

Date _____
Page _____

W.A.P 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).

Algorithm:

- Create an empty stack for the operators.
- Create an empty string for postfix operation.
- Read the infix expression.
- for each character ch in infix (infix expression)

a) if ch is an operand :

Add ch to postfin. string.

b) else if ch is '(' :

push '(' onto stack.

c) Else if ch is ')' :

while top of stack is not '(' :

pop from stack and add to postfin.

Pop ')' from stack.

d) If ch is an operator, push it onto stack.
~~else if ch is an operator~~

while stack is not empty.

AND

top of stack >= ch :

pop from stack and add to postfin.

push ch onto stack

e). while stack is not empty :

pop from stack & add to postfin.

- Print postfin expression.

```
#include <stdio.h> } (go next) numbered for  
#include <ctype.h> } (go next)  
#include <string.h> : 'H' (0)  
#define MAX 100 : 'L' (1)
```

```
char stack[MAX]; : 'T' (2)  
int top = -1; : 'V' (3)
```

```
void push(char c) { : 'A' (4)  
    if (top == MAX-1) {  
        printf("Stack overflow\n");  
        return; : 'B' (5)  
    }
```

```
    stack[++top] = c; : 'C' (6)  
}
```

```
char pop() { : 'D' (7)  
    if (top == -1) {  
        printf("Stack underflow\n");  
        return -1; : 'E' (8)  
    }
```

```
    return stack[top - 1]; : 'F' (9)
```

```
}
```

(C) after next, (D) after next) after previous bid
~~char peek() {~~ : 'G' (10)
 ~~if (top == -1) {~~ : 'H' (11)
 ~~printf("Stack underflow\n");~~ : 'I' (12)
 ~~return -1; }~~ : 'J' (13)
 ~~return stack[top]; }~~ : 'K' (14)

3 ('') : 'L' (15)

: 'M' (16)

```

int precedence (char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        case '(':
        case ')':
            return -1;
    }
}

```

else if

```

int associativity (char op) {
    if (op == '^') {
        return 1;
    }
    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;
        } else if (c == '(') {
            push(c);
        }
    }
}

```

```
else if (c == ')') {  
    while (peekl) != '(' {  
        postfix[k++] = pop();  
    }  
    pop();  
}  
else {  
    while (top != -1) && ((precedence(peek()) >  
        precedence(c)) ||  
        (precedence(peek()) == precedence(c)) &&  
        && associativity(c) == 0)) {  
        postfix[k++] = pop();  
    }  
    push(c);  
}  
while (top != -1) {  
    postfix[k++] = pop();  
}  
postfix[k] = '\0';
```

~~int main() {~~

~~char infix[MAX], postfix[MAX];~~
~~printf("Enter infix expression: ");~~
~~scanf("%s", infix);~~

~~infixtopostfix(infix, postfix);~~
~~printf("Postfix Expression: %s \n", postfix);~~
~~return 0;~~

output

$y(') = 1 \text{ (long) } \text{ value}$

Enter infix expression: $a + b * c / d$

postfix expression: $ab * cd / +$

Enter infix expression: $a / b + c (d * e)$

postfix expression: $ab / cd e * +$

$\langle (\text{long}) \text{ value} \rangle \rightarrow (b = 1, \text{val}) \text{ value}$

~~Enter infix expression: $(a + b * c - (d / e * f) + g) + h$~~

~~charbuff = (postfix expression): abc*delf*-g++h+~~

~~((0 = 10) 10 + 20 20~~

~~b/10/25~~

~~b/10/25 = [t + g] value~~

~~; (3) long~~

~~f~~

~~y(b = 1, pp) value~~

~~((pp) = [t + g] value~~

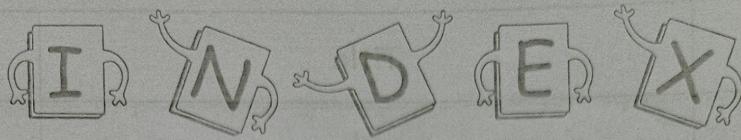
~~f~~

~~0/1 = (0) int~~

~~x(x) value f~~

~~(x(x)) int, (x(x)) int and~~

~~" " integers after each) f~~



Roll No. _____ Subject _____ School/College _____

School/College Tel. No. _____ Parents Tel. No. _____