模拟启明电梯运行系统

学习文档

学院：电子信息与通讯学院

班级：2308

学号：U202314612

姓名：曾庆浩

目录

0-2 学习文档

C 语言学习

学习资料：[程序设计入门——C 语言_中国大学 MOOC(慕课) (icourse163.org)](icourse163.org)
以下是笔记摘要：

基础运算

优先级	运算符	运算	结合关系	举例
1	+	单目不变	自右向左	a*+b
1	-	单目取负	自右向左	a*-b
2	*	乘	自左向右	a*b
2	/	除	自左向右	a/b
2	%	取余	自左向右	a%b
3	+	加	自左向右	a+b
3	-	减	自左向右	a-b
4	=	赋值	自右向左	a=b

图表 1 运算符优先级

复合取值 “+= -= *= /= %=”。
递增递减运算符 “++ --”　a++ ++a 要注意区别。

判断

“if（）{ }else{ }”
注意 else 与 if 的匹配问题。
补充 if(x)中如果 x 的值是 0 就不执行 if 里的语句，如果是非零就执行语句。

循环

大致分为 3 种：If else、while 以及 do while 。
学会使用循环的级联和嵌套，以及 break 和 continue 使用与区别。

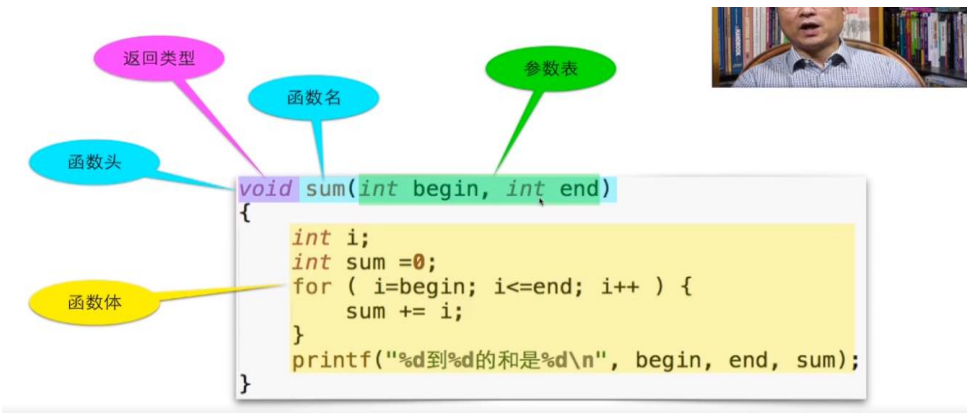
逻辑运算“&& ||”

优先级	运算符	结合性
1	()	从左到右
2	! + - ++ --	从右到左（单目的+和-）
3	* / %	从左到右
4	+ -	从左到右
5	< <= > >=	从左到右
6	== !=	从左到右
7	&&	从左到右
8	||	从左到右
9	= += -= *= /= %=	从右到左

图表 2 优先级

对于&&，如果左边的条件不满足，就不会进行右边的判断，所以右边表达式的附作用（比如 a++等）也会消失。

函数



图表 3 函数构成

使用函数需要有函数声明以及函数定义。

函数调用的值与参数值不同时不会引发编译错误，因此要自己检查。

在 c 语言中，函数不能嵌套使用，即函数内部无法再定义一个新的函数。

分清楚形参和实参，理解参数传递的是值的实质。

数组 指针 函数参数 &、[]运算符:

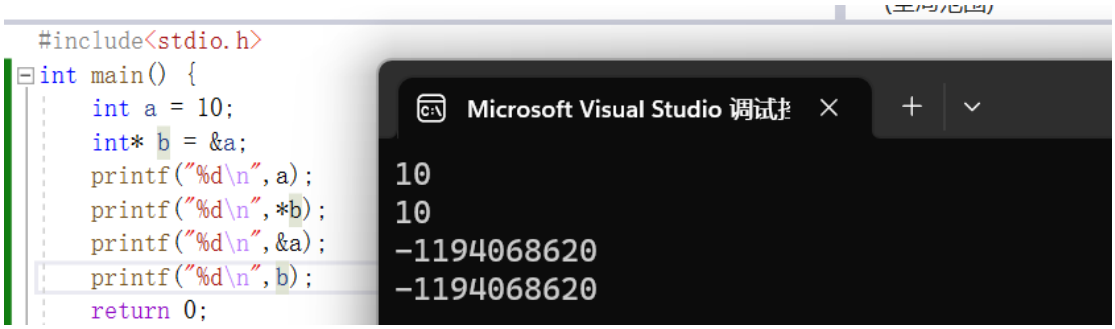
先说指针，指针的“值”是一个代表了地址的数字，指针是用来存储地址的变量，电脑能通过这个地址找到内存中存储的值并进行读取或修改。

而&运算符的结果就是某个变量的地址。

数组可以看作是一个特殊的指针，在通过参数传递到函数时，函数得到的值是一个指针，这个指针指向数组的第一个元素，即 array==&array[0]，因此再通过参数传递时无法传递数组的长度，常常需要另一个变量来表示数组的长度。

指针能用数组的运算符[]来进行运算，*p[0]==1。

明确易混概念：



图表 4 易混概念

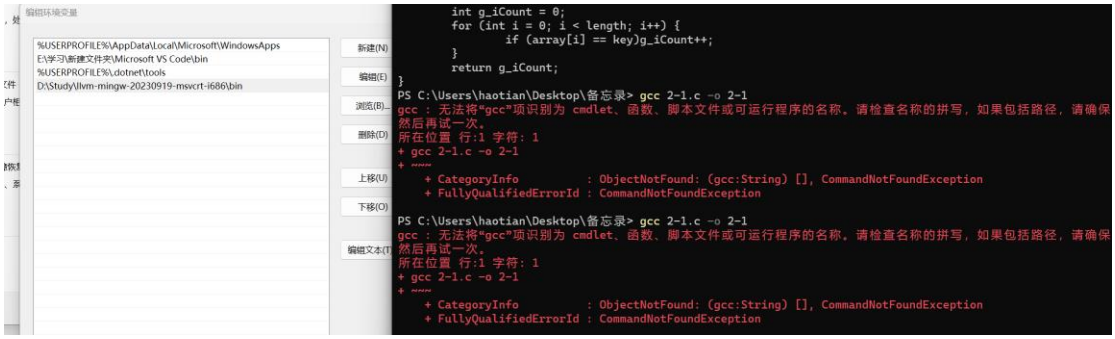
编译器学习

安装以及配置

在官网下载得到下图文件

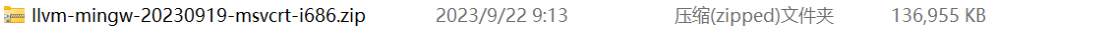
然后 [MinGW-W64 下载、安装与配置（支持最新版的 GCC，目前 GCC 13.2.0）_mingw64 配置_小青龍的博客-CSDN 博客]

根据这个尝试配置。



图表 5 错误 1

失败，于是重新下载文件。



尝试后再次失败，在搜索后了解到是库的选择出错了。

2. MSVCRT 和 UCRT 介绍

MSVCRT和UCRT都是用于Windows平台的C运行时库，提供了基本的C函数和类型，用于C程序的开发和运行。

MSVCRT（Microsoft Visual C Runtime）是Microsoft Visual C++早期版本使用的运行时库，用于支持C程序的运行。它提供了一些常用的C函数，如printf、scanf、malloc、free等。MSVCRT只能在32位Windows系统上运行，对于64位系统和Windows Store应用程序不支持。

UCRT（Universal C Runtime）是在Windows 10中引入的新的C运行时库，用于支持C程序的运行和开发。UCRT提供了一些新的C函数，同时还支持Unicode字符集和安全函数，如strcpy_s、strcat_s、_itoa_s等。UCRT同时支持32位和64位系统，并且可以与Windows Store应用程序一起使用。

总的来说，UCRT是Microsoft为了更好地支持Windows 10和Windows Store应用程序而开发的新一代C运行时库，相比于MSVCRT，UCRT提供了更多的功能和更好的兼容性。但对于旧的32位Windows系统，MSVCRT仍然是必需的。

图表 6 库函数介绍

2. 离线安装

从 GitHub或镜像站点下载编译好的安装程序包（【二、下载】中提供的有地址）

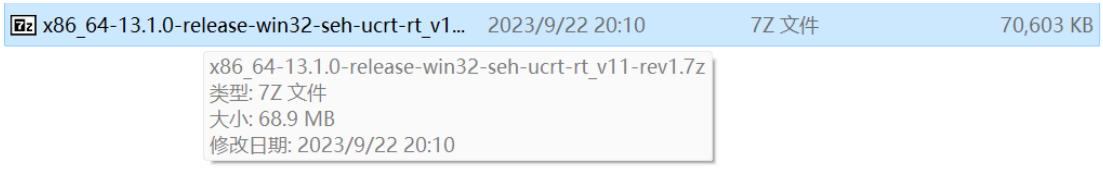
以 niXman/mingw-builds-binaries/releases 为例，离线安装比较简单，只需下载解压即可

1）下载 `MinGW-w64` 的安装包
根据你自己的需要选择适合的安装包

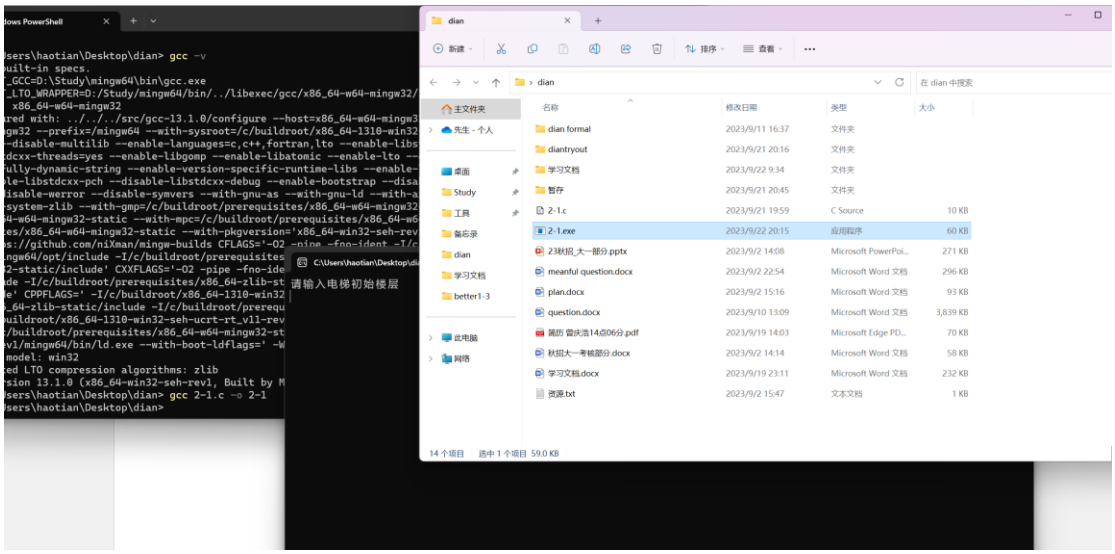
- 32位的操作系统，选择 `i686`，64位的操作系统，选择 `x86_64`；
- `13.1.0` 是GCC的版本号，其他版本的你需要往下找；
- `win32` 是开发windows系统程序的协议，`posix` 是其他系统的协议（例如Linux、Unix、Mac OS），更多信息参考：mingw-w64-threads-posix-vs-win32；
- 异常处理模型 `seh`（新的，仅支持64位系统），`sjlj`（稳定的，64位和32位都支持），`dwarf`（优于sjlj的，仅支持32位系统），更多信息参考：what-is-difference-between-sjlj-vs-dwarf-vs-seh/15685229；
- `msvcrt`、`ucrt` 运行时库类型，有关介绍请参考文章简介部分；
- `rt_v11` 运行时库版本；
- `rev1` 构建版本；

图表 7 版本选择

重新安装后尝试



图表 8 正确版本

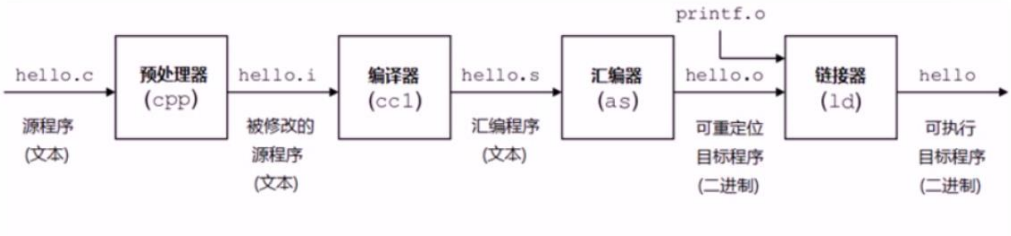


图表 9 成功

成功

指令学习

从源文件到可执行程序的过程



图表 10 过程介绍

参考 gptGCC 编译详解_错误: 在无返回值的函数中,'return'带返回值 [-werror=return-type]_junmuzi 的博客-CSDN 博客　GCC 编译详解_错误: 在无返回值的函数中,'return'带返回值 [-werror=return-type]_junmuzi 的博客-CSDN 博客

得到以下指令:

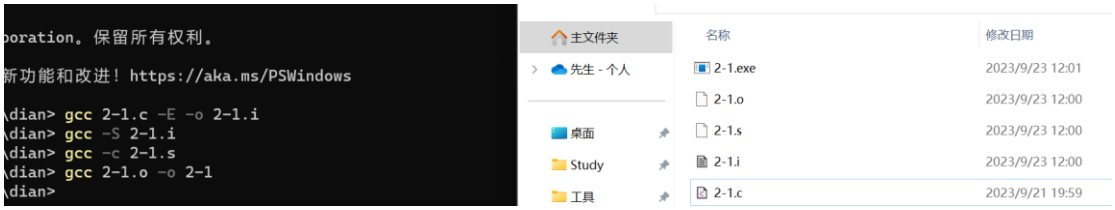
```
-E：只进行预处理，不编译
-S：只编译，不汇编
-c：只编译、汇编，不链接
-g：包含调试信息
-I：指定include包含文件的搜索目录
-o：输出成指定文件名
```

图表 11 指令介绍

gcc filename.c -o executable_file: 编译单个源文件 filename.c 生成可执行文件

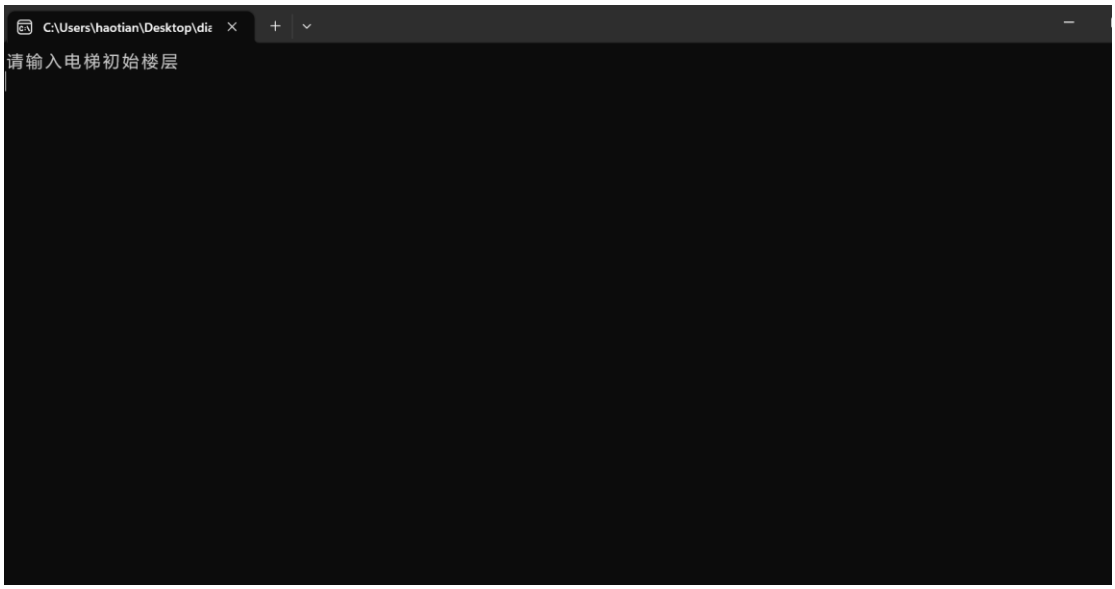
gcc file1.c file2.c -o executable_file: 编译多个源文件 file1.c 和 file2.c 生成可执行文件
gcc -c filename.c: 只编译不链接,生成对象文件 filename.o。
gcc filename.o -o executable_file: 链接单个对象文件 filename.o 生成可执行文件
gcc filename.o otherfile.o -o executable_file: 链接多个对象文件 filename.o 和 otherfile.o 生成可执行文件
gcc -Wall filename.c -o executable_file: 开启所有警告选项编译。
gcc -g filename.c -o executable_file: 包含调试信息编译。
gcc -static filename.c -o executable_file: 静态链接生成可执行文件。
gcc -shared filename.c -o libfilename.so: 生成动态共享库。
gcc -Iinclude_path filename.c -o executable_file: 添加包含路径 include_path。

尝试实践



图表 12 成功

打开 exe



图表 13 内容展示

成功

补充

后面运行时发现输入后消失，上网搜索后了解到是运行结束自动退出导致的，我的解

决方法: include<conio.h> + getch();

总结

掌握 c 语言基本语法。了解程序从源文件(.c)到可执行程序(.exe)的过程，并学习 gcc 等编译器的安装配置以及基本指令。并在 gcc 的安装过程中了解到非二进制安装包的离线安装及配置过程。

1-2 学习文档

C 语言基础语法：文件操作【全集】学了这么久还不会操作文件？一次带你掌握文件的读写拷贝等一系列操作！_哔哩哔哩_bilibili

C 语言文件读写操作（详解）_读文件和写文件 c 语言_放码过来呀!!! 的博客-CSDN 博客

根据以上资料以及还有一些搜索结果完成任务。

1-3 学习文档

算法选择

参考资料：大实话：等电梯的时候，90%的程序员都想过调度算法 - 知乎 (zhihu.com)
结合自身代码水平以及实际背景，选择 look 算法进行实现。

代码问题

Look 算法的代码实现中的问题：

若电梯下行时接到一个目标楼层在此时电梯上面的乘客，电梯不应向上走而是继续下行直到不需要再向下。解决方法：给电梯赋一个状态（direction）来控制电梯上行还是下行，直到没有需求之后再改变状态。

求最小值函数中的数组越界以及死循环：

死循环：1、将返回值初始化为 0，结果时若数组的第一个元素为 0 时便找不到最小值

2、再找数组中最小值时，由于初始化将数组所有元素赋值为 0，于是在通过判断电梯层数（不可能笔 0 小）是否小于需求的最小值从而判断是否需要向上时，结果是电梯永远处于向下的状态。

解决：初始化为-1，并且跳过数组中等于零的数。

数组越界：最开始解决死循环的方法是将返回值初始化为-1，然后再让返回值等于数组中不为零的最小值的下标，结果就是数组全为 0 的时候数组越界。但由于数组全为 0 有可能使函数 quit，于是代码出现时而正常时而报错的情况。

解决：每次使用最小值函数使先判断返回值是否为-1.

【解决方案】如果提示中的变量是指针类型变量，则大概率的就是数组访问越界，需要反复检查数组下标访问的合法性

关于报错：Run-Time Check Failure #2 - Stack around the variable 'xxx' was corrupted 的解决方式_run time check failure 2_Adam Xi 的博客-CSDN 博客

规范命名

交流时发现代码中命名情况过于混乱，因此学习命名规则并修改代码中的“magic number”。

驼峰命名法 匈牙利命名法 帕斯卡命名法

基本原则：

命名要用全称。中英文混合要用下划线分开。缩写要注释。变量不能用单字符命名（除非是用来做循环的局部变量）。命名规范统一。用正反义词（up down）来命名对应的函数

变量命名用匈牙利法则

(1)变量的命名规则要求用“匈牙利法则”。
即开头字母用变量的类型，其余部分用变量的英文意思、英文的缩写、中文全拼或中文全拼的缩写,要求单词的第一个字母应大写。
即： 变量名=变量类型+变量的英文意思(或英文缩写、中文全拼、中文全拼缩写)
对非通用的变量，在定义时加入注释说明，变量定义尽量可能放在函数的开始处：
bool 用b开头 b标志寄存器
int 用i开头 iCount
short int 用n开头 nStepCount
long int 用l开头 lSum
char 用c开头 cCount
unsigned char 用by开头
float 用f开头 fAvg
double 用d开头 dDeta
unsigned int(WORD) 用w开头 wCount
unsigned long int(DWORD) 用dw开头 dwBroad
字符串 用s开头 sFileName
用0结尾的字符串 用sz开头 szFileName

图表 14 命名规则

一重指针用 p+变量类型前缀+命名。
全局变量用 g_开头。
对常量(包括错误的编码)命名，要求常量名用大写，单词间用_连接。
对 const 的变量加 c_。
参考资料 C 语言：命名规范 - 知乎 (zhihu.com)

优化

对于寻找数组最小值下标的函数的优化：

原来的代码是

```
int indexOfMinimum(int* KnownDestation, int length) {//要找出非零的最小值
    int iNonZeroCount = 0;
    for (int i = 0; i < length; i++) {
        if (KnownDestation[i] != 0) {
            iNonZeroCount++;
        }
    }
    if (iNonZeroCount != 0) {//防止数组全为零
        int ret;
        for (int x = 0; x < length; x++) {if (KnownDestation[x] != 0)ret = x;}//防止下面做下标的ret没有值
        for (int t = 0; t < iNonZeroCount; t++) {
            if (KnownDestation[t] != 0 && KnownDestation[t] < KnownDestation[ret]) {
                ret = t;
            }
            return ret;
        }
    }
    else return -1;//注意使用时要判断，不然会引发越界
}
```

图表 15 初始代码

这个函数中用了 3 个循环，后来知道有更简单的方法

```
int indexOfMinimum(int* KnownDestation,int length) {//要找出非零的最小值
    int iNonZeroCount = 0;
    for (int i = 0;i < length; i++) {
        if (KnownDestation[i] != 0) {
            iNonZeroCount++;
        }
    }
    if (iNonZeroCount != 0) {//防止数组全为零
        int ret;
        //for (int x = 0; x < length; x++) {if (KnownDestation[x] != 0)ret = x;}有更简单的初始化
        for (int t = 0;t < iNonZeroCount;t++) {
            if (KnownDestation[t] == 0) {
                continue;
            }
            else {
                ret = t;
            }//防止ret没有值
            if (KnownDestation[t] != 0 && KnownDestation[t] < KnownDestation[ret]) {
                ret = t;
            }
            return ret;
        }
    }
    else return -1;//注意使用时要判断，不然会引发越界
}
```

图表 16 优化一

后记：最终优化版本。该代码相比于之前的优点在于 ret 只赋值了一次。

```
int indexOfMinimum(int* array,int length) {//要找出非零的最小值
    int iNonZeroCount = 0;
    for (int i = 0;i < length; i++) {
        if (array[i] != 0) {
            iNonZeroCount++;
        }
    }
    if (iNonZeroCount != 0) {//防止数组全为零
        int ret=-1;
        //for (int x = 0; x < length; x++) {if (KnownDestation[x] != 0)ret = x;}有更简单的初始化
        for (int t = 0;t < iNonZeroCount;t++) {
            if (array[t] == 0) {
                continue;
            }
            if(ret==-1) {
                ret = t;
            }//防止ret没有值
            if (array[t] != 0 && array[t] < array[ret]) {
                ret = t;
            }
            return ret;
        }
    }
    else return -1;//注意使用时要判断，不然会引发越界
}
```

图表 17 最终版本

总结

真正实践了之前学习到的 c 语言代码，在修改代码解决 bug 的过程中提高写代码的熟练度。并且学习了代码中变量，函数等的命名，使代码规范化，提高可读性。

2-1 学习文档

代码修改

赋值的初始方法：设置 intervaltime 和 waitingfloor 数组，每当电梯运动一次，intervaltime 数组中不为零的值就减一，当 intervaltime==0 时将 waitingfloor 数组中对应的值赋给 requestfloor 数组，并清零 waitingfloor。

Bug：当循环进行第二遍的时候会将 requestfloor 数组的数据重新赋为零，清空了原有数据。

解决：使 intervaltime 数组减一重复至值为-1 才停。

当电梯中无人但未来将有人呼叫时程序出现 bug，原有的向上向下数组不能满足该场景，因此要设置第三种状态——停靠。

方法：request 和 destation 全部为零时将状态设为停靠。

Bug：若电梯开始时没人呼叫，则 request 和 destation 数组全部是零，此时 isElevatorUp 函数异常。原因是最大值为零，电梯一定比最大值大，导致方向一定向下。

解决：判断输入的数组内容是否全部为零,若是，则停靠。后面了解到这是所谓异常保护。

总结

意识到写代码不能钻营取巧，否则会使代码的拓展性大大降低，并且出现 bug 时提高修改难度。修改 bug 时发现几个问题经常出现（如数组的越界，数组初始化的 0 对于运算的影响），以后写代码时会更加注意，尽量更加规范。