



第6章 函数

——代码风格



哈尔滨工业大学

苏小红

sxh@hit.edu.cn

基本的代码规范

- **Basic rules and guidelines of Coding Style**
 - * 程序版式
 - * 程序注释
 - * 命名规则
- **追求**
 - * 清晰、整洁、美观、一目了然
 - * 容易阅读，容易测试

程序版式

■ 不良的风格

```
#include <stdio.h>
#include <math.h>
main()
{int i;
for (i=2;i<100;i++)
{if(isprime(i))
printf("%d\t",i); }
}
int isprime(int n)
{int k,i;
if (n == 1) return 0;
k=sqrt((double)n);
for (i=2;i<=k;i++)
{if(n%i==0) return 0;}
return 1;
}
```

■ 良好的风格

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

int main()
{
    int i;

    for (i=2; i<100; i++)
    {
        if (isprime(i))
        {
            printf("%d\t", i);
        }
    }
    return 0;
}
```

```
int isprime(int n)
{
    int k, i;

    if (n == 1) return 0;

    k = (int)sqrt((double)n);

    for (i=2; i<=k; i++)
    {
        if (n % i == 0)
        {
            return 0;
        }
    }
    return 1;
}
```

程序版式

■ 对齐（Alignment）与缩进（indent）

——保证代码整洁、层次清晰的主要手段

- * 位于同一层{和}之内的代码在{右边数格处左对齐
- * 同层次的代码在同层次的缩进层上
 - * 现在的许多开发环境、编辑软件都支持“自动缩进”
 - * VC中有自动整理格式功能(ALT+F8)
- * 一般用设置为4个空格的Tab键缩进，不用空格缩进

```
int isprime(int n)
{
    int k, i;

    if (n == 1) return 0;

    k = (int)sqrt((double)n);

    for (i=2; i<=k; i++)
    {
        if (n % i == 0)
        {
            return 0;
        }
    }
    return 1;
}
```

<pre>void Function(int x) { ... // program code }</pre>	<pre>void Function(int x){ ... // program code }</pre>
<pre>if (condition) { ... // program code } else { ... // program code }</pre>	<pre>if (condition){ ... // program code } else { ... // program code }</pre>
<pre>for (initial; condition; update) { ... // program code }</pre>	<pre>for (initial; condition; update){ ... // program code }</pre>
<pre>while (condition) { ... // program code }</pre>	<pre>while (condition){ ... // program code }</pre>
<pre>{ ... { ...//嵌套的 {} 用缩进对齐 } ... }</pre>	



程序版式

■ 变量的对齐规则

* 数据类型 + n个TAB + 变量名 + [n个TAB] + = +[初始化值] ;

* 例

- char name[20] ;
- char addr[30] ;
- char sex = 'F' ;
- int age = 20 ;
- float score = 90 ;

程序版式

■ 空行——分隔程序段落的作用

- * 在每个函数定义结束之后加空行
- * 在一个函数体内，相邻两组逻辑上密切相关的语句块之间加空行，语句块内不加空行

```
// 空行
void Function1(...)
{
    ...
}
// 空行
void Function2(...)
{
    ...
}
// 空行
void Function3(...)
{
    ...
}
```

```
// 空行
while (condition)
{
    statement1;
    // 空行
    if (condition)
    {
        statement2;
    }
    else
    {
        statement3;
    }
    // 空行
    statement4;
}
```

程序版式

■ 代码行内的空格——增强单行清晰度

- * 关键字之后加空格，但函数名之后不加空格
- * 赋值、算术、关系、逻辑等二元运算符前后各加一空格
- * 但一元运算符以及[] . ->前后不加空格
 - `sum = sum + term;`
- * (向后紧跟,) , ;向前紧跟, 紧跟处不留空格, , ;后留一个空格
 - `Function(x, y, z)`
 - `for (initialization; condition; update)`
- * 对表达式较长的for和if语句，为了紧凑可在适当地方去掉一些空格
 - `for (i=0; i<10; i++)`
 - `if ((a+b>c) && (b+c>a) && (c+a>b))`

程序版式

■ 代码行

* 一行只写一条语句，便于测试

```
x = a + b;
```

* 一行只写一个变量，便于写注释

```
y = c + d;
```

```
z = e + f;
```

- `int width;` //宽度

- `int height;` //高度

- `int depth;` //深度

```
x = a + b; y = c + d; z = e + f;
```



- `int width, height, depth;` //宽度高度深度（不建议）

* 尽可能在定义变量的同时，初始化该变量

- `int sum = 0;`

* `if`、`for`、`while`、`do`等语句各占一行，便于测试和维护

```
if (width < height)
```

```
{
```

```
    DoSomething(); //执行语句无论有几条都用{和}将其包含在内
```

```
}
```

程序版式

■ 长行拆分

- * 代码行不宜过长，应控制在10个单词或70~80个字符以内
 - *Studies show that up to ten-word text widths are optimal for eye tracking*
- * 实在太长时要在适当位置拆分，拆分出的新行要进行适当缩进

```
if ((veryLongVar1 >= veryLongVar2)
    &&(veryLongVar3 >= veryLongVar4))
{
    DoSomething();
}
double FunctionName(double variablename1,
                    double variablename2);
for (very_longer_initialization;
     very_longer_condition;
     very_longer_update)
{
    DoSomething();
}
```

标识符命名规则

- 按照执行级别分为：

- * 共性规则

- 必须执行

- * 简化规则

- 建议采用

- * 可选规则

- 灵活运用

标识符命名的共性规则

- 有意义，直观可拼读，见名知意，不必解码
- 最好采用英文单词或其组合，切忌用汉语拼音，尽量避免出现数字编号
- 不要出现仅靠大小写区分的相似的标识符
- 不要出现名字完全相同的局部变量和全局变量
- 用正确的反义词组命名具有互斥意义的变量或相反动作的函数

```
* int minValue;  
* int maxValue;  
* int GetValue(...);  
* int SetValue(...);
```

标识符命名的共性规则

- 尽量与所采用的操作系统或开发工具的风格保持一致
 - * 在Linux/Unix平台
 - 习惯用“小写加下划线”
 - `function_name`
 - `variable_Name`
 - * Windows风格
 - 大小写混排的单词组合而成
 - `FunctionName`
 - `variableName`

Windows应用程序命名规则

- Microsoft公司的Hungarian Notation

- 主要思想

- * 在变量和函数名前加上前缀，用于标识变量的数据类型
- * [限定范围的前缀] + [数据类型前缀] + [有意义的英文单词]
- * 限定范围的前缀
 - 静态变量前加前缀s_，表示static
 - 全局变量前加前缀g_，表示global
 - 默认情况为局部变量
- * 数据类型前缀
 - ch 字符变量前缀
 - i 整型变量前缀
 - f 实型变量前缀
 - p 指针变量前缀

灵活运用的命名规则

- 限定范围的前缀与数据类型前缀可要可不要
- 无特殊意义的循环变量可以直接定义成*i, j, k*等单字母变量

- `int i, j, k;`

- `float x, y, z;`

* 若采用匈牙利命名规则，则应写成

- `int iI, iJ, iK; //前缀i表示int类型`

- `float fX, fY, fZ; //前缀f表示float类型`

简化的Windows应用程序命名规则

■ 变量名形式

- * 小写字母开头，“名词”或者“形容词+名词”
 - `oldValue, newValue`

■ 函数名形式

- * 大写字母开头，“动词”或者“动词+名词”（动宾词组）
 - `GetValue(), SetValue()`

■ 宏和const常量全用大写字母，并用下划线分割单词

- * `#define ARRAY_LEN 10`
- * `const int MAX_LEN = 100;`

注释规范

■ 写注释给谁看？

- * 给自己看，使自己的设计思路得以连贯
- * 给继任者看，使其能够接替自己的工作

■ 写注释的最重要的功效在于传承

- * 要站在继任者的角度写
- * 简单明了、准确易懂、防止二义性
- * 让继任者可以轻松阅读、复用、修改自己的代码
- * 让继任者轻松辨别出哪些使自己写的，哪些是别人写的

不好的注释

```
/*以二进制只读方式打开文件并判断打开是否成功*/
if ((fin = fopen("cat.pic","rb") == NULL)
{
    puts("打开文件cat.pic失败");/*如果打开失败，则显示错误信息*/
    return -1;                    /*返回-1*/
}
.....
/*从图像的第1行到第400行循环*/
for (i=0; i<400; i++)
    /*从图像的第1列到第400列循环*/
    for (j=0; j<400; j++)
    {.....
        /*按 $Y = 0.299R + 0.587G + 0.114B$ 计算灰度值*/
        y = (299 * r + 587 * g + 114 * b) / 1000;
        .....
    }
.....
fclose(fin);    /*关闭文件*/
```

* 注释不是白话文翻译

Don't write comments that repeat the code

* 注释不是教科书

* 注释不是标准库函数参考手册

* 注释不是越多越好

好的注释

```
/*打开输入文件后判断文件长度是否符合格式要求*/
if ((fin = fopen("cat.pic","rb") == NULL)
{
    puts("打开文件cat.pic失败");
    return -1;
}
.....
/*
 * 利用RGB颜色空间到YUV颜色空间的变换公式实现彩色图像到灰度图像的转换
 * 公式为 $Y = 0.299R + 0.587G + 0.114B$ 
 */
for (i=0; i<400; i++)
    for (j=0; j<400; j++)
    {.....
        y = (299 * r + 587 * g + 114 * b) / 1000;
        .....
    }
.....
fclose(fin);
```

* 不写做了什么，写想做什么

*Do write illuminating
comments that explain
approach and rationale*

* 边写代码边注释

* 修改代码同时修改注释

在哪些地方写注释？

* 在重要的文件首部

- 文件名 + 功能说明 + [作者] + [版本] + [版权声明] + [日期]

* 在用户自定义函数前，对函数接口进行说明

- 函数功能 + 入口参数 + 出口参数 + 返回值（包括出错处理）

C风格

```
/* **** */
/*功能描述： 本函数用于实现xxx功能，目的是： */
/*入口参数： 参数xx，表示 */
/*出口参数： 参数xx，表示 */
/*返回值： 返回xx值，当返回xx值时，表示 */
/* **** */
/*
功能描述： 本函数用于实现xxx功能，目的是：
入口参数： 参数xx，表示
出口参数： 参数xx，表示
返回值： 返回xx值，当返回xx值时，表示
*/
```

C++风格

```
* //////////////////////////////////////
//功能描述： 本函数用于实现xxx功能，目的是：
//入口参数： 参数xx，表示
//出口参数： 参数xx，表示
//返回值： 返回xx值，当返回xx值时，表示
////////////////////////////////////
```

在哪些地方写注释？

- * 在一些重要的语句块上方
 - 对代码的功能、原理进行解释说明
- * 在一些重要的语句行右方
 - 定义一些非通用的变量，函数调用，较长的、多重嵌套的语句块结束处

```
/*  
 *C风格  
 */
```

```
/*  
*****  
/*下面代码是用来接收网络数据，其原理为*/  
/*          C风格          */  
*****  
*/
```

```
////////////////////////////////////  
//          Visual C++风格          //  
////////////////////////////////////
```

```
/* C风格 */
```

```
//Visual C++风格
```

```
i = j + 1; //代码行右方的注释
```

```
//代码行之上的注释
```

```
i = j + 1;
```

在哪些地方写注释？

- * 在修改的代码行旁边加注释
- * 在调试程序中对暂不使用的语句通常可先用注释符括起来，使编译器跳过这些语句

■ 可灵活运用的一些规则

- * 注释可长可短，但应画龙点睛，重点加在语义转折处
- * 简单的函数可以用一句话简单说明

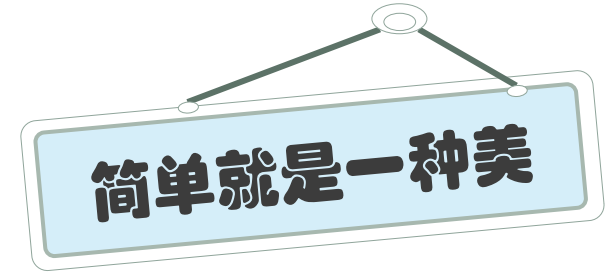
//两数交换

```
void Swap(int *x, int *y)
```

- * 内部使用的函数可以简单注释，供别人使用的函数必须严格注释，特别是入口参数和出口参数

Other rules and guidelines

- Avoid long functions.
- Avoid deep nesting.
 - * *KISS (Keep It Simple Software)*
 - *Correct is better than fast*
 - *Simple is better than complex*
 - *Clear is better than cute*
 - *Safe is better than insecure*
 - *Short is better than long*
 - *Flat is better than deep*



```
b = (a++) + (a++) + (a++);
```

```
printf("a++ = %d, ++a = %d, a = %d", a++, ++a, a);
```

```
Function(a++, ++a);
```

讨论

- 你对代码风格怎么看？你认为它是可有可无的吗？代码风格不影响程序的运行结果，可以不用特别关注，还是在一开始学习编程时就应养成良好的编程习惯？你能举出哪些不好的或者好的编程习惯？



