

2. 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>
```

```
#define MAX 100
```

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

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

```
// Push into stack
```

```
void push(char ch) {
```

```
    stack[++top] = ch;
```

```
}
```

```
// Pop from stack
```

```
char pop() {
```

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

```
}
```

```
// Check operator precedence
```

```
int precedence(char op) {
```

```
    if (op == '+' || op == '-')
```

```
        return 1;
```

```
    if (op == '*' || op == '/')
```

```

        return 2;
    return 0;
}

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

    int i = 0, k = 0;

    char ch;

    printf("Enter a valid parenthesized infix expression: ");
    scanf("%s", infix);

    while (infix[i] != '\0') {
        ch = infix[i];

        // If operand, add to postfix
        if (isalnum(ch)) {
            postfix[k++] = ch;
        }

        // If '(' push to stack
        else if (ch == '(') {
            push(ch);
        }

        // If ')', pop until '('
        else if (ch == ')') {
            while (stack[top] != '(') {
                postfix[k++] = pop();
            }

```

```

        pop(); // Remove '('
    }

    // If operator
    else {
        while (top != -1 && precedence(stack[top]) >= precedence(ch)) {
            postfix[k++] = pop();
        }
        push(ch);
    }
    i++;
}

// Pop remaining operators
while (top != -1) {
    postfix[k++] = pop();
}

postfix[k] = '\0';

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

return 0;
}

```

OUTPUT:

```
PS C:\Users\chait\OneDrive\Desktop\ds> cd "c:\Users\chait\  
Enter a valid parenthesized infix expression: (a+b)*(c-d)  
Postfix expression: ab+cd-*  
PS C:\Users\chait\OneDrive\Desktop\ds> 
```