

# “简易计算器”实验报告

---

## 项目git地址

---

(在pta提交后又进行了一些小的修改，麻烦助教大大以这个最新版本为准，或以git仓库中的最新版本为准)

url: <https://github.com/dendenxu/npl> git command: git clone <https://github.com/dendenxu/npl>  
lgit

## 正文

---

### 写在前面

---

为了后期方便调试，在最后的几个版本中将stack.c stack.h main.c几个文件进行了合并，内容没有发生实际变动。前期开发是分成两个模块进行的，stack.c和main.c，后期加入函数功能后，main.c变得庞大，为了方便直接编译调试，以及提高开发效率，将三个文件合并。助教大大可通过git上版本回退检查这一点。

### 功能及主体思想

---

1. 通过将用户输入的中缀表达式转换为后缀表达式
2. 计算后缀表达式的值来实现简单的计算器功能（因此，对于不加括号的除法，会先进行后面的乘法运算）
3. 通过递归实现函数调用（主要是math.h中的函数）功能
4. 通过递归实现不加括号的函数调用功能
5. 通过循环进行错误输入处理，并以此为基础添加历史记录功能
6. 对有特殊含义的字符，pi，e等内置精确值
7. 建立了git仓库，助教大大可查验项目生成的完整过程（似乎整个过程也只有我一个人在push.....）

### 使用注意事项


---

1. 一切计算都是在假定用户输入格式正确的情况下进行的（即，表达式不会引起歧义（例如不可以写sinpi，需要写sin pi），括号成对出现，运算符需要几个写几个（例如不能出现++））。如果输入的格式不是严格的中缀表达式，谁知道会发生什么
2. 一切计算都应以等于号结束，若用户在敲回车前忘记了敲等于号，没关系，程序会跳过空白字符，再敲一次即可。
3. 支持的运算符：加 减 乘 除 左括号 右括号 等于号

4. 支持的数据类型：双精度浮点类型（注意不可以将0.3写成.3，但是可以将3.0写成3.），支持负数
5. 支持的函数：sin cos sqrt fabs tan atan asin acos exp log floor
6. 支持的关键字：e ans ANS Ans pi PI Pi
7. 支持处理的错误类型：
  - 函数表达式不正确
  - 输入了不支持的字符，例如中文字符
8. 未计算出有效结果时，请勿使用ANS关键字，否则谁知道会发生什么
9. 栈大小：10000\*sizeof(char)，浮点数组大小：10000\*sizeof(double)，用户需保证输入的计算式不会超出两者的限制，否则谁知道会发生什么
  - 超过栈大小限制时程序会报错
  - 超过浮点数组大小时会导致难以描述的事情发生
10. 用户输入的数据中可以在任意地方放入空白字符，比如用户输入1+1=或者1 + 1 =都可以得到正确的结果（当然，不可以在原本应连续输入的函数中插入奇怪的空白字符，例如函数和关键字中）
11. 用户调用函数时，请保证格式，可以采取的格式有
  - 直接加括号调用，最保险，但在linux系统中习惯敲完左右括号再左移的用户可能会遇到问题（shell会将左右移动符号解释为^)]D ^[[C)
  - 单值函数可以以无括号的形式传入格式合法的单值参数
  - 所有调用都可嵌套
12. 所有运算都是浮点运算，所以最终结果可能与实际结果有微小的差异，但这也是不可避免的……，最终的运算结果输出形式与最终结果有关（以%g格式输出），有可能为E型或浮点型
13. 为了方便使用（以及实现ans关键字），写入死循环，如果不手动Ctrl+C，会一直运行下去
14. 当计算器程序出现了奇怪的错误（一般情况下这种错误是由不严格的表达式造成的），请Ctrl+C
15. 理论上，程序可以在所有能编译C的环境下运行，经过测试的环境有：Windows10 x64 1809，Ubuntu 18.04 x64，CentOS7 x64，Windows7 x86
  - windows10

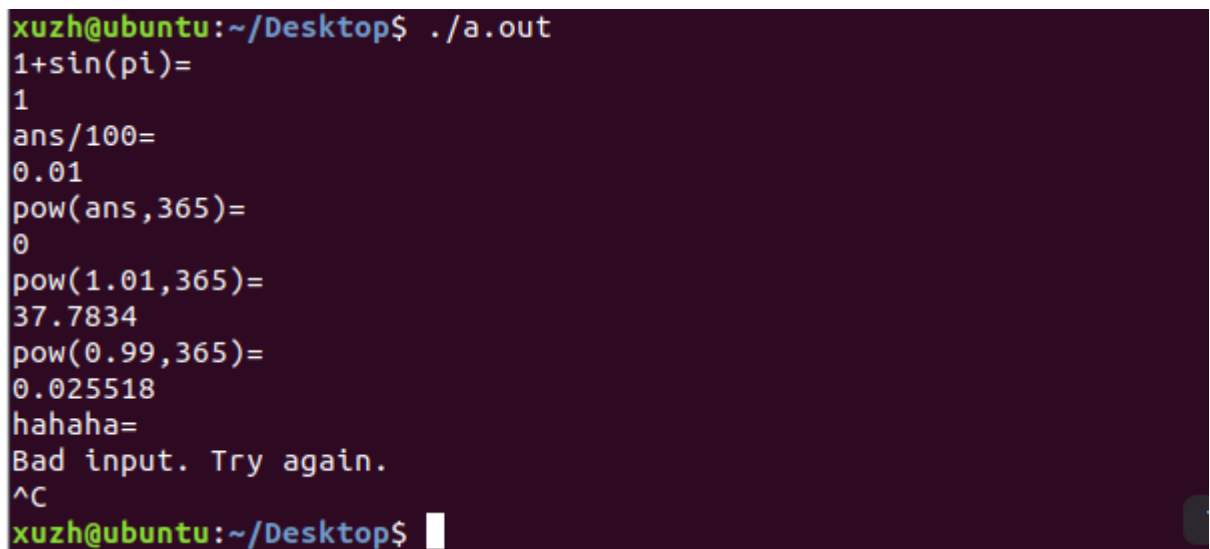
```
4 warnings generated.
-1=
-1
-1+1=
0
1-1=
0
^C
e:\Documents\Material\DENDEn\proj\cc++\qianhuiCbig\npl>
```

- windows7



```
C:\Users\67898\Desktop\main.exe
floor pi=
3
ans+sqrt 4=
5
```

- o ubuntu



```
xuzh@ubuntu:~/Desktop$ ./a.out
1+sin(pi)=
1
ans/100=
0.01
pow(ans,365)=
0
pow(1.01,365)=
37.7834
pow(0.99,365)=
0.025518
hahaha=
Bad input. Try again.
^C
xuzh@ubuntu:~/Desktop$
```

- o centos

```
[xuzh@localhost Desktop]$ ./a.out
1+1=
2
2+2=
4
e+e=
5.43656
pi+pi/2=
4.71239
sin(2*pi+pi)=
2.6938e-10
^C
[xuzh@localhost Desktop]$
```

经过测试的编译器：

- gcc 8.1.0 64-bit(mingw:windows10)
- LLVM-clang 64-bit(windows10)
- Microsoft cl 64-bit(windows10)
- Microsoft cl 32-bit(windows10)
- TDM gcc 4.9.2 32-bit(dev-cpp)
- TDM gcc 4.9.2 64-bit(dev-cpp)
- gcc 8.2.0 64-bit(linux:centos)
- gcc 7.3.0 64-bit(linux:ubuntu)

但是，每一种运行环境都有自己的特点（例如上述linux中的方向键解释，以及linux中gcc需要链接math库才可正常编译（gcc main.c -o a.out -lm）），本人能力有限，无法测试市面上所有的编译器及环境，因此也无法保证项目在您的机器上一定会完美编译运行。

16. 在项目设计者的机器上可以基本保证0 warnings（除了microsoft的SDL checks）

- clang warnings(微软的警告类似：原因在于，我的机器上clang调用的Visual Studio的windows SDK)

```
main.c:109:9: warning: 'scanf' is deprecated: This function or variable may
be unsafe. Consider using scanf_s instead. To disable deprecation, use
_CRT_SECURE_NO_WARNINGS. See online help for details. [-wdeprecated-
declarations]
    scanf("%lf", &re);
    ^
C:\Program Files (x86)\Windows
Kits\10\Include\10.0.17763.0\ucrt\stdio.h:1273:20: note: 'scanf' has been
explicitly marked deprecated here
    _Check_return_ _CRT_INSECURE_DEPRECATE(scanf_s)
    ^
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Professional\VC\Tools\MSVC\14.16.27023\include\vcruntime.h:255:5
5: note: expanded from macro '_CRT_INSECURE_DEPRECATE'
    #define _CRT_INSECURE_DEPRECATE(_Replacement) _CRT_DEPRECATED_TEXT(
    \
    ^
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Professional\VC\Tools\MSVC\14.16.27023\include\vcruntime.h:245:4
7: note: expanded from macro '_CRT_DEPRECATED_TEXT'
```

```

#define _CRT_DEPRECATED_TEXT(_Text) __declspec(deprecated(_Text))
^
main.c:247:17: warning: 'scanf' is deprecated: This function or variable may
be unsafe. Consider using scanf_s instead. To disable deprecation, use
_CRT_SECURE_NO_WARNINGS. See online help for details. [-wdeprecated-
declarations]
        scanf("%lf", &num[cnt++]); //<C0><FB><D3><C3><U+04BB><B8>
<F6><B6><EE><CD><E2><B5><C4><CA><FD><D7><E9><BD><F8><D0><D0><CA><FD><D7>
<U+05B5><U+0134><A2><B4><U+68EC><D2><U+0531><E3><U+05B1><BD><D3><C0><FB><D3>
<C3>scan...
^
C:\Program Files (x86)\Windows
Kits\10\Include\10.0.17763.0\ucrt\stdio.h:1273:20: note: 'scanf' has been
explicitly marked deprecated here
        _Check_return_ _CRT_INSECURE_DEPRECATED(scanf_s)
^
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Professional\VC\Tools\MSVC\14.16.27023\include\vcruntime.h:255:5
5: note: expanded from macro '_CRT_INSECURE_DEPRECATED'
        #define _CRT_INSECURE_DEPRECATED(_Replacement) _CRT_DEPRECATED_TEXT(
\
^
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Professional\VC\Tools\MSVC\14.16.27023\include\vcruntime.h:245:4
7: note: expanded from macro '_CRT_DEPRECATED_TEXT'
#define _CRT_DEPRECATED_TEXT(_Text) __declspec(deprecated(_Text))
^
main.c:277:13: warning: 'scanf' is deprecated: This function or variable may
be unsafe. Consider using scanf_s instead. To disable deprecation, use
_CRT_SECURE_NO_WARNINGS. See online help for details. [-wdeprecated-
declarations]
        scanf("%lf", &num[cnt++]); //<C0><FB><D3><C3><U+04BB><B8><F6>
<B6><EE><CD><E2><B5><C4><CA><FD><D7><E9><BD><F8><D0><D0><CA><FD><D7><U+05B5>
<U+0134><A2><B4><U+68EC><D2><U+0531><E3><U+05B1><BD><D3><C0><FB><D3>
<C3>scanf...
^
C:\Program Files (x86)\Windows
Kits\10\Include\10.0.17763.0\ucrt\stdio.h:1273:20: note: 'scanf' has been
explicitly marked deprecated here
        _Check_return_ _CRT_INSECURE_DEPRECATED(scanf_s)
^
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Professional\VC\Tools\MSVC\14.16.27023\include\vcruntime.h:255:5
5: note: expanded from macro '_CRT_INSECURE_DEPRECATED'
        #define _CRT_INSECURE_DEPRECATED(_Replacement) _CRT_DEPRECATED_TEXT(
\
^
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Professional\VC\Tools\MSVC\14.16.27023\include\vcruntime.h:245:4
7: note: expanded from macro '_CRT_DEPRECATED_TEXT'
#define _CRT_DEPRECATED_TEXT(_Text) __declspec(deprecated(_Text))
^
3 warnings generated.

```

## 项目分工

---

- 前期设计
  - 架构设计及实现：徐震
  - 主函数设计及实现：徐震
  - git项目创建及维护-徐震
  - stack.h设计及实现：徐震
  - 部分stack.c函数设计及实现：徐震
  - stack.c主体设计及实现：付仁泓
  - 项目总体调试及初步测试运行：徐震
  - 简单函数计算功能设计及实现：陈九润（待完成）（最终未完成……）
- 后期完善
  - 简单函数功能-徐震
  - 不加括号的单值函数-徐震
  - 现在能对单独参数是一个函数的情况进行处理-徐震
  - 对e,pi,Pi,PI等关键字进行处理-徐震
  - 历史记录功能，可以用ans,ANS,Ans调用-徐震
  - 最终调试-徐震
  - 实验报告-徐震

## 感想

---

- 整个项目的开发过程，有点类似编译器的编写过程，一步步添加对表达式，函数，关键字的支持（不知道我对编译器的理解对不对）
- 没有走开发图形界面的路子，因为个人认为，相对于大多数图形界面计算器（每次输入运算符都立即给出反馈进行计算），命令行的行缓冲给了用户更大的修改机会，并且若用户更习惯用键盘而不是鼠标，此种简约的设计方式会提高用户的使用效率以及使用舒适度。（反正写出来之后我自己一直在用它做物理题hhhh，一开始是没有while(1)的死循环的，是我做题时候觉得每次都要重新打开也太麻烦了叭，才加进去的）
- 没有在函数的实现细节上过分纠结，而是直接调用的标准的math库，是由于个人认为，除非是学习需要，没必要造太多轮子，且过分纠结于那部分内容偏离了项目本身目的（queue和stack的应用。）  
（还有个原因是……那也太硬核了叭，大一小菜鸡还不会FFT）
- **几乎没有UI……，所以，请使用前阅读“使用注意事项”**
- 开发过程的工作分配存在极大问题。几乎成了个人项目。真实心累。开发，管理，调试几乎全都一个人干的太心酸。于是下一个项目（Project2-1）直接当单人项目来写了。
- 注释真的挺重要的……看没有注释而且命名模糊的代码真是……唉……
- 代码过于臃肿，有很多想做的优化、想实现的新功能，都没有实现，例如
  - 用函数指针实现函数调用的内容
  - 用类似Python的字典的形式实现关键字的内容（就算是仅仅用struct来表示而不追求 $O(\log n)$ 的复杂度也可以大大提高扩展性）
  - 命名存在极大的规范空间。各种flag，temp满天飞，自己看还好，其他人可读性较差。

- 注释写的极其随意……
- 有时间的话我也想用FFT实现math里面的各类函数啊……
- 有时间的话我也想做个GUI啊……
- 渴望实现的memory功能没有实现
- 渴望对更多运算符增加支持，例如^
- 渴望用更规范的方式增加对运算符的支持（可能需要摒弃中缀转后缀的思想，而采用一边输入一边计算的思想，但这样……没有了计算式子的功能，会对用户的输入的准确度提出较高的要求。）
- 渴望实现的历史记录（不单单是记住上一次的ans，而是对每一个成功的运算都做完整的记录）功能没有实现。

## 测试样例

```
1+1=
1 + 1 =
1 + sin ( pi ) =
1 + sin sin sin (pow(sqrt(pi) , 2)) * 10000 + 1 / 2 / 3=
ans + 1 =
fabs ans =

// output
// 1+1=
// 2
// 1 + 1 =
// 2
// 1 + sin ( pi ) =
// 1
// 1 + sin sin sin (pow(sqrt(pi) , 2)) * 10000 + 1 / 2 / 3=
// 2.5
// ans + 1 =
// 3.5
// fabs ans =
// 3.5
// ^C
```

## 源码

**（在pta提交后又进行了一些小的修改，麻烦助教大大以这个版本或以git仓库中的最新版本为准）**

```
#include <stdio.h>
#include <stdlib.h>
```

```

#include <string.h>
#include <ctype.h>
#include <math.h>

#define MAXSTACKSIZE 10000 //用于创建栈和double数组

//用于清空以保守的方式清空stdin
#define CLEARSTDIN \
{ \
    int ch; \
    while ((ch = getchar()) != '\n' && ch != EOF) \
        ; \
}

typedef struct _Stack
{
    char array[MAXSTACKSIZE];
    int top; //栈顶index
} * Stack;

Stack create(void); //初始化栈（array和top都变为零）
char pop(Stack stack); //栈为空时返回'0'
void push(Stack stack, char c); //达到最大值后报错并结束程序
char peek(Stack stack); //栈为空时返回'0'
void destroy(Stack stack); //清空该栈的内容（其实没啥用因为C里面没有对象或者析构函数）
char peekmid(Stack stack, int index);
double npl(void);

int wrong;
double output;

int main()
{
    while (1)
    {
        do
        {
            wrong = 0;
            output = npl();
            CLEARSTDIN
        } while (wrong);
        printf("%g\n", output);
    }
}

Stack create(void)
{
    Stack S;
    S = (Stack)malloc(sizeof(*S));
    int i = 0;
    S->top = 0;
    return S;
}

```



```

char pop(Stack stack)
{
    if (stack->top == 0)
        return '0';
    else
    {
        char a = stack->array[--stack->top];
        return a;
    }
}

void push(Stack stack, char c)
{
    if (stack->top == MAXSTACKSIZE - 1)
    {
        printf("Stack is FULL");
        exit(1);
    }
    else
        stack->array[stack->top++] = c;
}

char peek(Stack stack)
{
    if (stack->top <= 0)
        return '0';
    else
    {
        return stack->array[stack->top - 1];
    }
}

void destroy(Stack stack)
{
    free(stack);
    stack = NULL;
}

char peekmid(Stack stack, int index)
{
    if (index >= stack->top)
        return '0';
    return stack->array[index];
}

double func()
{
    //读入用于表达函数的字符串
    char ch = getchar();
    if (isdigit(ch))
    {
        double re;

```

```

    ungetc(ch, stdin);
    scanf("%lf", &re);
    return re;
}
else if (isalpha(ch))
{
    char c;
    int i = 0;
    char temp[MAXSTACKSIZE];
    while (!wrong && isalpha(c = getchar()))
    {
        temp[i++] = ch;
        ch = c;
    }
    temp[i++] = ch;
    temp[i] = 0;

    if (!strcmp(temp, "pi") || !strcmp(temp, "PI") || !strcmp(temp, "Pi"))
    {
        ungetc(c, stdin);
        return 3.1415926535;
    }
    if (!strcmp(temp, "e"))
    {
        ungetc(c, stdin);
        return 2.71828;
    }
    if (!strcmp(temp, "ANS") || !strcmp(temp, "Ans") || !strcmp(temp, "ans"))
    {
        ungetc(c, stdin);
        return output;
    }

    if (wrong)
        return 1;

    //跳过空格
    while (isspace(c))
        c = getchar();

    double tempNum;
    //确定function应该计算的值是什么（这里会发生一系列函数的多次递归）
    if (c != '(') //上面的getchar读入了空格后面的一个东西
    {
        ungetc(c, stdin);
        tempNum = func();
    }
    else
        tempNum = npl();
    if (wrong)
        return 1;
    //这一段算灵活性比较高的，想要的话甚至可以把能调用的系统里所有的计算型的double函数都添加进去

```

//如果用比较高级的方式，比如哈希表？或许会大幅度提高性能？或者考虑函数指针，用switch建立类似的哈希表？但这里包含的函数实在太少.....利用哈希表提高的性能或许还满足不了complexity的常数项对性能的拖累

//对于pow这种函数，有两个参数，不会出现后面不加括号的情况

```
if (!strcmp(temp, "pow"))
{
    double
        temp1 = tempNum,
        temp2 =
            npl(); //最好不要直接在pow函数中调用两个npl，执行顺序是个变数。
    return pow(temp1, temp2);
}
else if (!strcmp(temp, "sin"))
    return sin(tempNum);
else if (!strcmp(temp, "cos"))
    return cos(tempNum);
else if (!strcmp(temp, "sqrt"))
    return sqrt(tempNum);
else if (!strcmp(temp, "fabs"))
    return fabs(tempNum);
else if (!strcmp(temp, "tan"))
    return tan(tempNum);
else if (!strcmp(temp, "atan"))
    return atan(tempNum);
else if (!strcmp(temp, "asin"))
    return asin(tempNum);
else if (!strcmp(temp, "acos"))
    return acos(tempNum);
else if (!strcmp(temp, "exp"))
    return exp(tempNum);
else if (!strcmp(temp, "log"))
    return log(tempNum);
else if (!strcmp(temp, "floor"))
    return floor(tempNum);
else
{
    printf("Bad input. Try again.\n");
    CLEARSTDIN
    wrong = 1;
    return 1;
}
}
return 1; //这种情况一般不会发生。
}
```

```
double npl()
{
    Stack infix = create();
    Stack postfix = create();
    double num[MAXSTACKSIZE] = {0}; // num数组用于储存读入的数字
    int cnt = 0;                      // cnt代表num的index

    //读入数据并直接转换为后缀表达式
    char ch;
```

```

int flag = 1;
int flag2 = 0;
int prefix = 1;
while ((!wrong && flag && (ch = getchar()) != '='))
{
    if (isspace(ch)) //跳过空格
        continue;
    if (isalpha(ch))
    {
        ungetc(ch, stdin);
        num[cnt++] = func();
        push(postfix, -cnt + 1);
        continue;
    }
    switch (ch)
    {
        case '*':
        case '/':
            push(infix, ch); //高优先级的运算符直接压栈到infix
            break;
        case '(':
            prefix = 0;
            push(infix, ch); //低优先级的运算符直接压栈到infix
            flag2 = 1; // flag2用于判断是否处于合法的括号当中（便于递归函数的实现）
            break;
        case '+':
            while (peek(infix) == '*' || peek(infix) == '/')
                push(postfix, pop(infix)); //低优先级的运算符先将infix中高优先级的出栈到postfix
再压栈到infix
            push(infix, ch);
            break;
        case '-':
            if (prefix)
            {
                ungetc(ch, stdin);
                scanf("%lf", &num[cnt++]); //利用一个额外的数组进行数字的储存，以便直接利用scanf
读入double类型功能的实现，cnt是该数组的指针（也可以用栈直接进行实现）
                push(postfix, -cnt + 1); //遇到数字直接进postfix
                break;
            }
            while (peek(infix) == '*' || peek(infix) == '/')
                push(postfix, pop(infix)); //低优先级的运算符先将infix中高优先级的出栈到postfix
再压栈到infix
            push(infix, ch);
            break;
        case ')':
            if (flag2)
            {
                while (peek(infix) != '(') //将infix中的运算符出栈，直到遇到'(', 注意'('也要出
栈，但不进入postfix
                    push(postfix, pop(infix));
                pop(infix);
            }
    }
}

```

```

        flag2 = 0;
    }
    else
        flag = 0;
    break;
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
case '0':
    ungetc(ch, stdin);
    scanf("%lf", &num[cnt++]); //利用一个额外的数组进行数字的储存，以便直接利用scanf读入
double类型功能的实现，cnt是该数组的指针（也可以用栈直接进行实现）
    push(postfix, -cnt + 1);    //遇到数字直接进postfix
    break;

case ',':
    flag = 0;
    break;
default:
    //输入格式有误，清空所有堆栈
    printf("Bad input. Try again.\n");
    CLEARSTDIN
    wrong = 1;
    break;
}
prefix = 0;
}

if (wrong)
    return 1;

while (peek(infix) != '0') //倒序postfix剩余元素进infix
    push(postfix, pop(infix));

//计算结果
double re; //用于储存每一次的结果
int index = 0;
double num1, num2;
while ((ch = peekmid(postfix, index++)) != '0')
{
    if (ch <= 0) //数字直接压栈到infix
    {
        push(infix, ch);
        continue;
    }
    switch (ch) //对postfix加减乘除直接进行运算，并将运算结果压栈到infix
    {

```

```

        case '+':
            re = num[-pop(infix)] + num[-pop(infix)];
            num[cnt++] = re;
            push(infix, -cnt + 1);
            break;
        case '-':
            num1 = num[-pop(infix)];
            num2 = num[-pop(infix)];
            re = num2 - num1;
            num[cnt++] = re;
            push(infix, -cnt + 1);
            break;
        case '*':
            re = num[-pop(infix)] * num[-pop(infix)];
            num[cnt++] = re;
            push(infix, -cnt + 1);
            break;
        case '/':
            num1 = num[-pop(infix)];
            num2 = num[-pop(infix)];
            re = num2 / num1;
            num[cnt++] = re;
            push(infix, -cnt + 1);
            break;
    }
}
double result = num[-pop(infix)]; //输出infix的最后结果
destroy(infix);
destroy(postfix);
wrong = 0;
return result;
}

```