Lab No: 3 **Date: 2081/12/08**

Title: Write a program to evaluate the user input postfix or prefix expression.

- **Infix notation:** X + Y

  Operators are written in-between their operands. This is the usual way we write expressions. An expression such as A * (B + C) / D is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer. "Infix notation needs extra information to make the order of evaluation of the operators clear: rules built into the language about operator precedence and associativity, and brackets ( ) to allow users to override these rules.

- **Postfix notation** (also known as "Reverse Polish notation"): X Y +

  Operators are written after their operands. The infix expression given above is equivalent to A B C + * D / The order of evaluation of operators is always left-to-right, and brackets cannot be used to change this order. Because the "+" is to the left of the "*" in the example above, the addition must be performed before the multiplication. Operators act on values immediately to the left of them. For example, the "+" above uses the "B" and "C". We can add (totally unnecessary) brackets to make this explicit:((A (B C +) *) D /)

- **Prefix notation** (also known as "Polish notation"): + X Y

  Operators are written before their operands. The expressions given above are equivalent to / * A + B C DAs for Postfix, operators are evaluated left-to-right and brackets are superfluous. Operators act on the two nearest values on the right. I have again added (totally unnecessary) brackets to make this clear:(/ (* A (+ B C)) D)

IDE: Visual Studio Code

Language: C

Source code (for postfix):

```
#include <stdio.h>
#include <math.h>
int stack[50], top = -1;
void sum();
void sub();
void mult();
void div();
void power();
int main()
{char st[50];
    int i;
    printf("Enter the postfix expression:\n");
    scanf("%[^\n]s", st);
    // Here, to take space as input we use '%[^\n]s'
    for (i = 0; st[i] != '\0'; i++)
    {
        if (st[i] != ' ')
        {
            switch (st[i])
            {
            case '+':
                sum();
                break;
            case '-':
                sub();
                break;
            case '*':
                mult();
                break;

            case '/':
                div();
                break;

            case '^':
                power();
                break;

            default:
                top++;
                stack[top] = st[i] - 48;
                /*since, st[i] contain characters so, we subtract
                them with 48 so that (char: '0','1','2',...) changed
                to their interger form */}
        }
    }
    printf("\nThe result is %d\n", stack[top]);
}
void sum()
{
    int res, op1, op2;
    op2 = stack[top];
    top--;
    op1 = stack[top];
```

```
    top--;
    res = op1 + op2;
    top++;
    stack[top] = res;
}
void sub()
{
    int res, op1, op2;
    op2 = stack[top];
    top--;
    op1 = stack[top];
    top--;
    res = op1 - op2;
    top++;
    stack[top] = res;
}
void mult()
{
    int res, op1, op2;
    op2 = stack[top];
    top--;
    op1 = stack[top];
    top--;
    res = op1 * op2;
    top++;
    stack[top] = res;
}
void div()
{
    int res, op1, op2;
    op2 = stack[top];
    top--;
    op1 = stack[top];
    top--;
    res = op1 / op2;
    top++;
    stack[top] = res;
}
void power()
{
    int op1, op2, i;
    op2 = stack[top];
    top--;
    op1 = stack[top];
    top--;
    stack[top++] = pow(op1, op2);
}
```

Output:

```
PS C:\Users\user\OneDrive\Desktop\DSA\ManjilBajgain>
cc postfic.c -o postfic } ; if ($?) { .\postfic }
Enter the postfix expression:
3 4+2/

The result is 3
```

Source code(for prefix):

```
#include <stdio.h>
#include <math.h>
#include <string.h>

int stack[50], top = -1;

void sum();
void sub();
void mult();
void div();
void power();
void reverse(char str[]);

int main()
{
    char st[50];
    int i;

    printf("Enter the prefix expression:\n");
    scanf("%[^\n]s", st);
    // Reverse the input string to process prefix from right to left
    strrev(st);

    for (i = 0; st[i] != '\0'; i++)
    {
        if (st[i] != ' ') // Skip spaces
        {

                switch (st[i])
                {
                case '+':
                    sum();
                    break;

                case '-':
                    sub();
                    break;

                case '*':
                    mult();
                    break;

                case '/':
                    div();
                    break;

                case '^':
                    power();
                    break;

                default:
                   top++;
                 stack[top] = st[i] - 48;

            }
        }
```

```
    }

    printf("\nThe result is %d\n", stack[top]);
    return 0;
}

void sum()
{
    int op1 = stack[top--];
    int op2 = stack[top--];
    stack[++top] = op1 + op2;

}

void sub()
{
    int op1 = stack[top--];
    int op2 = stack[top--];
    stack[++top] = op1 - op2;

}

void mult()
{
    int op1 = stack[top--];
    int op2 = stack[top--];
    stack[++top] = op1 * op2;

}

void div()
{
    int op1 = stack[top--];
    int op2 = stack[top--];
    stack[++top] = op1 / op2;

}

void power()
{
    int op1 = stack[top--];
    int op2 = stack[top--];
    stack[++top] = pow(op1, op2);

}
```

```
PS C:\Users\user\OneDrive\Desktop\DSA\ManjilBajgain> c
cc prfic.c -o prfic } ; if ($?) { .\prfic }
Enter the prefix expression:
+ - * + 1 2 / 4 2 1 ^ 4 2

The result is 21
```