

**WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), \* (multiply) and / (divide)**

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

```
#include <ctype.h>
```

```
#include <string.h>
```

```
#define MAX 100
```

```
char stack[MAX];
```

```
int top = -1;
```

```
void push(char c) {
```

```
    if (top == MAX - 1) {
```

```
        printf("Stack Overflow\n");
```

```
    } else {
```

```
        top = top + 1;
```

```
        stack[top] = c;
```

```
    }
```

```
}
```

```
char pop() {
```

```
    char val;
```

```
    if (top == -1) {
```

```
        printf("Stack Underflow\n");
```

```
        return -1;
```

```
    } else {
```

```
        val = stack[top];
```

```
        top = top - 1;
```

```
        return val;
```

```
}
```

```
}
```

```
char peek() {
```

```
    if (top == -1)
```

```
        return '\0';
```

```
    return stack[top];
```

```

}

int precedence(char c) {
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
}

void infixToPostfix(char infix[], char postfix[]) {
    int i, k = 0;
    char c;

    for (i = 0; infix[i] != '\0'; i++) {
        c = infix[i];
        if (isalnum(c)) {
            postfix[k] = c;
            k = k + 1;
        }
        else if (c == '(') {
            push(c);
        }
        else if (c == ')') {
            while (top != -1 && peek() != '(') {
                postfix[k] = pop();
                k = k + 1;
            }
            pop();
        }
    }

    else {
        while (top != -1 && precedence(peek()) >= precedence(c)) {
            postfix[k] = pop();
            k = k + 1;
        }
    }
}

```

```
        }

        push(c);

    }

}

while (top != -1) {

    postfix[k] = pop();

    k = k + 1;

}

postfix[k] = '\0';

}

int main() {

    char infix[MAX], postfix[MAX];

    printf("Enter a valid parenthesized infix expression: ");

    scanf("%s", infix);

    infixToPostfix(infix, postfix);

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

    return 0;

}
```

## Output

The screenshot shows a terminal window with the following content:

```
PS: C:\Users\WISCHAL\OneDrive\Documents\Desktop\dsa> cd "c:\Users\WISCHAL\OneDrive\Documents\Desktop\dsa"
PS: C:\Users\WISCHAL\OneDrive\Documents\Desktop\dsa> if ($?) { gcc infixToPostfix.c -o infixToPostfix }
PS: C:\Users\WISCHAL\OneDrive\Documents\Desktop\dsa> ./infixToPostfix
Enter a valid parenthesized infix expression: a*(b*c)/d
Postfix Expression: abc*cd/
PS: C:\Users\WISCHAL\OneDrive\Documents\Desktop\dsa> [ ]
```

At the bottom of the terminal window, status indicators show: Line 86, Col 1, Spaces: 4, UTF-8, CRLF, Signed out, Go Live, Prettier.

The screenshot shows a full IDE interface with the following components:

- File Explorer:** Shows files in the DSA folder: infixToPostfix.c, stack.c, stack.exe, and Write a program to stimulate a working of st...
- Code Editor:** Displays the `infixToPostfix.c` file with the following code:

```
int main() {
    // ...
    infixToPostfix(infix, postfix);
    printf("Postfix Expression: %s\n", postfix);
    return 0;
}
```

- Terminal:** Shows the command-line interface with the following session:

```
PS: C:\Users\WISCHAL\OneDrive\Documents\Desktop\dsa> if ($?) { gcc infixToPostfix.c -o infixToPostfix }
PS: C:\Users\WISCHAL\OneDrive\Documents\Desktop\dsa> ./infixToPostfix
Enter a valid parenthesized infix expression: (1+7)*(5-2)
Postfix Expression: 1752-
PS: C:\Users\WISCHAL\OneDrive\Documents\Desktop\dsa> [ ]
```

At the bottom of the terminal window, status indicators show: Line 86, Col 1, Spaces: 4, UTF-8, CRLF, Signed out, Go Live, Prettier.