

```

#include <iostream>
1  #include <algorithm>
2  #include <string.h>
3  #include <cctype>
4  using namespace std;
5
6  // making a node class
7  class Node {
8  public:
9      int data;
10     Node* next;
11 };
12
13 // making a Stack class
14 class Stack {
15
16 private:
17     Node* top;
18
19 public:
20     // making a pop operation
21     void Pop()
22     {
23         Node* temp;
24         if (top == NULL)
25         {
26             exit(1);
27         }
28         else
29         {
30             temp = top;
31             top = top->next;
32             temp->next = NULL;
33             free(temp);
34         }
35     }
36
37     // making a Push operation..
38     void Push(int data)
39     {
40         Node* temp;
41         temp = new Node();
42         if (!temp)
43         {
44             exit(1);
45         }
46         temp->data = data;
47         temp->next = top;
48         top = temp;
49     }
50
51     int Empty_stack()
52     {
53         // if top == null return true
54         return top == NULL;
55     }
56
57     int Stack_top()
58     {
59         if (!Empty_stack())
60             return top->data;
61         else
62             exit(1);
63     }
64 }
65

```

```

66 bool isOperator(char c)
67 {
68     return (!isalpha(c) && !isdigit(c));
69 }
70
71 int HigherPreference(char C)
72 {
73     if (C == '-' || C == '+')
74         return 1;
75     else if (C == '*' || C == '/')
76         return 2;
77     else if (C == '^')
78         return 3;
79     return 0;
80 }
81 string Conversion_to_postfix(string infix)
82 {
83     infix = '(' + infix + ')';
84     int length = infix.size();
85     string result;
86
87     // iterating over the complete entered expression
88     for (int i = 0; i < length; i++) {
89
90         if (isalpha(infix[i]) || isdigit(infix[i]))
91             result += infix[i];
92
93         else if (infix[i] == '(')
94             Push('(');
95
96         else if (infix[i] == ')') {
97             while (Stack_top() != '(') {
98                 result += Stack_top();
99                 Pop();
100             }
101             Pop();
102         }
103         else {
104             if (isOperator(Stack_top())) {
105                 // checking the operators preference
106                 while (HigherPreference(infix[i]) <= HigherPreference(Stack_top())) {
107                     result += Stack_top();
108                     Pop();
109                 }
110                 Push(infix[i]);
111             }
112         }
113     }
114     return result;
115 }
116
117 string infixToPrefix(string infix)
118 {
119     int l = infix.size();
120     reverse(infix.begin(), infix.end());
121     for (int i = 0; i < l; i++) {
122         if (infix[i] == '(') {
123             infix[i] = ')';
124             i++;
125         }
126         else if (infix[i] == ')') {
127             infix[i] = '(';
128             i++;
129         }
130     }
131 }
132 string prefix = Conversion_to_postfix(infix);

```

```
133     reverse(prefix.begin(), prefix.end());
134     return prefix;
135 }
136 };
137
138 int main()
139 {
140     string s;
141     Stack expression;
142     cout << "Enter Infix Operation:- ";
143     cin >> s;
144     cout << expression.infixToPrefix(s) << "\n";
145 }
```