# DATA STRUCTURES
# USING "C"

## (Polish Notation)

For

BCA Part-II (Session 2018-21) Students

*BY*

**ANANT KUMAR**
MCA, M. Phil, M. Tech.
Faculty Member
Department of Computer Science
J. D. Women's College, Patna

# Program to convert an infix to postfix expression

```
# include <string.h>

char postfix[50];
char infix[50];
char opstack[50];   /* operator stack */
int i, j, top =0;

int lesspriority(char, char);
void push(char);
void pop();

void main()
{
        char ch; clrscr();
        printf("\n Enter Infix Expression : ");
        gets(infix);
        while( (ch=infix[i++]) != '\0')
        {
            switch(ch)
            {
                case ' ' :
                        break;
                case '(' :
                case '+' :
                case '-':
                case '*':
                case '/':
                case '^' :
                case '%' :
                        push(ch);      /* check priority and push */
                        break;
                case ')' :
                        pop();
                        break;
                default :
                        postfix[j] = ch;
                        j++;
            }
```

```
        }
        while(top >= 0)
        {
                postfix[j] = opstack[--top];
                j++;
        }
        postfix[j] = '\0';
        printf("\nInfixExpression : %s ", infix);
        printf("\n Postfix Expression : %s ", postfix);
        getch();
}

int lesspriority(char op, char op_at_stack)
{
   int k;
   int pv1;    /* priority value of op*/
   int pv2;    /* priority value of op_at_stack */
   char operators[] = {'+', '-', '*', '/', '%', '^', '('};
   int priority_value[] = {0,0,1,1,2,3,4};
   if( op_at_stack == '(' )
        return 0;
        for(k = 0; k < 6; k ++)
        {
                if(op == operators[k])
                        pv1 = priority_value[k];
        }
        for(k = 0; k < 6; k ++)
        {
                if(op_at_stack == operators[k])
                        pv2 = priority_value[k];
        }
        if(pv1 < pv2)
                return 1;
        else
                return 0;
}
```

```
void push(char op) /* op - operator  */
{
        if(top == 0)
        {
                opstack[top] = op;
                top++;
        }
        else
        {

                if(op != '(' )
                {

                        while(lesspriority(op, opstack[top-1]) == 1 && top > 0)
                        {
                                postfix[j] = opstack[--top];
                                j++;
                        }
                }
                opstack[top]=op;    /* pushing onto stack */
                top++;
        }
}

void pop()
{
        while(opstack[--top] !='(')          /* pop until '(' comes*/
        {
                postfix[j] = opstack[top];
                j++;
        }
}
```

# Program to evaluate a postfix expression

```
# include <conio.h>
# include <math.h>
# define MAX 20

int isoperator(char);

void main(void)
{
        char postfix[MAX];
        int val;
        char ch;
        int i = 0, top = 0;
        float val_stack[MAX], val1, val2, res;
        clrscr();
        printf("\n Enter a postfix expression: ");
        scanf("%s", postfix);
        while((ch = postfix[i]) != '\0')
        {
                if(isoperator(ch) == 1)
                {
                        val2 = val_stack[--top];
                        val1 = val_stack[--top];
                        switch(ch)
                        {
                                case '+':
                                        res = val1 + val2;
                                        break;
                                case '-':
                                        res = val1 - val2;
                                        break;
                                case '*':
                                        res = val1 * val2;
                                        break;
                                case '/':
                                        res = val1 / val2;
                                        break;
```

```
                        case '^':
                                res = pow(val1, val2);
                                break;
                }
                val_stack[top] = res;
        }
        else
                val_stack[top] = ch-48; /*convert character digit to integer digit */

        top++;
        i++;
    }
    printf("\n Values of %s is : %f ",postfix, val_stack[0] );
    getch();
}


int isoperator(char ch)
{
    if(ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^')
        return 1;
    else
        return 0;
}
```