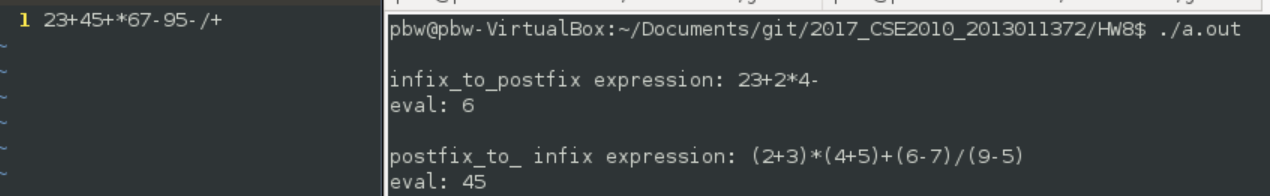
HW8: infix evaluation, postfix to infix conversion

2013011372 박병욱

1. 실행결과



1. 소스코드

#include <stdio.h>

#include <stdlib.h>

//#include<conio.h>

#include<ctype.h>

#define MAX\_STACK\_SIZE 50

#define MAX\_STACK\_NUM 50

typedef int element;

typedef struct {

element stack[MAX\_STACK\_SIZE];

int top;

}StackType;

char tmp[MAX\_STACK\_SIZE] = "";

// 스택 초기화 함수

void init(StackType \*s) {

s->top = -1;

}

// 공백 상태 검출 함수

int is\_empty(StackType \*s) {

//printf("---%d\n",(s->top == -1));

return (s->top == -1);

}

// 포화 상태 검출 함수

int is\_full(StackType \*s) {

return (s->top == (MAX\_STACK\_SIZE-1));

}

// 삽입함수

void push(StackType \*s, element item) {

//printf("push %d\n",item);

if ( is\_full(s) ) {

fprintf(stderr," \n");

return;

}else s->stack[++(s->top)] = item;

}

// 삭제함수

element pop(StackType \*s) {

//printf("pop %d\n",s->stack[(s->top)]);

if ( is\_empty(s) ) {

fprintf(stderr, "스택 공백 에러\n");

exit(1);

} else return s->stack[(s->top)--];

}

// 피크함수

element peek(StackType \*s) {

if ( is\_empty(s) ) {

fprintf(stderr, "스택 공백 에러\n");

exit(1);

} else return s->stack[s->top];

}

//Moves Stack from to

void moveStack(StackType\* from, StackType\* to){

StackType tmp;

init(&tmp);

while(!is\_empty(from))

push(&tmp,pop(from));

while(!is\_empty(&tmp))

push(to,pop(&tmp));

}

//Moves Stack from to with Bracket

void moveStackWithBraket(StackType\* from, StackType\* to){

StackType tmp;

init(&tmp);

push(&tmp,'('-'0');

while(!is\_empty(from))

push(&tmp,pop(from));

push(&tmp,')'-'0');

while(!is\_empty(&tmp))

push(to,pop(&tmp));

}

// 연산자의 우선순위를 반환한다.

int prec(char op) {

switch(op){

case '(': case ')': return 0;

case '+': case '-': return 1;

case '\*': case '/': return 2;

}

return -1;

}

//infix to postfix

void infix\_to\_postfix(char infix[],char postfix[])

{

StackType s;

char x,token;

int i,j; //i-index of infix, j-index of postfix

init(&s);

j=0;

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

{

token=infix[i];

if(isalnum(token))

postfix[j++]=token;

else

if(token=='(')

push(&s,'(');

else

if(token==')')

while((x=pop(&s))!='(')

postfix[j++]=x;

else

{

while(!is\_empty(&s) && (prec(token) <= prec(peek(&s))))

{

x=pop(&s);

postfix[j++]=x;

}

push(&s,token);

}

}

while(!is\_empty(&s))

{

x=pop(&s);

postfix[j++]=x;

}

postfix[j]='\0';

printf("\ninfix\_to\_postfix expression: %s", postfix);

}

char\* postfix\_to\_infix(char expression[])

{

/\* fill in the blank \*/

StackType s[MAX\_STACK\_NUM];

char token;

char\* ret;

int i,j;

StackType\* oa1;

StackType\* oa2;

StackType tmp;

j=0;

ret = malloc(sizeof(char)\*100);

for(i=0;i<MAX\_STACK\_NUM;i++)

init(&s[i]);

init(&tmp);

for(i=0; expression[i] != '\0'; i++)

{

token = expression[i];

if(isalnum(token))

{

push(&s[j],token-'0');

j++;

}else

{

oa1 = &s[j-1];

oa2 = &s[j-2];

//Push result depends on operation

switch (token){

case '+':

moveStack(oa2,&tmp);

moveStack(oa1,oa2);

push(oa2,'+'-'0');

moveStack(&tmp,oa2);

break;

case '\*':

moveStack(oa2,&tmp);

moveStackWithBraket(oa1,oa2);

push(oa2,'\*'-'0');

moveStackWithBraket(&tmp,oa2);

break;

case '-':

moveStack(oa2,&tmp);

moveStack(oa1,oa2);

push(oa2,'-'-'0');

moveStack(&tmp,oa2);

break;

case '/':

moveStack(oa2,&tmp);

moveStackWithBraket(oa1,oa2);

push(oa2,'/'-'0');

moveStackWithBraket(&tmp,oa2);

break;

}

j = j-1;

}

}

if(j != 1){

printf("\n postfix\_to\_infix error input file is incorrect\n"); return ret;

}

for(i=0; !is\_empty(&s[j-1]);i++)

{

token = pop(&s[j-1]) + '0';

ret[i] = (char)token;

}

ret[i] = '\0';

printf("\npostfix\_to\_ infix expression:");

printf(" %s\n",ret);

return ret;

}

int postfixEval(char exp[])

{

/\* fill in the blank \*/

StackType s;

char token;

int i;

int oa1, oa2;

init(&s);

for(i=0; exp[i] != '\0';i++)

{

token = exp[i];

if(isalnum(token))

push(&s,token -'0');

else{

oa1 = pop(&s);

oa2 = pop(&s);

//push result of operation depends on case

switch (token){

case '+':

push(&s,oa2+oa1);

break;

case '\*':

push(&s,oa2\*oa1);

break;

case '-':

push(&s,oa2-oa1);

break;

case '/':

push(&s,oa2/oa1);

break;

}

}

}

return pop(&s);

}

int main() {

FILE \*f;

char postfix[30];

char\* tmp = (char\*)malloc(sizeof(char)\*20);

f=fopen("input1.txt","r");

fscanf(f,"%s", tmp);

infix\_to\_postfix(tmp,postfix);

printf("\neval: %d \n", postfixEval(postfix));

f=fopen("./input3.txt","r");

fscanf(f,"%s", tmp);

postfix\_to\_infix(tmp);

printf("eval: %d \n", postfixEval(tmp));

return 0;

}

1. 소스코드 설명

* Topic1

Exp[]로 부터 토큰을 받고, isalnum()을 이용해, operand인지 operator인지 구분하고, operand일 경우 push, operator일경우 스택 상위2 operand를 이용해 연산한 값을 푸쉬하였다.

그리고 마지막 스택에있는 값을 리턴한다.

* Topic2

moveStack();

moveStackWithBracket()함수를 추가하였다.

StackType s[MAX\_STACK\_NUM]을 초기화하고,

J는 s전체스택(s[i]들을 담은 스택이라고 보았을 경우)의 top을 가리킨다.

Exp[]로 부터 토큰을 이용해, operand일경우 s[j]에 푸쉬하고 j++;

operator일경우 s[j-1], s[j-2]로 부터의 값 중간에 연산자를 넣고 s[j-2]에 푸쉬한다.

다만 operator가 ‘\*’나 ‘/’ 일 경우 bracket도 추가한다.

아래와 같은 매트릭을 이용했다.

Postfix to Infix conversion

Example

abc-+de-fg-h+/\*

|  |  |
| --- | --- |
| **Expression** | **Stack** |
| abc-+de-fg-h+/\* | NuLL |
| bc-+de-fg-h+/\* | "a" |
| c-+de-fg-h+/\* | |  | | --- | | "b" | | "a" | |
| -+de-fg-h+/\* | |  | | --- | | "c" | | "b" | | "a" | |
| +de-fg-h+/\* | |  | | --- | | "b - c" | | "a" | |
| de-fg-h+/\* | |  | | --- | | "a+b-c" | |
| e-fg-h+/\* | |  | | --- | | "d" | | "a+b-c" | |
| -fg-h+/\* | |  | | --- | | "e" | | "d" | | "a+b-c" | |
| fg-h+/\* | |  | | --- | | "d - e" | | "a+b-c" | |
| g-h+/\* | |  | | --- | | "f" | | "d - e" | | "a+b-c" | |
| -h+/\* | |  | | --- | | "g" | | "f" | | "d - e" | | "a+b-c" | |
| h+/\* | |  | | --- | | "f-g" | | "d - e" | | "a+b-c" | |
| +/\* | |  | | --- | | "h" | | "f-g" | | "d - e" | | "a+b-c" | |
| /\* | |  | | --- | | "f-g+h" | | "d - e" | | "a+b-c" | |
| \* | |  | | --- | | "(d-e)/(f-g-h)" | | "a+b-c" | |
| Null | |  | | --- | | "(a+b-c)\*(d-e)/(f-g+h)" | |