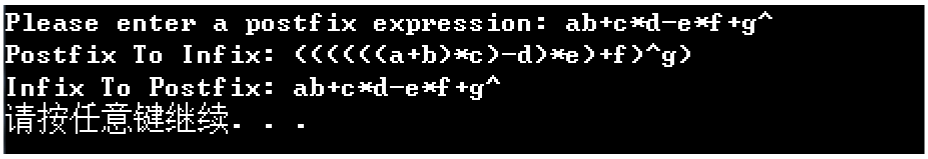
14346009 李志容

Assignment for Data Structures and Algorithm Analysis (3.21)

```
//Textbook 3.23
#include <cstring>
#include <string>
#include <cmath>
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

bool is_operator(string s);        //judge whether the parameters
string PostToInfix(vector<string> t);      //Postfix to Infix
string InToPostfix(string t);          //Infix To Postfix


bool is_operator(string s) {                          //judge whether the parameters
        if (s == "+" || s == "-" || s == "*" || s == "/" || s == "^") return true;
        else return false;
}

string PostToInfix(vector<string> t) {
        stack<string> res;                           //use stack to store the results
        for (int i = 0; i < t.size();i++) {           //traverse the string
                if (!is_operator(t[i])) {
                        res.push(t[i]);                  //if it isn't a opterator, push to the stack
                } else {

        string y = res.top();              //if it is a opterator, extract the top two elements from the stack
                        res.pop();
                        string x = res.top();
                        res.pop();
                        if (t[i][0] == '+' ) { //addition
                                x += "+";
                                x += y;
                                x += ")";
                                x = "(" + x;
                        }
                        else if (t[i][0] == '-') {x += "-";x += y;x += ")";x = "(" + x;}         //substraction
                        else if (t[i][0] == '*') {x += "*";x += y;}         //multiplication
                        else if (t[i][0] == '/') {x += "/";x += y;}         //division
                        else if (t[i][0] == '^') {x += "^";x += y;x += ")";x = "(" + x;}        //expenent
                res.push(x);          //push the new expression to the stack
            }
        }
        return res.top();               //the top of the stack is the result, return it
}

string InToPostfix(string t) {
        string res = "";
        stack<char> s;                  //using stack to save the operators
        for (int i = 0; i < t.length(); i++) {
            if (t[i] >= 'a' && t[i] <= 'z') {
                res += (t[i]);                //if it is an operation, add to the result string.

            } else {
                if (t[i] == ')') {            //if it is a right braket
                    while (!s.empty() && s.top() != '(') {
                        res += (s.top());
                        s.pop();
                    }
                    s.pop();

                } else if (t[i] == '(') {                //if it is a left braket, push to the stack
                    s.push(t[i]);
                } else if (t[i] == '+' || t[i] == '-') {            //if it is a + or - operator
                    while (!s.empty() && s.top() != '(') {
                        res += (s.top());
                        s.pop();
                    }
                    s.push(t[i]);
                } else if (t[i] == '*' || t[i] == '/') {            //if it is a * or / operator
                    while (!s.empty() && s.top() != '(' && s.top() != '+' & s.top() != '-') {
                        res += (s.top());
                        s.pop();
                    }
                    s.push(t[i]);
                } else if (t[i] == '^') {                //if it is a ^ operator
                    while (!s.empty() && s.top() != '(' && s.top() != '^') {
                        res += (s.top());
                        s.pop();
                    }
                    s.push(t[i]);
                }
            }
        }
```

```
    while (!s.empty()) {          //if there are still operators, add all of them to the results
        res += (s.top());
        s.pop();
    }
    return res;
}

int main() {
    vector<string> t;
    string re;
    cin >> re;
    for (int i = 0; i < re.length(); i++) {
        t.push_back(re.substr(i,1));
    }

    string s = PostToInfix(t);
    cout << "Postfix To Infix: " << s << endl;
    string tt = InToPostfix(s);
    cout << "Infix To Postfix: " << tt << endl;
    system ("pause");
    return 0;
}
```

Testing:



```
Please enter a postfix expression: ab+c*d-e*f+g^
Postfix To Infix: ((((((a+b)*c)-d)*e)+f)^g)
Infix To Postfix: ab+c*d-e*f+g^
请按任意键继续. . .
```