WRITE A PROGRAM TO CONVERT ((A+B)-C\*(D/E))+F FROM INFIX TO POSTFIX EXPRESSION.

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

if (top == MAX - 1) {

printf("Stack overflow\n");

return;

}

stack[++top] = c;

}

char pop() {

if (top == -1) {

printf("Stack underflow\n");

return -1;

}

return stack[top--];

}

int precedence(char c) {

if (c == '+' || c == '-') {

return 1;

} else if (c == '\*' || c == '/') {

return 2;

} else if (c == '^') {

return 3;

}

return 0;

}

void infixToPostfix(char\* infix, char\* postfix) {

int i = 0, j = 0;

char c;

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

if (isalnum(c)) {

postfix[j++] = c;

}

else if (c == '(') {

push(c);

}

else if (c == ')') {

while (top != -1 && stack[top] != '(') {

postfix[j++] = pop();

}

pop();

}

else {

while (top != -1 && precedence(stack[top]) >= precedence(c)) {

postfix[j++] = pop();

}

push(c);

}

}

while (top != -1) {

postfix[j++] = pop();

}

postfix[j] = '\0';

}

int main() {

char infix[MAX], postfix[MAX];

printf("Enter an infix expression: ");

scanf("%s", infix);

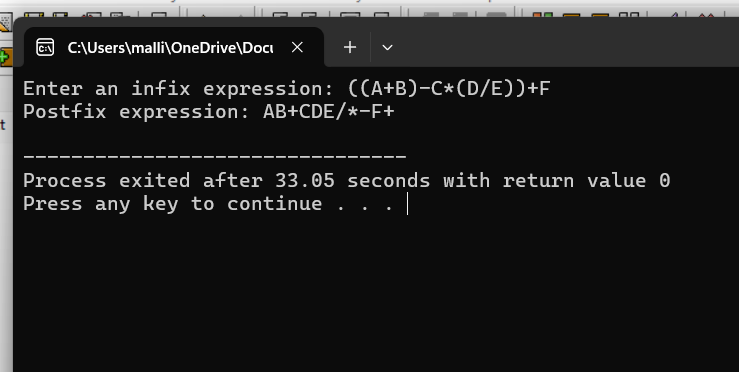
infixToPostfix(infix, postfix);

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

return 0;

}

OUTPUT:



2.CONVERT A+B\*C+D FROM INFINIX TO POSTFIX EXPRESSION.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

if (top == MAX - 1) {

printf("Stack overflow\n");

return;

}

stack[++top] = c;

}

char pop() {

if (top == -1) {

printf("Stack underflow\n");

return -1;

}

return stack[top--];

}

int precedence(char c) {

if (c == '+' || c == '-') {

return 1;

} else if (c == '\*' || c == '/') {

return 2;

} else if (c == '^') {

return 3;

}

return 0;

}

void infixToPostfix(char\* infix, char\* postfix) {

int i = 0, j = 0;

char c;

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

if (isalnum(c)) {

postfix[j++] = c;

}

else if (c == '(') {

push(c);

}

else if (c == ')') {

while (top != -1 && stack[top] != '(') {

postfix[j++] = pop();

}

pop();

}

else {

while (top != -1 && precedence(stack[top]) >= precedence(c)) {

postfix[j++] = pop();

}

push(c);

}

}

while (top != -1) {

postfix[j++] = pop();

}

postfix[j] = '\0';

}

int main() {

char infix[] = "A+B\*C+D";

char postfix[MAX];

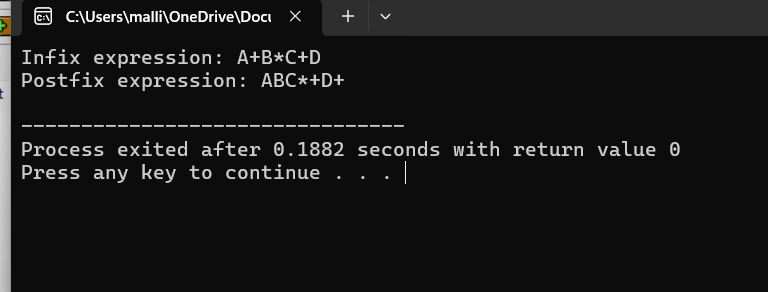
infixToPostfix(infix, postfix);

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

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

return 0;

}



BALANCED SYMBOLS:

#include<stdio.h>

#include <stdlib.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

if (top == MAX - 1) {

printf("Stack overflow\n");

return;

}

stack[++top] = c;

}

char pop() {

if (top == -1) {

printf("Stack underflow\n");

return '\0';

}

return stack[top--];

}

int isOpeningSymbol(char c) {

return c == '(' || c == '{' || c == '[';

}

int isClosingSymbol(char c) {

return c == ')' || c == '}' || c == ']';

}

int isMatchingPair(char opening, char closing) {

return (opening == '(' && closing == ')') ||

(opening == '{' && closing == '}') ||

(opening == '[' && closing == ']');

}

int areSymbolsBalanced(char\* expr) {

int i = 0;

char c;

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

if (isOpeningSymbol(c)) {

push(c);

} else if (isClosingSymbol(c)) {

if (top == -1 || !isMatchingPair(pop(), c)) {

return 0;

}

}

}

return top == -1;

}

int main() {

char expr[MAX];

printf("Enter an expression: ");

scanf("%s", expr);

if (areSymbolsBalanced(expr)) {

printf("The expression is balanced.\n");

} else {

printf("The expression is not balanced.\n");

}

return 0;

}

