

PRN: 2020BTEIT00041

Infix to Postfix:

```

1 // Program to convert infix expression to postfix expression
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 class Node{
7 public:
8     char data;
9     Node* next;
10
11     Node(){
12         this->data = 0;
13         this->next = NULL;
14     }
15
16     Node(int data){
17         this->data = data;
18         this->next = NULL;
19     }
20 };
21
22 // template
23
24 template <class stack>
25 class Stack{
26 public:
27     Node* top = NULL;
28     int count = 0;
29
30     // Constructor
31     Stack(){
32         this->top = NULL;
33         this->count = 0;
34     };
35
36     // Destructor
37     ~Stack(){
38         Node* temp = this->top;
39         while(temp != NULL){
40             Node* temp2 = temp->next;
41             delete temp;
42             temp = temp2;
43         }
44     };
45
46     void Push(int data);
47     int Pop();
48     void Display();

```

```

49     };
50
51     // Push
52     template<>
53     void Stack<Node>::Push(int data){
54
55         // Create a new node and store the data
56         Node* newNode = new Node(data);
57
58         // If stack is empty, then newNode will be the head
59         if(this->top == NULL){
60             top = newNode;
61         }
62
63         // If stack is not empty, then add the newNode to the top
64         else{
65             newNode->next = this->top;
66             this->top = newNode;
67         }
68
69         // Increment the count
70         count++;
71     }
72
73     // Pop
74     template<>
75     int Stack<Node>::Pop(){
76         // If stack is empty, then return -1
77         if(top == NULL){
78             cout<<"Stack Underflow\n";
79             return -1;
80         }
81
82         // If stack is not empty, then return the top and delete the top
83         else{
84             // Store the top data
85             int data = top->data;
86
87             // Store the top's next node
88             Node* temp = top;
89
90             // move the top to the next node
91             top = top->next;
92
93             // Delete the top
94             delete temp;
95

```

```

96         // Decrement the count
97         count--;
98
99         // Return the top data
100         return data;
101     }
102 }
103
104 // Display
105 template<>
106 void Stack<Node>::Display(){
107     // If stack is empty, then print "Stack is empty"
108     if(top == NULL){
109         cout<<"Stack is empty\n";
110     }
111
112     // If stack is not empty, then print the stack
113     else{
114         // Create a new node and store the top
115         Node* temp = top;
116
117         // Print the stack
118         while(temp != NULL){
119             cout<<temp->data<<" ";
120             temp = temp->next;
121         }
122         cout<<endl;
123     }
124 }
125
126 int getPrecedence(char c){
127     if(c == '+' || c == '-'){
128         return 1;
129     }
130     else if(c == '*' || c == '/'){
131         return 2;
132     }
133     else if(c == '^'){
134         return 3;
135     }
136     else{
137         return -1;
138     }
139 }
140
141 void convertToPostfix(string exp){

```

```

142     // Create a stack to store operators
143     Stack<Node> opStack;
144
145     // Create a string to store the postfix expression
146     string postfix = "";
147
148     // Iterate through the given expression
149     for(int i=0; i<exp.length(); i++){
150         // If the current character is an operand, then add it to the postfix string
151         if(exp[i] >= 'a' && exp[i] <= 'z'){
152             postfix += exp[i];
153         }
154
155         // If the current character is an '(', then push it to the stack
156         else if(exp[i] == '('){
157             opStack.Push(exp[i]);
158         }
159
160         // If the current character is an ')', then pop and add the operators to the postfix string
161         else if(exp[i] == '){
162             while(opStack.top != NULL && opStack.top->data != '('){
163                 postfix += opStack.Pop();
164             }
165             opStack.Pop();
166         }
167
168         // If the current character is an operator, then pop the stack and add the operators to the postfix string
169         else{
170             while(opStack.top != NULL && getPrecedence(opStack.top->data) >= getPrecedence(exp[i])){
171                 postfix += opStack.Pop();
172             }
173             opStack.Push(exp[i]);
174         }
175     }
176
177     // Pop the remaining operators from the stack and add them to the postfix string
178     while(opStack.top != NULL){
179         postfix += opStack.Pop();
180     }
181
182     // Print the postfix string
183     cout<<postfix<<endl;
184 }
185
186 int main(){
187     string s;
188
189     cout<<"Enter the infix expression: ";
190     cin>>s;
191
192     cout<<"Infix expression: "<<s<<endl;
193
194     cout<<"Postfix expression: ";
195     convertToPostfix(s);
196
197
198     return 0;
199 }

```

OUTPUT:

```

Enter the infix expression: a+b*(c^d-e)^(f+g*h)-i
Infix expression: a+b*(c^d-e)^(f+g*h)-i
Postfix expression: abcd^e-fgh*+^*+i-

```