目录

1. 需求分析

(1) 设计一个程序，以字符序列的形式从终端输入语法正确的、不含变量的整数表达式，演示用算符优先法对算术四则混合运算表达式的求值；

(2) 演示在求值中运算符栈、运算数栈、输入字符和主要操作的变换过程；

(3) 扩充运算符集，如增加乘方、开方等运算；

(4) 运算量可以是变量；

(5) 计算器的功能和仿真界面。

2. 项目亮点

(1) 通过 Python 调用 C 语言功能函数，实现了仿 windows 的简易版计算器；

(2) 建立了独立的工作栈库和计算功能库，使得项目的调用更加清晰合理；

(3) 运用 txt 文本记录求值过程中的操作变换过程，让使用者方便单独使用计算器和查看具体过程；

(4) 解决了在实数范围内的运算；

(5) 区分了除 0 无意义和输入错误两种情况；

(6) 输入过程可以添加空格；

(7) 运算量可以是已赋值的变量，程序会给出精确结果；

(8) 增加了乘方、开方运算。

3. 概要设计

表达式求值是程序设计语言编译的一个最基本问题，其中任何一个表达式都是由操作数、运算符、界限符组成。要实现算术表达式的演示，需要建立栈的顺序结构来实现栈的基本操作，并构建两个工作栈：一个寄存运算符，另一个寄存操作数和运算结果。在读取时，输出具体的进出栈过程和操作过程。

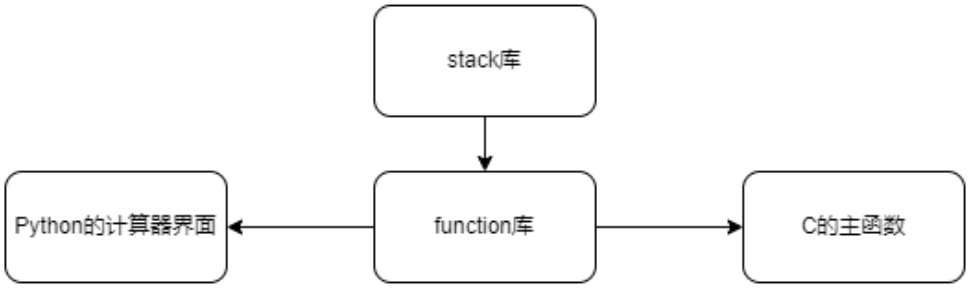

图 1: 结构体系图

4. 详细设计

4.1 工作栈库

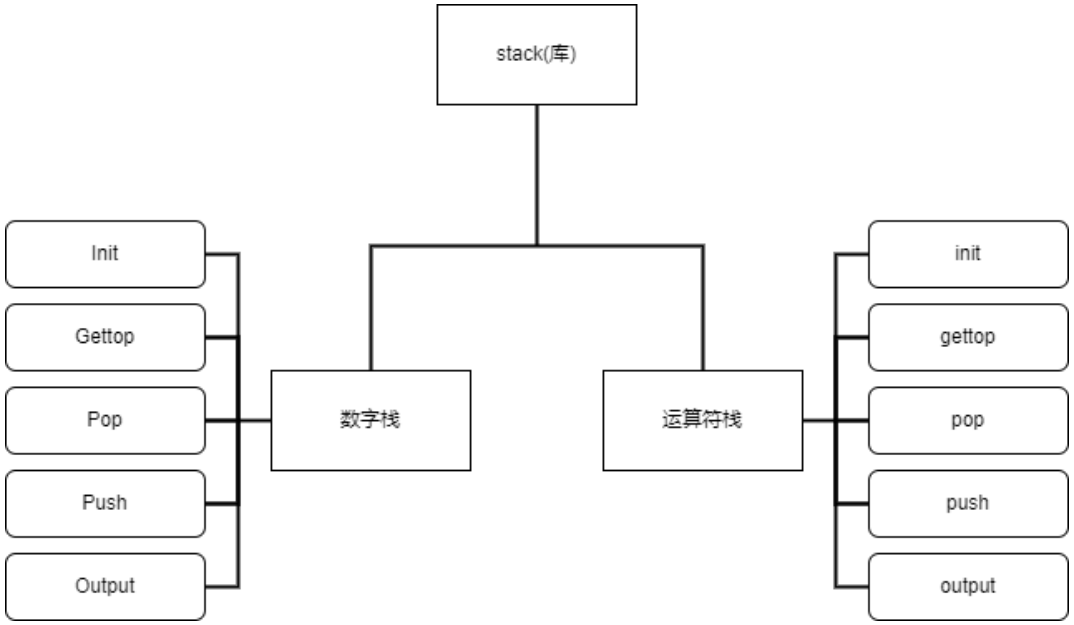

图 2: 栈库示意图

```
1 #include<stdio.h>
2 #include<string.h>
```

```
#include<stdlib.h>
#include<float.h>
#include<math.h>
#include<conio.h>
#define STACK_INIT_SIZE 100 //存储空间初始分配量
#define STACKNCREMENT 10 //存储空间分配增量
FILE *fp; //定义文件指针
typedef struct
{
    char *base;
    char *top;
    int stacksize;
}OPTR; //运算符栈
typedef struct
{
    double *base;
    double *top;
    int stacksize;
}OPND;//数字栈
int InitStack(OPTR *S); //构造运算符栈
int initStack(OPND *S); //构造数字栈
char GetTop(OPTR *S); //输出运算符栈栈顶元素
double getTop(OPND *S); //输出数字栈栈顶元素
int Push(OPTR *S,char e); //元素入运算符栈
int push(OPND *S,double e); //元素到数字栈
int Pop(OPTR *S,char *e); //元素出运算符栈
int pop(OPND *S,double *e); //元素出数字栈
int OutPut(OPTR *S); //输出运算符栈
int outPut(OPND *S); //输出数字栈
```

4.2 计算功能库

4.2.1 function.h

```
double Opreate(double a,double b,char optr); //对操作数和对应的运算符进行计算并返回结果
int In(char c,char *tr); //判断字符是否为运算符
char Precede(char m, char n, char *OP); //判断运算符的优先级
double EvaluateExpression(char *expr); //算术表达式求值的算符优先算法
void clear(char *c);//删除输入的空格
int check(char *c);//检查是否有非法字符输入
void negative(char *c);//处理负数问题
void change(char *c,double x);//将变量x赋值
int variable(char *c);//判断输入是否为变量
```

4.2.2 function.c

	+	-	*	/	^	(	)	#
+	>	>	<	<	<	<	>	>
-	>	>	<	<	<	<	>	>
*	>	>	>	>	<	<	>	>
/	>	>	>	>	<	<	>	>
^	>	>	>	>	>	<	>	>
(	<	<	<	<	<	<	=	
)	>	>	>	>	>		>	>
#	<	<	<	<	<	<		=

图 3: 算符优先级

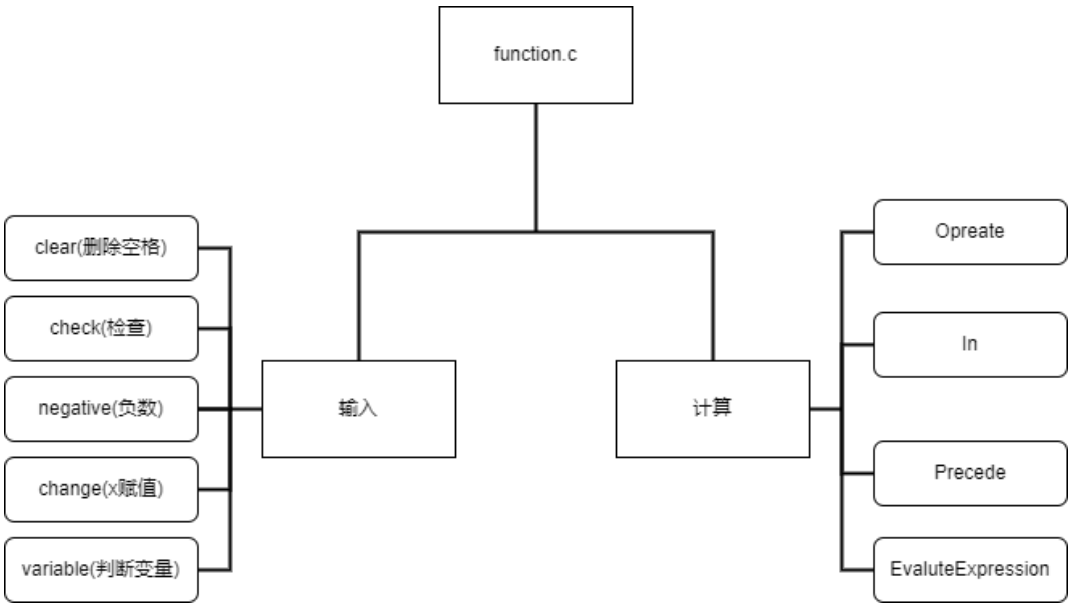

图 4: 计算功能示意图

```c
#include "function.h"
#include "stack.c"
double Opreate(double a,double b,char tr) //对操作数和对应的运算符进行计算并返回结果
{
    switch(tr)
    {
        case'+':return a+b;
        case'-':return a-b;
        case'/':return a/b;
        case'*':return a*b;
        case'^':return pow(a,b);
        default:return 0;
    }
}
```

```
15
16  int In(char c,char *tr) //判断字符是否为运算符
17  {
18      int i=0,flag=0;
19      while(tr[i]!='\0')
20      {
21          if(tr[i]==c) flag=1;
22          i++;
23      }
24      return flag;
25  }
26
27  char Precede(char m, char n, char *tr) //判断运算符的优先级
28  {
29      unsigned char
            Prior[8][8]={">>>>>>>","">>>>>>>","">>>>>>>","">>>>>>>","">>>>>>>","<<<<<<<
            ","">>>>> >>","<<<<<< ="};
30      int i=0,j=0;
31      while(m!=tr[i]) i++;
32      while(n!=tr[j]) j++;
33      return Prior[i][j];
34  }
35
36  double EvaluateExpression(char *expr) //算术表达式求值的算符优先算法
37  {
38      negative(expr);
39      OPTR TR;OPND ND;
40      InitStack(&TR);initStack(&ND);
41      char *c,cc[500]={},tr,x,OP[10]={'+','-','*','/','^','(',')','#','\0'};
42      double a,b,cf;
43      int i=1;
44      Push(&TR,'#');
45      c=strcat(expr,"# "); //拼接表达式使其以#结尾
46      while(*c!='#'||GetTop(&TR)!='#')
47      {
48          fprintf(fp,"————————————————————————————————第%d步————————————————————————————————\n",i)
49          if(!In(*c,OP)) //不是运算符进栈
50          {
51              while(!In(*c,OP))
52              {
53                  strcat(cc,c); //两位数以上数字输入
54                  c++;
55              }
56              cf=atof(cc);
57              OutPut(&TR);
58              outPut(&ND);
59              fprintf(fp,"输入字符: '%lf'\n",cf);
```

```
60            memset(cc,0x00,sizeof(char)*256); //清空临时字符串
61            push(&ND,cf);
62            fprintf(fp,"操作: Push(OPND,'%lf')\n",cf);
63        }
64        else
65        {
66            OutPut(&TR);
67            outPut(&ND);
68            fprintf(fp,"输入字符: '%c'\n",*c);
69            switch(Precede(GetTop(&TR),*c,OP))
70            {
71                case'<': //栈顶元素优先权低
72                    Push(&TR,*c);
73                    fprintf(fp,"操作: Push(OPTR,'%c')\n",*c);
74                    c++;
75                    break;
76                case'=': //脱括号指针移动到下一字符
77                    Pop(&TR,&x);
78                    fprintf(fp,"操作: Pop(OPTR){消去一对括号}\n",x);
79                    c++;
80                    break;
81                case'>': //退栈并将运算结果入栈
82                    Pop(&TR,&tr);
83                    pop(&ND,&b);
84                    pop(&ND,&a);
85                    push(&ND,Opreate(a,b,tr));
86                    fprintf(fp,"操作: Opreate('%lf','%c','%lf')\n",a,tr,b);
87                    break;
88            }
89        }
90        i++;
91    }
92    fprintf(fp,"————————————————————————————————第%d步————————————————————————————————\n",i);
93    OutPut(&TR);
94    outPut(&ND);
95    fprintf(fp,"输入字符: '%c'\n",*c);
96    fprintf(fp,"操作:return GetTop(OPND)\n");
97    return getTop(&ND);
98 }
99
100 void clear(char *c)//删除输入的空格
101 {
102   char *str=c;
103   int i,j;
104   for(i=0,j=0;c[i]!='\0';i++)
105   {
106     if(c[i]!=' ')
```

```
107         str[j++]=c[i];
108     }
109     str[j]='\0';
110     c=str;
111 }
112 
113 int check(char *c)//检查是否有非法字符输入
114 {
115     int i,j,t1=0,t2=0,flag=0;
116     for(i=0;i<strlen(c);i++)//括号输入不完整问题
117     {
118         if(c[i]=='(') t1++;
119         if(c[i]==')') t2++;
120         if(!In(c[i],"+-*/^()1234567890. "))
121         {
122             flag=1;
123             break;
124         }
125     }
126     if(t1!=t2) flag=1;
127     for(i=0,j=1;j<strlen(c);i++,j++)//运算符相邻问题
128     {
129         if(In(c[i],"+-*/^ ")&&In(c[j],"+-*/^ "))
130         {
131             flag=1;
132             break;
133         }
134     }
135     return flag;
136 }
137 
138 void negative(char *c)//处理负数问题
139 {
140     int i,j,k,l=0;
141     char str[100];
142     strcpy(str,c);
143     for(i=0,j=1;j<strlen(str);i++,j++)
144     {
145         if(str[i]=='('&&str[j]=='-')
146         {
147             c[j+l]='0';
148             for(k=j+l;c[k]!='\0';k++)
149                 c[k+1]=str[k-l];
150             l++;
151         }
152     }
153 }
```

```
154
155 void change(char *c,double x)//将变量x赋值
156 {
157    char p[1000],a[100]={};
158     int i,j,k,l,f=0;
159     strcpy(p,c);
160     sprintf(a,"%lf",x);
161    for(i=0;i<strlen(c);i++)
162     {
163       if(c[i]=='x')
164         {
165             for(k=i+f*(strlen(a)-1),j=0;j<strlen(a);k++,j++)
166                 p[k]=a[j];
167             for(l=i+1;l<strlen(c);l++,k++)
168                 p[k]=c[l];
169         p[k]='\0';
170         f++;
171         }
172     }
173     strcpy(c,p);
174 }
175
176 int variable(char *c)//判断输入是否为变量
177 {
178     int i,j,n=0;
179     for(i=0;i<strlen(c);i++)
180     {
181         if(c[i]=='x')
182         {
183             n=1;
184             break;
185         }
186     }
187     for(i=0,j=1;j<strlen(c);i++,j++)
188     {
189         if((c[i]=='x'||c[i]=='.')&&(c[j]=='.'||c[j]=='x'))
190         {
191             n=0;
192             break;
193         }
194     }
195     return n;
196 }
```

4.3 C 的主函数

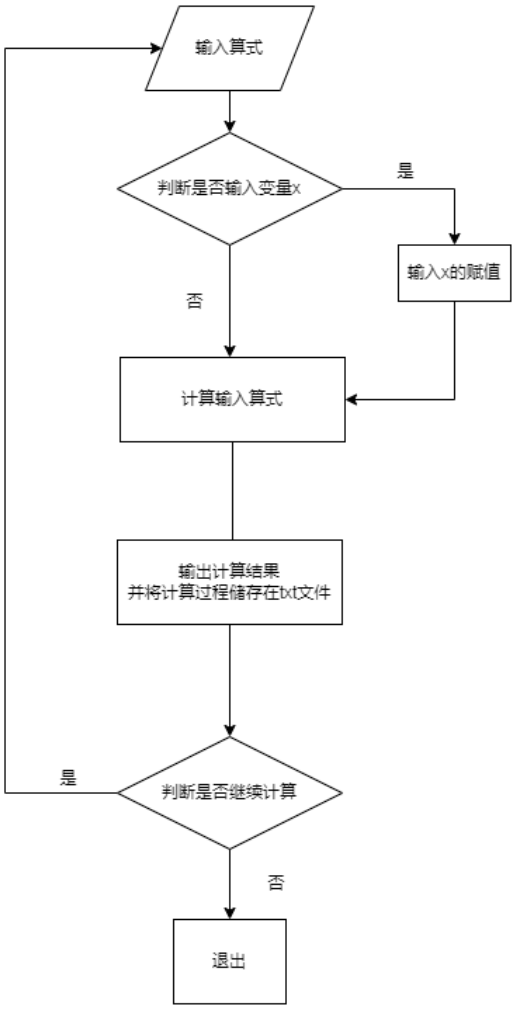

图 5: 主函数流程图

```
#include "function.c"
int main()
{
    int y=1,m=1,n=0;
    double x;
    char file[100],Algorite[500];
    while(y)
    {
```

```
        sprintf(file,"%d.txt",m);
        if((fp=fopen(file,"w"))==NULL) return -1;
        printf("请输入算式(^_^):");
        gets(Algorite);
        clear(Algorite);
        n=variable(Algorite);
        if(n)
        {
            printf("请输入x的值:");
            scanf("%lf",&x);
            fprintf(fp,"变量表达式为:%s\nx=%lf\n",Algorite,x);
            change(Algorite,x);
        }
        int flag=0;
        flag=check(Algorite);
        if(flag) printf("输入有误，请重新输入(T_T)\n");
        else
        {
            fprintf(fp,"算式为: %s\n",Algorite);
            negative(Algorite);
            double result=EvaluateExpression(Algorite);
            if(fabs(result)>DBL_MAX) printf("发生除0错误(-_-)\n");
            else printf("%lf\n",result);
            fclose(fp);
        }
        printf("是否继续计算？ 1:继续 0:退出\n");
        scanf("%d",&y);
        getchar();
        m++;
    }
    system("pause");
    return 0;
}
```

4.4 Python 的计算器可视化

```
from tkinter import *
from functools import partial
from ctypes import *
cpy=CDLL('./function.dll')
cpy.EvaluateExpression.restype=c_double
cpy.check.restype=c_int
cpy.variable.restype=c_int
cpy.change.restype=c_void_p

```

```
10 #生成计算器主界面
11 def layout(root):
12     label=Label(root,width=29,height=1,bd=5,bg='#FFFACD',anchor='se',
13     textvariable=label_text) #标签，可以显示文字或图片
14     label.grid(row=0,columnspan=5) #布局器，向窗口注册并显示控件;
           rowspan：设置单元格纵向跨越的列数
15     entry=Entry(root,width=23,bd=5,bg='#FFFACD',justify="right",font=('微软雅黑', 12))
           #文本框（单行）
16     entry.grid(row=1,column=0,columnspan=5,sticky=N+W+S+E,padx=5,pady=5)
           #设置控件周围x、y方向空白区域保留大小
17     myButton=partial(Button,root,width=5,cursor='hand2',activebackground='#90EE90') #
           偏函数：带有固定参数的函数
18     #command指定按钮消息的回调函数
19     button_clear=myButton(text=' C ',command=lambda:clear(entry))
20     button_left=myButton(text='(',command=lambda:entry.insert(INSERT,'('))
21     button_right=myButton(text=')',command=lambda:entry.insert(INSERT,')'))
22     button_backspace=myButton(text='←',command=lambda:entry.delete(len(entry.get())-1))
           #删除文本框的最后一个输入值
23     button_x=myButton(text='\n x \n',command=lambda:entry.insert(INSERT,'x'))
24     button_clear.grid(row=3,column=0)
25     button_left.grid(row=3,column=1)
26     button_right.grid(row=3,column=2)
27     button_backspace.grid(row=3,column=3)
28     button_x.grid(row=3,column=4)
29     #第二行
30     button_7=myButton(text=' 7 ',command=lambda:entry.insert(INSERT,'7'))
31     button_8=myButton(text=' 8 ',command=lambda:entry.insert(INSERT,'8'))
32     button_9=myButton(text=' 9 ',command=lambda:entry.insert(INSERT,'9'))
33     button_divide=myButton(text=' / ',command=lambda:entry.insert(INSERT,'/'))
34     button_fx=myButton(text='赋值',command=lambda:label_text.set('x='))
35     button_7.grid(row=4,column=0)
36     button_8.grid(row=4,column=1)
37     button_9.grid(row=4,column=2)
38     button_divide.grid(row=4,column=3)
39     button_fx.grid(row=4,column=4)
40     #第三行
41     button_4=myButton(text=' 4 ',command=lambda:entry.insert(INSERT,'4'))
42     button_5=myButton(text=' 5 ',command=lambda:entry.insert(INSERT,'5'))
43     button_6=myButton(text=' 6 ',command=lambda:entry.insert(INSERT,'6'))
44     button_multi=myButton(text=' * ',command=lambda:entry.insert(INSERT,'*'))
45     button_sure=myButton(text='确认',command=lambda:sure(entry))
46     button_4.grid(row=5,column=0)
47     button_5.grid(row=5,column=1)
48     button_6.grid(row=5,column=2)
49     button_multi.grid(row=5,column=3)
50     button_sure.grid(row=5,column=4)
51
```

```
52      #第四行
53      button_1=myButton(text=' 1 ',command=lambda:entry.insert(INSERT,'1'))
54      button_2=myButton(text=' 2 ',command=lambda:entry.insert(INSERT,'2'))
55      button_3=myButton(text=' 3 ',command=lambda:entry.insert(INSERT,'3'))
56      button_subtract=myButton(text=' - ',command=lambda:entry.insert(INSERT,'-'))
57      button_equal=myButton(text=' \n = \n ',command=lambda:calculate(entry))
58      button_1.grid(row=6,column=0)
59      button_2.grid(row=6,column=1)
60      button_3.grid(row=6,column=2)
61      button_subtract.grid(row=6,column=3)
62      button_equal.grid(row=6,column=4,rowspan=2)
63      #第五行
64      button_power=myButton(text='^',command=lambda:entry.insert(INSERT,'^'))
65      button_0=myButton(text=' 0 ',command=lambda:entry.insert(INSERT,'0'))
66      button_point=myButton(text=' . ',command=lambda:entry.insert(INSERT,'.'))
67      button_plus=myButton(text=' + ',command=lambda:entry.insert(INSERT,'+'))
68      button_power.grid(row=7,column=0)
69      button_0.grid(row=7,column=1)
70      button_point.grid(row=7,column=2)
71      button_plus.grid(row=7,column=3)
72
73  #删除所有输入内容和显示内容
74  def clear(entry):
75      entry.delete(0,END)
76      label_text.set('')
77
78  def sure(entry):
79      global m
80      m=entry.get()
81      if m=='':
82          m=0
83      m=float(m)
84      clear(entry)
85      return m
86
87  # 点击 “=” 后进行计算
88  def calculate(entry):
89      try:
90          formula=entry.get()
91          x=formula
92          x=x.encode('utf8')
93          x=c_char_p(x)
94          if cpy.variable(x)==1:
95              cpy.change(x,c_double(m))
96          clear(entry)
97          if cpy.check(x)==1:
98              entry.insert(END,'出错(T_T)')
```

```
99          else:
100             result=cpy.EvaluateExpression(x)
101             if abs(result)==float('inf'):
102                 entry.insert(END,'发生除0错误(-_-)')
103             else:
104                 entry.insert(END,result)
105         label_text.set(''.join(formula+'='))
106     except:
107         clear(entry)
108         entry.insert(END,'出错(T_T)')
109
110
111 if __name__ == '__main__':
112     root=Tk() #生成窗口
113     root.title('计算器') #窗口的名字
114     root.resizable(0,0) #窗口大小可调性，分别表示x，y方向的可变性
115     global label_text #定义全局变量
116     label_text=StringVar()
117     layout(root)
118     root.mainloop() #进入消息循环（必需组件），否则生成的窗口一闪而过
```

5. 用户手册

5.1 测试数据

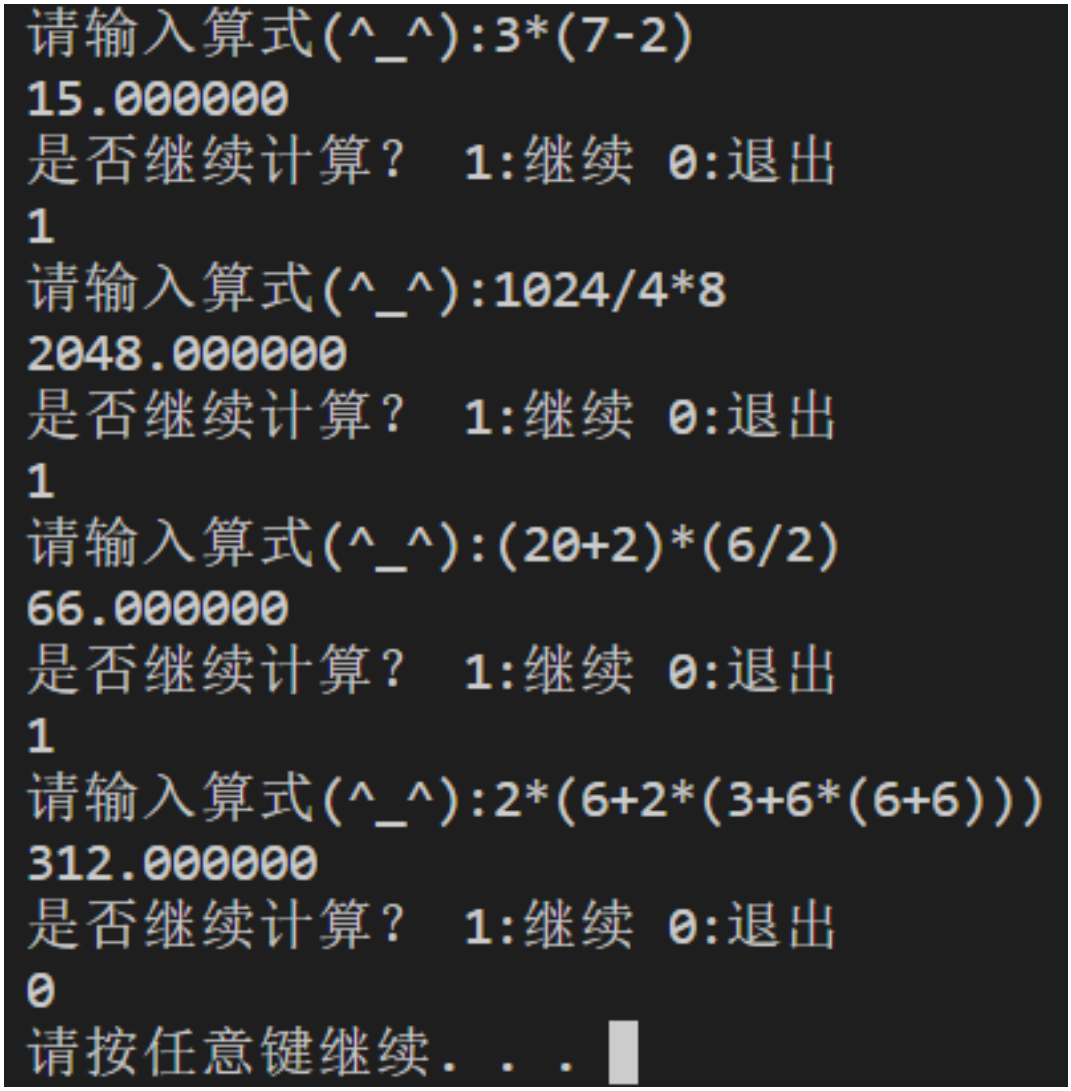

图 6: 测试数据

具体过程如下：

5.1.1 3*(7-2)

```
算式为: 3*(7-2)
————————————————————————————————第1步————————————————————————————————
OPTR栈: #
OPND栈:
输入字符: '3.000000'
操作: Push(OPND,'3.000000')
```

```
————————————————————————————————第2步————————————————————————————————
OPTR栈: #
OPND栈: 3.000000
输入字符: '*'
操作: Push(OPTR,'*')
————————————————————————————————第3步————————————————————————————————
OPTR栈: # *
OPND栈: 3.000000
输入字符: '('
操作: Push(OPTR,'(')
————————————————————————————————第4步————————————————————————————————
OPTR栈: # * (
OPND栈: 3.000000
输入字符: '7.000000'
操作: Push(OPND,'7.000000')
————————————————————————————————第5步————————————————————————————————
OPTR栈: # * (
OPND栈: 3.000000 7.000000
输入字符: '-'
操作: Push(OPTR,'-')
————————————————————————————————第6步————————————————————————————————
OPTR栈: # * ( -
OPND栈: 3.000000 7.000000
输入字符: '2.000000'
操作: Push(OPND,'2.000000')
————————————————————————————————第7步————————————————————————————————
OPTR栈: # * ( -
OPND栈: 3.000000 7.000000 2.000000
输入字符: ')'
操作: Opreate('7.000000','-','2.000000')
————————————————————————————————第8步————————————————————————————————
OPTR栈: # * (
OPND栈: 3.000000 5.000000
输入字符: ')'
操作: Pop(OPTR){消去一对括号}
————————————————————————————————第9步————————————————————————————————
OPTR栈: # *
OPND栈: 3.000000 5.000000
输入字符: '#'
操作: Opreate('3.000000','*','5.000000')
————————————————————————————————第10步————————————————————————————————
OPTR栈: #
OPND栈: 15.000000
输入字符: '#'
操作:return GetTop(OPND)
```

5.1.2 1024/4*8

```
算式为: 1024/4*8
————————————————————————————————————第1步————————————————————————————————————
OPTR栈: #
OPND栈:
输入字符: '1024.000000'
操作: Push(OPND,'1024.000000')
————————————————————————————————————第2步————————————————————————————————————
OPTR栈: #
OPND栈: 1024.000000
输入字符: '/'
操作: Push(OPTR,'/')
————————————————————————————————————第3步————————————————————————————————————
OPTR栈: # /
OPND栈: 1024.000000
输入字符: '4.000000'
操作: Push(OPND,'4.000000')
————————————————————————————————————第4步————————————————————————————————————
OPTR栈: # /
OPND栈: 1024.000000 4.000000
输入字符: '*'
操作: Opreate('1024.000000','/','4.000000')
————————————————————————————————————第5步————————————————————————————————————
OPTR栈: #
OPND栈: 256.000000
输入字符: '*'
操作: Push(OPTR,'*')
————————————————————————————————————第6步————————————————————————————————————
OPTR栈: # *
OPND栈: 256.000000
输入字符: '8.000000'
操作: Push(OPND,'8.000000')
————————————————————————————————————第7步————————————————————————————————————
OPTR栈: # *
OPND栈: 256.000000 8.000000
输入字符: '#'
操作: Opreate('256.000000','*','8.000000')
————————————————————————————————————第8步————————————————————————————————————
OPTR栈: #
OPND栈: 2048.000000
输入字符: '#'
操作:return GetTop(OPND)
```

5.1.3 (20+2)*(6/2)

```
算式为：(20+2)*(6/2)
————————————————————————————————————第1步————————————————————————————————————
OPTR栈：#
OPND栈：
输入字符：'('
操作：Push(OPTR,'(')
————————————————————————————————————第2步————————————————————————————————————
OPTR栈：# (
OPND栈：
输入字符：'20.000000'
操作：Push(OPND,'20.000000')
————————————————————————————————————第3步————————————————————————————————————
OPTR栈：# (
OPND栈：20.000000
输入字符：'+'
操作：Push(OPTR,'+')
————————————————————————————————————第4步————————————————————————————————————
OPTR栈：# ( +
OPND栈：20.000000
输入字符：'2.000000'
操作：Push(OPND,'2.000000')
————————————————————————————————————第5步————————————————————————————————————
OPTR栈：# ( +
OPND栈：20.000000 2.000000
输入字符：')'
操作：Opreate('20.000000','+','2.000000')
————————————————————————————————————第6步————————————————————————————————————
OPTR栈：# (
OPND栈：22.000000
输入字符：')'
操作：Pop(OPTR){消去一对括号}
————————————————————————————————————第7步————————————————————————————————————
OPTR栈：#
OPND栈：22.000000
输入字符：'*'
操作：Push(OPTR,'*')
————————————————————————————————————第8步————————————————————————————————————
OPTR栈：# *
OPND栈：22.000000
输入字符：'('
操作：Push(OPTR,'(')
————————————————————————————————————第9步————————————————————————————————————
OPTR栈：# * (
OPND栈：22.000000
```

```
输入字符: '6.000000'
操作: Push(OPND,'6.000000')
————————————————————————————————第10步————————————————————————————————
OPTR栈: # * (
OPND栈: 22.000000 6.000000
输入字符: '/'
操作: Push(OPTR,'/')
————————————————————————————————第11步————————————————————————————————
OPTR栈: # * ( /
OPND栈: 22.000000 6.000000
输入字符: '2.000000'
操作: Push(OPND,'2.000000')
————————————————————————————————第12步————————————————————————————————
OPTR栈: # * ( /
OPND栈: 22.000000 6.000000 2.000000
输入字符: ')'
操作: Opreate('6.000000','/','2.000000')
————————————————————————————————第13步————————————————————————————————
OPTR栈: # * (
OPND栈: 22.000000 3.000000
输入字符: ')'
操作: Pop(OPTR){消去一对括号}
————————————————————————————————第14步————————————————————————————————
OPTR栈: # *
OPND栈: 22.000000 3.000000
输入字符: '#'
操作: Opreate('22.000000','*','3.000000')
————————————————————————————————第15步————————————————————————————————
OPTR栈: #
OPND栈: 66.000000
输入字符: '#'
操作:return GetTop(OPND)
```

5.1.4 2*(6+2*(3+6*(6+6)))

```
算式为: 2*(6+2*(3+6*(6+6)))
————————————————————————————————第1步————————————————————————————————
OPTR栈: #
OPND栈:
输入字符: '2.000000'
操作: Push(OPND,'2.000000')
————————————————————————————————第2步————————————————————————————————
OPTR栈: #
OPND栈: 2.000000
输入字符: '*'
```

```
操作: Push(OPTR,'*')
————————————————————————————————第3步————————————————————————————————
OPTR栈: # *
OPND栈: 2.000000
输入字符: '('
操作: Push(OPTR,'(')
————————————————————————————————第4步————————————————————————————————
OPTR栈: # * (
OPND栈: 2.000000
输入字符: '6.000000'
操作: Push(OPND,'6.000000')
————————————————————————————————第5步————————————————————————————————
OPTR栈: # * (
OPND栈: 2.000000 6.000000
输入字符: '+'
操作: Push(OPTR,'+')
————————————————————————————————第6步————————————————————————————————
OPTR栈: # * ( +
OPND栈: 2.000000 6.000000
输入字符: '2.000000'
操作: Push(OPND,'2.000000')
————————————————————————————————第7步————————————————————————————————
OPTR栈: # * ( +
OPND栈: 2.000000 6.000000 2.000000
输入字符: '*'
操作: Push(OPTR,'*')
————————————————————————————————第8步————————————————————————————————
OPTR栈: # * ( + *
OPND栈: 2.000000 6.000000 2.000000
输入字符: '('
操作: Push(OPTR,'(')
————————————————————————————————第9步————————————————————————————————
OPTR栈: # * ( + * (
OPND栈: 2.000000 6.000000 2.000000
输入字符: '3.000000'
操作: Push(OPND,'3.000000')
————————————————————————————————第10步————————————————————————————————
OPTR栈: # * ( + * (
OPND栈: 2.000000 6.000000 2.000000 3.000000
输入字符: '+'
操作: Push(OPTR,'+')
————————————————————————————————第11步————————————————————————————————
OPTR栈: # * ( + * ( +
OPND栈: 2.000000 6.000000 2.000000 3.000000
输入字符: '6.000000'
操作: Push(OPND,'6.000000')
————————————————————————————————第12步————————————————————————————————
```

```
OPTR栈: # * ( + * ( +
OPND栈: 2.000000 6.000000 2.000000 3.000000 6.000000
输入字符: '*'
操作: Push(OPTR,'*')
————————————————————————————————第13步————————————————————————————————
OPTR栈: # * ( + * ( + *
OPND栈: 2.000000 6.000000 2.000000 3.000000 6.000000
输入字符: '('
操作: Push(OPTR,'(')
————————————————————————————————第14步————————————————————————————————
OPTR栈: # * ( + * ( + * (
OPND栈: 2.000000 6.000000 2.000000 3.000000 6.000000
输入字符: '6.000000'
操作: Push(OPND,'6.000000')
————————————————————————————————第15步————————————————————————————————
OPTR栈: # * ( + * ( + * (
OPND栈: 2.000000 6.000000 2.000000 3.000000 6.000000 6.000000
输入字符: '+'
操作: Push(OPTR,'+')
————————————————————————————————第16步————————————————————————————————
OPTR栈: # * ( + * ( + * ( +
OPND栈: 2.000000 6.000000 2.000000 3.000000 6.000000 6.000000
输入字符: '6.000000'
操作: Push(OPND,'6.000000')
————————————————————————————————第17步————————————————————————————————
OPTR栈: # * ( + * ( + * ( +
OPND栈: 2.000000 6.000000 2.000000 3.000000 6.000000 6.000000 6.000000
输入字符: ')'
操作: Opreate('6.000000','+','6.000000')
————————————————————————————————第18步————————————————————————————————
OPTR栈: # * ( + * ( + * (
OPND栈: 2.000000 6.000000 2.000000 3.000000 6.000000 12.000000
输入字符: ')'
操作: Pop(OPTR){消去一对括号}
————————————————————————————————第19步————————————————————————————————
OPTR栈: # * ( + * ( + *
OPND栈: 2.000000 6.000000 2.000000 3.000000 6.000000 12.000000
输入字符: ')'
操作: Opreate('6.000000','*','12.000000')
————————————————————————————————第20步————————————————————————————————
OPTR栈: # * ( + * ( +
OPND栈: 2.000000 6.000000 2.000000 3.000000 72.000000
输入字符: ')'
操作: Opreate('3.000000','+','72.000000')
————————————————————————————————第21步————————————————————————————————
OPTR栈: # * ( + * (
OPND栈: 2.000000 6.000000 2.000000 75.000000
```

```
105 输入字符: ')'
106 操作: Pop(OPTR){消去一对括号}
107 ————————————————————————————————第22步————————————————————————————————
108 OPTR栈: # * ( + *
109 OPND栈: 2.000000 6.000000 2.000000 75.000000
110 输入字符: ')'
111 操作: Opreate('2.000000','*','75.000000')
112 ————————————————————————————————第23步————————————————————————————————
113 OPTR栈: # * ( +
114 OPND栈: 2.000000 6.000000 150.000000
115 输入字符: ')'
116 操作: Opreate('6.000000','+','150.000000')
117 ————————————————————————————————第24步————————————————————————————————
118 OPTR栈: # * (
119 OPND栈: 2.000000 156.000000
120 输入字符: ')'
121 操作: Pop(OPTR){消去一对括号}
122 ————————————————————————————————第25步————————————————————————————————
123 OPTR栈: # *
124 OPND栈: 2.000000 156.000000
125 输入字符: '#'
126 操作: Opreate('2.000000','*','156.000000')
127 ————————————————————————————————第26步————————————————————————————————
128 OPTR栈: #
129 OPND栈: 312.000000
130 输入字符: '#'
131 操作:return GetTop(OPND)
```

5.2 功能测试

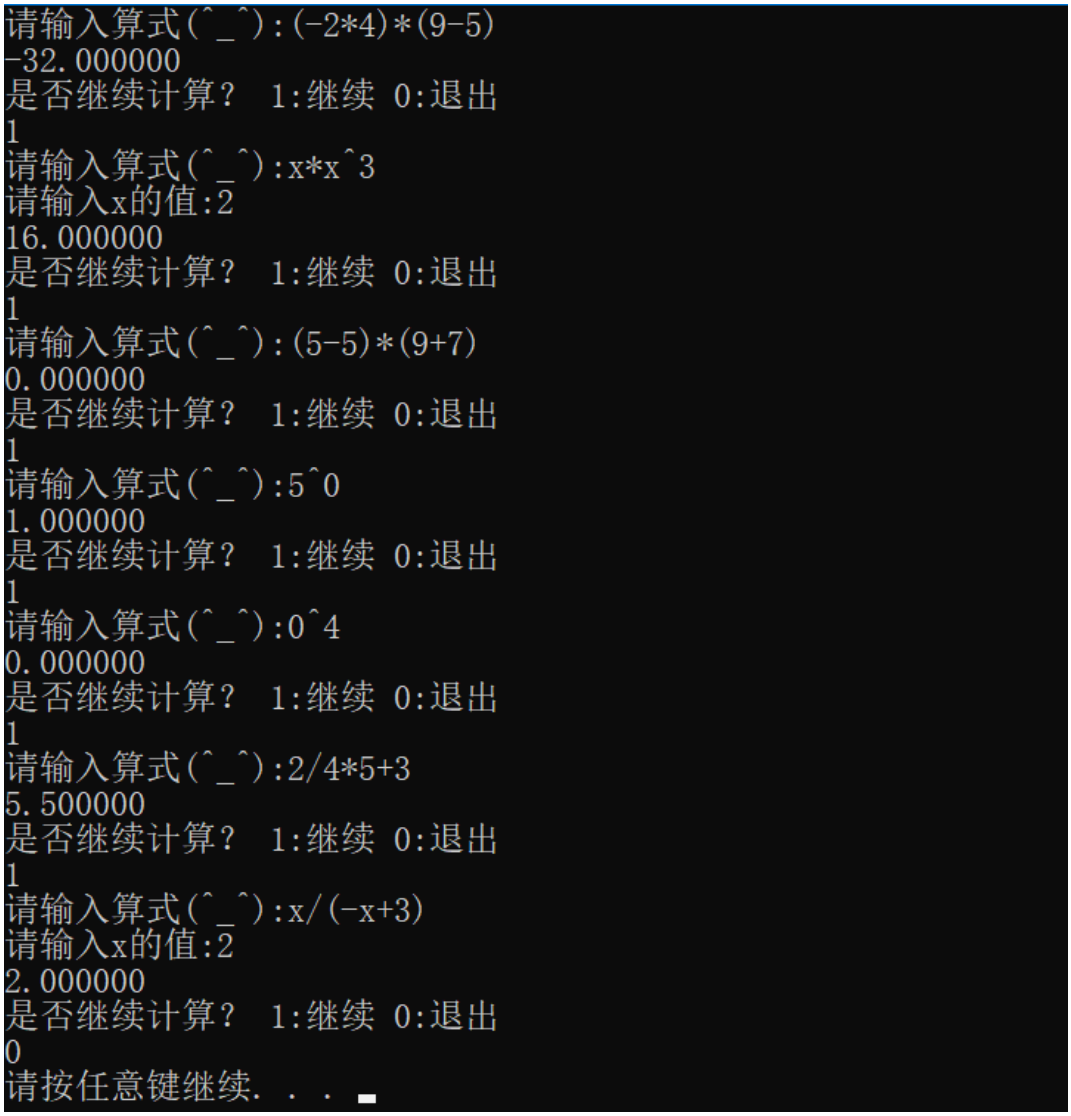

图 7: 功能测试

5.3 错误案例

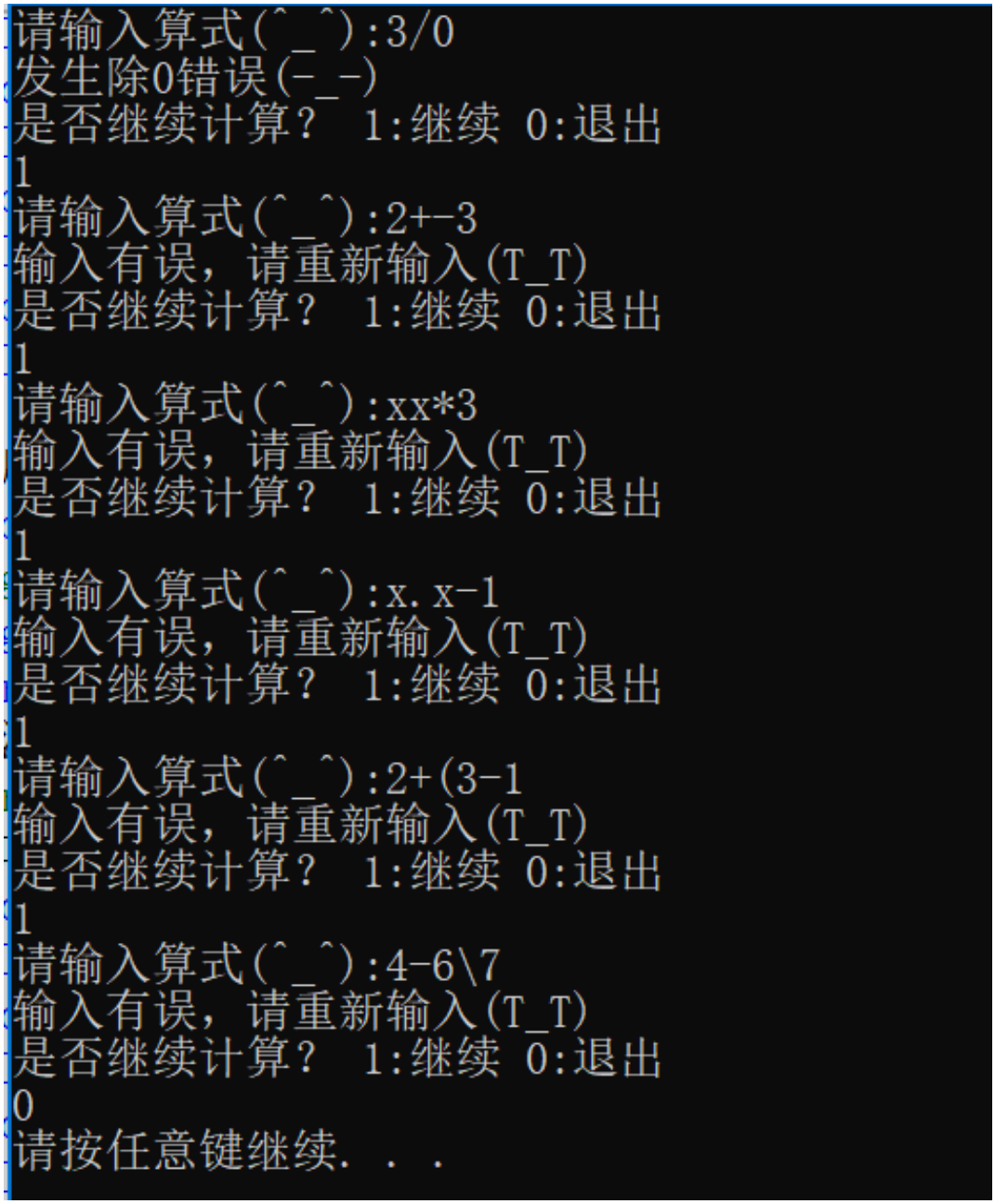

图 8: 错误案例

5.4 计算器测试

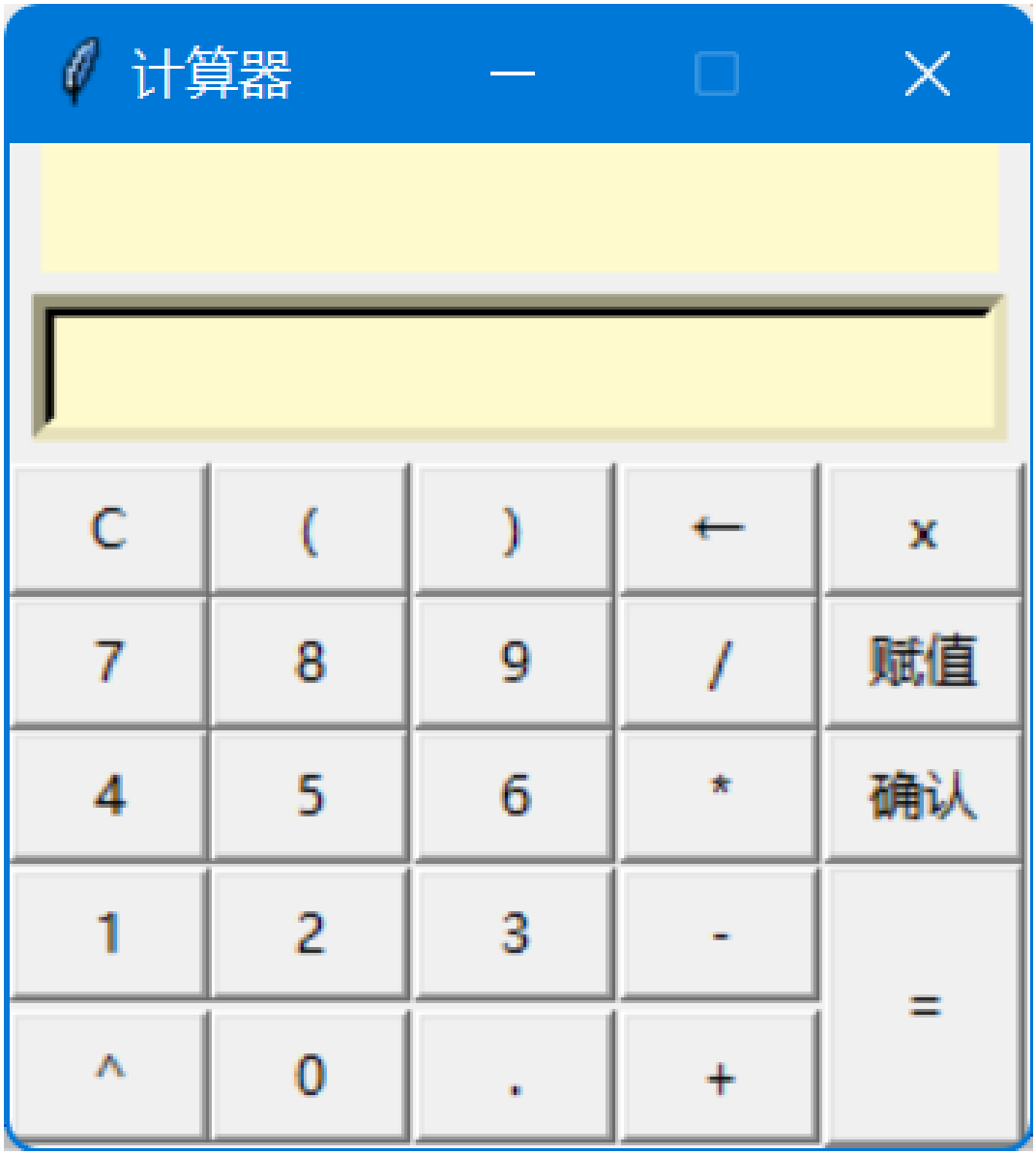

图 9: 计算器

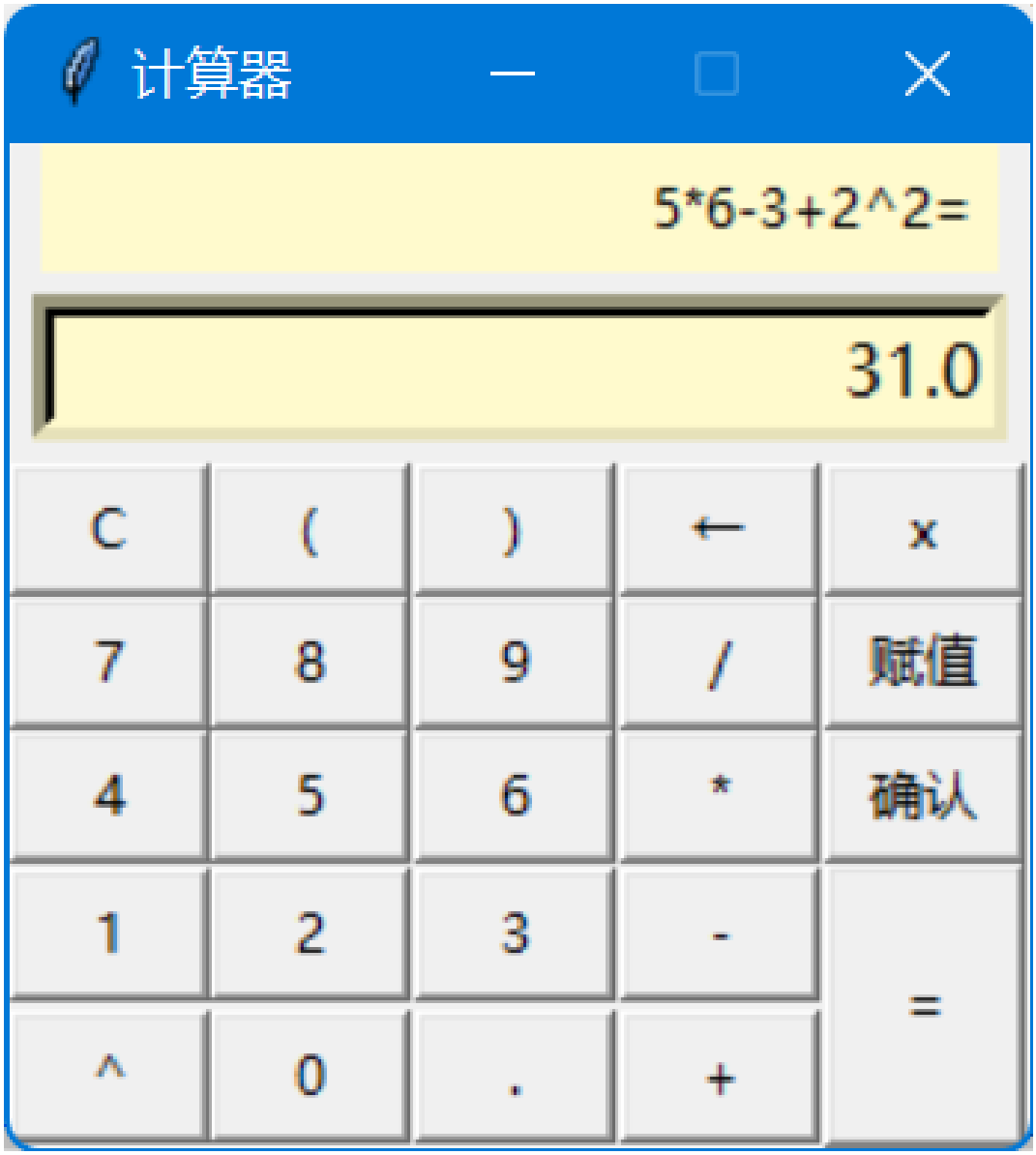

图 10: 输入案例

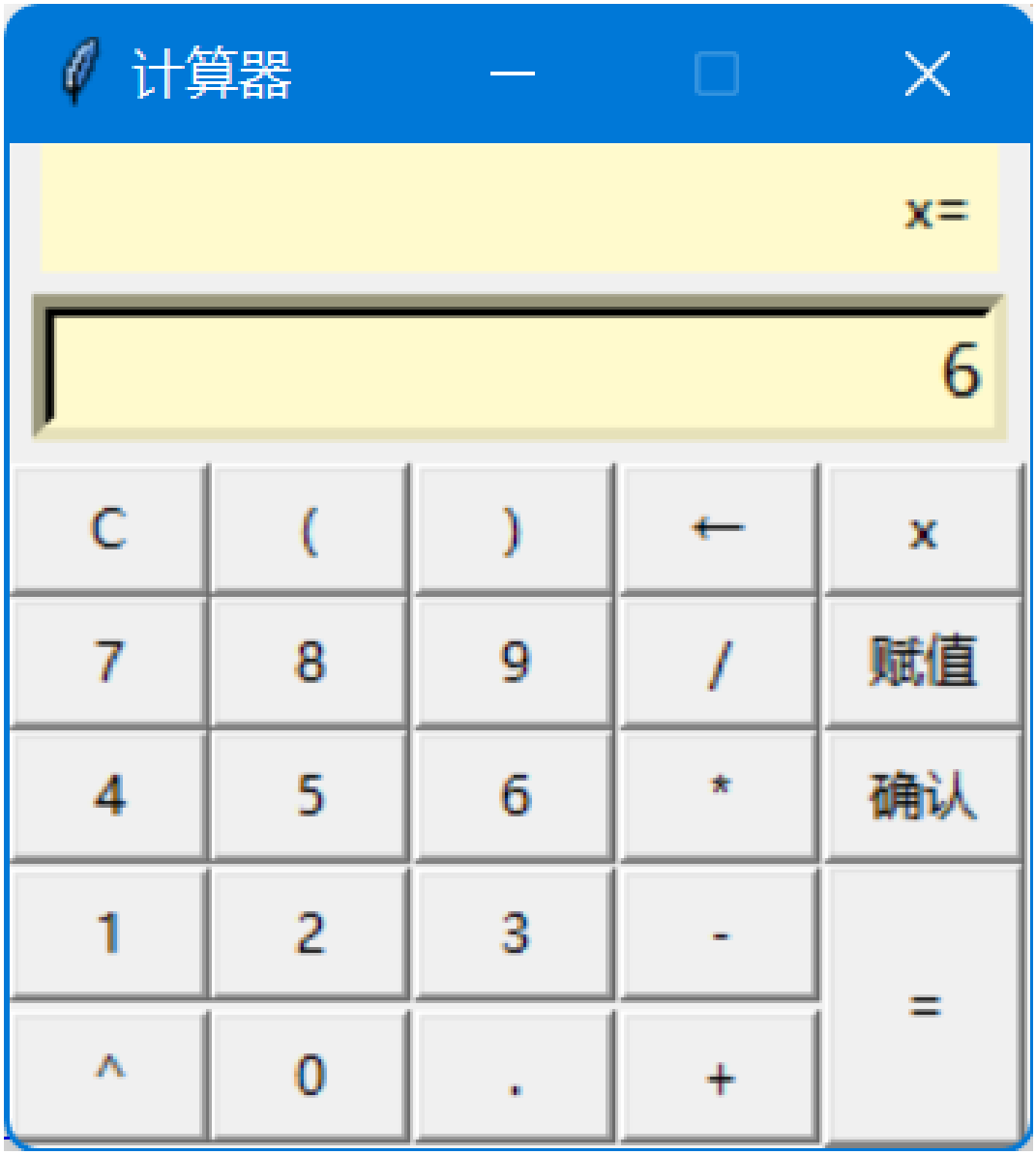

图 11: 变量赋值

6. 心得体会

这次课程设计的心得体会通过实践我们的收获如下：

1. 在这次的计算器算法及界面实现的过程中，我们更深刻的了解栈的特点与用法。

2. 在不断的修改程序 bug 的过程中，我们对程序运行的细节更加明了，提高了我们的查错，纠错能力。

3. 在面对含有变量的算式问题，我们发现可以使用之前课程设计的一元稀疏多项式运算程序来进行设计。

4. 在界面设计中我们学习了 DLL 库的设计，将设计的 C 语言程序加载到 Python 程序中，实现计算器界面。

7. 附录

7.1 stack.h

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<float.h>
#include<math.h>
#include<conio.h>
#define STACK_INIT_SIZE 100 //存储空间初始分配量
#define STACKNCREMENT 10 //存储空间分配增量
FILE *fp; //定义文件指针
typedef struct
{
    char *base;
    char *top;
    int stacksize;
}OPTR; //运算符栈
typedef struct
{
    double *base;
    double *top;
    int stacksize;
}OPND;//数字栈
int InitStack(OPTR *S); //构造运算符栈
int initStack(OPND *S); //构造数字栈
char GetTop(OPTR *S); //输出运算符栈栈顶元素
double getTop(OPND *S); //输出数字栈栈顶元素
int Push(OPTR *S,char e); //元素入运算符栈
int push(OPND *S,double e); //元素到数字栈
int Pop(OPTR *S,char *e); //元素出运算符栈
int pop(OPND *S,double *e); //元素出数字栈
int OutPut(OPTR *S); //输出运算符栈
int outPut(OPND *S); //输出数字栈
```

7.2 stack.c

```
#include "stack.h"
int InitStack(OPTR *S)//构造运算符栈
{
    S->base=(char *)malloc(STACK_INIT_SIZE*sizeof(char));
    if(!S->base) exit(-2);
    S->top=S->base;
```

```
7     S->stacksize=STACK_INIT_SIZE;
8     return 1;
9  }
10
11 int initStack(OPND *S)//构造数字栈
12 {
13    S->base=(double *)malloc(STACK_INIT_SIZE*sizeof(double));
14    if(!S->base) exit(-2);
15    S->top=S->base;
16    S->stacksize=STACK_INIT_SIZE;
17    return 1;
18 }
19
20 char GetTop(OPTR *S)//输出运算符栈栈顶元素
21 {
22    if(S->top==S->base) return 0;
23    return *(S->top-1);
24 }
25
26 double getTop(OPND *S)//输出数字栈栈顶元素
27 {
28    if(S->top==S->base) return 0;
29    return *(S->top-1);
30 }
31
32 int Push(OPTR *S,char e)//元素入运算符栈
33 {
34    if(S->top-S->base>=S->stacksize)
35    {
36       S->base=(char *)realloc(S->base,(S->stacksize+STACK_INIT_SIZE*sizeof(char)));
37       if(!S->base) exit(-2);
38       S->top=S->base+S->stacksize;
39       S->stacksize+=STACKNCREMENT;
40    }
41    *S->top++=e;
42    return 1;
43 }
44
45 int push(OPND *S,double e)//元素到数字栈
46 {
47    if(S->top-S->base>=S->stacksize)
48    {
49       S->base=(double*)realloc(S->base,(S->stacksize+STACK_INIT_SIZE*sizeof(double)));
50       if(!S->base) exit(-2);
51       S->top=S->base+S->stacksize;
52       S->stacksize+=STACKNCREMENT;
53    }
```

```
54      *S->top++=e;
55      return 1;
56  }
57
58  int Pop(OPTR *S,char *e)//元素出运算符栈
59  {
60      if(S->top==S->base) return 0;
61      *e=*(--S->top);
62      return 1;
63  }
64
65  int pop(OPND *S,double *e)//元素出数字栈
66  {
67      if(S->top==S->base)
68          return 0;
69      *e=*(--S->top);
70      return 1;
71  }
72
73  int OutPut(OPTR *S)//输出运算符栈
74  {
75      char *p;
76      p=S->base;
77      fprintf(fp,"OPTR栈: ");
78      while(p!=S->top)
79      {
80          fprintf(fp," %c ",*p);
81          p++;
82      }
83      fprintf(fp,"\n");
84  }
85
86  int outPut(OPND *S)//输出数字栈
87  {
88      double *p;
89      p=S->base;
90      fprintf(fp,"OPND栈: ");
91      while(p!=S->top)
92      {
93          fprintf(fp," %lf ",*p);
94          p++;
95      }
96      fprintf(fp,"\n");
97  }
```

7.3 function.h

```
double Opreate(double a,double b,char optr); //对操作数和对应的运算符进行计算并返回结果
int In(char c,char *tr); //判断字符是否为运算符
char Precede(char m, char n, char *OP); //判断运算符的优先级
double EvaluateExpression(char *expr); //算术表达式求值的算符优先算法
void clear(char *c);//删除输入的空格
int check(char *c);//检查是否有非法字符输入
void negative(char *c);//处理负数问题
void change(char *c,double x);//将变量x赋值
int variable(char *c);//判断输入是否为变量
```

7.4 function.c

```
#include "function.h"
#include "stack.c"
double Opreate(double a,double b,char tr) //对操作数和对应的运算符进行计算并返回结果
{
    switch(tr)
    {
        case'+':return a+b;
        case'-':return a-b;
        case'/':return a/b;
        case'*':return a*b;
        case'^':return pow(a,b);
        default:return 0;
    }
}

int In(char c,char *tr) //判断字符是否为运算符
{
    int i=0,flag=0;
    while(tr[i]!='\0')
    {
        if(tr[i]==c) flag=1;
        i++;
    }
    return flag;
}

char Precede(char m, char n, char *tr) //判断运算符的优先级
{
    unsigned char
        Prior[8][8]={">>><<<>>",">>><<<>>",">>>><<>>",">>>><<>>",">>>>><>>","<<<<<<=
        ",">>>>> >>","<<<<<< ="};
```

```
30      int i=0,j=0;
31      while(m!=tr[i]) i++;
32      while(n!=tr[j]) j++;
33      return Prior[i][j];
34  }
35
36  double EvaluateExpression(char *expr) //算术表达式求值的算符优先算法
37  {
38      negative(expr);
39      OPTR TR;OPND ND;
40      InitStack(&TR);initStack(&ND);
41      char *c,cc[500]={},tr,x,OP[10]={'+','-','*','/','^','(',')','#','\0'};
42      double a,b,cf;
43      int i=1;
44      Push(&TR,'#');
45      c=strcat(expr,"# "); //拼接表达式使其以#结尾
46      while(*c!='#'||GetTop(&TR)!='#')
47      {
48          fprintf(fp,"————————————————————————————第%d步————————————————————————————\n",i)
49          if(!In(*c,OP)) //不是运算符进栈
50          {
51              while(!In(*c,OP))
52              {
53                  strcat(cc,c); //两位数以上数字输入
54                  c++;
55              }
56              cf=atof(cc);
57              OutPut(&TR);
58              outPut(&ND);
59              fprintf(fp,"输入字符: '%lf'\n",cf);
60              memset(cc,0x00,sizeof(char)*256); //清空临时字符串
61              push(&ND,cf);
62              fprintf(fp,"操作: Push(OPND,'%lf')\n",cf);
63          }
64          else
65          {
66              OutPut(&TR);
67              outPut(&ND);
68              fprintf(fp,"输入字符: '%c'\n",*c);
69              switch(Precede(GetTop(&TR),*c,OP))
70              {
71                  case'<': //栈顶元素优先权低
72                      Push(&TR,*c);
73                      fprintf(fp,"操作: Push(OPTR,'%c')\n",*c);
74                      c++;
75                      break;
76                  case'=': //脱括号指针移动到下一字符
```

```
77                  Pop(&TR,&x);
78                  fprintf(fp,"操作: Pop(OPTR){消去一对括号}\n",x);
79                  c++;
80                  break;
81              case'>': //退栈并将运算结果入栈
82                  Pop(&TR,&tr);
83                  pop(&ND,&b);
84                  pop(&ND,&a);
85                  push(&ND,Opreate(a,b,tr));
86                  fprintf(fp,"操作: Opreate('%lf','%c','%lf')\n",a,tr,b);
87                  break;
88          }
89      }
90      i++;
91  }
92  fprintf(fp,"————————————————————————————————第%d步————————————————————————————————\n",i);
93  OutPut(&TR);
94  outPut(&ND);
95  fprintf(fp,"输入字符: '%c'\n",*c);
96  fprintf(fp,"操作:return GetTop(OPND)\n");
97  return getTop(&ND);
98 }
99
100 void clear(char *c)//删除输入的空格
101 {
102    char *str=c;
103    int i,j;
104    for(i=0,j=0;c[i]!='\0';i++)
105    {
106       if(c[i]!=' ')
107          str[j++]=c[i];
108    }
109    str[j]='\0';
110    c=str;
111 }
112
113 int check(char *c)//检查是否有非法字符输入
114 {
115     int i,j,t1=0,t2=0,flag=0;
116     for(i=0;i<strlen(c);i++)//括号输入不完整问题
117     {
118         if(c[i]=='(') t1++;
119         if(c[i]==')') t2++;
120         if(!In(c[i],"+-*/^()1234567890. "))
121         {
122             flag=1;
123             break;
```

```
124         }
125     }
126     if(t1!=t2) flag=1;
127     for(i=0,j=1;j<strlen(c);i++,j++)//运算符相邻问题
128     {
129         if(In(c[i],"+-*/^ ")&&In(c[j],"+-*/^ "))
130         {
131             flag=1;
132             break;
133         }
134     }
135     return flag;
136 }
137 
138 void negative(char *c)//处理负数问题
139 {
140     int i,j,k,l=0;
141     char str[100];
142     strcpy(str,c);
143     for(i=0,j=1;j<strlen(str);i++,j++)
144     {
145         if(str[i]=='('&&str[j]=='-')
146         {
147             c[j+l]='0';
148             for(k=j+l;c[k]!='\0';k++)
149                 c[k+1]=str[k-l];
150             l++;
151         }
152     }
153 }
154 
155 void change(char *c,double x)//将变量x赋值
156 {
157    char p[1000],a[100]={};
158     int i,j,k,l,f=0;
159     strcpy(p,c);
160     sprintf(a,"%lf",x);
161    for(i=0;i<strlen(c);i++)
162     {
163       if(c[i]=='x')
164         {
165             for(k=i+f*(strlen(a)-1),j=0;j<strlen(a);k++,j++)
166                 p[k]=a[j];
167             for(l=i+1;l<strlen(c);l++,k++)
168                 p[k]=c[l];
169         p[k]='\0';
170         f++;
```

```
        }
    }
    strcpy(c,p);
}

int variable(char *c)//判断输入是否为变量
{
    int i,j,n=0;
    for(i=0;i<strlen(c);i++)
    {
        if(c[i]=='x')
        {
            n=1;
            break;
        }
    }
    for(i=0,j=1;j<strlen(c);i++,j++)
    {
        if((c[i]=='x'||c[i]=='.')&&(c[j]=='.'||c[j]=='x'))
        {
            n=0;
            break;
        }
    }
    return n;
}
```

7.5 main.c

```
#include "function.c"
int main()
{
    int y=1,m=1,n=0;
    double x;
    char file[100],Algorite[500];
    while(y)
    {
        sprintf(file,"%d.txt",m);
        if((fp=fopen(file,"w"))==NULL) return -1;
        printf("请输入算式(^_^):");
        gets(Algorite);
        clear(Algorite);
        n=variable(Algorite);
        if(n)
        {
```

```
17          printf("请输入x的值:");
18          scanf("%lf",&x);
19          fprintf(fp,"变量表达式为:%s\nx=%lf\n",Algorite,x);
20          change(Algorite,x);
21      }
22      int flag=0;
23      flag=check(Algorite);
24      if(flag) printf("输入有误，请重新输入(T_T)\n");
25      else
26      {
27          fprintf(fp,"算式为：%s\n",Algorite);
28          negative(Algorite);
29          double result=EvaluateExpression(Algorite);
30          if(fabs(result)>DBL_MAX) printf("发生除0错误(-_-)\n");
31          else printf("%lf\n",result);
32          fclose(fp);
33      }
34      printf("是否继续计算？ 1:继续 0:退出\n");
35      scanf("%d",&y);
36      getchar();
37      m++;
38  }
39  system("pause");
40  return 0;
41 }
```

7.6 calculator.py

```
1  from tkinter import *
2  from functools import partial
3  from ctypes import *
4  cpy=CDLL('./function.dll')
5  cpy.EvaluateExpression.restype=c_double
6  cpy.check.restype=c_int
7  cpy.variable.restype=c_int
8  cpy.change.restype=c_void_p
9
10 #生成计算器主界面
11 def layout(root):
12     label=Label(root,width=29,height=1,bd=5,bg='#FFFACD',anchor='se',
13     textvariable=label_text) #标签，可以显示文字或图片
14     label.grid(row=0,columnspan=5) #布局器，向窗口注册并显示控件;
           rowspan：设置单元格纵向跨越的列数
15     entry=Entry(root,width=23,bd=5,bg='#FFFACD',justify="right",font=('微软雅黑', 12))
           #文本框（单行）
```

```
16      entry.grid(row=1,column=0,columnspan=5,sticky=N+W+S+E,padx=5,pady=5)
            #设置控件周围x、y方向空白区域保留大小
17      myButton=partial(Button,root,width=5,cursor='hand2',activebackground='#90EE90') #
            偏函数：带有固定参数的函数
18      #command指定按钮消息的回调函数
19      button_clear=myButton(text=' C ',command=lambda:clear(entry))
20      button_left=myButton(text='(',command=lambda:entry.insert(INSERT,'('))
21      button_right=myButton(text=')',command=lambda:entry.insert(INSERT,')'))
22      button_backspace=myButton(text='←',command=lambda:entry.delete(len(entry.get())-1))
            #删除文本框的最后一个输入值
23      button_x=myButton(text='\n x \n',command=lambda:entry.insert(INSERT,'x'))
24      button_clear.grid(row=3,column=0)
25      button_left.grid(row=3,column=1)
26      button_right.grid(row=3,column=2)
27      button_backspace.grid(row=3,column=3)
28      button_x.grid(row=3,column=4)
29      #第二行
30      button_7=myButton(text=' 7 ',command=lambda:entry.insert(INSERT,'7'))
31      button_8=myButton(text=' 8 ',command=lambda:entry.insert(INSERT,'8'))
32      button_9=myButton(text=' 9 ',command=lambda:entry.insert(INSERT,'9'))
33      button_divide=myButton(text=' / ',command=lambda:entry.insert(INSERT,'/'))
34      button_fx=myButton(text='赋值',command=lambda:label_text.set('x='))
35      button_7.grid(row=4,column=0)
36      button_8.grid(row=4,column=1)
37      button_9.grid(row=4,column=2)
38      button_divide.grid(row=4,column=3)
39      button_fx.grid(row=4,column=4)
40      #第三行
41      button_4=myButton(text=' 4 ',command=lambda:entry.insert(INSERT,'4'))
42      button_5=myButton(text=' 5 ',command=lambda:entry.insert(INSERT,'5'))
43      button_6=myButton(text=' 6 ',command=lambda:entry.insert(INSERT,'6'))
44      button_multi=myButton(text=' * ',command=lambda:entry.insert(INSERT,'*'))
45      button_sure=myButton(text='确认',command=lambda:sure(entry))
46      button_4.grid(row=5,column=0)
47      button_5.grid(row=5,column=1)
48      button_6.grid(row=5,column=2)
49      button_multi.grid(row=5,column=3)
50      button_sure.grid(row=5,column=4)
51
52      #第四行
53      button_1=myButton(text=' 1 ',command=lambda:entry.insert(INSERT,'1'))
54      button_2=myButton(text=' 2 ',command=lambda:entry.insert(INSERT,'2'))
55      button_3=myButton(text=' 3 ',command=lambda:entry.insert(INSERT,'3'))
56      button_subtract=myButton(text=' - ',command=lambda:entry.insert(INSERT,'-'))
57      button_equal=myButton(text=' \n = \n ',command=lambda:calculate(entry))
58      button_1.grid(row=6,column=0)
59      button_2.grid(row=6,column=1)
```

```
60      button_3.grid(row=6,column=2)
61      button_subtract.grid(row=6,column=3)
62      button_equal.grid(row=6,column=4,rowspan=2)
63      #第五行
64      button_power=myButton(text='^',command=lambda:entry.insert(INSERT,'^'))
65      button_0=myButton(text=' 0 ',command=lambda:entry.insert(INSERT,'0'))
66      button_point=myButton(text=' . ',command=lambda:entry.insert(INSERT,'.'))
67      button_plus=myButton(text=' + ',command=lambda:entry.insert(INSERT,'+'))
68      button_power.grid(row=7,column=0)
69      button_0.grid(row=7,column=1)
70      button_point.grid(row=7,column=2)
71      button_plus.grid(row=7,column=3)
72
73  #删除所有输入内容和显示内容
74  def clear(entry):
75      entry.delete(0,END)
76      label_text.set('')
77
78  def sure(entry):
79      global m
80      m=entry.get()
81      if m=='':
82          m=0
83      m=float(m)
84      clear(entry)
85      return m
86
87  # 点击“=”后进行计算
88  def calculate(entry):
89      try:
90          formula=entry.get()
91          x=formula
92          x=x.encode('utf8')
93          x=c_char_p(x)
94          if cpy.variable(x)==1:
95              cpy.change(x,c_double(m))
96          clear(entry)
97          if cpy.check(x)==1:
98              entry.insert(END,'出错(T_T)')
99          else:
100             result=cpy.EvaluateExpression(x)
101             if abs(result)==float('inf'):
102                 entry.insert(END,'发生除0错误(-_-)')
103             else:
104                 entry.insert(END,result)
105         label_text.set(''.join(formula+'='))
106     except:
```

```
107         clear(entry)
108         entry.insert(END,'出错(T_T)')
109 
110 
111 if __name__ == '__main__':
112     root=Tk() #生成窗口
113     root.title('计算器') #窗口的名字
114     root.resizable(0,0) #窗口大小可调性，分别表示x，y方向的可变性
115     global label_text #定义全局变量
116     label_text=StringVar()
117     layout(root)
118     root.mainloop() #进入消息循环（必需组件），否则生成的窗口一闪而过
```