

C 语言编程规范

北京创毅视讯科技有限公司 驱动组

Revision 0.1

2010-3-13

目 录

1. 版本历史.....	1
2. 程序排版.....	2
2.1. 缩进与对齐.....	2
2.2. 左花括号“{”的位置	2
2.3. 空格.....	3
2.3.1. 需要有空格的地方	3
2.3.2. 不要留空格的地方	4
2.4. 空行.....	4
2.5. 代码行	5
3. 注释.....	6
3.1. doxygen 风格的注释	6
3.2. 关于注释的一些建议	8
4. 命名规则.....	8
5. 变量.....	8
6. 函数.....	8
7. 其它.....	9
8. 参考资料.....	9

1. 版本历史

版本号	时间	作者	备注
V0.1	2010-3-13	李正明	李正明起草

2. 程序排版

2.1. 缩进与对齐

缩进代码统一采用 4 格缩进，4 个空格或 Tab 字符，若采用 Tab 字符，应调整编辑器将 Tab 宽度设置为 4。不要在代码中混合使用 Tab 和空格来进行缩进。

在拆行书写时，也要注意代码的缩进，以保证代码的可读性。

2.2. 左花括号“{”的位置

对于 if, while, for 等语句，以下两种风格均允许，请在程序中保持一致的风格，不要混合出现。

紧凑型	清晰型
<pre>if (argc < 2) { perror("input paramater not enough!\n"); exit(0); }</pre>	<pre>if (argc < 2) { perror("input paramater not enough!\n"); exit(0); }</pre>

但在定义函数时，左花括号“{”应另起独占一行，并与函数对齐。

<pre>int func(int x, int y, int z) { ... return 0; }</pre>	(√)
<pre>int func(int x, int y, int z) { ... return 0; }</pre>	(×)

2.3. 空格

2.3.1. 需要有空格的地方

➤ 双目运算符两侧，需要留有空格。如 “=”，“+=”，“>=”，“<=”，“+”，“&&”，“<<”，“^”等。

x += y;	(√)
x+=y;	(×)
val = (mask << 3)	(√)
val=(mask<<3);	(×)

➤ if, for, while, switch 等关键字之后，与紧接着的左括号 ‘(’ 之间，要留有一个空格。

if (true)	(√)
if(true)	(×)
for (i=0;i<10; i++)	(√)
for(i=0; i<10; i++)	(×)
while (x<10)	(√)
while(x<10)	(×)

➤ 左花括号“{”之前

当采用 2.2 above 紧凑型写法时,右括号“)”与左花括号“{”之间要留有一空格。

While (p != NULL) { ... }	(√)
while (p != NULL) { ... }	(×)

➤ 函数参数之间逗号 ‘,’ 分隔后面要留有空格。

void func(int x, int y, int z)	(√)
void func(int x,int y,int z)	(×)

➤ for 语句分号 ‘;’ 分隔后面要留有空格

for (i=0; i<10; i++)	(√)
for (i=0;i<10;i++)	(×)

2.3.2. 不要留空格的地方

➤ 函数名与紧跟的左括号 ‘(’ 之间，不应有空格。

<code>void func(int x, int y, int z)</code>	(√)
<code>void func (int x, int y, int z)</code>	(×)

➤ 左括号的右边，以及右括号的左边，不要有空格

<code>if (x == y)</code>	(√)
<code>if (x == y)</code>	(×)

➤ 一目运算符前后不加空格，如 ‘!’、‘~’、‘++’、‘&’(取地址符)等

<code>i++;</code>
<code>char *p = &tmp;</code>

➤ ‘[]’, ‘->’, ‘.’ 操作符前后不加空格

<code>tmp = p->size;</code>	(√)
<code>tmp = p -> size;</code>	(×)
<code>limit = packet.size;</code>	(√)
<code>limit = packet . size;</code>	(×)
<code>x = array[2];</code>	(√)
<code>x = array [2];</code>	(×)
<code>x = array [2];</code>	(×)

➤ 对于表达式较长的语句，应适当的去掉一些空格，以保证代码的紧凑性。

<code>if ((x>=100) ((x<50) && (y!=0)))</code>	(√)
<code>if (i=0; i<100; i++)</code>	(√)
<code>if (i=0; i < 100; i++)</code>	(×)

2.4. 空行

➤ 函数定义的前后，均需要适当的空行。

	<- 空行
<code>void func1(int x)</code>	
<code>{</code>	
<code>...</code>	

```

}

                                <- 空行

void func2(void)
{
    ...
}

                                <- 空行

```

- 在函数体内，在一组变量定义完之后，需要适当空行。

```

void func(void *p)
{
    int x = 0;
    void *tmp = NULL;

                                <- 空行

    if (p != NULL) {
        ...
    }
}

```

- 一段逻辑性相关的代码前后需要加空行。

2.5. 代码行

- 一行只写一条语句

<code>if (p != NULL)</code> <code> delete p;</code>	(√)
<code>if (p != NULL) delete p;</code>	(×)

- 一行代码宽度应不超过 80 个字符，对于较长的语句，应该分行书写，增加代码的可读性。

3. 注释

3.1.doxygen 风格的注释

当别的程序或用户会使用到我们程序中的接口或数据结构时,如果在构建代码的时候就有意识的加入 doxygen 风格的注释,那么当程序开发完,使用 doxygen 工具就可以自动生成对你程序接口和数据结构描述的文档。因此建议在程序中使用 doxygen 风格的注释来对接口函数和数据结构进行注释,而对于程序内部私有的数据结构,以及对程序逻辑性的一些注解,则采用传统的 C 语言的块注释/* */和行注释//就行了。

doxygen 的注释块

```
/**
 * @*** XXXXXXXXXXXX
 * @*** XXXXXXXXXXXXXXXXXXXX
 *
 * XXXXXXXXXXXXXXXXXXXX
 */
```

doxygen 支持好几种注释风格,我们仅使用上面这一种即可,即在注释块开始使用两个星号,该注释块是放在所要描述内容的上方的。

在后面我们还会看到如下所示的注释块,这种注释块是放在所要描述内容的右边的。

```
/**< XXXXXXXXXXXXXXXXXXXX */
```

doxygen 的常用指令

@file	文件名
@author	作者
@brief	简要说明
@param	用于函数说明中,后面接参数名字,然后再接关于该参数的说明
@return	对函数返回值进行说明
@note	注解信息
@attention	注意信息
@warning	警告信息
@see	表示交叉参考

➤ 文件头

```
/**
 * The operating system emulation layer
 *
 * Copyright (c) 2009 Innofidei Inc.
```



```
*  
* @file  
* @brief  
*      The operating system emulation layer.  
* @author      jimmy.li <lizhengming@innofidei.com>  
* @date      2009/10/16  
*  
*/
```

@file 后面可跟文件名，也可空着，如果是空着，如上例，则 doxygen 会自动认为 @file 为所在文件的文件名。

@brief 后面可添加对本文件的一些简要描述。

@author 即是作者

@date 创建文件日期

➤ 函数

列出函数的功能描述(@brief)，入口参数(@param[in])，出口参数(@param[out])，返回值(@return)。

```
/**  
 * @brief Write File operation.  
 *  
 * @param[in] fd - file description to an open file.  
 * @param[out] buff - pointer to the buffer that containing the data to write to the file.  
 * @param[in] size - number of bytes to write to the file.  
 * @return - if success return actual write bytes. otherwise return -1.  
 */  
Int INNO_OS_WriteFile(INNO_FileHandle_T fd, void* buff, INNOuint32 size);
```

➤ 结构体

```
/**  
 * @brief File Stat structure  
 */  
struct INNO_FileStat {  
    INNOuint32      attrib;      /**< file attribute */  
    INNOuint32      size;        /**< file size */  
    INNOuint32      atime;       /**< time of last access */  
    INNOuint32      mtime;       /**< time of last modified */  
    INNOuint32      ctime;       /**< time of create */  
};
```

➤ 枚举

```
/** @brief enum of return value */
```

```
typedef enum {  
    INNO_STATUS_INVALID_OPERATION = -7,      /**< invalid operation */  
    INNO_STATUS_NOIMPL = -6,                 /**< the function not implementation */  
    INNO_STATUS_CHECKSUM_ERR = -5,           /**< packet checksum error */  
    INNO_STATUS_NOMEM = -4,                  /**< memory not enough */  
    INNO_STATUS_TIMEOUT = -3,                /**< timeout */  
    INNO_STATUS_BAD_PARAM = -2,              /**< invalid parameter(s) */  
    INNO_STATUS_ERROR = -1,                  /**< other error */  
    INNO_STATUS_NOERR = 0,                   /**< successful */  
} INNO_STATUS_T;
```

➤ 变量

```
/** use to store platform callback functions */  
struct INNO_HIF_PlatformFuncs * g_client_funcs;
```

3.2. 关于注释的一些建议

- 保证每条注释都是有效的；没用的、过时的注释要删除；修改代码的同时，要记得更新相应的注释，以保证代码与注释的一致性。
- 避免注释泛滥，显而易见的代码就不用加注释。
- 注释的位置放在被描述的代码的上方或右边，不要放在代码的下面。

4. 命名规则

变量、函数的命名尽量与所采用的操作系统或开发工具的风格保持一致。

Linux 下开发，变量及函数的命名，尽可能的使用小写，单词间用下划线‘_’分隔。

5. 变量

- 尽可能的少使用全局变量，全局变量会增加代码间的耦合度。
-

6. 函数

- 定义函数时，如果函数没有参数，则用 void 填充。

- 如果函数参数是个指针，且仅作输入用，应在类型前加 `const`，以防止该指针在函数体内被意外修改。
- 函数体入口，应该对参数作有效性检查。
- 调用别的函数时，应该对返回值进行判断，对于错误返回值，要有相应错误处理。
- 仅限于在本文件中内被调用的函数，应该添加 `static` 进行修饰，避免污染名字空间。

7. 其它

- 在源文件中仅引用你需要的头文件。
- 使用表达式中用到多个运算符时，为避免默认优先级引发的错误，和为了提高代码的可读性，应当用括号来确定表达式的操作顺序。

8. 参考资料

- 1, 林锐. [2001]. 高质量 C++/C 编程指南.pdf