```cpp
#include <iostream>
#include <algorithm>
#include <string.h>
#include <cctype>
using namespace std;

// making a node class
class Node {
public:
 int data;
 Node* next;
};

// making a Stack class
class Stack {

private:
 Node* top;

public:
// making a pop operation
  void Pop()
 {
  Node* temp;
  if (top == NULL)
  {
   exit(1);
  }
  else
  {
   temp = top;
   top = top->next;
   temp->next = NULL;
   free(temp);
  }
 }

//   making a Push operation..
 void Push(int data)
 {
  Node* temp;
  temp = new Node();
  if (!temp)
  {
   exit(1);
  }
  temp->data = data;
  temp->next = top;
  top = temp;
 }

 int Empty_stack()
 {
  //  if top == null return true
  return top == NULL;
 }

 int Stack_top()
 {
  if (!Empty_stack())
   return top->data;
  else
   exit(1);
 }

```

```
bool isOperator(char c)
{
 return (!isalpha(c) && !isdigit(c));
}

int HigherPreference(char C)
{
 if (C == '-' || C == '+')
  return 1;
 else if (C == '*' || C == '/')
  return 2;
 else if (C == '^')
  return 3;
 return 0;
}
string Conversion_to_postfix(string infix)
{
 infix = '(' + infix + ')';
 int length = infix.size();
 string result;

 // iterasting over the complete entered expression
 for (int i = 0; i < length; i++) {

  if (isalpha(infix[i]) || isdigit(infix[i]))
   result += infix[i];

  else if (infix[i] == '(')
   Push('(');

  else if (infix[i] == ')') {
   while (Stack_top() != '(') {
    result += Stack_top();
    Pop();
   }
   Pop();
  }
  else {
   if (isOperator(Stack_top())) {
    // checking the operators preference
    while (HigherPreference(infix[i]) <= HigherPreference(Stack_top())) {
     result += Stack_top();
     Pop();
    }
    Push(infix[i]);
   }
  }
 }
 return result;
}

string infixToPrefix(string infix)
{
 int l = infix.size();
 reverse(infix.begin(), infix.end());
 for (int i = 0; i < l; i++) {
  if (infix[i] == '(') {
   infix[i] = ')';
   i++;
  }
  else if (infix[i] == ')') {
   infix[i] = '(';
   i++;
  }
 }
 string prefix = Conversion_to_postfix(infix);
```

```
  reverse(prefix.begin(), prefix.end());
  return prefix;
 }
};

int main()
{
 string s;
 Stack expression;
 cout << "Enter Infix Operation:- ";
 cin >> s;
 cout << expression.infixToPrefix(s) << "\n";
}
```