Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Information Technology

COURSE CODE: DJS22ITL302 **DATE:** 25-10-23

COURSE NAME: Data Structure Laboratory **CLASS:** I1-Batch1

NAME: Ayush Vinod Upadhyay **SAP ID:** 60003220131

ROLL NO.: I025

Experiment No. 3

CO/LO: CO1

Aim: Implementation of Infix to Postfix conversion and Implementation.

Theory: Infix: An infix operation is any operation of the format x op y format, such as x + y

Postfix:- An operation or expression can also be expressed as x y op i.e xy+, which is equivalent to writing x+y in infix. All we're trying to perform relocating the operator to the operand's right.

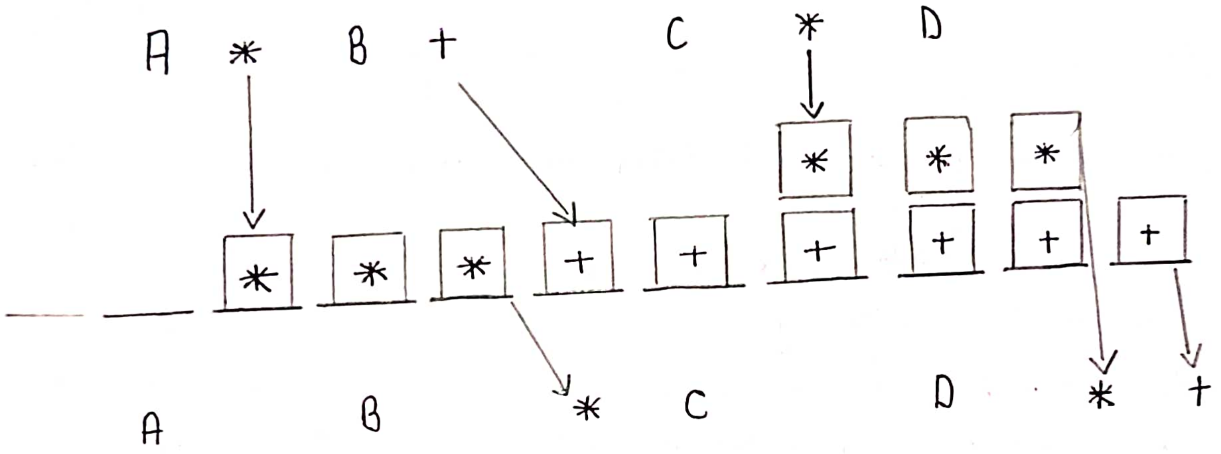
Algorithm

1. Scan the infix expression from left to right
2. If the scanned character is an operand, output it
3. Else,
   - If the precedence of the scanned operator is greator than the precedance of the operator in the stack, push it.
   - Else, Pop the operator from the stack until the precedence of the scanned operator is less - equal to the precedence of the ~~scanned~~ operator ~~using~~ residing on the top of the stack. Push the scanned operator to the stack.
4. If the scanned character is '(' push it to the stack.
5. If the scanned character is ')' pop and output ~~the~~ from the stack until an '(' is encountered.

6. Repeat Step 2-6 until infix expression is scanned.

7. Pop and output from the stack until it is not empty.

Output : A * B + C * D



AB* CD* +

Conclusion : I learnt the conversion from infix to postfix and their evaluation.

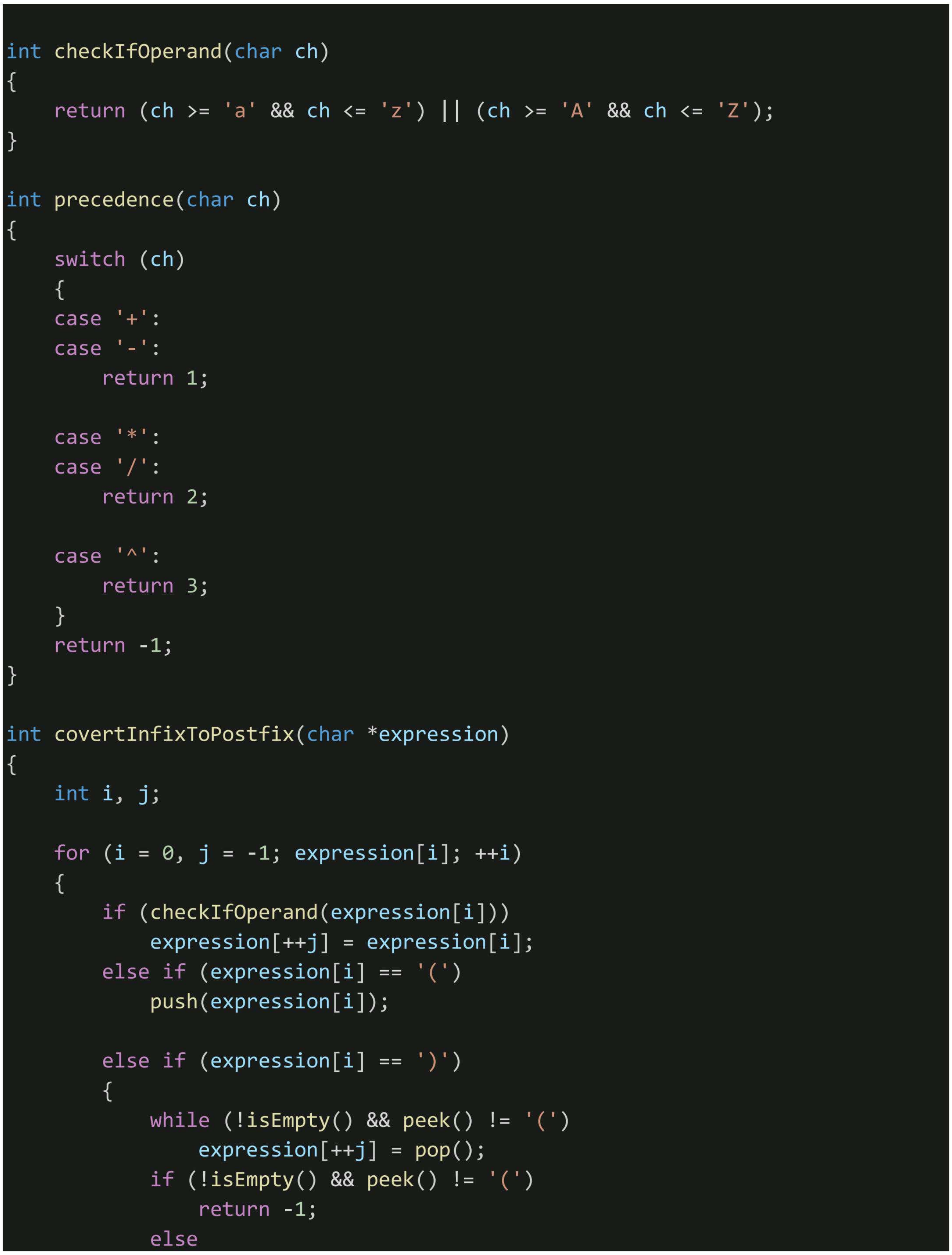
References : GeeksforGeeks, W3School for theory

Self implemented the code.

COURSE CODE: DJS22ITL302 **DATE:25/10/2023**
COURSE NAME: Data Structure Laboratory **CLASS: I1-Batch1**

Program:

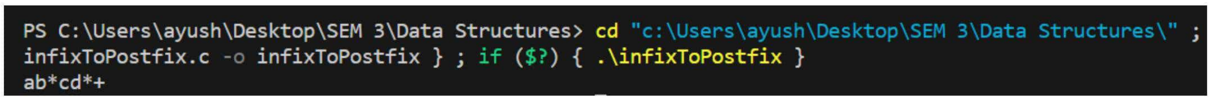
```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX 20

char stk[20];
int top = -1;
int isEmpty()
{
    return top == -1;
}
int isFull()
{
    return top == MAX - 1;
}

char peek()
{
    return stk[top];
}

char pop()
{
    if (isEmpty())
        return -1;

    char ch = stk[top];
    top--;
    return (ch);
}
void push(char oper)
{
    if (isFull())
        printf("Stack Full!!!!");

    else
    {
        top++;
        stk[top] = oper;
    }
}
```



```
int checkIfOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}

int precedence(char ch)
{
    switch (ch)
    {
    case '+':
    case '-':
        return 1;

    case '*':
    case '/':
        return 2;

    case '^':
        return 3;
    }
    return -1;
}

int covertInfixToPostfix(char *expression)
{
    int i, j;

    for (i = 0, j = -1; expression[i]; ++i)
    {
        if (checkIfOperand(expression[i]))
            expression[++j] = expression[i];
        else if (expression[i] == '(')
            push(expression[i]);

        else if (expression[i] == ')')
        {
            while (!isEmpty() && peek() != '(')
                expression[++j] = pop();
            if (!isEmpty() && peek() != '(')
                return -1;
            else
```

```
                pop();
        }
        else
        {
            while (!isEmpty() && precedence(expression[i]) <=
precedence(peek()))
                expression[++j] = pop();
            push(expression[i]);
        }
    }
    while (!isEmpty())
        expression[++j] = pop();

    expression[++j] = '\0';
    printf("%s", expression);
}

int main()
{
    char expression[] = "(a*b)+(c*d)";
    covertInfixToPostfix(expression);
    return 0;
}
```

Output screenshots:



```
PS C:\Users\ayush\Desktop\SEM 3\Data Structures> cd "c:\Users\ayush\Desktop\SEM 3\Data Structures\" ;
infixToPostfix.c -o infixToPostfix } ; if ($?) { .\infixToPostfix }
ab*cd*+
```