

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 approach, 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 opening pair is found
            while (!isEmpty() && peek() != '(') {
                expr[++j] = pop();
            }
            pop(); // Pop the '('
        } else {
            // If an operator, handle precedence
            while (!isEmpty() && precedence(expr[i]) <= precedence(peek())) {
                expr[++j] = pop();
            }
        }
    }
    expr[++j] = pop();
}

```

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

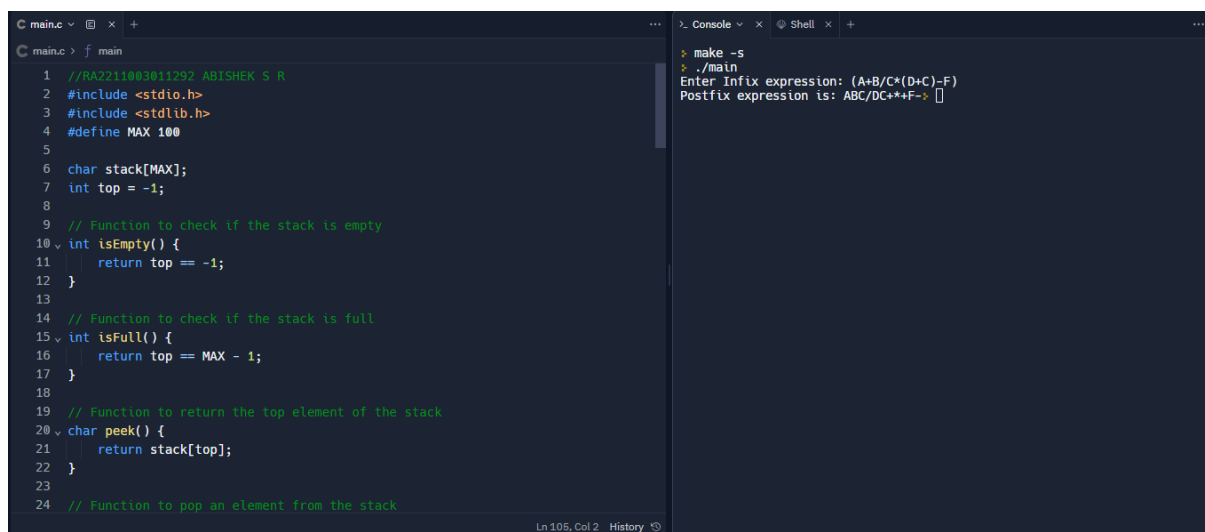
// Once all initial expression characters are traversed, add
all remaining elements from the stack to the expression
while (!isEmpty()) {
    expr[++j] = pop();
}
expr[++j] = '\0';
printf("%s", expr);
}

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-



The screenshot shows a C++ IDE with two panels. The left panel displays the source code for a program that converts an infix expression to a postfix expression using a stack. The code includes headers for `stdio.h` and `stdlib.h`, defines a stack size of 100, and implements functions for checking if the stack is empty or full, peeking at the top element, and popping an element. The `main` function prompts the user to enter an infix expression and calls the `convertInfixToPostfix` function. The right panel shows the terminal output, which displays the prompt 'Enter Infix expression: (A+B/C*(D+C)-F)' and the resulting postfix expression 'Postfix expression is: ABC/DC+*+F-'. The terminal also shows the commands `make -s` and `./main` being executed.

```
1 //RA2211003011292 ABISHEK S R
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define MAX 100
5
6 char stack[MAX];
7 int top = -1;
8
9 // Function to check if the stack is empty
10 int isEmpty() {
11     return top == -1;
12 }
13
14 // Function to check if the stack is full
15 int isFull() {
16     return top == MAX - 1;
17 }
18
19 // Function to return the top element of the stack
20 char peek() {
21     return stack[top];
22 }
23
24 // Function to pop an element from the stack
```

```
> make -s
> ./main
Enter Infix expression: (A+B/C*(D+C)-F)
Postfix expression is: ABC/DC+*+F-
```

ABISHEK S R

RA2211003011292