Assignment 2

Student Name:Rajiv Paul **UID:20BCS1812**
Branch: CSE **Section/Group:709A**
Semester: 3rd **Subject Name:Data Structure**

Q1

Write a function to convert an infix expression to a postfix expression. Pass a one-dimensional character array P to the function as input (infix exp) and return character array Q (postfix exp).

Aim/Overview of the practical:

To write a function to convert an infix expression to a postfix expression. Pass a one-dimensional character array P to the function as input (infix exp) and return character array Q (postfix exp).

Software required:

Vs Code

Source Code:

```
#include<iostream>
#include<stack>
#include<string>

using namespace std;


string InfixToPostfix(string expression);


int HasHigherPrecedence(char operator1, char operator2);


bool IsOperator(char C);


bool IsOperand(char C);
```

```
int main()
{
    string expression;
    cout<<"\nEnter Infix Expression: ";
    getline(cin,expression);
    string postfix = InfixToPostfix(expression);
    cout<<"\nOutput in postfix = "<<postfix<<endl<<endl;
}


string InfixToPostfix(string expression)
{

    stack<char> S;
    string postfix = "";
    for(int i = 0;i< expression.length();i++) {


        if(expression[i] == ' ' || expression[i] == ',') continue;


        else if(IsOperator(expression[i]))
        {
            while(!S.empty() && S.top() != '(' && HasHigherPrecedence(S.top(),expression[i]))
            {
                postfix+= S.top();
                S.pop();
            }
            S.push(expression[i]);
        }
```

```
        else if(expression[i] == ')')
        {
            while(!S.empty() && S.top() !=  '(') {
                postfix += S.top();
                S.pop();
            }
            S.pop();
        }
    }

    while(!S.empty()) {
        postfix += S.top();
        S.pop();
    }

    return postfix;
}

bool IsOperand(char C)
{
    if(C >= '0' && C <= '9') return true;
    if(C >= 'a' && C <= 'z') return true;
    if(C >= 'A' && C <= 'Z') return true;
    return false;
}


bool IsOperator(char C)
{
    if(C == '+' || C == '-' || C == '*' || C == '/' || C== '$')
        return true;

    return false;
}
```

```
int IsRightAssociative(char op)
{
    if(op == '$') return true;
    return false;
}


int GetOperatorWeight(char op)
{
    int weight = -1;
    switch(op)
    {
    case '+':
    case '-':
        weight = 1;
    case '*':
    case '/':
        weight = 2;
    case '$':
        weight = 3;
    }
    return weight;
}

int HasHigherPrecedence(char op1, char op2)
{
    int op1Weight = GetOperatorWeight(op1);
    int op2Weight = GetOperatorWeight(op2);


    if(op1Weight == op2Weight)
    {
        if(IsRightAssociative(op1)) return false;
        else return true;
    }
    return op1Weight > op2Weight ?  true: false;
}
```

Output:

```
Enter Infix Expression: ( A - (B / C ) * D + E ) * F % G
Output in postfix = ABC/D*E+FG*
```

Compiled successfully!