# Experiment 2.2

**Student Name:Rajiv Paul** **UID:20BCS1812**
**Branch: CSE** **Section/Group:709A**
**Semester: 3rd** **Date of Performance: 06/10/2021**
**Subject Name:Data Structure Lab** **Subject Code:20CSP-219**

**Q1. Write a program to convert an infix expression to a postfix expression using two precedence function .**

**1) Aim/Overview of the practical:**

**To write a program to convert an infix expression to a postfix expression using two precedence function.**

**2) Software required:**

**Vs Code**

## 3) Source Code:

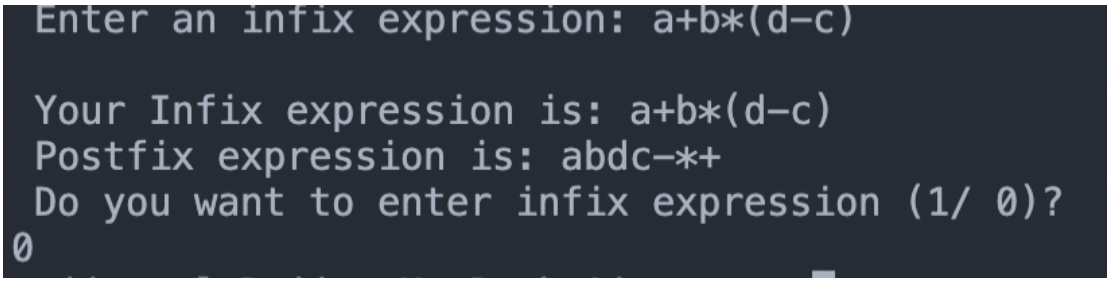
```cpp
#include <iostream>
#include <stack>
using namespace std;
bool IsOperator(char);
bool IsOperand(char);
bool eqlOrhigher(char, char);
string convert(string);
int main()
{
    string infix_expression, postfix_expression;
    int ch;
    do
    {
        cout << " Enter an infix expression: ";
        cin >> infix_expression;
        postfix_expression = convert(infix_expression);
        cout << "\n Your Infix expression is: " << infix_expression;
        cout << "\n Postfix expression is: " << postfix_expression;
        cout << "\n Do you want to enter infix expression (1/ 0)?"<<endl;
        cin >> ch;
    } while (ch == 1);
    return 0;
}
```

```cpp
bool IsOperator(char c)
{
    if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
        return true;
    return false;
}
bool IsOperand(char c)
{
    if (c >= 'A' && c <= 'Z')
        return true;
    if (c >= 'a' && c <= 'z')
        return true;
    if (c >= '0' && c <= '9')
        return true;
    return false;
}
int precedence(char op)
{
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    if (op == '^')
        return 3;
    return 0;
}
```

```
bool eqlOrhigher(char op1, char op2)
{
    int p1 = precedence(op1);
    int p2 = precedence(op2);
    if (p1 == p2)
    {
        if (op1 == '^')
            return false;
        return true;
    }
    return (p1 > p2 ? true : false);
}
string convert(string infix)
{
    stack<char> S;
    string postfix = "";
    char ch;
    S.push('(');
    infix += ')';
    for (int i = 0; i < infix.length(); i++)
```

```
    {
        ch = infix[i];
        if (ch == ' ')
            continue;
        else if (ch == '(')
            S.push(ch);
        else if (IsOperand(ch))
            postfix += ch;
        else if (IsOperator(ch))
        {
            while (!S.empty() && eqlOrhigher(S.top(), ch))
            {
                postfix += S.top();
                S.pop();
            }
            S.push(ch);
        }
        else if (ch == ')')
        {
            while (!S.empty() && S.top() != '(')
            {
                postfix += S.top();
                S.pop();
            }
            S.pop();
        }
    }
    return postfix;
}
```

**4. Output:**


Enter an infix expression: a+b*(d-c)
Your Infix expression is: a+b*(d-c)
Postfix expression is: abdc-*+
Do you want to enter infix expression (1/ 0)?
0

**Q2. Write a program to convert the expression "a+b" into "+ab".**

**1) Aim/Overview of the practical:**

**To write a program to convert the expression "a+b" into "+ab".**

**2) Software required:**

**Vs Code**

## 3) Source Code:

```
#include <iostream>
#include<stack>
using namespace std;
int pre(char c);
string infixToPostfix(string s);
int main()
{
string s;
cout << "Enter the infix value: ";
cin >> s;
cout << "postfix is :";
cout << infixToPostfix(s);
cout<<endl;
return 0;
}
int pre(char c)
{
if (c == '^')
return 3;
else if (c == '*' || c == '/')
return 2;
else if (c == '+' || c == '-')
return 1;
else
return 0;
}
```

```
string infixToPostfix(string s)
{
stack<char> s1;
string value = "";
for (int i = 0; i < s.length(); i++)
{
if (isalpha(s[i]))
{
    value += s[i];
}
else if (s[i] == '(')
{
s1.push(s[i]);
}
else
{
if (s[i] == ')')
{
while (s1.top() != '(')
{
value += s1.top();
s1.pop();
}
s1.pop();
}
```

```
else if (s1.empty() == true || pre(s[i]) > pre(s1.top()))
s1.push(s[i]);
else
{
while (s1.empty() == false && pre(s[i]) <= pre(s1.top()))
{
value += s1.top();
s1.pop();
}
s1.push(s[i]);
}
}
}
while (s1.empty() == false)
{
value += s1.top();
s1.pop();
}
return value;
}
```

**4. Output:**

```
Enter the infix value: a+b
postfix is :ab+
```