Penguin 编码规范

目的：为了保证在在软件开发过程中，全体成员的代码风格一致，便于维护，提高软件产品的质量和保持开发产品的延续性，特制定本编码规范。

要求：

1、 统一交流

采用 http://im.qq.com/qq/linux/download.shtml 中的 deb 版本的 qq 进行交流

2、 目录规范

Penguin

+src

+lib

+include

+doc

README

ChangeLog

RELEASE

Others（其它工程文件如 IDE 相关的项目配置文件）

Makefile

3、 文件名：

代码文件使用下列扩展名：

C 源文件 .c

C 头文件 .h

代码文件常用文件名：

Makefile： Makefile 的首选名称

RELEASE：程序发行说明 ,描述各个版本的主要更改、问题解决状况、尚未解决的问题等等 .首选文件名 .

README：总结项目目录的内容。 ChangeLog：代码修改记录 ,详细描述对每个源文件所做的修改 ,最好精确到函数级别。结构上 ,可采用如下形式：

ChangeLog：

+ 日更改记录

+ 文件修改记录

4、 文件头规范

文件头:

版权说明

公司或组织的名称

开发人员

地址（联系方式）

模块目的 /功能

文件版本

修改日志

5、 代码文件结构

C 头文件

文件头

版本历史

#include 区（可选）

常量定义

全局宏定义

全局数据类型定义

全局变量定义

外部引用定义

全局函数原型定义

C 源文件

文件头

版本历史

#include 区

#define 区

宏定义区(Macros)

本地数据类型定义区(Local data types)

本地变量区 (Local variables)

本地函数原型 (Local function prototypes)

局部函数(Local functions)

全局函数(Global functions)

6、 排版

-函数、结构、循环、判断等语句都需要采用缩进， 缩进请使用 4 个空格

-代码行宽度，推荐限制在 80 个字符内

-相对独立的程序块之间、变量说明之后必须加空行。示例:

```
void function1()
{
   int var1,var2;
   var1 = 1;
   var2 = 2;
}
```

-不允许把多个短语句写在一行中

-若函数或过程中的参数较长，则要进行适当的划分

-If, for, while 等的写法,{与 if, for, do 在同一行：

```
if (a == b) {
    do something;
}
```

-在两个以上的关键字、变量、常量进行对等操作时 ,它们之间的操作符之前、之后或者前后要加空格 ;进行非对等操作时,如果是关系密切的立即操作符(如->),后不应加空格。

– 逗号、分号只在后面加空格。

```
int a, b, c;
```

– 比较操作符 , 赋值操作符 "=" 、 "+=", 算术操作符 "+" 、 逻辑操作符"&&", %", "&",位域操符"<<"、 "^"等双目操作符的前后加空格。

```
if (current_time >= MAX_TIME_VALUE)
a = b + c;
a *= 2;
a = b ^ 2;
```

– 空格

– "!"、"~"、"++"、"--"、"&"(地址运算符)等单目操作符前后不加空格。

```
*p = 'a';              // 内容操作"*"与内容之间
flag = !isEmpty;  // 非操作"!"                与内容之间
p = &mem;          // 地址操作"&" 与内容之间
i++;                    // "++","--"与内容之间
```

– "->"、"."前后不加空格

```
p->id = pid; // "->"指针前后不加空格
```

– if、for、while、switch 等与后面的括号间应加空格, 使 if 等关键字更为突出、明显。

```
if (a >= b && c > d)
```

– 针对变量的条件判断语句,

7、 注释

对于函数要给出必要的注释 ,列出:函数的名称、功能、入口参数、出口参数、返回值、调用说明 .

```
/**
 * sample_func – summary this function
 * Parameter:
 *     param1(IN/OUT):      description of param1
 *     param2(IN):      description of param2
 *  return value:
 *      1 -
 *      0 -
 * * Detail descript the usuage of sample_func.
 */
int sample_func(int param1, int param2);
```

– 注释的格式使用//

– 将注释与其上面的代码用空行隔开

```
/* code block one comment */
code block one
/* code block two comment */
code block two
```

– 每 1 个注释都必须有用，尽量少写注释

8、 标识符命名

– 标识符的命名要清晰、明了，有明确含义

– 命名规范必须与所使用的系统风格保持一致，采用骆驼式，比如复制字符串到用变量表示为 strTo（第一个单词小写，第二个单词用首字母大写）

– #define constants:

– #define macros:

– typedefs:

– enum tags:

所有字符都必须大写

9、 变量要求

– 尽可能少定义全局变量

– 严禁使用未经初始化的变量作为右值

– 局部变量与全局变量不要同名

– 当向全局变量传递数据时,要十分小心,防止赋与不合理的值或越界现象发生

10、 函数过程

– 对所调用函数的错误返回码要仔细、全面地处理

– 编写可重入函数时,应注意局部变量的使用

– 在函数内部慎用 static 变量

– 局部变量与全局变量不要同名

– 编写可重入函数时,若使用全局变量,注意加以保护

– 所有函数调用参数判断全部由函数内部实现。出错时直接提示并退出程序即可。

– 函数的规模尽量限制在 100 行以内

11、 宏

– 用宏定义表达式时,要使用完备的括号

示例：如下定义的宏都存在一定的风险。

```
#define RECTANGLE_AREA( a, b ) a * b
#define RECTANGLE_AREA( a, b ) (a * b)
#define RECTANGLE_AREA( a, b ) (a) * (b)
```

正确的定义应为：

```
#define RECTANGLE_AREA( a, b ) ((a) * (b))
```

– 使用宏时,不允许参数发生变化

示例 :如下用法可能导致错误。

#define SQUARE(a) ((a) * (a))

int a = 5;

int b;

b = SQUARE(a++); // 结果 :a = 7, 即执行了两次增 1。

正确的用法是:

b = SQUARE(a);

a++; // 结果:a = 6, 即只执行了一次增 1。

– 将宏所定义的多条表达式放在 do{}while(0)

例如： #difne DO(a,b) a+b;\

a++;

应用时：

if(....)

DO(a,b); //产生错误

else

解决方法: #difne DO(a,b) do{a+b;\

a++;}while(0)

12、 可测试性

在同一项目组或产品组内,要有一套统一的为集成测

试与系统联调准备的调测开关及相应打印函数 ,并且

要有详细的说明。

示例： 以下使用 DEBUG 作为调试开关 ,输出调试信息

#ifdef DEBUG

#define DPRINTF(s) PRINTF(s)

#else

#define DPRINTF(s)

#endif

如果要将调试开关打开，” 只需要传递给编译器 -DDEBUG” 标记。

– 调试信息的格式必须保持一致和有意义：至少要有所在模块名 (或源文件名)及行号

– 单元测试代码应在一个子模块

– 正式发布去点测试部分

– 测试部分不能影响正常功能和效率

13、 其他

– 对库函数增加 C++支持声明：

示例 :#ifdef __cplusplus

extern "C" {

#endif

int function1();

int function2();

#ifdef __cplusplus

}

#endif

– 为防止重复包含， 请在头文件加上条件 INCLUDE:

示例：

```
#ifndef __LINUX_FILE_H
#define __LINUX_FILE_H
#include <stdio.h>
#include <include/mydefineheader.h>
struct file_struct {
    //....
    //....
};
#endif
```

– 仅引用你需要的头文件

– 尽量不要在头文件中暴露数据结构 :这样可以用户对

你的实现的依赖，也减少了用户的编译时间

```
typedef struct my_State my_State;
my_State *my_open (void);
void        my_close (my_State *L);
```

– *函数声明前加 XXX_API 有利于拓展:

```
#ifndef MY_API
#define MY_API              extern
#endif
MY_API my_State *my_open (void);
```