|  |  |  |  |
| --- | --- | --- | --- |
|  | **文 档 编 号** | **产品版本** | **受控状态** |
| DC-SB-2003-1005 | **V 1.0** | 内部 |
| **产品名称： 软件编程规范** | | **共 页** |

**软件编程规范**

**(仅供内部使用)**

**北京世纪百合科技有限公司**

Beijing Centurial Lily Technology Co.,Ltd.

**版权所有 不得复制**

**文档修改记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本号 | 日期 | 所改页 | 注 记 | 提交人 | 批准人 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目 录

1. 引言 4

1.1 目的 4

1.2 范围 4

2. 规范 4

2.1 文件 4

[2.2 版面风格 4](#_To )

2.3 标识符命名 8

2.4 函数与宏 10

2.5 代码的可靠性 14

3. 附录：通用类型的公共定义 19

# 引言

## 目的

本规范的目的在于增加源代码的可读性，减少程序员对代码理解上的偏差，使程序员能够编写出可靠的代码，降低代码维护成本。

## 范围

本规范内容涉及范围包括：文件、版面、注释、标识符、变量和结构、函数、宏以及可理解性等。本规范适用于公司开发的所有软件产品。在新软件的编码过程中本规范必须执行。

# 规范

## 文件

### 头文件的名称一律为小写，格式为“子系统名\_文件名.h”。例如：ipf\_protocol.h等。

### 头文件的格式如下：

* 注释头，格式参见软件编程规范；
* 头文件预编译开关开始，格式为：

#ifndef 预编译开关

#define 预编译开关

其中预编译开关格式为：“ \_文件名\_H”，其中文件名一律大写

* 头文件内容；
* 头文件预编译开关结束，格式为：

#endif

用来和头文件预编译开关的开始对应。

例如：以下为ipf\_ip.h头文件的内容：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Copyright (c) Lily Of The Century Technology Co., LTD.

ALL RIGHTS RESERVEDDescription: // 用于详细说明此程序文件完成的主要功能

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

#ifndef \_IPF\_IP\_H

#define \_IPF\_IP\_H

...

<头文件正文>

...

#endif

### 头文件的定义要有层次，禁止交叉引用。

说明：头文件的层次设置为公共模块、私有模块。头文件的引用次序为下层头文件引用上层头文件、私有头文件引用公共头文件，声明结构时尤其要注意，不允许出现交叉引用的情况。

示例：如下定义不符合规范

头文件isdn\_a.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Copyright (c) Lily Of The Century Technology Co., LTD.

ALL RIGHTS RESERVEDDescription: 定义配置数据结构

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

#ifndef \_ISDN\_A\_H

#define \_ISDN\_A\_H

#include “isdn\_b.h”

typedef struct

{

...;

ISDN\_CONTROL stIsdnControl;

...;

}ISDN\_ CONFIG;

#endif

头文件isdn\_b.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Copyright (c) Lily Of The Century Technology Co., LTD.

ALL RIGHTS RESERVEDDescription: 定义控制数据结构

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

#ifndef \_ISDN\_B\_H

#define \_ISDN\_B\_H

#include “isdn\_a.h”

typedef struct

{

...;

ISDN\_CONFIG stIsdnConfig;

...;

}ISDN\_CONTROL;

#endif

为了解决上述矛盾，可以将两个结构合并到一个文件中声明。

### 文件中如果引用系统头文件，必须使用“<“和“>“；如果引用自定义的头文件，必须使用“”“和“”“。

说明：系统头文件是指由编译系统提供的头文件。

示例：如下书写不符合规范。

#include “stdlib.h”

#include <isdn\_config.h>

应该改作：

#include <stdlib.h>

#include “isdn\_config.h”

### 头文件中只能声明变量类型，禁止定义变量。

说明：如果在头文件中定义变量，当有多个源文件引用该头文件时，会出现重复定义的错误。

示例：如下头文件是不规范的。

头文件isdn\_a.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Copyright (c) Lily Of The Century Technology Co., LTD.

ALL RIGHTS RESERVEDDescription: 定义配置数据结构

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

#ifndef \_ISDN\_A\_H

#define \_ISDN\_A\_H

typedef struct

{

...;

}ISDN\_ CONFIG;

ISDN\_CONFIG g\_stIsdnConfig;

#endif

结构变量g\_stIsdnConfig不能在头文件中定义，只能在源文件中定义。可以在相应的源文件定义后，头文件作如下改动：

ISDN\_CONFIG g\_stIsdnConfig;改为

extern ISDN\_CONFIG g\_stIsdnConfig;

### 头文件的声明顺序，应该有层次感。

说明：头文件的声明顺序一般是宏、结构、函数、变量。函数在头文件中声明时开头可以不加“extern”

### 源文件名称一律小写，格式为：子系统名\_文件名.c，

示例：ipf\_ pkt.c表示在IPF子系统的包处理文件。

### 源文件必须加注释头。

源文件注释头格式为：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Copyright (c) Lily Of The Century Technology Co.,LTD.

ALL RIGHTS RESERVEDDescription: // 用于详细说明此程序文件完成的主要功能

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

## 版面风格

### 程序块采用缩进风格编写，缩进使用TAB符号，TAB符号规定为4个空格。变量说明之后必须加空行。

说明：由开发工具自动生成的代码例外。示例：如下例子不符合规范。

void isdn\_InitConfig(void)

{

WORD wConfigNumner;

...; //program code

}

应如下书写

void isdn\_InitConfig(void)

{

WORD wConfigNumner;

...; //program code

}

### 不允许把多个短语句写在一行中，一行只写一条语句。

示例：如下例子不符合规范。

wLength = wWidth = 0;

或wLength = 0, wWidth = 0;

应如下书写

wLength = 0;

wWidth = 0;

禁止如下写法：

int i = j = 0;

或int i = 0,j = 0;

应该写作：

int i = 0;

int j = 0;

或

int i,j;

i = 0;

j = 0;

### do、while、switch、case、default、if、else、for等语句自占一行，且if、else语句在同一列。

示例1：如下例子不符合规范。do {

...; //program code

} while (pstUser != NULL);应如下书写：do

{

...; //program code

} while (pstUser != NULL);

示例2：如下例子不符合规范。while (pstUser != NULL) {

...; //program code

}应如下书写：while (pstUser != NULL)

{

...; //program code

}

示例3：如下例子不符合规范。switch (dwCounter) {

case 1: dwCounter++;

...; //program code

break;

default:break;

}应如下书写：switch (dwCounter)

{

case 1:

dwCounter++;

...; //program code

break;

default:

break;

}

示例4：如下例子不符合规范。if (pstUser == NULL) return;else {

...; //program code

}

应如下书写：if (pstUser == NULL){

return;}

else

{

...; //program code

}

示例5：如下例子不符合规范。for (i = 0;i < 10;i++) {

...; //program code

}应如下书写：for (i = 0;i < 10;i++)

{

...; //program code

}

### 函数的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格。

示例：如下例子不符合规范。

int isdn\_Config(void)

{

...; // program code

return 0;

}

应该改为

int isdn\_Config(void)

{

...; // program code

return 0;

}

示例：如下例子不符合规范。

typedef struct

{

WORD wSlot;

WORD wPort;

}ISDN\_CONFIG;

应该改为

typedef struct

{

WORD wSlot;

WORD wPort;

}ISDN\_CONFIG;

### 在switch的处理程序块中，case和default语句下的处理语句也要遵从语句缩进要求。

示例：

以下写法是不规范的

switch (wEvent)

{

case 1:

...; //program code

break;

}

应该改为：

switch (wEvent)

{

case 1:

...; //program code

break;

default:

break;

}

### 程序块的分界符（大括号'{'和'}'）应各独占一行并且位于同一列，同时与引用它们的语句左对齐。

示例：如下例子不符合规范。

if (...)

{

... // program code

}

void isdn\_ExampleFun( void ) {

... // program code

}

应如下书写。

if (...)

{

... // program code

}

void isdn\_ExampleFun( void )

{

...; // program code

}

### 函数头部应进行注释，列出：函数的名称、功能、输入参数、输出参数、返回值等。

示例：编写函数时候按照下面这段注释编写。/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Func Name: // 函数名称 Description: // 函数功能的描述 Input: // 输入参数说明 Output: // 对输出参数和函数返回值的说明\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面。

示例：如下例子不符合规范。isdn\_Init();

//初始化ISDN任务

应如下书写：//初始化ISDN任务

isdn\_Init();

### 对有实际含义的变量或者常量的注释，应放在其上方相邻位置或右方。对宏的注释，应放在上面，不可放在其右方或下方。对数据结构的声明(包括数组、结构、类、枚举等)，注释应放在其上方相邻位置，不可放在下面；对结构中的每个域的注释放在此域的右方。

示例：

以下写法不规范

#define ERROR\_MSG\_ON //错误信息打印输出开关

应改为

// 错误信息打印输出开关

#define ERROR\_MSG\_ON

### 允许使用”//”进行注释

示例：以下注释是允许的

// 根据CFM所存的配置信息更新接口配置信息

void isdn\_SetConfigData(void)

{

...; //program code

}

## 标识符命名

### 局部变量采用大小写混排的匈牙利方式命名，命名格式为前缀+变量名称，其中变量名由一个或一个以上的单词组成，每个单词首字母大写，其余一律小写。

说明：前缀要表明变量类型。下表为定义变量时候使用的前缀。

|  |  |  |
| --- | --- | --- |
| 类型 | 前缀 | 类型名称 |
| BOOLEAN | b | 布尔型 |
| BYTE | by | 无符号字符型 |
| WORD | w | 无符号短整型 |
| DWORD | dw | 无符号长整型 |
| char | c | 字符型 |
| short | s | 短整型 |
| int | i | 整型 |
| long | l | 长整型 |
|  | st | 结构类型 |
|  | un | 联合类型 |
|  | p | 指针 |
|  | pby | BYTE的指针 |
|  | pw | WORD的指针 |
|  | pdw | DWORD的指针 |
|  | pst | 结构的指针 |
|  | pun | 联合的指针 |
|  | pa | 数组的指针 |
|  | pfn | 函数指针 |
|  | pm | 消息的指针 |
|  | pc | 字符型的指针 |
|  | ps | 短整型的指针 |
|  | pi | 整型的指针 |
|  | pl | 长整型的指针 |
|  | pp(根据类型添加其他前缀) | 指针的指针 |
|  | a | 数组 |
|  | aby | BYTE数组 |
|  | aw | WORD数组 |
|  | adw | DWORD数组 |
|  | ast | 结构数组 |
|  | aun | 联合数组 |
|  | afn | 函数数组 |
|  | ac | 字符型数组 |
|  | as | 短整型数组 |
|  | ai | 整型数组 |
|  | al | 长整型数组 |
|  | sz | 以null结尾的字符串型 |

 注：类成员变量前应加前缀 m\_ ;

示例： 以下的书写不规范

WORD slot;

BYTE \*byAccess;

DWORD arp\_frame\_count;

应该写作：

WORD wSlot;

BYTE \*pbyAccess;

DWORD dwArpFrameCount;

### 普通宏的定义使用全大写字母加下划线的方式，结构为：子系统+下划线+宏内容名称。

示例：

#define IP\_MAX\_HWA\_LEN 6

#define IP\_VER(x) ((x >>4) & 0xF0)

### 函数的命名为：前缀+下划线+字符串。其中前缀为全小写的子系统名称，字符串由一个或多个单词组成，每个单词首字母大写，其他字母小写。

示例：

void MsgCreate(void);

void icmp\_PktRecv(void);

### 调试用的编译开关是开头为下划线的全大写字符串，具体格式是：下划线+DEBUG+子系统名称+下划线+名称。

示例：**\_DEBUG\_IPF\_IP\_PACKET、\_DEBUG\_TCP\_OPTION**等。

### 结构联合类型命名规则为：子系统名称+下划线+结构名称、子系统名称+下划线+联合名称，其中所有字母大写。

示例：

typedef struct

{

...;

}IP\_TRACEROUTE;

typedef union

{

...;

}ICMP\_PKT**;**

### 全局变量命名的具体格式是：前缀+下划线+子系统名+下划线+字符串，其中前缀为小写的“g”，子系统名称全部小写，字符串由一个或多个单词组成，每个单词首字母大写，其他字母小写。

示例：**g\_icmp\_TraceRouteQueue**、**g\_ospf\_InterfList**等。

### 消息宏定义的采用以下格式：前缀+下划线+字符串+下划线+后缀。其中前缀采用小写的“mm”，字符串标记消息宏的具体含义，由一个或多个单词组成，每个单词首字母大写，其他字母小写，后缀为XXXtoYYY，表示消息的传送方向，XXX表示源子系统名，YYY为目的子系统名，全部用大写字母表示。

示例：**mm\_AddRoute\_BGPtoRTMGT**表示从BGP协议发送到路由管理实体的增加一条路由的消息，**mm\_ShowIpRoute\_CLItoRTMGT**表示从CLI发送到路由管理子系统的要求显示路由表的消息。

## 函数与宏

### 对函数的返回值要仔细、全面地处理。

说明：对提供返回值的函数，尤其是接口函数，其返回值必须检查。

示例：以下函数是不规范的。

void isdn\_SendMessage(void)

{

...; //program code

MsgPost(...); // 发送消息

}

应该改作

void isdn\_SendMessage(void)

{

INT32 iPost;

...; //program code

iPost = MsgPost(...); // 发送消息

if (iPost == OK)

{

...; //处理代码

}

else

{

...; //处理代码

}

}

### 接口函数的输入参数（尤其是指针和数组下标）、非输入参数的合法性必须检查。

说明：函数的输入主要有两种：一种是参数输入；另一种是非参数输入，包括全局变量、数据文件等，这些参数都需要作检查。示例：下函数的实现不符合规范。RESULT isdn\_Receive(NI\_CELL \*pstCell,SK\_BUF \*pstSkBuf)

{

...; //接收数据代码

return OK;

}

应改作以下写法。

RESULT isdn\_Receive(NI\_CELL \*pstCell,SK\_BUF \*pstSkBuf)

{

if ((pstCell == NULL) || (pstSkBuf == NULL))

{

return FALSE;

}

...; //接收数据代码

return OK;

}

### 禁止把函数的参数作为工作变量。

说明：需要做改变的参数，应该先用局部变量代之，最后再将该局部变量的内容赋给该参数。示例：下函数的实现不符合规范。BOOLEAN isdn\_SumData(WORD wNumber,WORD \*pwData, WORD \*pwSum)

{

WORD wCount;

if ((pwData == NULL) || (pwSum == NULL))

{

return FALSE;

}

\*pwSum = 0;

for (wCount = 0; wCount < wNumber; wCount++)

{

\*pwSum += pwData[wCount]; // pwSum成了工作变量，不规范

}

return TRUE;

}

应改作以下写法。

BOOLEAN isdn\_SumData(WORD wNumber,WORD \*pwData, WORD \*pwSum)

{

WORD wCount;

WORD wSumTemp;

if ((pwData == NULL) || (pwSum == NULL))

{

return FALSE;

}

wSumTemp = 0;

for (wCount = 0; wCount < wNumber; wCount++)

{

wSumTemp += pwData[wCount];

}

\*pwSum = wSumTemp;

return TRUE;

}

### 对于没有参数的函数，要定义为void。

说明：目的减少函数间接口的复杂度。

示例：如下函数写作不规范

BOOLEAN isdn\_Init()

{

...; //program code

return TRUE;

}

应改作：

BOOLEAN isdn\_Init(void)

{

...; //program code

return TRUE;

}

### 明确函数的返回值，当函数不需要返回值时要定义为void。

示例：如下函数写作不规范

isdn\_InitCtrlBlock(void)

{

...; //program code

}

应改作：

void isdn\_InitCtrlBlock(void)

{

...; //program code

}

### 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，不符合规范。

#define MAX(a,b) a > b ? a : b

#define MAX(a,b) (a > b ? a : b)

#define MAX(a,b) (a) > (b) ? (a) : (b)

正确的定义应为：

#define MAX(a,b) ((a) > (b) ? (a) : (b))

### 将宏所定义的多条表达式放在大括号中。

示例：下面的定义不规范

#define ISDN\_INIT\_VALUE(a, b) \

a = 0;\

b = 0;

正确的用法应为：

#define ISDN\_INIT\_VALUE(a, b) \

{\

a = 0;\

b = 0;\

}

### 使用宏时，不允许参数发生变化。

示例：如下用法导致错误。#define MAX(a, b) ((a) > (b) ? (a) : (b))

iResult = MAX(iRx++,iTx);

将被预处理器解释为

iResult = ((iRx++) > (iTx) ? (iRx++) : (iTx));

此时如果输入

iRx = 6;

iTx = 5;

代码执行结果iResult = 7，此时iRx为8。

应该改作如下用法：

iResult = MAX(iRx,iTx);

iRx++;

## 代码的可靠性

### 系统运行之初，要初始化所有本系统的全局变量，禁止未经初始化的全局变量被引用。

说明：使用未初始化的数据，容易使系统进入混乱状态。

示例：以下的全局变量如果没有在系统运行时初始化，在被函数使用时会出错

ISDN\_CELL \*g\_pstIsdnCell[ISDN\_MAX\_CELL\_NUM];

void isdn\_ShowChannel(WORD wSlot,WORD wPort)

{

ISDN\_CELL \*pstIsdnCell = 0;

int i;

for (i = 0; i < ISDN\_MAX\_CELL\_NUM;i++)

{

if (g\_stpIsdnCell[i] != NULL)

{

pstIsdnCell = g\_pstIsdnCell[i];

// 如果g\_pstIsdnCell没有初始化为空，以下语句会出错

if ((pstIsdnCell->wSlot == wSlot) \

&& (pstIsdnCell->wPort == wPort))

break;

}

}

...; //other program code

}

### 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）

说明：申请内存时，如果申请失败，应该避免使用该指针。必须使用

if (p == NULL) 或if (p != NULL)进行防错处理。

### 禁止将未被初始化的内存作为零值使用。

说明：创建的数组或动态申请的内存，其初始值是不确定的，不能当作零处理。

示例：以下代码是不规范的。

void isdn\_CheckIe(BYTE \*pbyIe,WORD wIeLen)

{

WORD wTempLen;

//wTempLen未初始化，其初始值不一定为0

while(wTempLen < wIeLen)

{

...; //other program code

wTempLen += 2;

}

}

应作如下改写：

void isdn\_CheckIe(BYTE \*pbyIe,WORD wIeLen)

{

WORD wTempLen;

wTempLen = 0;

while (wTempLen < wIeLen)

{

...; //other program code

wTempLen += 2;

}

}

### 动态内存的申请与释放要配对，防止内存泄漏。

说明：内存泄漏问题要高度重视。内存泄漏会引起系统死机或崩溃，一般在系统启动很长一段时间后才发作，不易察觉，一般的测试手段也检测不到。

### 内存的申请和释放应该使用函数malloc( )和free( )

说明：内存的申请和释放应该使用函数malloc( )和free( )

，

或使用new( )和delete( )函数（特殊需求的系统函数除外）。

绝对禁止malloc( )和delete( )、new( )和free( )混用，否则会引起任务挂起等系统错误。

示例：如下写法是绝对禁止的。

void isdn\_ShowRunning(void)

{

BYTE \*pbyData;

pbyData = (BYTE \*)malloc();

if (pbyData == NULL)

{

return;

}

...; //programm code

delete(pData); //malloc与delete混用

return;

}

应该改作如下书写：

void isdn\_ShowRunning(void)

{

BYTE \*pbyData;

pbyData = (BYTE \*)new BYTE;

if (pbyData == NULL)

{

return;

}

...; //programm code

free(pbyData);

return;

}

### 防止引用已经释放的内存空间或未初始化的指针变量。函数中分配的内存，如果以后不再使用，在退出函数前应该释放。

示例：下函数在退出之前，没有把分配的内存释放。

int ExampleFunction(BYTE byLen)

{

BYTE \*pbyBuf;

pbyBuf = (BYTE \*)malloc(MAX\_LENGTH);

...; //program code, include check pBuf if or not NULL

if (byLen > MAX\_LENGTH)

{

return LENGTH\_ERROR; // 忘了释放pbyBuf

}

...; // other program code

}

应改为如下。

int ExampleFunction(BYTE byLen)

{

BYTE \*pbyBuf;

pbyBuf = (BYTE \*)malloc(MAX\_LENGTH);

...; //program code, include check pBuf if or not NULL

if (byLen > MAX\_LENGTH)

{

free(pBuf); // 退出之前释放pbyBuf

return LENGTH\_ERROR;

}

...; // other program code

}

### 在switch的处理程序块中，必须有default语句；并且每个case语句都要有对应的break。

说明：如果一个case下的处理语句需要调用下一个case的处理语句，可以将下一个case的处理语句copy过来。

示例：以下写法是不规范的

switch (wEvent)

{

case 1:

isdn\_SendEvent(wEvent);

case 2:

wEvent++;

break;

default:

break;

}

应该改为：

switch (wEvent)

{

case 1:

isdn\_SendEvent(wEvent);

wEvent++;

break;

case 2:

wEvent++;

break;

default:

break;

}

### 不要滥用goto语句。

说明：goto语句会破坏程序的结构性，除非确实需要，不要使用goto语句。

### 留心程序机器码大小（如指令空间大小、数据空间大小、堆栈空间大小等）是否超出系统有关限制。

### 使用显式的数据类型转换，避免让编译器进行隐式的数据类型转换。

说明：使用显示的数据转换一方面增加可读性，有利于代码的维护；另一方面避免隐式的数据类型转换引起的编译告警。

示例：

以下的书写不符合规范

NI\_CELL \*isdn\_Load(void)

{

NI\_CELL \*pstNiCell;

pstNiCell = malloc(sizeof(NI\_CELL));

...; //program code

return pstNiCell;

}

应改作：

NI\_CELL \*isdn\_Load(void)

{

NI\_CELL \*pstNiCell;

pstNiCell = (NI\_CELL\*)malloc(sizeof(NI\_CELL));

...; //program code

return pstNiCell;

}

### 不要使用难懂的技巧性很高的语句。

说明：高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。示例：如下表达式，考虑不周就可能出问题，也较难理解。

\*pbyState ++ += 1;

\* ++ pbyState += 1;

应分别改为如下。

\*pbyState += 1;

pbyState ++; // 此二语句功能相当于" \* pbyState ++ += 1; "

++ pbyState;

\*pbyState += 1; // 此二语句功能相当于" \* ++ pbyState += 1; "

# 附录：通用类型的公共定义

1. **常用数据类型的定义**

typedef int INT32;

typedef unsigned int UINT32;

typedef short INT16;

typedef unsigned short UINT16;

typedef char INT8;

typedef unsigned char UINT8;

typedef signed long LONG;

typedef unsigned long DWORD;

typedef unsigned short WORD;

typedef unsigned char BYTE;

typedef unsigned char BOOLEAN;

当使用无符号数时，应该优先使用BYTE、WORD、DWORD等宏定义。