**Week 3 & 4**

**3. C Program to convert Infix to Postfix Notation**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h> *// for isalpha and isdigit*

#define SIZE 100

char stack[SIZE];

int top = -1;

*// Stack operations*

void **push**(char c)

{

if (top == SIZE - 1)

{

printf("Stack Overflow\n");

} else {

stack[++top] = c;

}

}

char **pop**()

{

if (top == -1)

{

return '\0';

} else {

return stack[top--];

}

}

char **peek**()

{

if (top == -1)

return '\0';

return stack[top];

}

*// Function to return precedence of operators*

int **precedence**(char op)

{

switch (op)

{

case '^': return 3;

case '\*':

case '/': return 2;

case '+':

case '-': return 1;

default: return 0;

}

}

*// Function to check if the character is an operator*

int **isOperator**(char c)

{

return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

}

*// Function to convert infix to postfix*

void **infixToPostfix**(char\* infix)

{

char postfix[SIZE];

int i = 0, j = 0;

char symbol, temp;

printf("Infix Expression: %s\n", infix);

while ((symbol = infix[i++]) != '\0')

{

if (isalnum(symbol))

{

postfix[j++] = symbol;

} else if (symbol == '(') {

push(symbol);

} else if (symbol == ')') {

while ((temp = pop()) != '(')

{

postfix[j++] = temp;

}

} else if (isOperator(symbol)) {

while (top != -1 && precedence(peek()) >= precedence(symbol))

{

postfix[j++] = pop();

}

push(symbol);

}

}

while (top != -1)

{

postfix[j++] = pop();

}

postfix[j] = '\0';

printf("Postfix Expression: %s\n", postfix);

}

*// Main function*

int **main**()

{

char infix[SIZE];

printf("Enter Infix Expression: ");

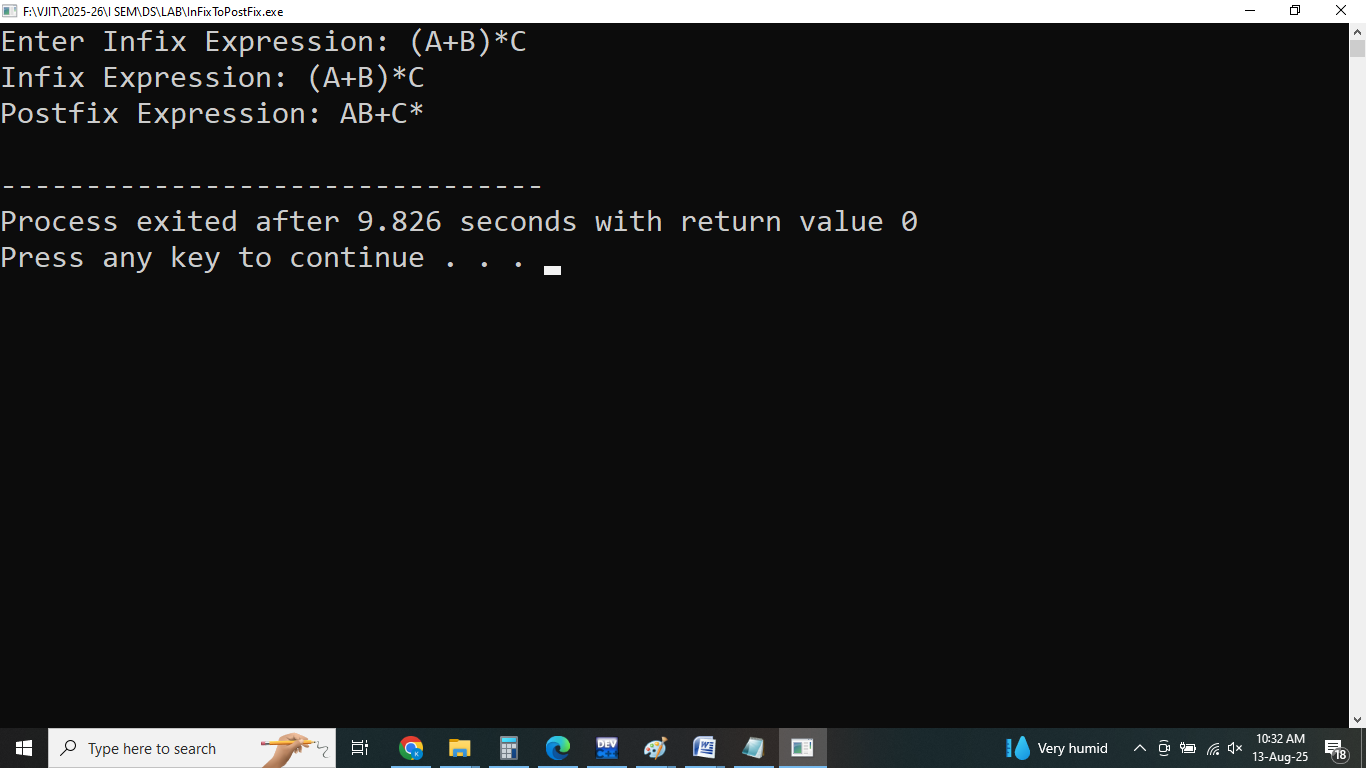
scanf("%s", infix);

infixToPostfix(infix);

return 0;

}

**Output:**

****

**4. Program to Evaluate Postfix Expression**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h> // for isdigit

#define SIZE 100

int stack[SIZE];

int top = -1;

*// Stack Operations*

void **push**(int value)

{

if (top == SIZE - 1)

{

printf("Stack Overflow\n");

} else {

stack[++top] = value;

}

}

int **pop**()

{

if (top == -1)

{

printf("Stack Underflow\n");

return -1;

} else {

return stack[top--];

}

}

*// Evaluate postfix expression*

int **evaluatePostfix**(char\* expr)

{

int i = 0;

char ch;

int op1, op2, result;

while ((ch = expr[i++]) != '\0')

{

if (isdigit(ch))

{

push(ch - '0'); *// Convert char to int*

} else {

op2 = pop();

op1 = pop();

switch (ch)

{

case '+': result = op1 + op2; break;

case '-': result = op1 - op2; break;

case '\*': result = op1 \* op2; break;

case '/': result = op1 / op2; break;

default:

printf("Unsupported operator: %c\n", ch);

return -1;

}

push(result);

}

}

return pop();

}

*// Main function*

int **main**()

{

char postfix[SIZE];

printf("Enter Postfix Expression: ");

scanf("%s", postfix);

int result = evaluatePostfix(postfix);

printf("Result of Postfix Expression: %d\n", result);

return 0;

}

**Output:**

