《数据结构与算法分析》

实验报告

实验名称	基于栈的中缀算术表达式求值
姓名	曾庆文
学号	**23060218**
院系	自动化学院
专业	人工智能
班级	**23061011**
实验时间	**2024.4**

一、实验目的

1. 掌握栈的基本操作算法的实现，包括栈初始化、进栈、出栈、取栈顶元素等。
2. 掌握利用栈实现中缀表达式求值的算法。

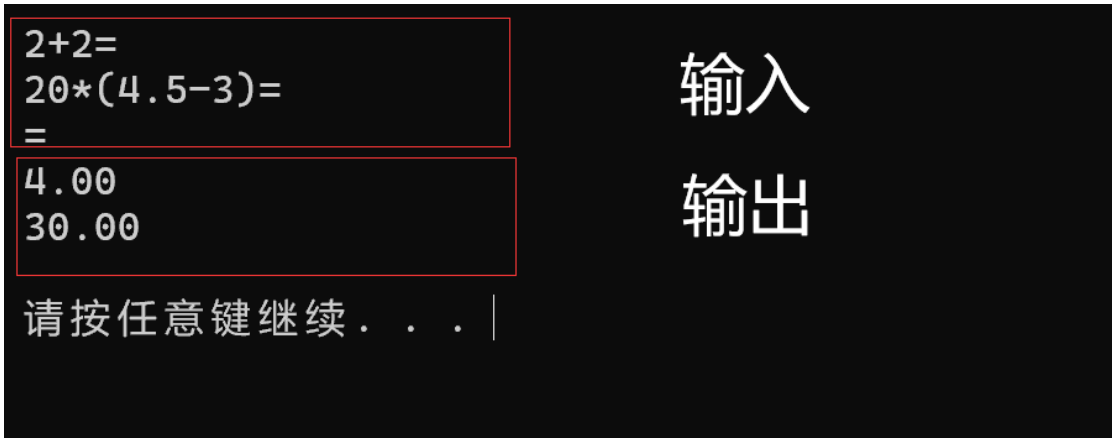
二、实验设备与环境

编辑器：Visual Studio Code
编译器：gcc

三、实验内容与结果

基本操作的输入输出结果如下：

1. 测试样例：



2. 补充测试：
   2.1 结果为负的减法：

2.2 与小数有关的除法：

2.3 有括号的表达式：

实验全部代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define maxsize 100

//使用顺序栈定义
typedef struct
{
    char *base;
    char *top;
    int stacksize;
}SqStackC;
    //操作符栈结构体类型
typedef struct
{
    double *base;
    double *top;
    int stacksize;
}SqStackN;
    //操作数栈结构体类型

//共 10 个功能函数
//C 结尾函数为对运算符操作的函数
//N 结尾函数为对运算数操作的函数
void InitStackC(SqStackC *s);
    //操作符栈初始化函数
void InitStackN(SqStackN *s);
    //操作数栈初始化函数
void PushC(SqStackC *s,char e);
    //操作符栈入栈函数
void PushN(SqStackN *s,double e);
    //操作数栈入栈函数
void PopC(SqStackC *s,char *e);
    //操作符栈出栈函数
void PopN(SqStackN *s,double *e);
    //操作数栈出栈函数
char GetTopC(SqStackC s);
    //取操作符栈栈顶元素函数
double GetTopN(SqStackN s);
    //取操作数栈栈顶元素函数
char Precede(char theta1,char theta2);
    //判定操作符栈的栈顶元素与读入的操作符之间优先关系的函数
```

```
32.double Operate(double a,char theta,double b);
     //进行二元运算的函数
33.
34.
35.//主函数
36.int main()
37.{
38.     SqStackC OPTR;
     //操作符栈
39.     SqStackN OPND;
     //操作数栈
40.     char ch,theta;
41.     int lastIsPoint=0;
     //上一个字符是否是'.'
42.     int cntExpression=0;
     //记录表达式个数
43.     int cntPoint=1;
     //记录小数点后几位
44.     double a,b;
45.     double num=0;
46.     double result[maxsize]={0};
     //存储每个表达式的结果
47.     InitStackC(&OPTR);
48.     InitStackN(&OPND);
49.     PushC(&OPTR,'=');
50.     scanf("%c",&ch);
51.     while (ch!='=')
52.     {
53.         while (ch!='=' || GetTopC(OPTR)!='=')
54.         {
55.             //ch 是操作数或小数点
56.             if((ch>='0' && ch<='9') || ch=='.')
57.             {
58.                 if(ch=='.')
59.                 {
60.                     lastIsPoint=1;
61.                     scanf("%c",&ch);
62.                 }
63.                 else
64.                 {
65.                     //上一次输入的 ch 为'.'
66.                     if(lastIsPoint)
67.                     {
```

```
68.                         num=num+pow(0.1,cntPoint)*(ch-
   '0');
69.                         cntPoint++;
70.                         scanf("%c",&ch);
71.                     }
72.                     //上一次输入的ch 不为'.'
73.                     else
74.                     {
75.                         num=num*10+ch-
   '0';                    //-'0'不可省略，否则会ch 会转换成ASCII
   码值，远大于ch 强制转换成double 的值
76.                         scanf("%c",&ch);
77.                     }
78.                 }
79.             }
80.             //ch 是操作符
81.             else
82.             {
83.                 //重置有关小数及小数位数处理的变量
84.                 if(lastIsPoint)
85.                 {
86.                     lastIsPoint=0;
87.                     cntPoint=1;
88.                 }
89.                 //之前有数字
90.                 if(num!=0.0)
91.                 {
92.                     PushN(&OPND,num);
93.                     num=0.0;
     //重置num
94.                 }
95.                 switch (Precede(GetTopC(OPTR),ch))
96.                 {
97.                     case '<':
98.                         PushC(&OPTR,ch);
99.                         scanf("%c",&ch);
100.                          break;
101.                    case '>':
102.                        PopC(&OPTR,&theta);
       //弹出操作符
103.                        PopN(&OPND,&a);
       //弹出操作数
104.                        PopN(&OPND,&b);
       //弹出操作数
```

```
105.                             PushN(&OPND,Operate(a,theta,b));
    //将运算结果压入 opnd 栈
106.                             break;
107.                         case '=':
108.                             PopC(&OPTR,&theta);
109.                             scanf("%c",&ch);
110.                             break;
111.                     }
112.                 }
113.             }
    //里层 while
114.             result[cntExpression]=GetTopN(OPND);
115.             cntExpression++;
116.             scanf("%c",&ch);
    //处理缓冲区的回车
117.             scanf("%c",&ch);
    //输入 ch
118.         }
    //外层 while
119.         //输出各个表达的值
120.         for(int i=0;i<cntExpression;i++)
121.         {
122.             printf("%.2f\n",result[i]);
123.         }
124.         return 0;
125.     }
126.
127.     //操作符栈初始化函数
128.     void InitStackC(SqStackC *s)
129.     {
130.         s->base=(char *)malloc(sizeof(char));
131.         if(!s->base){
132.             exit(0);
133.         }
134.         s->top=s->base;
135.         s->stacksize=maxsize;
136.     }
137.
138.     //操作数栈初始化函数
139.     void InitStackN(SqStackN *s)
140.     {
141.         s->base=(double *)malloc(sizeof(double));
142.         if(!s->base){
143.             exit(0);
```

```
144.     }
145.     s->top=s->base;
146.     s->stacksize=maxsize;
147. }
148.
149. //操作符栈入栈函数
150. void PushC(SqStackC *s,char e)
151. {
152.     if(s->top-s->base==s->stacksize){
153.         exit(0);
154.     }
155.     *(s->top)=e;
156.     s->top++;
157. }
158.
159. //操作数栈入栈函数
160. void PushN(SqStackN *s,double e)
161. {
162.     if(s->top-s->base==s->stacksize){
163.         exit(0);
164.     }
165.     *(s->top)=e;
166.     s->top++;
167. }
168.
169. //操作符栈出栈函数
170. void PopC(SqStackC *s,char *e)
171. {
172.     if(s->top==s->base){
173.         exit(0);
174.     }
175.     s->top--;
176.     *e=*(s->top);
177. }
178.
179. //操作数栈出栈函数
180. void PopN(SqStackN *s,double *e)
181. {
182.     if(s->top==s->base){
183.         exit(0);
184.     }
185.     s->top--;
186.     *e=*(s->top);
187. }
```

```
188.
189.  //取操作符栈栈顶元素函数
190.  char GetTopC(SqStackC s)
191.  {
192.      if(s.top!=s.base){
193.          return *(s.top-1);
194.      }
195.  }
196.
197.  //取操作数栈栈顶元素函数
198.  double GetTopN(SqStackN s)
199.  {
200.      if(s.top!=s.base){
201.          return *(s.top-1);
202.      }
203.  }
204.
205.  //判定操作符栈的栈顶元素与读入的操作符之间优先关系的函数
206.  char Precede(char theta1,char theta2)
207.  {
208.      if((theta1=='(' && theta2==')') || (theta1=='=' && th
   eta2=='='))
209.          return '=';
210.      else if(theta1=='(' || theta1=='=' || theta2=='(' ||
   ((theta1=='+' || theta1=='-
   ') && (theta2=='*' || theta2=='/')))
211.          return '<';
212.      else
213.          return '>';
214.  }
215.
216.  //进行二元运算的函数
217.  double Operate(double a,char theta,double b)
218.  {
219.      switch (theta)
220.      {
221.          case '+':
222.              return a+b;
223.          case '-':
224.              return b-
   a;                                    //由于栈先入后出的特
   性，这里要调换 a，b 顺序
225.          case '*':
226.              return a*b;
```

```
227.          case '/':
228.              return b/a;
      //由于栈先入后出的特性，这里要调换 a，b 顺序
229.      }
230.  }
```

四、实验总结及体会

本次实验对我个人来说有以下几个难点：

1. 多位数的处理：课本中的代码只有十位数以内的数据处理，而十位数之后便无法处理。我的处理方法是新增一个 num 变量，对输入的操作数强制转换成 double 型，存入 num 中。下一次输入如果还是操作数，只需 num*10+操作数即可。这个方法有效解决了多位数的数据处理问题。
2. 小数点的处理：因为输出的数据可能是小数，就会包含小数点，因此要对小数点的情况进行处理。我的处理思路是设置一个 lastIsPoint 变量作为 flag，看看上一次输入是否为小数点。如果上一次输入是小数点，那么下一次输入的操作数在强制转换后，就需要*0.1。
3. 多位小数的处理：当程序可以输入小数点后，为了提高程序的鲁棒性，就要考虑多位小数的问题。经过思考后，我新增了一个变量 cntPoint=1，记录小数点后数字的位数。如果上一次输入是小数点，那么下一次输入的操作数在强制转换后，就需要*0.1。只要接下来的输入不是操作符，那么每次输入的操作数就要*pow(0.1,cntPoint++)；如果接下去的输入是操作符，那么要重置 cntPoint，使其=1，保证下一次输入小数时，cntPoint 是从 1 开始。
4. 多行表达式的处理：课本中提供的代码只能处理单个表达式，因此要想输入多行表达式并得到正确输出，必须修改代码。一开始我的思路是多设置几个变量作为 flag，来判断输入是否结束。经过改善后，我发现只需要使用两层 while 循环即可，里层 while 控制每一行表达式，外层 while 控制整体。但是有一个小细节我没注意到，导致调试很久都没得到正确结果，就是每行表达式输入完并且换行后，要先 scanf 一次，处理缓冲区的换行符，处理好换行符后再输入表达式。