Q.1 Write a program for infix to prefix conversion,and vice versa.

INFIX -> PREFIX

```
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}
string infixToPrefix(string infix) {
    stack < char > operators;
    string prefix;
    reverse(infix.begin(), infix.end());
    unordered_map < char, int > priority;
    priority['+'] = priority['-'] = 1;
    priority['*'] = priority['/'] = 2;
    for (char & c: infix) {
        if (isalnum(c))
            prefix += c;
        else if (c == ')')
            operators.push(c);
        else if (c == '(') {
            while (operators.top() != ')') {
                prefix += operators.top();
                operators.pop();
            }
            operators.pop();
        } else {
            while (!operators.empty() && precedence(operators.top())
> precedence(c)) {
                prefix += operators.top();
                operators.pop();
            }
            operators.push(c);
        }
    }
    while (!operators.empty()) {
        prefix += operators.top();
        operators.pop();
```

```
    }
    reverse(prefix.begin(), prefix.end());
    return prefix;
}
int main() {
    string s;
    cout << "Enter the String";
    cin >> s;
    cout << "Infix: " << s << endl;
    cout << "Prefix: " << infixToPrefix(s) << endl;
    return 0;
}
```

Outputs: a+b*c-d/e
Infix: a+b*c-d/e Prefix: -+a*bc/de

PREFIX -> INFIX

```
string prefixToInfixConversion(string & s) {
    stack < string > st;
    for (int i = s.size() - 1; i >= 0; i--) {
        char c = s[i];
        if (c == '+' || c == '-' || c == '*' || c == '/') {
            string first = st.top();
            st.pop();
            string second = st.top();
            st.pop();
            st.push('(' + first + c + second + ')');
        } else {
            string t(1, c);
            st.push(t);
        }
    }
    return st.top();
}
int main() {
    string s;
    cin >> s;
    cout << prefixToInfixConversion(s);
    return 0;
```

```
}
```

Input: /-ab+-cde

Output: ((a-b)/((c-d)+e))

Q.2 Write a program for infix to postfix conversion,and vice versa.

INFIX → POSTFIX

```
int chack(char c){
if (c == '+' || c == '-') return 1;
else if (c == '*' || c == '/') return 2;
else if (c == '^') return 3;
else return -1;
}
string infixToPostfix(string s) {
    stack < char > st;
    string ans;
    int n = s.size();
    for (int i = 0; i < n; i++) {
        char c = s[i];
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c
<= '9')) {
            ans += c;
```

```
        } else if (c == '(') {
            st.push(c);
        } else if (c == ')') {
            while (st.top() != '(') {
                ans += st.top();
                st.pop();
            }
            st.pop();
        } else {
            while (!st.empty() && chack(c) <= chack(st.top())) {
                ans += st.top();
                st.pop();
            }
            st.push(c);
        }
    }
    while (!st.empty()) {
        ans += st.top();
        st.pop();
    }
    return ans;
}
int main() {
    string s;
    cin >> s;
    cout << infixToPostfix(s);
```

```
    return 0;
}
```

Input:a+b+c+d-e

Output:ab+c+d+e-

POSTFIX → INFIX

```
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}
string postfixToInfix(string postfix) {
    stack<string> s;
    for (char &c : postfix) {
        if (isOperator(c)) {
            string op2 = s.top();
            s.pop();
            string op1 = s.top();
            s.pop();
            string result = "(" + op1 + c + op2 + ")";
            s.push(result);
        } else {
            s.push(string(1, c));
        }
    }
    return s.top();
}
int main() {
    string postfix_expression;
```

```
    cout << "Enter postfix expression: ";
    getline(cin, postfix_expression);

    string infix_expression = postfixToInfix(postfix_expression);
    cout << "Infix expression: " << infix_expression << endl;

    return 0;
}
```

Postfix: ab*c+d/ Infix: ((a*b)+(c/d))

Q3.
i 7 – 6* 3^ 2/ 8+9

infix→ prefix

```
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}
string infixToPrefix(string infix) {
    stack < char > operators;
    string prefix;
    reverse(infix.begin(), infix.end());
    unordered_map < char, int > priority;
    priority['+'] = priority['-'] = 1;
    priority['*'] = priority['/'] = 2;
```

```
    for (char & c: infix) {
        if (isalnum(c))
            prefix += c;
        else if (c == ')')
            operators.push(c);
        else if (c == '(') {
            while (operators.top() != ')') {
                prefix += operators.top();
                operators.pop();
            }
            operators.pop();
        } else {
            while (!operators.empty() && precedence(operators.top()) >
precedence(c)) {
                prefix += operators.top();
                operators.pop();
            }
            operators.push(c);
        }
    }
    while (!operators.empty()) {
        prefix += operators.top();
        operators.pop();
    }
    reverse(prefix.begin(), prefix.end());
    return prefix;
```

```
}

int main() {

    string s;

    cout << "Enter the String";

    cin >> s;

    cout << "Infix: " << s << endl;

    cout << "Prefix: " << infixToPrefix(s) << endl;

    return 0;

}
```

Infix: 7-6*3^2/8+9 Prefix: ^-7*63+/289

infix→postfix

```
#include<bits/stdc++.h> using namespace std; int chack(char c){


if (c == '+' || c == '-') return 1;

else if (c == '*' || c == '/') return 2;

else if (c == '^') return 3;

else return -1;

}

string infixToPostfix(string s) {

    stack < char > st;

    string ans;

    int n = s.size();

    for (int i = 0; i < n; i++) {

        char c = s[i];
```

```
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c
<= '9')) {
            ans += c;
        } else if (c == '(') {
            st.push(c);
        } else if (c == ')') {
            while (st.top() != '(') {
                ans += st.top();
                st.pop();
            }
            st.pop();
        } else {
            while (!st.empty() && chack(c) <= chack(st.top())) {
                ans += st.top();
                st.pop();
            }
            st.push(c);
        }
    }
    while (!st.empty()) {
        ans += st.top();
        st.pop();
    }
    return ans;
}
int main() {
```

```
    string s;
    cin >> s;
    cout << "infix" << s << endl;
    cout << "Postfix" << infixToPostfix(s);
    return 0;
}
```

7 - 6 * 3 ^ 2 / 8 + 9 infix7 - 6 * 3 ^ 2 / 8 + 9 Postfix7632 ^ * 8 / -9 +

3.

ii.

3 5 5 5 / * +2 2 ^ -#include<bits/stdc++.h> using namespace std;

```
    bool isOperator(char c) {
        return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
    }
string postfixToInfix(string postfix) {
    stack < string > operands;
    for (char & c: postfix) {
        if (isalnum(c)) {
            operands.push(string(1, c));
        } else if (isOperator(c)) {
            string operand2 = operands.top();
            operands.pop();
            string operand1 = operands.top();
            operands.pop();
            string expression = "(" + operand1 + c + operand2 + ")";
            operands.push(expression);
```

```
        }
    }
    return operands.top();
}
int main() {
    string postfix;
    cout << "Enter a postfix expression: ";
    cin >> postfix;
    cout << "Postfix: " << postfix << endl;
    cout << "Infix: " << postfixToInfix(postfix) << endl;
    return 0;
}
```

Enter a postfix expression: 3555/*22^- Postfix: 3555/*22^-
Infix: ((5*(5/5))-(2^2))

3.
iii.

- +5/ * 6234

```
string prefixToInfixConversion(string & s) {
    stack < string > st;
    for (int i = s.size() - 1; i >= 0; i--) {
        char c = s[i];
        if (c == '+' || c == '-' || c == '*' || c == '/') {
            string first = st.top();
            st.pop();
            string second = st.top();
```

```
            st.pop();
            st.push('(' + first + c + second + ')');
        } else {
            string t(1, c);
            st.push(t);
        }
    }
    return st.top();
}
int main() {
    string s;
    cin >> s;
    cout << prefixToInfixConversion(s);
    return 0;
}
```

-+5/*6234

((5+((6*2)/3))-4)