

ASSIGNMENT-05

/*Syntax Parsing in Programming Languages:

Parsing expressions is a key step in many compilers and language processors. When a language's syntax requires parsing mathematical or logical expressions, converting between infix and postfix notation ensures that expressions are evaluated correctly. Accept an infix expression and show the expression in postfix form. */

```
#include <iostream>
#include <cstring>
#include <string>
#define N 10
using namespace std;
class stack
{
    public:
    char arr[10];
    string ex;
    int top;
    stack()
    {
        top=-1;
    }
    void push(char c)
    {
        if(top==(N-1))
        {cout<<"stack overflow";}
        else{top+=1;
        arr[top]=c;}
    }
    char pop()
    { char c= ' ';
    if(top==-1)
    { //cout<<"Stack underflow";
    }
    else
    {
        c=arr[top];
        top--;
    }
    return c;
}
int precedence(char opr){
if (opr=='*' || opr=='/')
```

```

    return 2;
    if (opr=='+' || opr=='-')
        return 1;
    if(opr=='(') return 0;
    }
    char associativity(char opr){
    if (opr=='*' || opr=='/'||opr=='+' || opr=='-')
        return 'L';
    else
        return 'R';
    }
    char peek(){return arr[top]; }
    string InfixToPostfixConversion(string ex);
};


```

```

string stack::InfixToPostfixConversion(string ex){
    int l=ex.length();
    int i=0,j=0;
    char op_exp[20];
    char ch,ch1;
    while(i<l){

        //cout<<ex[i];
        if(ex[i]=='+'||ex[i]=='-'||ex[i]=='*' || ex[i]=='/'){
            if(top==-1) push(ex[i]);
            else{
                ch=peek();
                //if(ch== ' ')
                while(precedence(ex[i])<=precedence(ch)){
                    ch1=pop();
                    op_exp[j++]=ch1;
                    ch=peek();
                }
                push(ex[i]);
            }
        }
        else if(ex[i]=='(')
            push(ex[i]);
        else if(ex[i]==')'){
            ch1=pop();
            while(ch1!= '('){
                op_exp[j++]=ch1;

```

```

        ch1=pop();
    }
}
else {
    op_exp[j++]=ex[i];
    // var++;
}
i++;

}

do{
    ch=pop();
    op_exp[j++]=ch;
}while(ch!=' ');
op_exp[j]='\0';
return op_exp;

}

int main() {
    stack s;
    string ex;
    cout<<"\n Enter an expression:";
    cin>>ex;
    string op_exp=s.InfixToPostfixConversion(ex);
    cout<<"\n Postfix Expression is: "<<op_exp;
    return 0;
}

```

output:

Enter an expression:a+b*c/d

Postfix Expression is: abc*d/+

==== Code Execution Successful ===

GITHUB: