# Application of Stack - Infix to Postfix

*Source Code:*

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

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

// Function to check if the stack is empty
int isEmpty() {
    return top == -1;
}

// Function to check if the stack is full
int isFull() {
    return top == MAX - 1;
}

// Function to return the top element of the stack
char peek() {
    return stack[top];
}

// Function to pop an element from the stack
char pop() {
    if (isEmpty()) {
        return 0; // Return 0 for an empty stack (not the best ap
proach, consider using error handling)
    }
    char ch = stack[top];
    top--;
    return ch;
}

// Function to push an element onto the stack
void push(char a) {
    if (isFull()) {
        printf("Stack Full");
    } else {
        top++;
        stack[top] = a;
    }
}
```

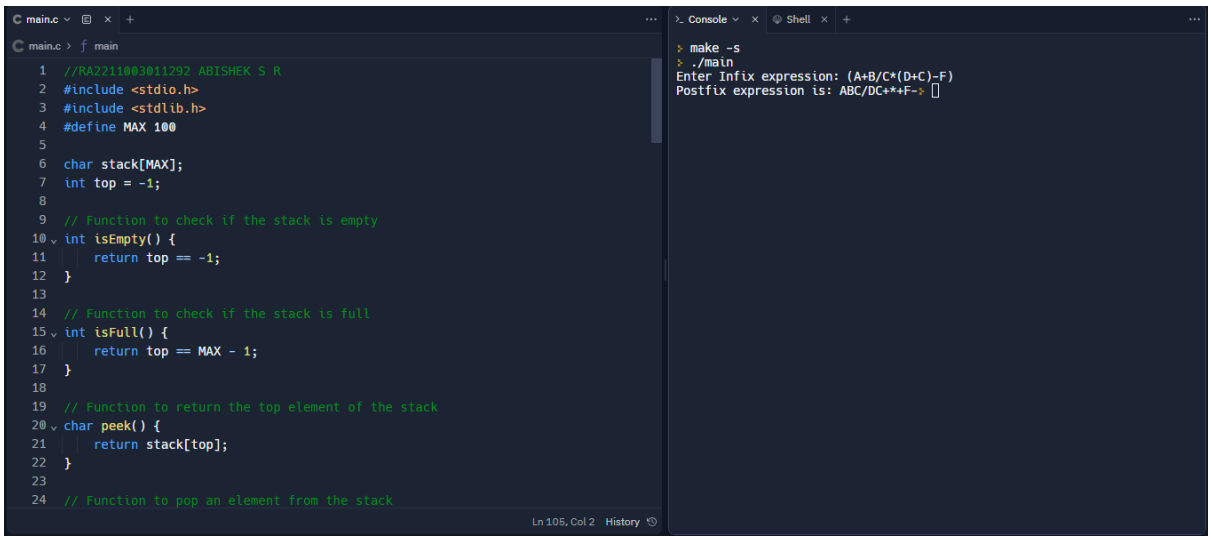
```
// Function to check if the given character is an operand
int checkIfOperand(char ch) {
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') |
| (ch >= '0' && ch <= '9');
}

// Function to compare the precedence of operators
int precedence(char ch) {
    switch (ch) {
        case '+':
        case '-':
            return 1;

        case '*':
        case '/':
            return 2;

        case '^':
            return 3;
    }
    return -1;
}

// Function for infix to postfix conversion
int convertInfixToPostfix(char *expr) {
    printf("Postfix expression is: ");
    int i, j;

    for (i = 0, j = -1; expr[i]; ++i) {
        if (checkIfOperand(expr[i])) {
            expr[++j] = expr[i];
        } else if (expr[i] == '(') {
            push(expr[i]);
        } else if (expr[i] == ')') {
            // Keep popping and adding to expression until openin
g pair is found
            while (!isEmpty() && peek() != '(') {
                expr[++j] = pop();
            }
            pop(); // Pop the '('
        } else {
            // If an operator, handle precedence
            while (!isEmpty() && precedence(expr[i]) <= precedenc
e(peek())) {
                expr[++j] = pop();
```

```
            }
            push(expr[i]);
        }
    }

    // Once all initial expression characters are traversed, add
all remaining elements from the stack to the expression
    while (!isEmpty()) {
        expr[++j] = pop();
    }
    expr[++j] = '\0';
    printf("%s", expr);
}

int main() {
    char expression[MAX];
    printf("Enter Infix expression: "); // Input infix expression
    scanf("%s", expression);
    convertInfixToPostfix(expression);
    return 0;
}
```

*Output:*

Enter Infix expression: (A+B/C*(D+C)-F)

Postfix expression is: ABC/DC+*+F-




**ABISHEK S R**

**RA2211003011292**