质量体系

技术文件

**文件名称：C语言程序编码规范**

**文件编号：**RC-SG-84

**版本号：**A

**编制部门：**运作与质量管理部

|  |  |  |  |
| --- | --- | --- | --- |
| 会签部门 | 会签人/日期 | 会签部门 | 会签人/日期 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

审核：日期：

批准：日期：

分发号：受控标识：

瑞斯康达科技发展股份有限公司

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **瑞斯康达科技发展股份有限公司**  文件更改记录 | | | | | 编号 | | RC-QP-01-F09 |
| 版本/更改 | | **D/0** |
| 序号 | |  |
| 文件名称 | | C语言程序编码规范 | | | | | |
| 版本 | 修订内容 | | | 更改日期 | | 更改申请号 | |
| A.00 | 创建 | | | 2012-07-11 | |  | |
| A.01 | 1、修该部分规则；  2、修改版权显示方式； | | | 2014-04-22 | |  | |
| A.02 | 1. 强调规则必须遵守 | | | 2015-6-18 | |  | |
|  |  | | |  | |  | |
|  |  | | |  | |  | |
|  |  | | |  | |  | |
|  |  | | |  | |  | |
|  |  | | |  | |  | |
| 分  发  范  围 | 05-研发中心 | |  |  | |  | |
|  | |  |  | |  | |
|  | |  |  | |  | |

**目录**

1 引言 1

**1.1** **目的** 1

**1.2** **一致性** 1

**1.3** **术语** 2

2 文件目录 2

3 文件结构 2

**3.1** **版权和版本的声明** 3

**3.2** **头文件的结构** 4

**3.3** **源文件的结构** 4

**3.4** **对第三方软件的修改记录** 5

4 程序版式 6

**4.1** **空行** 6

**4.2** **代码行** 6

**4.3** **代码行内的空格** 7

**4.4** **对齐** 8

**4.5** **长行拆分** 9

5 命名规则 9

**5.1** **一般规则** 9

**5.2** **变量命名** 11

**5.3** **函数命名** 11

**5.4** **缩略语** 11

**5.5** **常量命名** 11

6 常量与宏 11

**6.1** **一般规则** 11

**6.2** 常量定义规则 12

**6.3** 宏定义规则 12

**6.4** 断言 12

7 变量与类型 13

**7.1** **变量定义** 13

**7.2** **数据类型** 13

**7.3** **对数据类型的操作** 14

**7.4** **类型转换** 14

8 表达式与语句 14

**8.1** **运算符的优先级** 15

**8.2** **复合表达式** 15

**8.3** **反例：** 15

**8.4** **循环语句** 17

**8.5** **switch语句** 17

**8.6** **goto语句** 18

9 注释 18

**9.1** **注释规则** 18

**9.2** **函数头注释** 18

**9.3** **变量注释** 19

10 函数 19

**10.1** **参数规则** 19

**10.2** **返回值规则** 20

**10.3** **其它建议** 22

11 内存 22

12 编译与代码审查 23

13 附录 24

**13.1** **附录A：代码审查表** 24

**13.2** **附录B：源文件举例** 27

**13.3** **附录C：头文件举例** 29

**13.4** **附录D：第三方文件修改记录举例（rfc2574.c）** 31

**13.5** **附录E：常用缩写表** 33

**13.6** **附录F：常用简写表** 34

**13.7** **附录G：基本数据类型** 35

**13.8** **附录H：README文件模板** 37

# 引言

## **目的**

使用本规范是为了支持下述软件品质：

**无错误、有失效保障、可靠**

一个软件产品，如果严格符合它的规范就是正确的，如果在异常环境下仍然能够工作就是健壮的（robust）。无错误、有失效保障、可靠就是指一个软件产品既正确又健壮。

**可维护**

一个软件产品，如果它可以随着规范的改变而方便地进行修正就是可扩展的；如果一个熟悉该软件规程的新程序员能够轻松地理解它的实现就是可读的；如果能够整个或部分地被一个新的应用所重用就是可重用的。可维护就是指一个软件产品可扩展、可读并且可重用。

**可移植**

一个软件产品，如果可以方便地转移到一个不同的硬件或软件环境中应用，那么它是可移植的。

**可配置**

一个软件产品，如果能够轻松地选择源代码的正确版本和其它联编一个软件产品所需的部件，那么它是可配置的。

**可验证**

一个软件产品，如果能够在验证阶段轻松地准备测试数据和检测故障的程序，那么它就是可验证的。

**易调试**

一个软件产品，如果能够轻松地从故障上溯查找到错误所在，那么它是容易调试的。

## **一致性**

凡所有开发的C软件都必须遵循本文档的要求，第三方软件可以有例外。其中，规则必须遵守，而建议则希望被遵守，如不能遵守，其原因必须作书面记载。

## **术语**

**标识符（identifier）**：指C中的一个变量、常量、函数或类型。在本文档中它与“名称”同义。

**定义（define）**：变量或常量的定义是指编译器为其分配的内存。函数的定义是指函数体的实现（也是为代码分配内存）。

**声明（declare）**：指一些实体对编译器的声明所在的位置，如函数原型、类型（由typedef定义）、变量和常量。声明不分配内存。

**宏（macro）**：指由#define语句定义的代替一个文本串的名字。当该名字在源程序中出现时，预处理器用对应文本串代替它。

**固有数据类型**（或预定义或基本数据类型）：由语言本身定义的数据类型，如int。

**用户定义数据类型**：编程人员用struct、union、enum或typedef定义的数据类型。

**范围**：标识符的范围是指其可见的环境。环境在这里指可以使用该标识符的函数或程序块。

**全局**：使用范围没有限制。

**文件级局部**：使用范围局限在一个文件中。

**函数级局部**：使用范围局限在一个函数中。

# 文件目录

【建议2-1-1】每个模块的代码(包括头文件)放在以“系统+模块”方式命名的目录下。

【建议2-1-2】文件的命名使用“系统+模块(+功能)”的方式命名。

例：RNC系统RANAP模块的C语言代码应该为rnc\_ranap.c和rnc\_ranap.h，如

果程序太大仍需分为不同的文件，则按功能区分，比如：rnc\_ranap\_dispatch.c。

【规则2-1-1】实现同一功能的不同函数应在同一文件内。

【规则2-1-2】目录、文件名使用小写字母、数字、符号，不要使用大写字母。

【规则2-1-3】在每个子目录内要编写README文件对文件及子目录的组织结构作细致的描述。

README文件模板见13.8**附录H：README文件模板**

# 文件结构

C++/C程序的头文件以“.h”为后缀，C程序的定义文件（源文件）以“.c”为后缀。

## **版权和版本的声明**

版权和版本的声明位于头文件和定义文件的开头（参见示例3-1），主要内容有：

1. 版权信息。

我司版权描述有两种方式：

1）Copyright(c)20\*\*-20\*\*, Raisecom Technology Co., Ltd.（版权可描述两个时间点，如Copyright(c)2008-2014，2008代表版权开始年份，2014指此最新软件的发布年份；）

2）Copyright(c)20\*\*, Raisecom Technology Co., Ltd.)（版权描述1个时间点，如Copyright(c)2014，代表版权产生的时间为2014年；）

（2）文件名，作者/修改者，完成日期。

（3）最终版本号（必须跟最新修改记录的版本号相同），实现目的。

（4）修改记录。

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

/\* 瑞斯康达科技发展股份有限公司 版权所有 \*/

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

/\* @@BEGAIN\_INTERNAL\_LEGAL@@ \*/

/\* \*/

/\* Copyright(C) Description \*/

/\* Copyright(C)20\*\*-20\*\*, Raisecom Technology Co.,Ltd. \*/

/\* Unpublished work-rights reserved under the China Copyright Act. \*/

/\* Use, duplication, or disclosure by the government is subject to \*/

/\* restrictions set forth in the CDTT commercial license agreement. \*/

/\* \*/

/\* @@END\_INTERNAL\_LEGAL@@ \*/

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

/\* 程序文件名：xxxx.c或者xxxx.h \*/

/\* Filename：xxxx.c或者xxxx.h \*/

/\* 作者： \*/

/\* Author： \*/

/\* 日期：MM/DD/YYYY \*/

/\* Date：MM/DD/YYYY \*/

/\* 当前版本：1.0 \*/

/\* Version：1.0 \*/

/\* 实现目的： \*/

/\* Purpose： \*/

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

/\* 修改记录： \*/

/\* 序号,MM/DD/YYYY,版本，修改者，修改内容(最近的修改记录在最上面) \*/

/\* Modification history： \*/

/\* Index，Date，Author,content \*/

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

示例3-1参考的版权和版本的声明

## **头文件的结构**

头文件由三部分内容组成：

（1）头文件开头处的版权和版本声明（参见示例3-1）。

（2）预处理块。

（3）函数和结构声明等。

版本和版权声明

#ifndef \_文件名\_H

#define \_文件名\_H

其它头文件包含声明

常数和类型定义

宏定义

变量定义声明

函数声明

#endif

示例3-2头文件结构

【规则3-2-1】为了防止头文件被重复引用，应用ifndef/define/endif结构产生预处理块。

【规则3-2-2】用#include<filename.h>格式来引用标准库的头文件（编译器将从标准库目录开始搜索）。

【规则3-2-3】用#include"filename.h"格式来引用非标准库的头文件（编译器将从用户的工作目录开始搜索）。

【规则3-2-4】头文件中只存放“声明”而不存放“定义”。

## **源文件的结构**

源文件有三部分内容：

（1）定义文件开头处的版权和版本声明（参见示例3-1）。

（2）对一些头文件的引用。

（3）程序的实现体（包括数据和代码）。

版权和版本声明

头文件的引用

全局变量

局部常数和类型定义

局部宏定义

局部变量

局部函数声明

函数定义

示例3-3源文件的结构

## **对第三方软件的修改记录**

在软件开发过程中，可能需要对第三方软件进行修改，甚至覆盖第三方软件的某些文件。为了维护第三方软件的原始性，同时也为了方便软件后续的开发和维护，可以对修改进行记录。修改记录紧接着原有文件的版权和版本信息。包括：

（1）修改部分的版权和版本信息。

（2）修改记录。

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

/\* 瑞斯康达科技发展股份有限公司 版权所有 \*/

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

/\* Copyright(C) Description \*/

/\* Copyright(C)20\*\*-20\*\*, Raisecom Technology Co.,Ltd. \*/

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

/\* 当前版本：1.0 \*/

/\* Version：1.0 \*/

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

/\* 修改记录： \*/

/\* 序号,MM/DD/YYYY,版本，修改者，修改内容(最近的修改记录在最上面) \*/

/\* Modification history： \*/

/\* Index，Date，Author,content \*/

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

示例3-4对第三方软件修改的文件头

# 程序版式

版式虽然不会影响程序的功能，但会影响**可读性**。程序的版式追求清晰、美观，是程序风格的重要构成因素。

## **空行**

空行起着分隔程序段落的作用。空行得体（不过多也不过少）将使程序的布局更加清晰。空行不会浪费内存，虽然打印含有空行的程序是会多消耗一些纸张，但是值得。所以不要舍不得用空行。

【规则4-1-1】在每个函数定义结束之后都要加空行。参见示例4-1（a）

【规则4-1-2】在一个函数体内，逻辑上密切相关的语句之间不加空行，其它地方应加空行分隔。参见示例4-1（b）

|  |  |
| --- | --- |
| //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;  } |

示例4-1(a)函数之间的空行 示例4-1(b)函数内部的空行

## **代码行**

【规则4-2-1】一行代码只做一件事情，如只定义一个变量，或只写一条语句。这样的代码容易阅读，并且方便于写注释。

【规则4-2-2】if、for、while、do等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加{}。这样可以防止书写失误。

示例4-2（a）为风格良好的代码行，示例4-2（b）为风格不良的代码行。

|  |  |
| --- | --- |
| int width; /\*宽度\*/  int height; /\*高度\*/  int depth; /\*深度\*/ | int width,height,depth;//宽度高度深度 |
| x = a + b;  y = c + d;  z = e + f; | X＝a+b;y=c+d;z=e+f; |
| if (width < height)  {  dosomething();  } | if(width<height)dosomething(); |
| for (initialization; condition; update)  {  dosomething();  }  //空行  other(); | for(initialization;condition;update)  dosomething();  other(); |

示例4-2(a)**风格良好**的代码行 示例4-2(b)**风格不良**的代码行

## **代码行内的空格**

【规则4-3-1】关键字之后要留空格。像const、static、case等关键字之后至少要留一个空格，否则无法辨析关键字。像if、for、while等关键字之后应留一个空格再跟左括号‘（’，以突出关键字。参见示例4-2

【规则4-3-2】函数名之后不要留空格，紧跟左括号‘（’，以与关键字区别。

【规则4-3-3】‘（’向后紧跟，‘）’、‘，’、‘;’向前紧跟，紧跟处不留空格。

【规则4-3-4】‘，’之后要留空格，如Function(x, y, z)。如果‘;’不是一行的结束符号，其后要留空格，如for(initialization; condition; update)。

【规则4-3-5】赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=”“>=”、“<=”、“+”、“\*”、“%”、“&&”、“||”、“<<”,“^”等二元操作符的**前后**应当加空格。

【规则4-3-6】一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等**前后**不加空格。

【规则4-3-7】像“［］”、“.”、“->”这类操作符前后不加空格。

【建议4-3-1】对于表达式比较长的for语句和if语句，为了紧凑起见可以适当地去掉一些空格，如for (i=0; i<10; i++)和if ((a<=b) && (c<=d))

|  |
| --- |
| void Func1(int x, int y, int z); // 良好的风格  void Func1 (int x,int y,int z); // 不良的风格 |
| if (year >= 2000) // 良好的风格  if(year>=2000) // 不良的风格  if ((a>=b) && (c<=d)) // 良好的风格  if(a>=b&&c<=d) // 不良的风格 |
| for (i=0; i<10; i++) // 良好的风格  for(i=0;i<10;i++) // 不良的风格  for (i = 0; i < 10; i ++) // 过多的空格 |
| x = a < b ? a : b; // 良好的风格  x=a<b?a:b; // 不好的风格 |
| int \*x = &y; // 良好的风格  int \* x = & y; // 不良的风格 |
| array[5] = 0; // 不要写成 array [ 5 ] = 0;  a.Function(); // 不要写成 a . Function();  b->Function(); // 不要写成 b -> Function(); |

示例4-3代码行内的空格

## **对齐**

【规则4-4-1】程序的分界符‘{’和‘}’应独占一行并且位于同一列，同时与引用它们的语句左对齐。

【规则4-4-2】{}之内的代码块在‘{’右边数格处左对齐。

示例4-4（a）为风格良好的对齐，示例4-4（b）为风格不良的对齐。

|  |  |
| --- | --- |
| void Function(intx)  {  …/\*program code\*/  } | void Function(intx){  …/\*programcode\*/  } |
| if (condition)  {  …/\*programcode\*/  }  else  {  …/\*programcode\*/  } | if (condition){  …/\*programcode\*/  }  else{  …/\*programcode\*/  } |
| for(initialization; condition; update)  {  …/\*programcode\*/  } | for (initialization; condition; update){  …/\*programcode\*/  } |
| while (condition)  {  …/\*programcode\*/  } | while (condition){  …/\*programcode\*/  } |
| 如果出现嵌套的｛｝，则使用缩进对齐，如：  {  …  {  …  }  …  } |  |

示例4-4(a)**风格良好**的对齐 示例4-4(b)**风格不良**的对齐

## **长行拆分**

【规则4-5-1】代码行最大长度宜控制在120个字符宽度以内。代码行不要过长，否则眼睛看不过来，也不便于打印。

【规则4-5-2】长表达式要在**低优先级操作符处**拆分成新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

|  |
| --- |
| if ((very\_longer\_variable1 >= very\_longer\_variable12)  && (very\_longer\_variable3 <= very\_longer\_variable14)  && (very\_longer\_variable5 <= very\_longer\_variable16))  {  dosomething();  } |
| virtual CMatrix CMultiplyMatrix (CMatrix leftMatrix,  CMatrix rightMatrix); |
| for (very\_longer\_initialization;  very\_longer\_condition;  very\_longer\_update)  {  dosomething();  } |

示例4-5长行拆分

# 命名规则

## **一般规则**

【规则5-1-1】标识符应当直观且可以拼读，可望文知意，不必进行“解码”。

标识符最好采用英文单词或其组合，便于记忆和阅读。切忌使用汉语拼音来命名。程序中的英文单词一般不会太复杂，用词应当准确。例如不要把CurrentValue写成NowValue。

【规则5-1-2】标识符的长度应当符合“min-length&&max-information”原则。

一个变量的作用域越大，其名字所携带的信息就应该越多。另外，全局函数、结构也应该使用说明性的名字。一般来说，长名字能更好地表达含义，所以函数名、变量名长达十几个字符不足为怪。那么名字是否越长约好？不见得!**例如变量名maxval就比maxValueUntilOverflow好用。而象简单的循环变量用约定俗成的i、j即可**。

【规则5-1-3】命名要保持一致性和简明

相关的名字能更好说明相互关系和差异，以下代码中同一个词“队列（queue）”被分别写为Q、Queue或queue，应该使用一致的写法，另外由于只能在结构UserQueue中访问，其成员中的名字不必提到“队列”，减少了名字的冗余信息。

|  |  |
| --- | --- |
| 正例：  structUserQueue  {  int iItems;  int iFront;  float fCapacity;  int iUsers;  } | 反例：  structUserQueue  {  int iNoOfItemsInQ;  int iFrontOfTheQueue;  floatfqueueCapacity;  int iOfUsersInQueue;  } |

【规则5-1-4】程序中不要出现仅靠大小写区分的或者形状容易混淆的标识符。

反例1：

intx,X; /\*变量x与X容易混淆\*/

反例2：

void foo(int x);

void FOO(float x);/\*函数foo与FOO容易混淆\*/

【建议5-1-1】尽量避免名字中出现数字编号，如Value1,Value2等，除非逻辑上的确需要编号。这是为了防止程序员偷懒，不肯为命名动脑筋而导致产生无意义的名字（因为用数字编号最省事）。

【建议5-1-2】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等。

正例：

int minValue;

int maxValue;

int SetValue(…);

int GetValue(…);

【规则5-1-5】全局变量或者函数名要加上“模块名\_”或者“子系统名\_”作为前缀来避免调用混乱，从而避免运行时错误。

正例：

SNMP模块内的MD5Init算法函数定义为：

void SNMP\_MD5Init(MD5\_CTX \*mdContext);

这样它就可以和dox1中的md5算法相区别了。

## **变量命名**

【规则5-2-1】程序中不要出现标识符完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但会使人误解。

【规则5-2-2】变量的名字应当使用“名词”或者“形容词＋名词”。

正例：

float value;

float oldValue;

float newValue;

## **函数命名**

【建议5-3-1】函数的名字应当使用“名词+动词”或者“动词＋名词”

正例：

valueGet(); /\*全局函数\*/

getValue()；

## **缩略语**

【规则5-4-1】标识符中的缩略语应采用约定俗成或符合模块统一规定的缩略语。

缩略语见附录E和F

【规则5-4-2】标识符中的缩写（全部为大写字母）之间或者与大写字母单词之间用下划线‘\_’分隔。

正例：

const int P\_TMSI\_TYPE\_DIMENSION4;

int CN\_DataVolumeReportRequest(MessageType, RABsDataVolumeReport);

## **常量命名**

【规则5-5-1】常量命名全部使用大写。

正例：

const int MAX\_VALUE = 65536;

# 常量与宏

常量是一种标识符，它的值在运行期间恒定不变。常量可以用#define或者const来定义。前者称为宏常量，后者称为const常量。

## **一般规则**

【规则6-1-1】尽量使用含义直观的常量来表示那些将在程序中多次出现的数字或字符串。

正例：

#define MAX 100 /\*宏常量\*/

const int MAX = 100; /\*const常量\*/

const float PI = 3.14159; /\*const常量\*/

【建议6-1-1】用const或enum定义一个常量，避免用#define。

因为用const来定义常量比用#define来定义常量有更多的优点：

1. const常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且字符替换可能会产生意料不到的错误（边际效应）。
2. 有些集成化的调试工具可以对const常量进行调试，但是不能对宏常量进行调试。

## 常量定义规则

【规则6-2-1】需要对外公开的常量放在头文件中（宏常量），不需要对外公开的常量放在定义文件的头部。

【规则6-2-2】如果某一常量与其它常量密切相关，应在定义中包含这种关系，而不应给出一些孤立的值。

正例：

const float RADIUS=100;

const float DIAMETER=RADIUS\*2\*PI;

## 宏定义规则

【规则6-3-1】带参数的宏一定要用括号，用括号括住宏中的参数和表达式及调用的表达式。

|  |  |
| --- | --- |
| 正例：  #define SQUARE(x) ((x)\*(x)) | 反例：  #define SQUARE(x) (x)\*(x)  则1/ SQUARE(x)会被展开为如下的错误内容：1/(x)\*(x)  #define SQUARE(x) (x\*x)  则SQUARE（a+b）会被展开为如下的错误内容：a+b\*a+b |

## 断言

断言assert是仅在Debug版本起作用的宏，它用于检查“不应该”发生的情况。在运行过程中，如果assert的参数为假，那么程序就会中止（一般地还会出现提示对话，说明在什么地方引发了assert）。如果程序在assert处终止了，并不是说含有该assert的函数有错误，而是调用者出了差错，assert可以帮助我们找到发生错误的原因。

【规则6-4-1】为了不在程序的Debug版本和Release版本引起差别，assert不应该产生任何副作用。

# 变量与类型

## **变量定义**

【建议7-1-1】尽可能在定义变量的同时初始化该变量（就近原则）

如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。**如果引用了未被初始化的变量，可能会导致程序错误。**本建议可以减少隐患。在一些特殊情况下，一个变量由一个复杂的表达式来赋值，这时就不必赋予初始值。

如果不赋初值而使用，那么变量的值是随机的！

正例：

int width = 10; /\*定义并初绐化width\*/

指针：

int \*p = NULL;

数组：

char strAddr[20];

memset(strAddr, 0, sizeof(strAddr));

作为字符串使用的数组更应该注意初始化变量！因为对于字符串来说，strlen()等函数是以遇到’/0’为结尾的！

结构体：

struct client\_id cid;

memset(&cid, 0, sizeof(cid));

## **数据类型**

【规则7-2-1】要使用在项目级定义的类型别名，不要用固有数据类型（enum例外）。

基本数据类型见附录G，模块或子系统内部用户定义的数据类型应加“模块名\_”作为前缀。

【规则7-2-2】代码应当与数据类型在内存中的表示无关。

要具有可移植性，程序必须与其整型或浮点型数的字节存放顺序无关。

【建议7-2-1】不要对字符集的顺序做任何假设。

反例：假定字符的取值按字母表的顺序递增。

for(i=‘A’;i<=‘Z’;i++)

【建议7-2-2】要显式定义带符号或无符号字符。

【规则7-2-3】只能将enum型变量与同类型变量相比较。

虽然不同enum类型的变量相比较不会出现语法错误，但是会影响程序的可读性和逻辑性。

## **对数据类型的操作**

【规则7-3-1】如果操作数符号不同，要用标准库函数div和ldiv代替除/取模操作。

【规则7-3-2】只能对无符号整数进行移位操作。

【规则7-3-3】移动的位数只能大于等于0并小于等于被移变量的长度。

【建议7-3-1】在除/取模操作中，用显式检查避免被0除或模0。

【建议7-3-2】不要用移位操作代替乘/除2的幂的操作。

【建议7-3-3】不要用&（按位与）代替取模操作。

## **类型转换**

类型转换分为隐式(Implicit)类型转换和显式(Explicit)类型转换。

【规则7-4-1】避免依赖于结果的类型转换。

|  |  |
| --- | --- |
| 正例：  short i = 0xFFFF;  short j = i;/\*j=-1\*/ | 反例：  long a = 0x7FFFFFFFL;  short b = a;/\*b=-1\*/  即使告警级别已设为最高，SunSPARCC++4.1的编译器也不对下列语句发出任何警告 |

**【规则7-4-2】切不可将指针转换为非指针类型，反之亦不可。**

**【规则7-4-3】切不可将指向函数的指针转换为指向结构或其他任何类型的指针，反之亦不可。**

【规则7-4-4】不能对const定义的常量进行类型转换。

【建议7-4-1】当表达式中混有带符号数和无符号数时要特别小心。

当带符号数和无符号数混和运算时，有符号数有可能转变为无符号数。如果有符号数是个负数，转换后可能变为正数。这可能导致不可预料的结果。

**【建议7-4-2】尽量避免将一个结构类型的指针转换为指向另一个结构类型的指针。**

【建议7-4-3】尽量避免使用union。

# 表达式与语句

表达式属于C++/C的短语结构语法。它们看似简单，但使用时隐患比较多。本节归纳了正确使用表达式的一些规则与建议。

## **运算符的优先级**

【规则8-1-1】如果代码行中的运算符比较多，用括号确定表达式的操作顺序，避免使用默认的优先级。

由于将运算符优先级熟记是比较困难的，为了防止产生歧义并提高可读性，应当用括号确定表达式的操作顺序。

正例：

word=(high<<8)|low

if((a|b)&&(a&c))

【规则8-1-2】不要在一条语句中两次改变一个变量。

反例1：

一条语句中对同一变量的多次修改

y=（x++）\*（x++）;

反例2：

scanf(“%d%d”,&yr,&profit[yr]);

## **复合表达式**

如a=b=c=0这样的表达式称为复合表达式。允许复合表达式存在的理由是：

（1）书写简洁；

（2）可以提高编译效率。但要防止滥用复合表达式。

【规则8-2-1】不要编写太复杂的复合表达式（考虑能否优化）。

反例：

subkey=subkey>>(bitoff–((bitoff>>3)<<3);

以上代码就比较晦涩。它实际上是使bitoff最低3位置为0，再被bitoff减，最终得到bitoff的最低3位，然后以此确定subkey的右移位数，而同样功能的以下代码就简明清晰很多。

正例：

subkey>>=bitoff&0x7;

【规则8-2-2】不要有多用途的复合表达式。

|  |  |
| --- | --- |
| 正例：  a=b+c;  d=a+r; | 反例：  d=(a=b+c)+r;  该表达式既求a值又求d值,  应该拆分为两个独立的语句。 |

## **if语句**

【规则8-3-1】不可将布尔变量直接与宏定义TRUE、FALSE或者1、0进行比较。

根据布尔类型的语义，零值为“假”（记为FALSE），任何非零值都是“真”（记为TRUE）。TRUE的值究竟是什么并没有统一的标准。例如VisualC++将TRUE定义为1，而VisualBasic则将TRUE定义为-1。

假设布尔变量名字为flag，它与零值比较的标准if语句如下：

|  |  |
| --- | --- |
| 正例：  if(flag)/\*表示flag为真\*/  if(!flag)/\*表示flag为假\*/ | 反例：  if(flag==TRUE)  if(flag==1)  if(flag==FALSE)  if(flag==0) |

【规则8-3-2】应当将整型变量用“==”或“！=”直接与0比较。

假设整型变量的名字为value，它与零值比较的标准if语句如下：

|  |  |
| --- | --- |
| 正例：  if(0==value)  if(0!=value) | 反例：  if(!value) /\*会让人误解value是布尔变量\*/ if(value) |

【规则8-3-3】不可将浮点变量用“==”或“！=”与任何数字比较。

千万要留意，无论是float还是double类型的变量，都有精度限制。所以一定要避免将浮点变量用“==”或“！=”与数字比较，应该设法转化成“>=”或“<=”形式。

假设浮点变量的名字为x：

|  |  |
| --- | --- |
| 正例：  if((x>=-EPSINON)&&(x<=EPSINON))  其中EPSINON是允许的误差（即精度）。 | 反例：  if(x==0.0) /\*隐含错误的比较\*/ |

【规则8-3-4】应当将指针变量用“==”或“！=”与NULL比较。

指针变量的零值是“空”（记为NULL）。尽管NULL的值与0相同，但是两者意义不同。假设指针变量的名字为p，它与零值比较的标准if语句如下：

|  |  |
| --- | --- |
| 正例：  if(NULL == p)  if(p!=NULL) p与NULL显式比较，强调p是指针变量 | 反例1:  if(p==0)  if(p!=0)  容易让人误解p是整型变量  反例2:  if(p)  if(!p)  容易让人误解p是布尔变量 |

【建议8-3-1】判断条件为常量与变量进行比较时，应将常量作为左值，变量作为右值。

例如使用if(NULL==p)替代if(p==NULL)，这样做的目的是防止误写成if(p=NULL)。

编译器认为if(p=NULL)是合法的，但是会指出if(NULL=p)是错误的，因为NULL不能被赋值。所以，有意把p和NULL颠倒。

【建议8-3-2】if有多个判断条件时，把高效的条件放在最前。

例如，在比较两个字符串str1(长度len1),str2（长度len2）是否相等时

正例：

if((len1==len2)&&（strcmp(str1,str2)＝＝0）)

反例：

if（strcmp(str1,str2)＝＝0）

因为，如果len1不等于len2，就不会进行（strcmp(str1,str2)＝＝0）的判断，从而提高了效率。

## **循环语句**

【规则8-4-1】不可在for循环体内修改循环变量，防止for循环失去控制。

【建议8-4-1】建议for语句的循环控制变量的取值采用“半开半闭区间”写法。

|  |  |
| --- | --- |
| 正例：  for(int x=0; x<N; x++) | 反例：  for(int x=0; x<=N-1; x++) |

## **switch语句**

有了if语句为什么还要switch语句？

switch是多分支选择语句，而if语句只有两个分支可供选择。虽然可以用嵌套的if语句来实现多分支选择，但那样的程序冗长难读。这是switch语句存在的理由。

【建议8-5-1】每个case语句的结尾不要忘了加break，否则将导致多个分支重叠（除非有意使多个分支重叠）。

【规则8-5-1】不要忘记最后那个default分支。即使程序真的不需要default处理，也应该保留语句default:break;这样做并非多此一举，而是为了防止别人误以为你忘了default处理。

【建议8-5-2】switch语句的每个case分支都要加注释，尤其在有意使多个分支重叠的情况下。

正例：

switch(expr)

{

caseABC:/\*comment\*/

caseDEF:/\*comment\*/

statement;

break;

caseUVW:/\*comment\*/

statement;

/\*comment\*/

caseXYZ:/\*comment\*/

statement;

break;

default:

break;

}

【规则8-5-2】**在switch语句中，case的所有标识变量要用同一个enum类型。**

## **goto语句**

【规则8-6-1】尽量不使用goto语句。

【规则8-6-2】严禁使用longjmp。

# 注释

## **注释规则**

【规则9-1-1】注释应该尽量使用**汉语**，并和程序保持一致。

如果在操作系统不支持汉语的情况下使用英语。英语的注释不要使用复杂的长句，尽量用简单的“谓语+宾语”格式的短句。

【规则9-1-2】临时不用的大段代码，用＃if 0注释掉，而不用“/\*\*/”。

【规则9-1-3】将不再使用的代码删除。

【规则9-1-4】采用统一的格式注释模块、函数、函数参数、变量、typedef、枚举和结构/联合。

## **函数头注释**

建议函数头模板：

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

\*函数名:some\_function \*

\*描述:<描述函数实现的功能>\*

\*相关文档:<描述此函数的相关协议名称、版本及在协议中的位置>\*

\*参数: \*

\* \*

\*参数名 类型 输入/输出 描述 \*

\*-------- ---- --- ----------- \*

\* …… \*

\*返回值:(各个返回值的注释) \*

\*说明：（可选，标识本函数未完成或者可能改动，在本函数相应位置要加“/\*TBD\*

\*…TBD\*/”批注，即留待完成） \*

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

示例9-1函数头

## **变量注释**

enum Error\_Code

{

OUT\_OF\_MEMORY,/\*注释 \*/

INSUFFICIENT\_DISK\_SPACE,/\*注释\*/

LOGIC\_ERROR,/\*注释\*/

FILE\_NOT\_FOUND/\*注释 \*/

};

enum Error\_Code LocalVar;/\*注释\*/

示例9-2变量注释

# 函数

函数接口的两个要素是参数和返回值。C语言中，函数的参数和返回值的传递方式有两种：值传递（passbyvalue）和指针传递（passbypointer）。本节论述函数参数和返回值的一些规则。

## **参数规则**

【规则10-1-1】函数声明中参数的书写要完整，不要贪图省事只写参数的类型而省略参数名字。

|  |  |
| --- | --- |
| 正例：  void SetValue(intwidth, intheight); | 反例：  void SetValue(int, int); |

【规则10-1-2】参数命名要恰当，顺序要合理。

|  |  |
| --- | --- |
| 正例：  StrCopy(char \*strDest, char \*strSrc) | 反例：  StrCopy(char \*str1, char \*str2);  我们很难搞清楚究竟是把str1拷贝到str2中，还是刚好倒过来。 |

【规则10-1-3】如果函数参数占据空间很大的结构体或数组时，采用其指针作为输入参数来提高效率。

函数参数的值传递是将整个值复制一份作为输入参数。指针传递的只是该值的地址，一般只有4个字节。

|  |  |
| --- | --- |
| 正例：  void function(const int\* parray)  {  }  参数传递只需要拷贝sizeof(int \*)个字节 | 反例：  void function(int array[256])  {  }  参数传递需要拷贝256×sizeof(int)个字节 |

【规则10-1-4】如果参数是指针，且仅作输入用，则应在类型前加const，以防止该指针在函数体内被意外修改。  
正例：

void StrCopy(char \* strDest，const char \*strSrc);

【规则10-1-5】在函数体的“入口处”，对参数的有效性进行检查。

【建议10-1-1】避免函数有太多的参数，参数个数尽量控制在6个以内。如果参数太多，在使用时容易将参数类型或顺序搞错。可以使用结构体指针代替多个参数。

控制入参个数更为重要的原因是尽量使用寄存器传递入参，避免通过栈来传递。常见CPU架构的传递入参的寄存器都比较有限。如：MIPS有4个、ARM有4个、PPC有7个。

【建议10-1-2】尽量不要使用类型和数目不确定的参数。

反例：

C标准库函数printf是采用不确定参数的典型代表，其原型为：  
int printf(const chat \*format[, argument]…);  
这种风格的函数在编译时丧失了严格的类型安全检查。

## **返回值规则**

【规则10-2-1】不要省略返回值的类型。

C语言中，凡不加类型说明的函数，一律自动按整型处理。这样做不会有什么好处，却容易被误解为void类型。

【规则10-2-2】函数名字与返回值类型在语义上不可冲突。  
违反这条规则的典型代表是C标准库函数getchar。

反例：  
char c;  
c = getchar();  
if (c == EOF)  
…

按照getchar名字的意思，将变量c声明为char类型是很自然的事情。但不幸的是getchar的确不是char类型，而是int类型，其原型如下：

int getchar(void);

由于c是char类型，取值范围是[-128，127]，如果宏EOF的值在char的取值范围之外，那么if语句将总是失败，这种“危险”人们一般哪里料得到！导致本例错误的责任并不在用户，是函数getchar误导了使用者。

【规则10-2-3】当函数同时返回正常值和错误标志时，正常值用输出参数获得，而错误标志用return语句返回。

回顾上例，C标准库函数的设计者为什么要将getchar声明为令人迷糊的int类型呢？

在正常情况下，getchar的确返回单个字符。但如果getchar碰到文件结束标志或发生读错误，它必须返回一个标志EOF。为了区别于正常的字符，只好将EOF定义为负数（通常为负1）。因此函数getchar就成了int类型。

我们在实际工作中，经常会碰到上述令人为难的问题。为了避免出现误解，我们应该**将正常值和错误标志分开。即：正常值用输出参数获得，而错误标志用return语句返回**。

函数getchar可以改写成BOOL GetChar(char \*c);

虽然gechar比GetChar灵活，例如putchar(getchar());但是如果getchar用错了，它的灵活性又有什么用呢？

【规则10-2-4】带返回值的函数在其每个出口都必须用return语句返回结果值。

void类型的函数，结尾处也要显式地加return。

【规则10-2-5】return语句不可返回指向“栈内存”的“指针”，因为该内存在函数体结束时被自动销毁。

反例：

char \* Func(void)  
{

char str[] = “hello world”; /\* str的内存位于栈上 \*/  
…  
return str; /\* **将导致错误** \*/

}

【建议10-2-1】有时候函数原本不需要返回值，但为了增加灵活性如支持链式表达，可以附加返回值。

正例：

Char \*strcpy(char \*strDest, const char \*strSrc);  
strcpy函数将strSrc拷贝至输出参数strDest中，同时函数的返回值又是strDest。这样做并非多此一举，可以获得如下灵活性：  
char str[20];  
int length = strlen( strcpy(str, “Hello World”) );

【建议10-2-2】在函数体的“出口处”，对return语句的**正确性和效率**进行检查。

如果函数的返回值是一个空间占据很大的结构体，函数尽量采用指针类型的返回值来提高效率。

【建议10-2-3】用STATUS类型（OK/ERROR）来返回函数执行的错误标志；用int来返回正常值。

## **其它建议**

【建议10-3-1】函数的功能要单一，不要设计多用途的函数。

【建议10-3-2】函数体的规模要小，尽量控制在250行代码之内。

【建议10-3-3】尽量避免函数带有“记忆”功能。相同的输入应当产生相同的输出。

带有“记忆”功能的函数，其行为可能是不可预测的，因为它的行为可能取决于某种“记忆状态”。这样的函数既不易理解又不利于测试和维护。在C/C++语言中，函数的static局部变量是函数的“记忆”存储器。建议尽量少用static局部变量，除非必需。

【建议10-3-4】不仅要检查输入参数的有效性，还要检查通过其它途径进入函数体内的变量的有效性，例如全局变量。

【建议10-3-5】用于出错处理的返回值一定要清楚，让使用者不容易忽视或误解错误情况

。

【建议10-3-6】合理规划函数的作用域，作用域仅为本文件的函数，可以合理使用LOCAL(即static)关键字

# 内存

内存越界是最难定位的。

指针有一些“似是而非”的特征：

**（1）指针消亡了，并不表示它所指的内存会被自动释放。**（示例11－1）

**（2）内存被释放了，并不表示指针会消亡或者成了NULL指针。**（示例11－2）

|  |
| --- |
| void Func(void)  {  char \*p=(char\*)malloc(100); //动态内存会自动释放吗？  } |

示例11-1试图让动态内存自动释放

【规则11-1-1】用malloc申请内存之后，应该立即检查指针值是否为NULL。防止使用指针值为NULL的内存。

【建议11-1-1】不要忘记为数组和动态内存赋初值。防止将未被初始化的内存作为右值使用。

【规则11-1-2】避免数组或指针的下标越界，特别要当心发生“多1”或者“少1”操作。

【规则11-1-3】动态内存的申请与释放必须配对，防止内存泄漏。

【建议11-1-2】用free释放了内存之后，立即将指针设置为NULL，防止产生“野指针”。

|  |
| --- |
| char \*p=(char\*)malloc(100);  strcpy(p, “hello”);  free(p); //p所指的内存被释放，但是p所指的地址仍然不变  …  if(p != NULL) //没有起到防错作用  {  strcpy(p, “world”); //**出错**  } |

示例11-2 p成为野指针

【规则11-1-4】在程序的关键部分必须加充分的检查和保护，以防止上溢和下溢。

如对于为字符串动态分配内存的时候，考虑到strlen等关于字符串的函数不计入结尾的’\0’字符，而strcpy等函数却复制它，所以习惯的写法如下：

char strBuffer[256];

…… /\*初始化\*/

p = malloc(strlen(strBuffer)+1);

strcpy(p, strBuffer);

# 编译与代码审查

【规则12-1-1】编译时将告警级别设置为最高并消除所有警告。

让编译器尽可能多的帮你检查程序错误。

【规则12-1-2】代码merge到主干前要进行代码审查并编译通过。

代码审查表见附录A。

【规则12-1-3】每一次代码审查后修改了代码，都要重新编译测试代码。

# 附录

## **附录A：代码审查表**

* 文件目录和结构

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
| 重要 | **每个子目录是否都有README文件** |  |
|  | 头文件和定义文件的名称是否合理？ |  |
|  | 头文件和定义文件的目录结构是否合理？ |  |
| 重要 | 版权和版本声明是否完整？ |  |
| 重要 | 修改记录是否跟上程序修改的进度？ |  |
| 重要 | 头文件是否使用了ifndef/define/endif预处理块? |  |
|  | 头文件中是否只存放“声明”而不存放“定义” |  |
|  | 对第三方软件的修改是否添加了修改记录 |  |
| 重要 | 目录下Readme文件是否存在并且更新？ |  |

* 程序的版式

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
|  | 空行是否得体？ |  |
|  | 代码行内的空格是否得体？ |  |
|  | 长行拆分是否得体？ |  |
|  | “{”和“}”是否各占一行并且对齐于同一列？ |  |
| 重要 | 一行代码是否只做一件事？如只定义一个变量，只写一条语句。 |  |
| 重要 | If、for、while、do等语句自占一行，不论执行语句多少都要加“{}”。 |  |
| 重要 | 在定义变量（或参数）时，是否将修饰符\*和＆紧靠变量名？ |  |

* 命名规则

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
|  | 标识符是否直观且可以拼读？ |  |
|  | 标识符的长度应当符合“min-length&&max-information”原则？ |  |
| 重要 | 程序中是否出现相同名字的局部变量和全部变量？ |  |
|  | 函数名、变量和参数、常量的书写格式是否遵循一定的规则？ |  |
|  | 易重名的静态变量、全局变量是否加前缀？ |  |
|  | 缩略语是否得当？ |  |

* 常量与宏

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
|  | 是否使用含义直观的常量来表示那些将在程序中多次出现的数字或字符串 |  |
|  | 是否用const或enum定义一个常量，避免用#define |  |
|  | 是否在定义中包含相关常量的相互关系？ |  |
|  | 带参数的宏是否用括号括住宏中的参数和表达式及调用的表达式？ |  |
| 重要 | 断言是否起副作用？ |  |

* 变量与类型

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
|  | 是否使用register和auto修饰符？ |  |
|  | 每个变量初始化和缺省值是否正确？ |  |
|  | 是否对每个变量使用一个单独的定义语句 |  |
|  | 是否使用项目级别定义的类型别名？ |  |
|  | 是否显式定义带符号或无符号字符 |  |
|  | enum型变量是否只与同类型变量相比较 |  |
|  | 是否显式检查除数为0？ |  |
|  | 移位操作是否只对无符号整数进行？ |  |
|  | 移位操作的位数是否超过被移变量的长度？ |  |
| **重要** | 检查指针是否进行了非正常的类型转换？ |  |
| **重要** | 是否存在不同数据类型的赋值和比较？ |  |

* 表达式与基本语句

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
| 重要 | 如果代码行中的运算符比较多，是否已经用括号清楚地确定表达式的操作顺序？ |  |
|  | 是否编写太复杂或者多用途的复合表达式？ |  |
| 重要 | 是否用隐含错误的方式写if语句?例如 （1）将布尔变量直接与TRUE、FALSE或者1、0进行比较。 （2）将浮点变量用“==”或“！=”与任何数字比较。 （3）将指针变量用“==”或“！=”与NULL比较。 |  |
| 重要 | Case语句的结尾是否忘了加break？ |  |
| 重要 | 是否忘记写switch的default分支？ |  |
| 重要 | 重叠的case分支是否添加注释 |  |
|  | If语句的判定条件是否为最高效的？ |  |
| 重要 | 是否在一个语句中两次操作同一个变量？ |  |

* 注释

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
|  | 注释是否清晰并且必要？ |  |
| 重要 | 注释是否有错误或者可能导致误解？ |  |
| 重要 | 是否将不再使用的代码删除？ |  |
|  | 是否用#if0/#endif代替/\*\*/来注释掉大块代码 |  |
| 重要 | 函数头注释是否符合规范 |  |
|  | 函数参数注释是否符合规范 |  |
|  | 关键变量是否注释 |  |

* 函数

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
| 重要 | 函数声明中参数是否书写完整 |  |
|  | 参数命名是否要恰当，顺序是否合理 |  |
| 重要 | 占据空间很大的结构体或数组参数是否采用其指针作为输入参数 |  |
| 重要 | 仅做输入用的指针参数是否加了const？ |  |
| 重要 | 在函数体的“入口处”是否对参数的有效性进行了检查？ |  |
|  | 函数参数是否过多？ |  |
|  | 是否使用类型和数目不确定的参数？ |  |
| 重要 | 是否省略了返回值的类型？ |  |
|  | 函数名字与返回值类型在语义上是否冲突？ |  |
|  | 函数返回的正常值是否和错误标志区分开？ |  |
| 重要 | 函数的每个出口是否都有return语句 |  |
| 重要 | return语句是否返回了指向“栈内存”的“指针”？ |  |

* 内存

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
| 重要 | 字符串是否为‘\0’申请空间？（尤其是函数调用时） |  |
| 重要 | 用malloc申请内存之后，应该立即检查指针值是否为NULL。 |  |
|  | 是否为数组和动态内存赋初值？ |  |
| 重要 | 动态内存的申请与释放是否配对？ |  |
| 重要 | 用free释放了内存之后是否立即将指针设置为NULL？ |  |
| 重要 | 是否把没有初始化的指针用作右值 |  |

* 文件I/O

|  |  |  |
| --- | --- | --- |
| **重要性** | **审查项** | **结论** |
| 重要 | 是否对不存在的或者错误的文件进行操作吗？ |  |
| 重要 | 文件是否以正确的方式打开吗？ |  |
| 重要 | 文件结束判断正确吗？ |  |
| 重要 | 是否正确地关闭文件吗？ |  |

## **附录B：源文件举例**

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

/\* 瑞斯康达科技发展股份有限公司 版权所有 \*/

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

/\* @@BEGAIN\_INTERNAL\_LEGAL@@ \*/

/\* \*/

/\* Copyright(C) Description \*/

/\* Copyright(C)20\*\*-20\*\*, Raisecom Technology Co.,Ltd. \*/

/\* Unpublished work-rights reserved under the China Copyright Act. \*/

/\* Use, duplication, or disclosure by the government is subject to \*/

/\* restrictions set forth in the CDTT commercial license agreement. \*/

/\* \*/

/\* @@END\_INTERNAL\_LEGAL@@ \*/

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

/\* 程序文件名：demo.c \*/

/\* Filename：demo.c \*/

/\* 作者：李琼 \*/

/\* Author：liqiong \*/

/\* 日期：10/20/2004 \*/

/\* Date：10/20/2004 \*/

/\* 当前版本：1.0 \*/

/\* Version：1.0 \*/

/\* 实现目的：源文件范例 \*/

/\* Purpose：sourcefiledemo \*/

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

/\* 修改记录： \*/

/\* 01,10/20/2004,lq，创建demo.c \*/

/\* Modification history： \*/

/\* Index，Date，Author,content \*/

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

#include<stdio.h> /\* 操作系统标准头文件 \*/

#include<xxxxx.h> /\* 其它头文件 \*/

#include"yyyyy.h" /\* 用户头文件 \*/

#ifdef SS\_SWITCH /\*头文件切换\*/

#include<zzzzz.h>

#endif

/\* 局部常数和类型定义 \*/

enum Error\_Code

{

OUT\_OF\_MEMORY, /\* 注释 \*/

INSUFFICIENT\_DISK\_SPACE,/\* 注释 \*/

LOGIC\_ERROR, /\* 注释 \*/

FILE\_NOT\_FOUND /\* 注释 \*/

};

/\* 局部宏定义 \*/

#define LOCAL\_MACRO\_DEMO 1

/\* 局部变量 \*/

enum Error\_Code LocalVar; /\* 注释 \*/

/\* 函数声明 \*/

Any\_typesome\_fuction(Type\_1 first\_param, Type\_2 \*second\_param);

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

\* 函数名: some\_function \*

\* 描述: 函数举例 \*

\* 相关文档:<描述此函数的相关协议名称、版本及在协议中的位置> \*

\* 参数: \*

\* 参数名 类型 输入/输出 描述 \*

\* -------- ---- --- ----------- \*

\* first\_param Type\_1 输入 \*

\* second\_param Type\_2\* 输出 \*

\* \*

\* 返回值:Any\_type \*

\* return1 返回值1，表明函数...... \*

\* return2 返回值2，表明函数...... \*

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

Any\_typesome\_function(Type\_1first\_param,Type\_2\*second\_param)

{

…

return…;

}

…

## **附录C：头文件举例**

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

/\* 瑞斯康达科技发展股份有限公司 版权所有 \*/

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

/\* @@BEGAIN\_INTERNAL\_LEGAL@@ \*/

/\* \*/

/\* Copyright(C) Description \*/

/\* Copyright(C)20\*\*-20\*\*, Raisecom Technology Co.,Ltd. \*/

/\* Unpublished work-rights reserved under the China Copyright Act. \*/

/\* Use, duplication, or disclosure by the government is subject to \*/

/\* restrictions set forth in the CDTT commercial license agreement. \*/

/\* \*/

/\* @@END\_INTERNAL\_LEGAL@@ \*/

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

/\* 程序文件名：demo.h \*/

/\* Filename：demo.h \*/

/\* 作者：李琼 \*/

/\* Author: liqiong \*/

/\* 日期：10/13/2004 \*/

/\* Date：10/13/2004 \*/

/\* 当前版本：1.0 \*/

/\* Version：1.0 \*/

/\* 实现目的：头文件范例 \*/

/\* Purpose：Headfiledemo \*/

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

/\* 修改记录： \*/

/\* 02,10/20/2004,1.0, lq, 修改注释为中文 \*/

/\* 01,10/13/2004,1.0, lq, 创建头文件demo.h \*/

/\* ModiRecord: \*/

/\* Index, Date, Version, Author, content \*/

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

#ifndef \_DEMO\_H

防止demo.h被重复引用

#define \_DEMO\_H

/\*头文件\*/

#include<math.h> 引用标准库的头文件

…

#include"myheader.h" 引用非标准库的头文件

…

/\*常量和类型\*/

typedef struct demo\_struct\_s

{

常数和类型的定义

…

}demo\_struct\_t;

/\*宏定义\*/

#define DEMO\_MACRO(X) (1/((X)\*(X)))

宏定义

#define DEMO\_MACRO\_NAME "raisecom"

/\*变量定义\*/

ulong\_t demo\_variable; 变量定义

/\*函数声明\*/

void DEMO\_FUNC(…); 全局函数声明

…

#endif

## **附录D：第三方文件修改记录举例（rfc2574.c）**

/\*rfc2574.c-IDB+Modulesourceforrfc2574.mi2.\*/

/\*Copyright1998-2000WindRiverSystems,Inc.\*/

/\*Copyright2000WindRiver,Inc.\*/

#include"copyright\_wrs.h"

/\*

modificationhistory

--------------------

xxx,25Nov00,idbCreatedbyIDBGEN.Ver2.27

%%BeginUserCoderfc2574\_Modifications

原文件头

00a,26jul00,kwConvertfromzeroindexbasescalarsto1basedindex.

%%EndUserCoderfc2574\_Modifications

\*/

/\*

DESCRIPTION

ThismoduledefinestheIDB+structuresandIDB+Objecthandlerroutinesto

accesstheIDB+objectsdefinedinrfc2574.mi2.

INCLUDES:

rfc2574.h

\*/

增加的文件头

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

/\* 瑞斯康达科技发展股份有限公司 版权所有 \*/

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

/\* Copyright(C) Description \*/

/\* Copyright(C)20\*\*-20\*\*, Raisecom Technology Co.,Ltd. \*/

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

/\* 当前版本：1.0 \*/

/\* Version：1.0 \*/

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

/\* 修改记录： \*/

/\* 序号,MM/DD/YYYY,版本，修改者，修改内容(最近的修改记录在最上面) \*/

/\* Modification history： \*/

/\* Index，Date，Author,content \*/

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

原有程序

…

添加的代码

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

\*函数名:statususmUserInit() \*

\*描述:USM出厂用户初始化，主要计算出厂用户的密钥 \*

\*相关文档:rfc2574 \*

\*参数:无 \*

\* \*

\*参数名 类型 输入/输出 描述 \*

\*-------- ---- --- ----------- \*

\* …… \*

\*返回值: 0：执行成功 \*

\* 1：执行失败 \*

\*说明：无 \*

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

STATUS usmUserInit(void)

{

…添加的函数体

}

原有程序

…

## **附录E：常用缩写表**

BOOTP: BOOTSTRAP PROTOCOL

CIDR: Classless Inter Domain Routing

CoS: Class of Service

DHCP: Dynamic Host Configuration Protocol

DSCP: Differentiated Services Code Point

EAPOL: Extensible Authentication Protocol over LAN

END: Enhanced Network Device

EGP: Exterior Gateway Protocol

GARP: Generic Attribute Registration Protocol

GMRP: GARP Multicast Registration Protocol

GVRP: GARP VLAN Registration Protocol

LACP: Link Aggregation Control Protocol

ICMP: Internet Control Message Protocol

IGMP: Internet Group Management Protocol

IGP: Interior Gateway Protocol

InARP: Inverse ARP

MAC: Medium Access Control

MIB: Management Information Base

NMS: Network Management station

NVM: Nonvolatile Memory

OSPF: Open Shortest Path First

PDU: Protocol Data Unit

QoS: Quality of Service

RIP: Routing Information Protocol

RMON: Remote Network Monitoring

SNMP: Simple Network Management Protocol

SSP： Switch Support Package

STP: Spanning Tree Protocol

ToS: Type of Service

VLAN: Virtual LAN

WRR: Weighted Round Robin

## **附录F：常用简写表**

2: to

addr: address

blk: block

cfg： config

clk: clock

cmd: command

cmp: compare

cpy: copy

ctl: control

defs: definitions

dev: device

drv: driver

err: error

func: function

info: information

init: initial

lib: library

mem: memory

MSG: message

nfy: notify

param: parameter

PTR： pointer

pwd: password

rcv: receive

req: request

str: string

sync: synchronization

sys: system

tbl: table

tgt: target

usr： user

var: variable

## **附录G：基本数据类型**

**1、ENVOY模块基本数据类型**

（其它数据类型参见$WINBASE\target\src\epilogue\common\h\types.h或者文档envoyprog.pdf）

**sbits8\_t** 有符号8位整数。

**sbits16\_t** 有符号16位整数。（或者**INT\_16\_T**）

**sbits32\_t** 有符号32位整数，MIB变量的整数类型必须为此类型。（或者**INT\_32\_T**）

**OCTET\_T** 无符号8位字符。

**OIDC\_T** OID值，32位无符号整数。

**bits8\_t** 无符号8位整数。（或者**UINT\_8\_T**）

**bits16\_t** 无符号16位整数。(或者**UINT\_16\_T**）

**bits32\_t** 无符号32位整数，MIB变量的Counter、Gauge和TimeTicks必须为此类型。（或者**UINT\_32\_T**）

**UINT\_64\_T** 无符号64位整数。

**OBJ\_ID\_T** 一个ASN.1对象标识。

**PTR\_T** 空指针void\*。

**SNMPADDR\_T** 网络地址结构（snmpdefs.h）

**ALENGTH\_T** Asn1conf.h中定义

**ATVALUE\_T** Asn1conf.h中定义

**EBUFFER\_T** Asn1conf.h中定义

**2、TMS模块基本数据类型**

（$WINBASE\target\h\types\vxTypes.h）

**uchar\_t**  8位无符号整数

**ushort\_t** 16位无符号整数

**uint\_t** 32位无符号整数

**ulong\_t** 32位无符号整数

**char** 8bits

**short** 16bits

**int** 32bits

**long**  32bits

**3、驱动模块基本数据类型**

（$WINBASE\target\h\types\vxTypesold.h）

typedef char **INT8**;

typedef short **INT16**;

typedef int **INT32**;

typedef unsignedchar **UINT8**;

typedef unsignedshort **UINT16**;

typedef unsignedint **UINT32**;

**4、OTP模块基本数据类型**

**OTP\_UINT8**  8位无符号整数

**OTP\_INT8**  8位有符号整数

**OTP\_UINT16**  16位无符号整数

**OTP\_INT16** 16位有符号整数

**OTP\_UINT32**  32位无符号整数

**OTP\_INT32** 32位有符号整数

**OTP\_ULONG32**  32位无符号整数

**OTP\_LONG32** 32位有符号整数

**OTP\_ULONG64**  64位无符号整数

**OTP\_LONG64** 64位有符号整数

基本数据类型一览表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 基本数据类型 | 位数 | bootrom | TMS | ENVOY | **OTP** |
| signed char | 8 | **INT8** | **char** | **sbits8\_t** | **OTP\_INT8** |
| signed short | 16 | **INT16** | short | **sbits16\_t** | **OTP\_INT16** |
| signed long | 32 |  | **long** | **sbits32\_t** | **OTP\_INT32** |
| unsigned char | 8 | **UINT8** | **uchar\_t** | **bits8\_t** | **OTP\_UINT8** |
| unsigned short | 16 | **UINT16** | **ushort\_t** | **bits16\_t** | **OTP\_UINT16** |
| unsigned long | 32 | **ULONG** | **ulong\_t** | **bits32\_t** | **OTP\_UINT32** |
| unsigned int | 32 | **UINT32** | **uint\_t** |  |  |
| int | 32 | **INT32** | **int** |  |  |
| unsigned char  (用于字符) | 8 | **UCHAR** |  | **OCTET\_T** |  |
|  | 64 |  |  | **UINT\_64\_T** |  |
| int  (用于布尔量) | 32 | BOOL | BOOL | **BOOL** |  |

**5、函数返回数据类型**

STATUS 只用来返回函数执行的错误标志(vxTypesOld.h中定义为int)

int 用来返回函数的正常整型值

## **附录H：README文件模板**

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

/\* 瑞斯康达科技发展股份有限公司 版权所有 \*/

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

/\* @@BEGAIN\_INTERNAL\_LEGAL@@ \*/

/\* \*/

/\* Copyright(C) Description \*/

/\* Copyright(C)20\*\*-20\*\*, Raisecom Technology Co.,Ltd. \*/

/\* Unpublished work-rights reserved under the China Copyright Act. \*/

/\* Use, duplication, or disclosure by the government is subject to \*/

/\* restrictions set forth in the CDTT commercial license agreement. \*/

/\* \*/

/\* @@END\_INTERNAL\_LEGAL@@ \*/

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

目录名称：*XXX*

目录描述：*<说明本目录隶属于哪个模块，实现什么功能等>*

子目录列表：

子目录名称 描述

*xxxx yyyy*

*… …*

文件列表：

文件名称 描述

*xxxx yyyy*

*… …*