Program to convert infix to prefix

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

// Stack data structure

struct Stack {

int top;

char items[MAX\_SIZE];

};

// Check if a character is an operator (+, -, \*, /, ^)

int isOperator(char ch) {

if (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^') {

return 1;

}

return 0;

}

// Get precedence of an operator

int precedence(char ch) {

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

return 1;

}

else if (ch == '\*' || ch == '/') {

return 2;

}

else if (ch == '^') {

return 3;

}

return -1;

}

// Push an item to the stack

void push(struct Stack\* stack, char item) {

if (stack->top == MAX\_SIZE - 1) {

printf("Stack Overflow\n");

}

else {

stack->top++;

stack->items[stack->top] = item;

}

}

// Pop an item from the stack

char pop(struct Stack\* stack) {

if (stack->top == -1) {

printf("Stack Underflow\n");

return '\0';

}

else {

char item = stack->items[stack->top];

stack->top--;

return item;

}

}

// Convert infix expression to prefix expression

void infixToPrefix(char\* infix, char\* prefix) {

int i, j;

char item;

struct Stack stack;

// Initialize stack

stack.top = -1;

// Reverse infix expression

strrev(infix);

// Traverse infix expression

for (i = 0, j = 0; infix[i] != '\0'; i++) {

// If current character is an operand

if (isalnum(infix[i])) {

prefix[j] = infix[i];

j++;

}

// If current character is an operator

else if (isOperator(infix[i])) {

while (stack.top != -1 && precedence(infix[i]) < precedence(stack.items[stack.top])) {

item = pop(&stack);

prefix[j] = item;

j++;

}

push(&stack, infix[i]);

}

// If current character is a parenthesis

else if (infix[i] == ')' || infix[i] == '(') {

if (infix[i] == ')') {

push(&stack, infix[i]);

}

else {

while (stack.items[stack.top] != ')') {

item = pop(&stack);

prefix[j] = item;

j++;

}

pop(&stack);

}

}

}

// Pop remaining items from stack and append to prefix expression

while (stack.top != -1) {

item = pop(&stack);

prefix[j] = item;

j++;

}

// Reverse prefix expression to get final result

strrev(prefix);

prefix[j] = '\0';

}

// Main function

int main() {

char infix[MAX\_SIZE], prefix[MAX\_SIZE];

printf("Enter infix expression: ");

scanf("%s", infix);

infixToPrefix(infix, prefix);

printf("Prefix expression: %s\n", prefix);

return 0;

}

