }
}

void infixToPostfix(char* infix, char* postfix) {
    struct Stack* stack = createStack(MAX_SIZE);
    int i, j;

    for (i = 0, j = 0; infix[i]; ++i) {
        if (infix[i] == ' ' || infix[i] == '\t') {
            continue;
        } else if (isdigit(infix[i]) || isalpha(infix[i])) {
            postfix[j++] = infix[i];
        } else if (isOperator(infix[i])) {
            while (!isEmpty(stack) && precedence(infix[i]) <= precedence(peek(stack))) {
                postfix[j++] = pop(stack);
            }
```

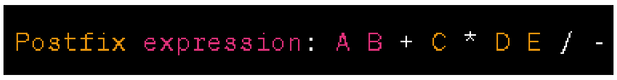


```c
                push(stack, infix[i]);
            } else if (infix[i] == ')') {
                while (!isEmpty(stack) && peek(stack) != '(') {
                    postfix[j++] = pop(stack);
                }
                pop(stack);
            }
    }

    while (!isEmpty(stack)) {
        postfix[j++] = pop(stack);
    }

    postfix[j] = '\0';
}

int main() {
    char infix[MAX_SIZE], postfix[MAX_SIZE];

    printf("Enter infix expression: ");
    fgets(infix, sizeof(infix), stdin);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

Output:

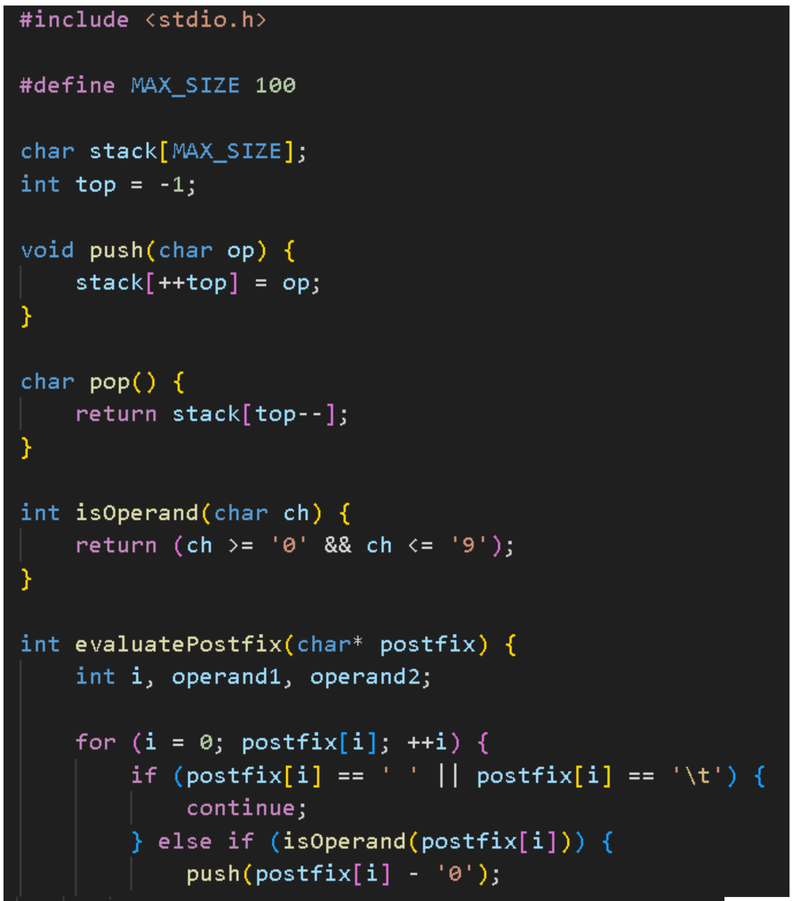


```
Enter infix expression: ( A + B ) * C - D / E
```

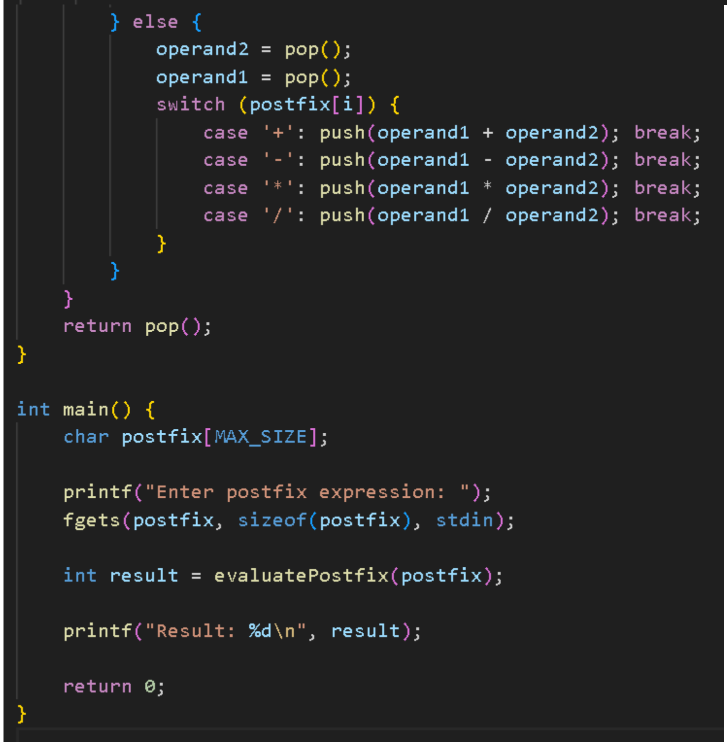


```
Postfix expression: A B + C * D E / -
```

2) Postfix evaluation using stacks

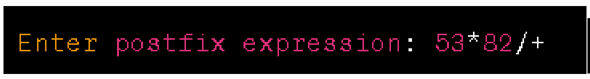


```c
#include <stdio.h>

#define MAX_SIZE 100

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

void push(char op) {
    stack[++top] = op;
}

char pop() {
    return stack[top--];
}

int isOperand(char ch) {
    return (ch >= '0' && ch <= '9');
}

int evaluatePostfix(char* postfix) {
    int i, operand1, operand2;

    for (i = 0; postfix[i]; ++i) {
        if (postfix[i] == ' ' || postfix[i] == '\t') {
            continue;
        } else if (isOperand(postfix[i])) {
            push(postfix[i] - '0');
```

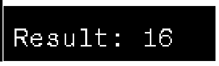


```c
        } else {
            operand2 = pop();
            operand1 = pop();
            switch (postfix[i]) {
                case '+': push(operand1 + operand2); break;
                case '-': push(operand1 - operand2); break;
                case '*': push(operand1 * operand2); break;
                case '/': push(operand1 / operand2); break;
            }
        }
    }
    return pop();
}

int main() {
    char postfix[MAX_SIZE];

    printf("Enter postfix expression: ");
    fgets(postfix, sizeof(postfix), stdin);

    int result = evaluatePostfix(postfix);

    printf("Result: %d\n", result);

    return 0;
}
```

Output



```
Enter postfix expression: 53*82/+
```



```
Result: 16
```