

14/10/25

Q) WAP to convert second line program Description to be created for Today's Result.

Q) WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operand and binary operator.

+, -, *, /

.

Pseudo Code

Define N 100

int Stack[N];

int top = -1;

void push (char c) {

Stack[++top] = c;

}

~~char~~ ~~char~~ pop (char c)

char pop () {

return Stack[top--];

}

char peek () {

return Stack[top];

}

int precedence (char o) { } (c ==) also

switch (o) { = : long int w

: (0 == - case 't':

Case '-' :
return 1;

Case '*':

Case '/': } also

: (0 == /) : long int w
return 2;

Case '(':

: (0 == () : long int w
return 0;

: (0 == () : long int w
return 0;

: (0 == () : long int w
return -1;

int associativity (char o) { } also

if (0 == ('n') : int) also

: (0 == () : long int w
return 1;

return 0;

}

Conversion (char infix[], char postfix[]) { } also

void

int k = 0;

char c;

for (int i = 0; infix[i] != '\0'; i++) { } also

if (c == 'is num ()' { } also

postfix[i + k] = c;

else if (c == 'is sym ()' { } also

no longer int top
push(c); } also

}

else if ($c == ')'$) { } (0 marks) returning to;

while ($peek != ')'$) { } (0 marks)

Postfix[$++i$] = pop();

} (0 marks)
Therefore

0

else {

while ($top != -1$) { } (0 marks)

while ($top != -1$ & (precedence(peek()) > precedence(c) || associativity(c) == 0))

Postfix[$++i$] = pop();

push(c); (0 marks) pushing to

while ($top != -1$) { } (0 marks)

Postfix[$++i$] = pop();

0 marks

} (0 marks)

int main() {

char infix[N];

char postfix[N];

// Get infix input expression

Conversion(infix[N], postfix[N]);

// Print postfix expression

0

Code:

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

```
#define N 100
```

```
int stack[N];
```

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

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

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

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

```
        return;
```

```
}
```

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

```
char pop() {
```

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

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

```
        return -1;
```

```
    } else {
```

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

```
}
```

```
char peek() {
```

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

```
        return -1;
```

```
    } else {
```

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

```
}
```

```

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

```

$\leftarrow \text{left2} \rightarrow \text{subnif}$
 $\leftarrow \text{right2} \rightarrow \text{subnif}$
 $\leftarrow \text{left1} \rightarrow \text{subnif}$
 001 VA \leftarrow subnif.
 $\leftarrow [n] \rightarrow$ bi
 $\leftarrow n = \text{opt} \rightarrow$ bi

$\leftarrow (S \text{ next}) \rightarrow$ bi or bio
 $\leftarrow (n = \text{opt}) \rightarrow$ bi
 $\leftarrow (n / \text{bio/bio} \text{ next}) \rightarrow$ bio
 $\leftarrow \text{next}$
 $\leftarrow \text{next}$

```

int associativity (char o) {
    if (o == '^') {
        return 1;
    }
    return 0;
}

```

$\leftarrow [opt +] \rightarrow$ left2
 $\leftarrow (1 \rightarrow \text{opt}) \rightarrow$ right
 $\leftarrow (n / \text{next} \text{ next}) \rightarrow$ next
 $\leftarrow n = \text{opt} \rightarrow$ next

```

void conversion (char infix[], char post fix[])
{
    int k = 0;
    char c;
    for (int i = 0; infix[i] != ']' ; i++) {
        c = infix[i];
        if (isalnum (c)) {
            postfix[k++] = c;
        } else if (c == '(')
            push(c);
    }
}

```

$\leftarrow [opt +] \rightarrow$ left2
 $\leftarrow (1 \rightarrow \text{opt}) \rightarrow$ right
 $\leftarrow (n / \text{next} \text{ next}) \rightarrow$ next
 $\leftarrow n = \text{opt} \rightarrow$ next

else if ($c == ')'$) {

 while ($\text{peek}() != '('$) {

$\text{postfix}[k++] = \text{pop}();$

}

$\text{pop}();$

}

else {

 while ($\text{top} != -1$ && ($\text{precedence}(\text{peek}()) > \text{precedence}(c)$))

 if ($\text{precedence}(\text{peek}()) == \text{precedence}(c)$ &&
 $\text{associativity}(c) == 0$)

{

$\text{postfix}[k++] = \text{pop}();$

}

$\text{push}(c);$

 while ($\text{top} != -1$) {

$\text{postfix}[k++] = \text{pop}();$

}

$\text{postfix}[k] = '10';$

int main() {

 char infix[N], postfix[N];

 printf("Enter a postfix expression: ");

 scanf("%s", infix);

 conversion(infix, postfix);

 post

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

 return 0;

}

Output \Rightarrow Enter infix Expression:

$(3+2/3 + (9-4))$

Postfix Expression: 3 2 3 / + 9 4 - +