DSA_College\infix_to_postfix.cpp

```cpp
#include<iostream>
#include<stack>
#include<string>
using namespace std ;

// Function to check the precedence of operators
int precedence(char op)
{
    if(op == '+' || op == '-')
    return 1 ;

    if(op == '*' || op == '/')
    return 2 ;

    if(op == '^')
    return 3;

    return 0; // for non-operators
}

bool isOperand(char x)
{
    return ((x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z')) ;
}

string infixToPostfix(const string &infix)
{
    stack<char> st ;
    string postfix = "" ;
    for(int i=0; i<infix.length(); i++)
    {
        char c = infix[i] ;

        if (isspace(c))
        continue;

        // if the scanned character is an operand, add to o/p string
        if(isOperand(c))
        {
            postfix += c ;
        }

        // if the scanned character is '(' , add to stack
        else if(c == '(')
        {
            st.push('(') ;
        }

        // if the scanned character is ')', pop and add to o/p string
        // from the stack until an '(' is found
        else if(c == ')')
```

```cpp
        {
            while(st.top() != '(')
            {
                postfix += st.top() ;
                st.pop() ;
            }
            st.pop() ;
        }

        // if an operator is scanned, add to the stack
        else{
            while(st.empty() != 1 && precedence(c) <= precedence(st.top()))
            {
                postfix += st.top() ;
                st.pop() ;
            }
            st.push(c) ;
        }

    }

    // pop all the remaining elements from stack
    while(!st.empty())
    {
        postfix += st.top() ;
        st.pop() ;
    }

    return postfix ;
}

int main()
{
    string infix = "((a+(b*c))-d)" ;
    cout << "Infix Expression: " << infix << endl ;
    cout << "Postfix Expression: " << infixToPostfix(infix)<< endl ;
    return 0;
}

```