Assignment 3

# Name – Akshat Chandrapatle

# Roll no. – 45

# Div. – CS-A

# PRN – 12111449

Write a program to convert a given Infix expression into its

equivalent Postfix expression and evaluate it using stack

#include <stdio.h>

#include<stdlib.h>

#include<string.h>

#include <ctype.h>

struct stack

{

char data;

struct stack \*link;

};

struct stack \*top=NULL;

void push(char x)

{

struct stack \*temp=(struct stack\*)malloc(sizeof(struct stack));

temp->data=x;

temp->link=NULL;

temp->link=top;

top=temp;

}

void pop()

{

if(top==NULL) return;

struct stack \*temp=top;

top=top->link;

free(temp);

}

char peek()

{

if(top==NULL) return 0;

return top->data;

}

int IsOpeningParentheses(char C)

{

if(C == '(' || C == '{' || C=='[')

return 1 ;

return 0;

}

int IsClosingParentheses(char C)

{

if(C == ')' || C == '}' || C==']')

return 1 ;

return 0;

}

int isoperand(char C)

{

if(C >= '0' && C <= '9') return 1;

if(C >= 'a' && C <= 'z') return 1;

if(C >= 'A' && C <= 'Z') return 1;

return 0;

}

int isoperator(char c)

{

if(c=='+' || c=='\*' || c=='-' || c=='/') return 1;

return 0;

}

int priority(char operator) {

switch (operator) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return -1;

}

}

int IsMatchingPair(char character1, char character2) {

if (character1 == '(' && character2 == ')') {

return 1;

} else if (character1 == '{' && character2 == '}') {

return 1;

} else if (character1 == '[' && character2 == ']') {

return 1;

}

return 0;

}

int isValidInfixExpression(char\* infix\_expression) {

int i, length;

char character;

length = strlen(infix\_expression);

while(top!=NULL){

pop();

}

for(i = 0; i < length; i++) {

character = infix\_expression[i];

if(isoperand(character)) {

}

else if(isoperator(character)) {

if(i > 0 && isoperator(infix\_expression[i - 1])) {

return 0;

}

if(i == length - 1) {

return 0;

}

if(i==0){

return 0;

}

if (i < length - 1 && !isoperand(infix\_expression[i + 1]) && !IsOpeningParentheses(infix\_expression[i + 1])) {

return 0;

}

}

else if(IsOpeningParentheses(character)) {

if(i>0 && !isoperator(infix\_expression[i -1])){

return 0;

}

else{

push(character);

}

}

else if(IsClosingParentheses(character)) {

if(top == NULL || !IsMatchingPair(top->data, character)) {

return 0;

}

else {

pop();

}

if (i < length - 1 && !isoperator(infix\_expression[i + 1]) && !IsClosingParentheses(infix\_expression[i + 1])) {

return 0;

}

}

}

if(top != NULL) {

return 0;

}

return 1;

}

char\* infixToPostfix(char \*A,int n)

{

char postfix[100];

int j=0;

for(int i=0;i<n;i++)

{

if(A[i] == ' ' || A[i] == ',')

continue;

if(isoperand(A[i])>0){

postfix[j++]=A[i];

}

else if(isoperator(A[i])>0)

{

if(priority(peek())>=priority(A[i]))

{

while(top!=NULL && !IsOpeningParentheses(peek()))

{

postfix[j++]=peek();

pop();

}

}

push(A[i]);

}

else if(IsOpeningParentheses(A[i]))

{

push(A[i]);

}

else if(IsClosingParentheses(A[i]))

{

while(top!=NULL && !IsOpeningParentheses(peek())) {

postfix[j++]=peek();

pop();

}

pop();

}

}

while(top!=NULL)

{

postfix[j++]=peek();

pop();

}

printf("Postfix:%s",postfix);

}

void main() {

char arr[100];

printf("Infix: ");

fgets(arr, sizeof(arr), stdin);

arr[strcspn(arr, "\n")] = '\0';

int n = strlen(arr);

if (isValidInfixExpression(arr)) {

infixToPostfix(arr, n);

} else {

printf("Invalid infix expression. Please enter a valid infix expression.\n");

main();

}

}

