实验二 栈、队列及其应用

题目：算术表达式求值演示

班级：少年班学院少年班 姓名：邱子悦 学号：PB18000028 完成日期：2019.10.31

一.实验要求

编写一个程序，该程序计算输入的算术表达式，然后输出计算结果。

基本要求:

1. 计算表达式的值。

选做要求:

1. 增加乘方、单目减等运算;

2. 运算量可以是变量或实数类型。

二.设计思路

思路：

基础框架是用带头结点的链栈，将中缀表达式求值，其中要考虑处理括号和算符优先级，写成了 precede()函数，9*9=81 种情况对应的返回值，列表如下：

1\2	+	-	*	/	(	)	#	^
+	>	>	<	<	<	>	>	<
-	>	>	<	<	<	>	>	<
*	>	>	>	>	<	>	>	<
/	>	>	>	>	<	>	>	<
(	<	<	<	<	<	=	>	<
)	>	>	>	>	=	>	>	>
#	<	<	<	<	<	<	=	<
^	>	>	>	>	<	>	>	>

要借助两个栈：数值栈和运算符栈。

由于数值的类型为 double，运算符类型为 char，基础函数要分别写两种，有 CreateStack()，Push()，Pop()，GetTop()四类。

另外还有，isOperator()函数判断一个字符是否是操作符（是否属于（，），+，-，

*, /, ^, # 中一个），f()函数将 a β b 计算出来。

最核心的 transform()函数，是将输入的表达式字符串，利用前面这些函数，计算出结果。

transform()函数算法原理：

0. 创建数值栈 head1 和运算符栈 head2，将#压进运算符栈两次，并将表达式尾端添上一个#。

1. 从左至右逐个读取表达式的每一项α

2. 如果α是#，且运算符栈的栈顶元素是#，结束。

3. α是数：Push(head1, α); goto1;

4. α是运算符：

β =GetTop(head2)

比较β与α的优先级：

β<α Push(head2,α); goto 1;
β=α Pop(head2,α); goto 1;
β>α Pop(head1,b); Pop(head1,a); Pop(head2,β); Push(head1, a β b); goto 4;

考虑到附加的变量求值功能，我在 main 函数中加入了“过滤”功能，即，将 x 替换为 x 被赋的数值。

考虑到乘方运算，我把‘^’加入了算符优先级比较表。

考虑到单目减运算，对于表达式开头作了一次判断，判断有没有负号开头，并且检测了每一次‘(-’的连续出现。

三.关键代码讲解

->两种结构体：

```
typedef struct StackNode1{
    double data;
    struct StackNode1 *next;
```

```
}StackNode1;

typedef struct StackNode2{
	char data;
	struct StackNode2 *next;
}StackNode2;
```

->创建栈头结点

```
StackNode1* CreateStack1(){
	StackNode1* head = (StackNode1*)malloc(sizeof(StackNode1));
	if(head == NULL){
		printf("Memory1 allocate failed.\n");
		return NULL;
	}
	head->data = 0;
	head->next = NULL;
	return head;
}

StackNode2* CreateStack2(){
	StackNode2* head = (StackNode2*)malloc(sizeof(StackNode2));
	if(head == NULL){
		printf("Memory2 allocate failed.\n");
		return NULL;
	}
	head->data = '\0';
	head->next = NULL;
	return head;
}
```

->入栈

```
void Push1(StackNode1* head, double e){
	if(head == NULL)
		return;
	StackNode1* node = (StackNode1*)malloc(sizeof(StackNode1));
	if(node == NULL){
		printf("Memory1 allocate failed.\n");
		return;
	}
	node->data = e;
	node->next = head->next;
	head->next = node;
```

```
}
```

->出栈

```
double Pop1(StackNode1* head){
    if(head == NULL || head->next == NULL){
        printf("Error1.\n");
        return 0;
    }
    StackNode1* node = (StackNode1*)malloc(sizeof(StackNode1));
    if(node == NULL){
        printf("Memory1 allocate failed.\n");
        return 0;
    }
    StackNode1* temp = head->next;
    head->next = temp->next;
    double val = temp->data;
    free(temp);
    return val;
}
```

->获得顶端元素值

```
char GetTop2(StackNode2* head){
    if(head == NULL || head->next == NULL){
        printf("Error GetTop2.\n");
        return 0;
    }
    return head->next->data;
}
```

->优先级分析（把前面的表格翻译成 switch-case 和 if-else 语句）

```
char precede(char ch1, char ch2){
    switch(ch1){
        case '+':
        case '-':
            if(ch2=='+'||ch2=='-'||ch2==')'||ch2=='#')
                return '>';
            else if(ch2=='*'||ch2=='/'||ch2=='('||ch2=='^')
                return '<';
            break;
        case '*':
        case '/':
```

```
                if(ch2=='('||ch2=='^')
                        return '<';
                else
                        return '>';
        case '(':
                if(ch2==')')
                        return '=';
                else if(ch2=='#')
                        return '>';
                else
                        return '<';
        case ')':
                if(ch2=='(')
                        return '=';
                else
                        return '>';
        case '#':
                if(ch2=='#')
                        return '=';
                else
                        return '<';
        case '^':
                if(ch2=='(')
                        return '<';
                else
                        return '>';
    }
}
```

->判断是不是运算符

```
int isOperator(char ch){
    char a[]="()+-*/^#";
    for(int i=0;a[i]!='\0';i++){
        if(a[i]==ch)
            return 1;
    }
    return 0;
}
```

->计算

```
double f(double a, double b, char ch){
    //计算 a ch b
```

```
        switch(ch){
            case '+': return a+b;
            case '-': return a-b;
            case '*': return a*b;
            case '/': return a/b;
            case '^': return pow(a,b);
            default: printf("Error.");
        }
        return 0;
}
```

->核心函数

```
double transform(char exp[]){//exp 是输入的表达式字符串

        StackNode1* head1 = NULL;//数值栈 OPND

        StackNode2* head2 = NULL;//运算符栈 OPTR

        head1 = CreateStack1();
        head2 = CreateStack2();

        Push2(head2,'#');
        Push2(head2,'#');

        int k=0,j=0;//从前往后读取

        char* p;//不影响 exp

        char* save[LEN];//用在将字符串转为 double 保存

        double val,b,a;

        char ch;//存运算符

        int flag;

        for(p=exp;*p!='\0';p++)
            ;
        *p='#';
        *(p+1)='#';
        *(p+2)='\0';（一定要记得最后存为‘\0’，不然容易出意外的错）

        p=exp;

        if(*p=='-'){//单目减功能，句首
```

```
        while(*(p+j)!='\0' && !isOperator(*(p+j))){
            j++;
        }
        memset(save,'\0', sizeof(char)*LEN);
        strncpy(save,p,j* sizeof(char));
        val = atof(save);
        Push1(head1, val);
        k=j;
}

while(*(p+k)!='#' || GetTop2(head2)!='#'){
        if(!isOperator(*(p+k)) || (*(p+k)=='('&&*(p+k+1)=='-')){
            if(!isOperator(*(p+k))){
                j = k;

                while(*(p+j)!='\0' && !isOperator(*(p+j))){//交换位置

                    j++;
                }
                memset(save,'\0', sizeof(char)*LEN);
                strncpy(save,p+k,(j-k)* sizeof(char));
                val = atof(save);
                Push1(head1, val);
                k=j;
            }
            else{
                j = k+2;

                while(*(p+j)!=')'){//交换位置

                    j++;
                }
                memset(save,'\0', sizeof(char)*LEN);
                strncpy(save,p+k+1,(j-k-1)* sizeof(char));
                val = atof(save);
                Push1(head1, val);
                k=j+1;

            }（单目减功能，句中）

        }
        else{
            flag=1;
            while(flag==1){

                flag=0;（相当于手动实现了前面算法的 goto 4）

                ch=GetTop2(head2);
```

```
                switch(precede(ch,*(p+k))){
                    case '<': Push2(head2,*(p+k)); break;
                    case '=': Pop2(head2); break;
                    case '>': b=Pop1(head1); a=Pop1(head1);
                              ch=Pop2(head2); Push1(head1,f(a,b,ch));
                              flag=1; break;
                }
            }
            k++;
        }
    }
    printf("%g\n",head1->next->data);
}
```

->主函数中输入变量功能的“过滤”函数，核心想法就是把 x 替换成数字

```
    while(scanf("%s",s)){
        /*
          过滤替换 x
         */
        for(i=0;*(s+i)!='\0';i++){
            if(*(s+i)=='x'){
                printf("x=");
                scanf("%s",x);
                break;
            }
        }
        for(i=0;*(s+i)!='\0';i++) {
            if(*(s+i)=='x'){
                memset(save,'\0', sizeof(char)*LEN);
                strncpy(save,s+i+1,LEN * sizeof(char));//把后半段存起来
                strncpy(s+i,x,strlen(x) * sizeof(char));
                strncpy(s+i+strlen(x),save, LEN* sizeof(char));
```

（相当于把 x 之后的部分用 save 存起来，把数值接上去，再把 save 中的部分接到数值的后面）

```
            }
        }
        transform(s);
    }
```

四.调试分析

1. 优先级表没考虑清楚

1\2	+	-	/	(	)	#	^
+	>	>	<	<	>	>	<
-	>	>	<	<	>	>	<
*	>	>	>	<	>	>	<
/	>	>	>	<	>	>	<
(	<	<	<	<	=	>	<
)	>	>	>	=	>	>	>
#	<	<	<	<	<	=	<
^	>	>	>	<	>	>	>

2. LEN 取小了，输出为 0

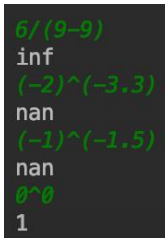
LEN 为存输入的表达式的数组长度，最初 define 为 20，输入(((6+6)*6+3)*2+6)*2

刚好 19 个字符，错误就显现了，改为 50 就没有出错了。

3. isOperator()中最初忘了考虑#

4. 乘方运算要限定指数和底数。比如底数为负数，而指数很多情况都会使得结

果为“nan”，即非实数，也可以有 0^0 这种数学上无意义的输入。如图：



有两种处理方式：有非法输入就报错并结束程序，或者人为限制输入。

我选择后者，主要原因是程序无需过度复杂，我考虑到这类情况，并且如上图尝试了一些输入了解了可能发生的情况即可。人为限制输入，和限制输入的表达式是标准的，是同一个道理。但是在实际应用中，不可以忽视这些情况。

5. 分母为 0 的除法是不合规范的，有两种处理方式：打印“错误！分母为 0！”

并结束程序，或者强行输出。我选择了后者，如图： 

inf 表示无穷。

6. 替换 x 为数值

```
for(i=0;*(s+i)!='\0';i++) {
    if(*(s+i)=='x'){
        memset(save,'\0', sizeof(char)*LEN);
        strncpy(save,s+i+1,LEN * sizeof(char));//把后半段存起来
        strncpy(s+i,x,strlen(x) * sizeof(char));
        strncpy(s+i+strlen(x),save, LEN* sizeof(char));
    }
}
```

memset 函数进行初始化，和每个小段的边界值取上还是取下，都是要注意的小地方，写代码的时候要先想清楚。

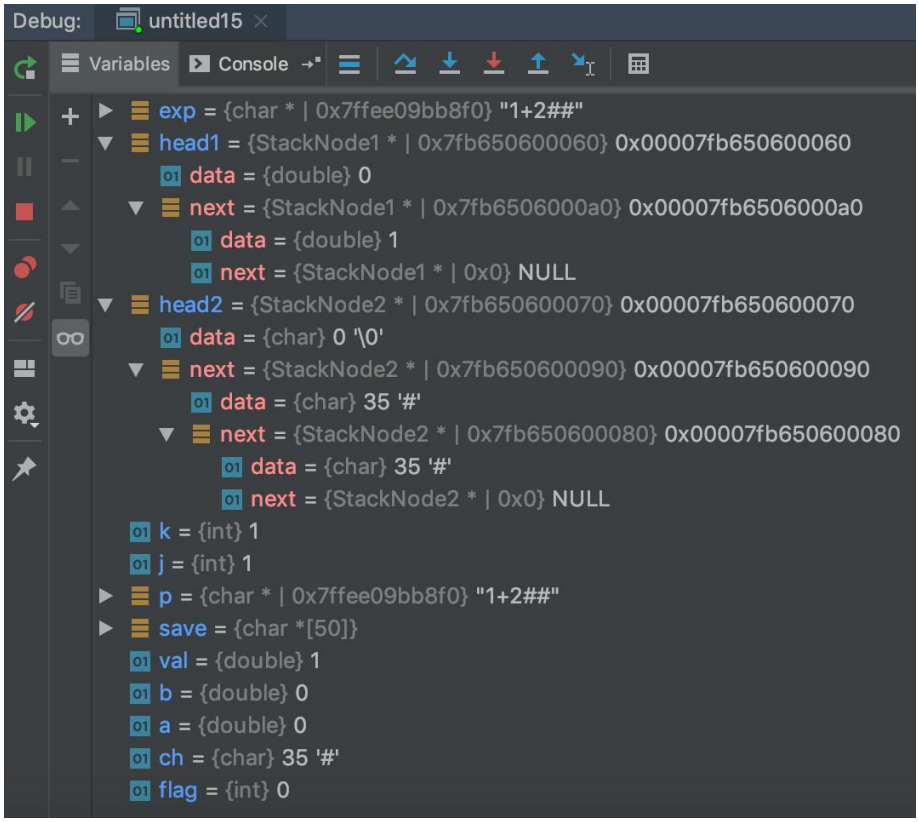
7.这次的报错我总能很快找到问题出在哪，原因在于 printf 出每个部分独特的报错信息，例如：

```
//出栈
double Pop1(StackNode1* head){
    if(head == NULL || head->next == NULL){
        printf("Error1.\n");
        return 0;
    }
    StackNode1* node = (StackNode1*)malloc(sizeof(StackNode1));
    if(node == NULL){
        printf("Memory1 allocate failed.\n");
        return 0;
    }
    StackNode1* temp = head->next;
    head->next = temp->next;
    double val = temp->data;
    free(temp);
    return val;
}
```

```
//获得顶端元素值
char GetTop2(StackNode2* head){
    if(head == NULL || head->next == NULL){
        printf("Error GetTop2.\n");
        return 0;
    }
    return head->next->data;
}
```

```
double f(double a, double b, char ch){
    //计算a ch b
    switch(ch){
        case '+': return a+b;
        case '-': return a-b;
        case '*': return a*b;
        case '/': return a/b;
        case '^': return pow(a,b);
        default: printf("Error.");
    }
    return 0;
}
```

并且要善用 CLion 的调试功能，可以很容易地观测链栈的变化过程，如图：



五.代码测试

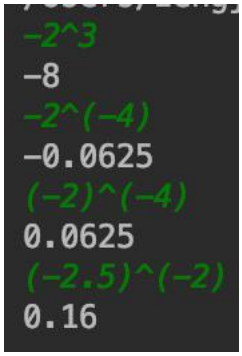
1.整数、双目运算，测试样例来自《数据结构题集》

```
8
8
1+2+3+4
10
88-1*5
83
1024/(4*8)
32
(20+2)*(6/2)
66
3-3-3
-3
8/(9-9)
inf
```

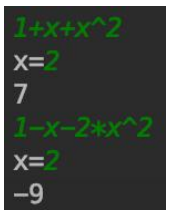
```
2*(6+2*(3+6*(6+6)))
312
```

```
(((6+6)*6+3)*2+6)*2
312
```

2.乘方运算与单目减，并适当输入了非整数



3.输入含变量表达式，并计算



我认为这个功能能写出来就可以了，出于健壮性考虑，我假设 x 的输入非负。

如果想要输入负数，只需要加一层判断，并且在两端加上括号，就能计算了，做重复工作没有太大必要。

六. 实验总结

第一次尝试较大的栈程序，使用 CLion 书写和执行。

从最初的思路混乱到后来功能清晰、细节处理到位，收获很大，逐渐体会到了思考全面以及迅速找到出错的地方的方法。

实验进行较为顺利，按时完成实验。

七.附录

main.c　　//主程序