**infix-to-postfix.c**

//write a program to convert infix to postfix.

#include <stdio.h>

#include <stdlib.h>

typedef struct List{

float data;

char op;

struct List \*next;

}list;

typedef struct charStack{

char data;

struct charStack \*next;

}cstack;

void push(list \*\*l1, char op, float number){

list \*new\_node = (list\*)malloc(sizeof(list));

new\_node->data = number;

new\_node->op = op;

if(!\*l1){

new\_node->next = \*l1;

\*l1 = new\_node;

return;

}

new\_node->next = NULL;

list \*temp = \*l1;

while(temp->next){

temp = temp->next;

}

temp->next = new\_node;

}

void push\_char\_top(cstack \*\*st, char data){

cstack \*new\_node = (cstack\*)malloc(sizeof(cstack));

new\_node->data = data;

new\_node->next = \*st;

\*st = new\_node;

}

void push\_char\_bottom(cstack \*\*st, char data){

cstack \*new\_node = (cstack\*)malloc(sizeof(cstack));

new\_node->data = data;

if(!\*st){

new\_node->next = \*st;

\*st = new\_node;

return;

}

new\_node->next = NULL;

cstack \*temp = \*st;

while(temp->next){

temp = temp->next;

}

temp->next = new\_node;

}

char pop\_top(cstack \*\*st){

cstack \*temp = \*st;

\*st = (\*st)->next;

char data = temp->data;

free(temp);

return data;

}

void scan\_char(cstack \*\*head){

char c;

while(1){

scanf("%c",&c);

if(c == 32)continue;

push\_char\_bottom(head,c);

if(c == 10)break;

}

}

void displayExp(list \*head) {

while (head != NULL) {

if(head->op == '1'){

long num = head->data;

if(num < head->data)

printf("%.2f ",head->data);

else

printf("%d ",num);

}else{

printf("%c ", head->op);

}

head = head->next;

}

printf("\n");

}

int isNumber(char c){

return (c >= 48 && c <= 57) || c == 46;

}

int isOperator(char c){

return (c == 45 || c == 43 || c == 42 || c == 47 || c == 37 || c == 94);

}

int isBrace(char c){

return (c == 40 || c == 41);

}

int opPref(char c){

switch(c){

case 94:

return 6;

case 47:

return 5;

case 42:

case 37:

return 3;

case 43:

case 45:

return 1;

case 40:

case 41:

return 0;

default:

exit(0);

}

}

float toNumber(cstack \*st){

float num = 0, point = 0;

int isNeg = 0, isPoint = 0;

if(st->data == '-'){

isNeg = 1;

st = st->next;

}

while(st->next != NULL){

if(st->data == 46){

isPoint = 1;

st = st->next;

break;

}

int n = st->data-48;

num = (num\*10) + n;

st = st->next;

}

int len = 1;

if(isPoint){

while(st->next != NULL){

if(st->data == 46){

printf("multiple '.' found. \nProgram Crashed.");

exit(0);

}

int n = st->data-48;

point = (point\*10)+n;

len \*= 10;

st = st->next;

}

}

num = (num\*len+point)/len;

if(isNeg){

return num\*-1;

}

return num;

}

list\* charToList(cstack \*st){

cstack\* number = NULL, \*prev = NULL;

list \*l1 = NULL;

int isNeg = 0;

char c;

while(st != NULL){

c = st->data;

if(isNumber(c)){

push\_char\_bottom(&number,c);

}

if(isOperator(c) || isBrace(c)){

if(number != NULL){

push\_char\_bottom(&number,c);

push(&l1,'1',toNumber(number));

number = NULL;

}

if(c == '-'){

if(prev == NULL || isOperator(prev->data) || isBrace(prev->data)){

push\_char\_bottom(&number,c);

}else{

push(&l1,c,0);

}

}else{

push(&l1,c,0);

}

}

prev = st;

st = st->next;

}

if(number != NULL){

push\_char\_bottom(&number,c);

push(&l1,'1',toNumber(number));

number = NULL;

}

return l1;

}

void infixToPostfix(list \*\*l){

list \*l1 = \*l, \*postlist = NULL;

cstack \*oplist = NULL;

while(l1 != NULL){

if(l1->op == '1'){

push(&postlist,'1',l1->data);

}else{

if(oplist == NULL || l1->op == 40 || opPref(l1->op) > opPref(oplist->data)){

push\_char\_top(&oplist,l1->op);

}else{

if(l1->op == ')'){

while(oplist != NULL && oplist->data != '('){

push(&postlist,pop\_top(&oplist),0);

}

if(oplist != NULL && oplist->data == '(')

char garbg = pop\_top(&oplist);

}else{

while(oplist != NULL && opPref(l1->op) <= opPref(oplist->data)){

push(&postlist,pop\_top(&oplist),0);

}

push\_char\_top(&oplist,l1->op);

}

}

}

l1 = l1->next;

}

while(oplist != NULL){

push(&postlist,pop\_top(&oplist),0);

}

\*l = postlist;

}

void main(){

cstack \*st1 = NULL;

printf("Enter the Infix Expression: ");

scan\_char(&st1);

list \*l1 = charToList(st1);

printf("Your Infix Expression is:\n");

displayExp(l1);

printf("The Postfix Version of the Expression is:\n");

infixToPostfix(&l1);

displayExp(l1);

}

**OUTPUT**

PS S:\WorkSpace\CollegeWork\DataStructure\Temp> gcc .\infix-to-postfix.c

PS S:\WorkSpace\CollegeWork\DataStructure\Temp> ./a

Enter the Infix Expression: ((10\*((-13+17)/1.5-17)^3+10)\*-1)

Your Infix Expression is:

( ( 10 \* ( ( -13 + 17 ) / 1.50 - 17 ) ^ 3 + 10 ) \* -1 )

The Postfix Version of the Expression is:

10 -13 17 + 1.50 / 17 - 3 ^ \* 10 + -1 \*

PS S:\WorkSpace\CollegeWork\DataStructure\Temp>