

# 计算概论B

## C语言代码规范

王 昭

北京大学信息科学技术学院

[wangzhao@pku.edu.cn](mailto:wangzhao@pku.edu.cn)



# 可读性

- 软件的规模越来越大，一个系统通常需要**几代程序员**来开发维护
- 程序员的抱怨：这些代码简直无法维护，还不如推到重写？
- 代码的文体属于说明文或记叙文，着眼于把事请说清楚
- 白居易：“每成篇，必令其家老妪读之，问解则录”

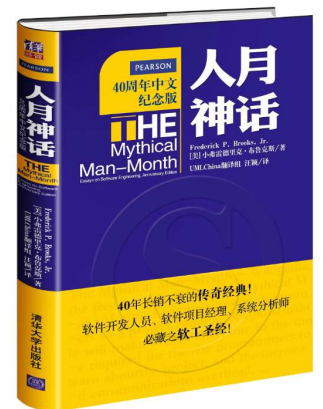


# C代码规范的目标

- 缩短开发时间
- 便于维护
- 避免程序的漏洞
- 所有软件的开发项目中，开发成本占总成本的比例为**20%**，维护成本占**67%**。

——《人月神话》，Frederick P. Brooks

- Frederick P. Brooks, IBM 360系统之父”，Brooks博士创立了[北卡罗莱纳大学](#)的计算机科学系，并在1964-1984年期间担任系主任。




# C语言代码规范

- 一个好的程序，不仅要**算法正确**、**效率高**，而且还应该**可读性好**。所谓程序的**可读性**，就是程序是否能让人容易读懂。在开发实践中，许多情况下可读性与代码效率同等重要。
- **标识符命名**、**书写格式**、**注释**三个方面加以注意



# C语言代码规范

- 标识符命名 
- 程序的书写格式
- 注释
- 一些需要注意的问题



# 标识符命名注意事项-1

1. 标识符名称包括函数名、常量名、变量名等。这些名字应该能反映它所代表的实际东西，具有一定的意义，使其能够见名知义，对于函数名说明其在“做什么”，对于变量说明“是什么”，有助于对程序功能的理解。

□用词准确，

- 任务：assignment\mission\duties\task
- 有些场合也可以使用拼音，绝密、机密、秘密  
JueMi\JiMi\Mimi



# 标识符命名注意事项-2

2. 复合词中每个单词的第一个字母大写。也可以在复合词里可以用下划线隔开每个词。

- `myFirstName`      **//驼峰式命名**

- `my_first_name`      `ARRAY_SIZE`      **//短横线式**

□ 骆驼式命名法（**Camel-Case**）又称**驼峰式命名法**，骆驼式命名法就是当变量名或函数名是由一个或多个单词连结在一起，而构成的唯一识别字时，第一个单词以小写字母开始；从第二个单词开始以后的每个单词的首字母都采用大写字母，例如：

**myFirstName**、**myLastName**，这样的变量名看上去就像骆驼峰一样此起彼伏，故得名。



# 标识符命名注意事项-3

3. 所有宏定义、枚举常量和**const**常变量，用**大写字母命名**。

```
#define ARRAY_SIZE 24
```

4. **typedef**定义的类型名用大写字母表示。

```
typedef int INTEGER;
```

5. 为全局变量取长的，描述信息多的名字，为局部变量取稍短的名字。
6. 名字太长时可以适当**采用单词的缩写**。比如 单词**Number**，如果在某个变量里缩写成了：**int nDoorNum**；那么最好包含**Number**单词的变量都缩写成 **Num**。





# 标识符命名注意事项-4

- 7. 循环变量可采用*i*, *j*, *k*等, 不受上述规则限制
- 8. 对结构体内的变量命名, 遵循变量的具体含义命名原则
- 9. 通常, 函数的命名也是以能表达函数的动作意义为原则的, 一般是由动词打头, 然后跟上表示动作对象的名词, 各单词的首字母可以大写。 **createNullList\_seq**
- 10. 对于返回值为真或假的函数, 加“**is**”前缀如:

```
int isalpha();    // C语言标准库函数
```



# 美程序员枪击4同事，竟因代码不写注释？产品汪薪水高过码农



AI财经社

发布时间：18-09-23 19:44 | 北京小犀快跑科技有限公司

现实版“杀一名程序员祭天”在现实中上演。

9月19日，一名程序员在美国某办公楼向4名同事开枪，导致一人情况危机，两人伤情严重，一人被子弹擦伤。目前，凶手已死，身份被警方查明。

目前，码农持枪杀人的动机仍然是个谜。有人猜测道：“同事不写注释，不遵循驼峰命名，括号换行，最主要还天天git push -f等因素”激怒了这名行凶者。

有网友评论道：“别看平时默默敲键盘，反手一枪谁也受不了。沟通，多沟通，感情深才能代码真。”

## 作者最新文章

北京文化回应财务造假：娄晓峰涉嫌挪用资金罪出逃，已立案调查

《流浪地球》《战狼2》出品方北京文化董事长宋歌被举报财务造假



# C语言代码规范

- 标识符命名
- 程序的书写格式 ←
- 注释
- 一些需要注意的问题



# 程序的书写格式

- 应该特别注意程序的书写格式，让它的形式反映出其内在的意义结构。好的格式能使程序结构一目了然，帮助你和别人理解它，帮助你的思维，也帮助你发现程序中不正常的地方，使程序中的错误更容易被发现。
- 程序编写首先应考虑清晰性，不要刻意追求技巧性而使得程序难以理解。
- 利用集成开发环境（**IDE**）或者其他程序编辑器的功能，可以很方便地维护好程序的良好格式。请注意下面这几个键，在写程序中应该经常用到它们：**Enter**键（换一行），**Tab**键（将输入光标移到下一个对齐位置——进入新的一个层次），**Backspace**键（回到前一个对齐位置——退到外面的一个层次）。



# 间隔相关-I

- 为保证语句结构的清晰和程序的可读性，在编写软件程序时应注意以下几个方面的问题：
  - ① 在一行内只写一条语句，并采用空格、空行和移行保证清楚的视觉效果。
  - ② 文件之中不得存在无规则的空行，比如说连续十个空行。
  - ③ 一般来讲函数与函数之间的空行为2-3行；
  - ④ 在函数体内部，在逻辑上独立的两个函数块可适当空行，一般为1-2行。



# 间隔相关-II

⑤ 赋值符号“=”或其他各类运算符两边各加一个空格，或者不加空格，或者加若干个空格使代码块保持对齐

□运算符两边各加一个空格。

```
total = int1 + int2;
```

□运算符两边不加空格。

```
total=int1+int2;
```

□运算符两边加若干空格。

```
total          = int1 + int2 ;  
moth_salary    = 0 ;  
day_salary     = 0 ;
```



## 间隔相关-III

- ⑥ 行尾的“;”前加一个空格，或者不加

```
total = int1 + int2 ; //分号前有一个空格
```

```
total = int1 + int2; //分号前没有空格
```

- ⑦ 定义函数时，函数的名称和后面的小括号“(”之间加一个空格，或者不加

```
int function (int x ) //小括号前有空格
```

```
int function(int x) //小括号无有空格
```



# 间隔相关-IV

- 插入太多空格会使代码不易理解

⑧ 不要在一元运算符（++、--）与操作数之间插入空隔

`++P` ✓

`++ P` //不建议插入空格



⑨ 行尾的“;”前不要插入多个空格

`Num1 = Num2`

;





## 间隔相关-V

- ⑩ 罗列函数参数时，需要用逗号区分各参数，逗号后插入一个空格，以更好地区分各参数，但不要插入太多空格。

```
scanf("%d %d %d",&num1,&num2,&count,&county);
```

```
scanf("%d %d %d",&num1, &num2, &count, &county);
```



```
printf("总计%d      合计%d", totalSum,      netSum ) ;
```



# 缩进相关-I

- ① 人们常用的格式形式是：逻辑上属于同一个层次的互相对齐；逻辑上属于内部层次的推到下一个对齐位置。每一个嵌套的逻辑层次，使用一个**TAB**缩进（可以设定为**4**个空格）。

```
# include <stdio.h>
int gcd(int x,int y);
int main()
{
    int a,b,c;
    scanf("%d,%d",&a,&b);
    c=gcd(a,b);
    printf("%d\n",c);
}

int gcd(int x,int y)
{
    int z;
    while (y)
    {
        z=x%y;
        x=y;
        y=z;
        printf("%d,%d\n",y,z);
    }
    return (x);
}
```

# 不好的书写格式

- ❑ 代码块没有缩进时  
结构混乱，难以理解

```
# include <stdio.h>
int gcd(int x,int y);
int main()
{
int a,b,c;
scanf("%d,%d",&a,&b);
c=gcd(a,b);
printf("%d\n",c);
}
int gcd(int x,int y)
{
int z;
while (y)
{
z=x%y;x=y;y=z;
printf("%d,%d\n",y,z);
}
return(x);
}
```



## 缩进相关-II

② **if**、**for**、**do...while**、**while**、**switch...case**等语句的括号，函数的括号可以有如下风格

□左括号“{”，在第一行的最右边。

```
for ( int i=0;i<10;i++ ){  
    .....  
}
```

```
int function ( int x ){  
    .....  
}
```

□左括号“{”，另起一行放在最左边，并与上一行的行首对齐。

```
for ( int i=0;i<10;i++ )  
{  
    .....  
}
```

```
int function( int x )  
{  
    .....  
}
```



# 缩进相关-III

## ③ 不要毫无意义的缩进

```
int main(){  
    printf("不要毫无意义的缩进\n");  
    return 0;  
}
```



## ④ 不要使用凸出形式的代码

```
    int  (num1 == num2){  
num1 = 10;  
num2 = 100;  
    }
```




# 行宽与折行

- ① **行宽与折行**：行不要太长，不能超过显示区域。以免阅读不便。太长则应折行。折行最好发生在运算符前面，不要发生在运算符后面。如

```
if( Condition1() && Condition2()  
    && Condition3() ) {  
}
```



# C语言代码规范

- 标识符命名
- 程序的书写格式
- 注释 
- 一些需要注意的问题



# 数据和函数说明

- 数据说明次序应当规范化，使数据属性容易查找，也有利于测试、排错和维护。说明的先后次序应固定，应按逻辑功能排序，逻辑功能块内建议采用下列顺序：**整型说明、实型说明、字符说明、逻辑量说明**。
- 如果设计了一个复杂的数据结构，应当通过注释对其变量的含义、用途进行说明





# 程序注释

- 程序注释是程序员与日后的程序读者之间通信的重要手段之一，注释分为文件注释、函数注释和功能注释。正规程序的注释应注意：注释行的数量占到整个源程序的1/3到1/2。
- 文件注释位于整个源程序的最开始部分，注释后空两行开始程序正文。它包括：
  - 程序标题。
  - 目的、功能说明。
  - 文件作者、最后修改日期等说明。



# 文件注释

- 例:

```
/******空一行)
```

**标题:** chengjiabi.cpp

**功能:** 从12枚银币中判断假币.

**说明:**

根据三次称量的结果, 确定哪一个是假币, 并指出其轻重。

**当前版本:** x.x

**修改信息:** 2004.08.05 Anni, Initial Version

2004.08.20 Tom, Bug xxxx fixed

```
*****/
```

(空2行, 开始程序正文)



# 函数注释

- 函数注释通常置于每函数或过程的开头部分，它应当给出函数或过程的整体说明，对于理解程序本身具有引导作用。一般包括如下条目：
  - 模块标题。
  - 有关本模块功能和目的的说明。
  - 调用格式
  - 接口说明：包括输入、输出、返回值、异常。
  - 算法。如果模块中采用了一些复杂的算法。



# 函数注释示例

- （注释开头与上一函数最后一行间隔两行）

```
/******
```

**标题：** `isHeavy`

**功能：** 判断银币x是否是重假币，假设银币x是重假币，看输入的条件是否满足

**格式：** `int isHeavy(char x)`

**输入：** 硬币x

**输出：** 如果x是重假币，返回1；如果x不是重假币，返回0

**返回值：** 1是重假币，0不是重假币

```
*****/
```

（ 注释后直接开始程序正文，不空行。）



# 功能性注释

- 功能性注释嵌在源程序体中，用于描述其后的语句或程序段做什么工作，也就是解释下面要做什么，或是执行了下面的语句会怎么样。而不要解释下面怎么做，因为解释怎么做常常与程序本身是重复的。

例：

```
/*把 amount 加到 total中*/
```

```
total = amount + total;
```

这样的注释仅仅是重复了下面的程序，对于理解它的工作并没有什么作用。而下面的注释，有助于读者理解。

```
/*将每月的销售额amount加到年销售额total中*/
```

```
total = amount + total;
```



# 在==运算符旁添加注释

- 比较是否相等的运算符`==`，很容易与赋值运算符`=`混淆，可以在以下例句旁添加注释

```
if ( isThis == TRUE ){ ..... } //如果isThis的值为真
```



# 在大括号闭合处添加注释

- 随着程序、函数、条件表达式长度的增加，判断代码块范围会越来越困难，也就是，看到大括号的闭合符}，很难判断它表示的是哪个代码块的结束。应该，用**end if**、**end while**、**end main**这种以**end**开头的注释。

```
int main(){  
    if () {  
        if() {  
            .....  
        } //end inner if  
        .....  
    } // end outer if  
} //end main
```



# C语言代码规范

- 标识符命名
- 程序的书写格式
- 注释
- 一些需要注意的问题 ←





# C语言中可能出现的问题

- 除数为0会出现问题
- 使用空指针会出现问题
- 可能再次访问已释放的动态分配内存
- 字符串末尾必须加空字符 ‘\0’
- 向超出数组范围的数组元素赋值会出现问题
- 数据类型不同的变量间可以进行运算
- 数据类型不同的指针访问内存会出现意向不到的结果
- 使用指针变量前必须初始化



# 一些需要注意的问题-1

- ① 随时注意表达式计算过程和类型。注意运算符的**优先级**和**结合顺序**，不同类型的运算对象将怎样转换，运算的结果是什么类型的，等等。在必要的时候加上括号或显式的类型强制转换。
- ② **C**语言的运算符很多，优先级定义也不尽合理，很难完全记清楚，因此要特别注意。需要时查一查（不要怕麻烦，相关网页有运算符表），或者直接按照自己的需要加上几个括号。  
稍复杂的表达式中要积极使用括号，以免优先级理解上的混乱以及二义性。

```
n = k +++ j;    //不好  
n = ( k ++ ) + j;  //好一点
```



## 一些需要注意的问题-2

- ③ 绝不去写依赖于运算对象求值顺序的表达式。对于普通二元运算符的运算对象，函数调用的各个实际参数，C语言都没有规定特定求值顺序。因此，我们不应该写那种依赖于特定求值顺序的表达式，因为不能保证它一定得到什么结果。

```
#include <stdio.h>

int main()
{
    int i;
    i=3;
    printf("%d ,%d\n",i,++i);
    printf("%d ,%d\n",i,i++);
    return 0;
}

j=i++;
printf("%d ,%d\n",i,j);
```

4, 4

5, 4



## 一些需要注意的问题-3

- ④ 总注意检查数组的界限和字符串（也以数组的方式存放）的结束。C语言内部根本不检查数组下标表达式的取值是否在合法范围内，也不检查指向数组元素的指针是不是移出了数组的合法区域。写程序的人需要自己保证对数组使用的合法性。越界访问可能造成灾难性的后果。

例：在写处理数组的函数时一般应该有一个范围参数；处理字符串时总检查是否遇到空字符 ‘\0’。

- ⑤ 绝不对空指针或者悬空的指针做间接访问。这种访问的后果不可预料，可能造成系统的破坏，也可能造成操作系统发现这个程序执行非法操作而强制将它终止。



# 一些需要注意的问题-4

⑥ 对于所有通过返回值报告运行情况或者出错信息的库函数，都应该检查其执行是否正常完成。如果库函数没有完成操作（可能因为各种原因），随后的操作有可能就是非法的。这种错误也可能在程序运行中隐藏很长时间，到很后来才暴露出来，检查错误非常困难。

- 参考书：

- 《程序设计实践》，（The Practice Of Programming, Brian W. Kernighan & Bob Pike 1999）。机械工业出版社2000。



# 一些好的编程习惯-1

1. 尽量不要用立即数，而用**#define**定义成常量，以便以后修改。例如：

```
#define MAX_STUDENTS 20
```

```
struct SStudent aStudents [MAX_STUDENTS];
```

比

```
struct SStudent aStudents [20];
```

好。



# 一些好的编程习惯-2

2. 使用**sizeof()**宏，不直接使用变量所占字节数的数值。如：  
应该写：

```
int Student;  
  
for( j = 0; j < 100; j++ )  
    fwrite( aStu, sizeof(student), 7, fpSrc);
```

不应该写：

```
for( j = 0; j < 100; j++ )  
    fwrite( aStu, 4, 7, fpSrc);
```



# 一些好的编程习惯-3

3. 不很容易理解的表达式应分几行写：

`n = ( k ++ ) + j;` 应该写成：

`n = k + j;`

`k ++;`

4. 单个函数的程序行数最好不要超过**100行**（两个屏幕高）。

5. 尽量使用标准库函数和公共函数，除非题目要求。

6. 不要随意定义全局变量，全局变量会增大理解难度，除非需要。

7. 保持注释与代码完全一致，改了代码别忘改注释。

8. 循环、分支层次最好不要超过五层。





