

WAP to convert a given parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators +, -, *, /. Write the pseudo code.

⇒ ~~MAX SIZE~~

Define max size

initialise stack of max size

initialise top to -1

initialise char c

Function Pop()

if top = -1 then

display "Stack Underflow"

else

stack[top] = c

top--

return c

Function Peek()

if top = -1 then

return "Stack empty"

else

return operator stack[top]

Function Push()

if top = max size - 1

return "Stack overflow"

pop

Function Precedence(op)

switch (op)

case '+': return 1;

case '*': return 2;

case '^': return 3;

case '/': return 4;

default: return -1;

Function associativity(op)

if op = '^' then

return 1;

else

return 0;

Function infix-to-Postfix(infix, postfix)

initially int i, k = 0

initialise char c;

for (i = 0 to infix[i] != '\0', i++)

c = infix[i]

if c is an operand

postfix[k] = c

k++

while (top != -1 && peek() != '(')

postfix[k++] = pop();

push(c)

increment i

else

while (top != -1 && and

precedence(peek()) > precedence(c)

or precedence(peek()) == precedence(c)

and associativity(c) == 0.

push(c):

pop

while (top != -1)

postfix[k++] = pop();

postfix[k] = '\0';

Main function

char infix[MAX] & postfix[MAX]

display("Enter valid parentheses of infix")

read(infix)

infix-to-Postfix(infix, postfix)

```

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

```

```

3
3 while (top != -1) {
    postfix[k++] = pop();
}
postfix[k] = '\0';
3
int main() {
    char infix[MAX];
    char postfix[MAX];
    printf("Enter a valid Paranthesized of Infix Expression\n");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("Your Postfix Expression is: %s\n", postfix);
}

```

o/p

Enter a Valid Paranthesized of Infix Expression
A+B+C

Your Postfix Expression is: AB+C+

```

2;
:
3;
:
4;
t: Enter a Valid Paranthesized of Infix Expression
0: A+B+C
Your Postfix Expression is :AB+C+

ativ Process returned 0 (0x0)   execution time : 4.605 s
== Press any key to continue.
turn
0;
Topo
0;
0;
;
nt

```