# 第1章文件组织

## 1.1文件命名

1.文件名最长不得超过16字节（不包括扩展名）；文件名和目录名只能使用小写字母、数字和下划线；

2.文件名要遵照“模块名\_功能名”的格式；

## 1.2文件组织结构

1.文件组织以模块为单位，一个模块一个独立的源代码和资源文件目录，一般结构如下：

XXXXXX\ //XXXXXX模块的源代码目录

XXXXXX.mk //makefile for module

include\ //private header files deirectory

source\ //source files directory

resource\ //resource files directory

2.一个模块一般包含四种文件

(1).工程文件（.mk）；

(2).头文件（.h）；

(3).C文件（.c）；

(4).RC文件（.rc）。

3.一个模块一般只有一个工程文件；

4.一个模块的头文件一般分成两部分：

(1).提供给其他模块调用的接口的公共头文件（xxx\_pub.h），存放在\public\include目录下；

(2).提供给本模块内部使用的私有头文件（xxx\_xxx.h），存放在本模块对应的\include\目录系

5..c文件一般按照功能模块进行划分，一个模块还可以拆分成多个子模块，由多个.c文件组成；

6.由于利用c语言的文件编译链接的方式进行裁减，一个模块的多个可供裁减的部分不能写成一个c文件，而应该各自写成独立的.c文件，并且不能直接引用其他C文件内部的函数或全局变量。

## 1.3文件格式

1.任何一个文件都应该有如下的文件头部说明：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) yyyy,Corparation Name Co.,Ltd. All rights reserved.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* @file

\* @brief

\* Project Name:

\* <br>

\* Module Name:

\* <br>

\*

\* @author

\* @date 2008/08/16

\* @version

\* @par

\* detailed description

\*

\* @par Modification History

\* -------------------------------------- <br>

\* Date Name Desciption <br>

\* -------------------------------------- <br>

\* YYYY/MM/DD

\*

\*/

2.任何一个函数都应该有如下的函数头部说明：

/\*\*

\* @brief

\* @author

\* @param [IN]

\* @exception

\* @return

\* @par

\* detailed description

\*

\* @remarks

\* @par Modification History

\* -------------------------------------- <br>

\* Date Name Desciption <br>

\* -------------------------------------- <br>

\* YYYY/MM/DD

\*

\*/

3.任何一个.c文件都应该包含如下形式的头部和尾部：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) yyyy,Corparation Name Co.,Ltd. All rights reserved.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* @file

\* @brief

\* Project Name:

\* <br>

\* Module Name:

\* <br>

\*

\* @author

\* @date 2008/08/16

\* @version

\* @par

\* detailed description

\*

\* @par Modification History

\* -------------------------------------- <br>

\* Date Name Desciption <br>

\* -------------------------------------- <br>

\* YYYY/MM/DD

\*

\*/

#ifdef \_\_cplusplus

extern "C"

{

#endif

/\* please include system header files here \*/

#include <Windows.h>

/\* please include module header files here \*/

#include "utility.h"

/\* please define all global variables here \*/

UINT32 g\_uTest;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

file body here

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef \_\_cplusplus

}

#endif /\* end of \_cplusplus \*/

4.任何一个.h文件都应该包含如下形式的头部和尾部：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) yyyy,Corparation Name Co.,Ltd. All rights reserved.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* @file

\* @brief

\* Project Name:

\* <br>

\* Module Name:

\* <br>

\*

\* @author

\* @date 2008/08/16

\* @version

\* @par

\* detailed description

\*

\* @par Modification History

\* -------------------------------------- <br>

\* Date Name Desciption <br>

\* -------------------------------------- <br>

\* YYYY/MM/DD

\*

\*/

#ifndef \_XXX\_H\_

#define \_XXX\_H\_

#ifdef \_\_cplusplus

extern "C"

{

#endif

/\* 头文件不允许引用任何其他头文件\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Header file body here

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef \_\_cplusplus

}

#endif /\* end of \_cplusplus \*/

#endif /\* end of \_XXX\_H\_ \*/

5.标准文件的模板见如下两个文件：

template.c template.h

## 1.4文件大小

为增加代码的可读性和可维护性，一个.c文件不宜过大。建议一个. C文件不要超过5000行。

# 第2章 文件引用

## 2.1文件引用

1.禁止采用extern外部声明方式引用模块外部的任何函数和变量；

2.引用任何外部函数、数据结构、变量和宏都应该首先引用对方提供的公共头文件（xxx\_pub.h）；

3.引用外部公共头文件可以直接使用文件名，不必包括其相对路径，其路径已经在工程文件makefile中指定；

4.引用内部头文件需要包括其相对路径，以本模块的根目录为根；

5.头文件常常彼此有引用关系，由各个模块自己保证其相对依赖的引用关系；

6. 头文件应include自身编译所必须的头文件。

7.C文件应该只引用需要的头文件，禁止引用多余头文件；

8.禁止引用C文件。

# 第3章 数据类型

## 3.1基本数据类型

禁止直接使用C的基本数据类型，应该使用typedef后的基本数据类型，基本数据类型如下：

|  |  |
| --- | --- |
| VOID | 无类型 |
| BOOL | 布尔型，取值TRUE或FALSE |
| CHAR | 8位有符号字符，占1字节 |
| INT16 | 16位有符号整数，占2字节 |
| UINT16 | 16位无符号整数，占2字节 |
| INT32 | 32位有符号整数，占4字节 |
| UINT32 | 32位有符号整数，占4字节 |

注意：CHAR不一定会被定义成有符号的，这取决于编译器。所以，永远不要用CHAR去保存负数。

## 3.2复杂数据类型

结构、联合和枚举一般定义如下：

typedef struct tagXXXX\_XXX

{

...

}XXXX\_XXX\_S;

typedef union tagXXXX\_XXX

{

...

}XXXX\_XXX\_U;

typedef enum tagXXXX\_XXX

{

...

}XXXX\_XXX\_E;

1.所有复杂数据类型都以”tag”为前缀，”tag”后面一般要紧跟模块名，并以下划线分开后续的结构名，形式如”tagXXXX\_XXX”；

2.重定义的类型一般形式为”XXXX\_XXX”，必须全部为大写，其中：前两部分一般就是模块名加结构名，一般就是前面所列”tagXXXX\_XXX”中”tag”后面的部分；

3.由下划线分割的最后一个字母为后缀，一般用”\_S”表示结构、”\_U”表示联合、”\_E”表示枚举；

4.数据结构中的额元素个数要适中，若结构中元素过多，可以依据某种原则拆分成若干个子结构；

5.数据结构设计中，应该合理安排元素的顺序，使易于理解，并尽量能节省空间；

6.联合类型的各字段必须具备相同的本质特征，禁止把毫不相关的字段组合成联合类型；

7.避免数据结构之间通过“虚拟字段”直接访问，保证数据结构的封装性和完整性。

typedef struct tagTEST\_TEST

{

UINT32 cchCount;/\* 字符个数\*/

CHAR pszStr[0]; /\* 指向字符串\*/

}TEST\_TEST\_S;

上述结构通过pszStr引用该结构后续内容，破坏了数据结构的封装性，禁止使用。

8.同一模块禁止定义大量雷同的数据结构，要求进行复用和优化。

# 第4章 变量和常量

## 4.1变量名必须使用如下前缀：

|  |  |
| --- | --- |
| VOID | v |
| BOOL | b |
| CHAR | c |
| UCHAR | uc |
| INT16,INT32 | n |
| UINT16,UINT32 | u |
| Structures | st |
| Unions | un |
| Enumerations | en |
| Pointers | p |
| Pointer to function | pf |
| String | sz |
| Array | a |
| Global Viriables | g\_ |

注意：g\_、a、p可以用在其他前缀之前，当有多个是，采用”g\_ap”顺序定义。

## 4.2变量命名

1.变量名要采用大小写交替的方式，最长不得超过32字节，并反映出实际意义，以增强程序可读性。对于一些约定俗成的变量可以采用简单命名，如循环变量定义成i，开关中断的变量定义成s等。

## 4.3类型转换

1.禁止一切隐式类型转换。

2.如果同时转换字长和有无符号类型，则需要明确指出转换的顺序：

例如在不同的编译器下，如下的代码的结果是不确定的：

UINT32 uTemp;

INT16 nTemp = -2;

uTemp = nTemp;

正确的写法是：

uTemp = (UINT32)((UINT16)nTemp); //uTemp = 0xFFE;

或

uTemp = (UINT32)((INT32)nTemp); //uTemp = 0xFFFFFFFE

3.双目运算符两端的操作数数据类型必须一致，如果不一致，必须进行强制转换；

4.进行强制转换时，应该保证数据的取值范围不大于其中最小的那个数据类型，以免截取时发生有效数据位的丢失。

## 4.4移位

如果一个整型变量会做移位操作，不要定义成有符号数，除非有特殊的目的。

例如，对于下面的代码：

CHAR cTwoLength;

…

cTwoLength >> 4;

结果到底是正是负，是不可预料的。当cTwoLength为39时，其结果是一个整数；而当cTwoLength为153时，其结果将是一个负数。

## 4.5变量

1.所有变量使用前必须初始化，严禁将没有初始化的变量作为右值；

2.变量初始化采用“就近原则”，即在开始使用的地方进行初始化，特别是循环控制变量；

3.每个变量只用于单一用途，避免具有多重含义的变量或参数。

## 4.6常量

1.禁止使用魔幻数，应用const或#define 定义常量；

2.在使用宏来代替常数时，应该包括显示的类型转换，例如对于TRUE和FALSE的定义：

#define FALSE ((BOOL)0)

#define TRUE ((BOOL)1)

## 4.7负数

尽量避免使用负数，更不要用“-1”代替“全1”。

# 第5章 宏

1. 宏在命名时要以模块名带下划线开头，全部大写并以下划线分割。

2. 定义宏时，参数和宏体都要用圆括号括起来。

3. 宏定义包括多条语句是，用do{…}while(FALSE)包起来。

# 第6章 函数

1. 函数名应以“大写模块名\_”开始，以大小写交错的形式，表达出函数的实际意义，一般为动宾词组。

2. 函数的参数按照IN、INOUT、OUT的顺序排列。

1. 参数设计要体现函数接口的抽象性，并且参数之间应该保证正交性。

3. 参数合法性一般由调用者，但为了健壮性，函数本身也要进行参数合法性检查。

尽可能保证函数的参数不超过7个。

4. 禁止将函数的参数作为工作变量。

5. 执行不会失败或失败后没有影响的函数的返回值，可以不判断以简化代码。但要在函数调用前明确使用(VOID)。

函数的功能应明确且单一

6. 函数应该只有一个入口一个出口，为此应使用RETURN宏：

1. 用RETURN宏替代return:

#define RETURN(label) (goto (label))

2. 函数中只能有多个RETURN和末尾一个return，不得再使用goto。

3. label只能定义在函数末尾，即label后不能有跟释放资源无关的代码。

3. label的命名规范：函数名大写\_动词\_资源，动词一般是Close，Free，Release等，首字母大写，资源名也首字母大写，其余小写。

4. 为配合RETURN，函数的第一行一定是定义一个result变量(目前我把这个变量名字定死为result，从pascal学来的)。

5. 每次RETURN前，记得将result赋相应的值。

6. 申请资源和函数末尾的释放资源的顺序是相反的，这有点不合直觉，不过是由goto的特点决定的。

1. 断言一般用来检查函数入口的参数，特别是模块内部之间的函数接口，连续顺序调用的函数接口。
2. 已经使用正常代码进行错误处理的地方不必再增加断言。
3. 禁止将断言作为错误处理，断言仅在Debug版本中有效。

7. 在正常错误流程，不应使用断言，应该用条件判断等进行错误处理。

# 第7章 编写风格

## 7.1 换行

1. 一行只写一条语句。

2. 一行代码的长度不应该超过120个字符，超过的应该换行。

3. 如果换行，应按照如下原则操作：

在逗号后换行；

在操作符后换行；

长表达式要在低优先级操作符后换行；

## 7.2 缩进

1. 在函数体的开始、结构的定义、枚举的定义以及if、for、do、while、switch、case语句中的程序都要采用缩进，缩进的距离是4个空格。

2. 禁止使用Tab进行缩进，用4个空格代替。

## 7.3 大括号

1. if、else、for、do、while、switch、case、default等语句占一行，其后的执行语句，一律要用{}括起来，哪怕只有一句。

2. 大括号”{”和”}”各自独占一行，并且要与if、for等左对齐。

## 7.4 空格

2. 函数调用中，使用逗号分隔多个参数时，逗号后必须有一个空格。

3． 所有的双目操作符，除了”.”和”->”，两边都必须有空格，单目运算符和操作数之间不能有空格。

## 7.5 空行

以下情况应该使用空行：函数之间；局部变量和第一条可执行语句之间；相对独立的程序块之间。

## 7.6 注释

2. 短注释可以跟在代码行的后面，但必须有足够间隔；如果多个连续短注释要对齐。

3. 长注释要放在其所描述的代码之前，和前面的代码用空行分开，和所描述的代码段保持相同的缩进。

4. 一个文件的注释（注释行占总行数的百分比）应该在30%以上。

6. 如果变量、常量，如果其命名不是充分自注释的，在声明时都应该加以注释。注释可以放在上方或右方。

7. 数据结构声明，如果其命名不是充分自注释的，在声明时都应该加以注释。注释只可放在上方。

8. 全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数或过程存取它以及存取时注意事项等的说明。

9. 不要在一行代码内部插入注释。

10. 禁止使用#if 0 和 /\*…\*/方式保留废弃的代码或注释，废弃的代码和注释必须删除。

11. 代码维护记录注释要求：

/\* Added/Modified by 姓名拼音+工号，日期 of 问题单号 \*/

bFlag = TRUE; /\* Modified by wangxiaoming4153, 2008-8-30 of HSD01234 \*/

其他方式均采用如下封闭型格式：

/\* Begin Added/Modified by 姓名拼音+工号，日期 of 问题单号 \*/

/\* End Begin Added/Modified by 姓名拼音+工号，日期 of 问题单号 \*/

例如：

/\* Begin Modified by wangxiaoming4153, 2008-8-30 of HSD01234 \*/

Program Code

/\* End Modified by wangxiaoming4153, 2008-8-30 of HSD01234 \*/

## 7.7 goto

禁止使用goto。

## 7.8 相等判断

判断一个变量和一个常量是否相等，必须把常量放在“==”的左边，对“!=”的处理也如此。

## 7.9 switch

1.case后的语句应使用{}括起来。

2.case分支如果没有break，必须添加注释说明。

3.每个switch都必须包含分支。如果default分支不应该被走到，在其中添加断言。

## 第8章

3. 除特殊要求外，禁止将对数据结构中的字段的赋值操作分散。

2. 释放内存后，指针必须清零。

3. 第一时间释放内存资源。

4.合理使用栈内存和堆内存。

## 8.4 语言编程类

2. 全局变量的访问应该严格受控，对其访问应该通过宏或函数等封装手段。

3. 只在本C文件中使用的函数和全局变量，应使用static修饰符。

4. use const whenever needed.

6. 避免依赖运算符的优先级，使用括号明确表达式的运算顺序。

7. 禁止使用?:运算符。

9. 与0比较应使用如下形式：

整形变量与零值比较：if(0 == value)或if(0 != value)

指针变量与零值比较：if(NULL == p)或if(NULL != p)

浮点变量与零值比较：if(abs(x) <= EPSILON)或if(abs(x) >= EPSILON)