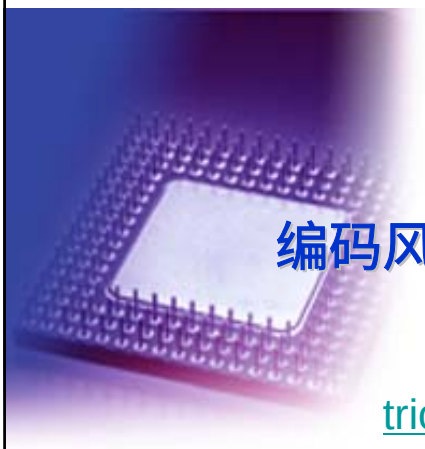
CNASIC



第 七 讲
编码风格（Coding Style）

凌 明

trio@seu.edu.cn

东南大学国家专用集成电路系统工程技术研究中心

CNASIC

目录

CNASIC

简介及说明

- 正确性
- 易维护性
- 易移植性
- 代码的高效性

CNASIC

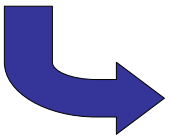
程序实例

```
float b, c[10];
void abc(void)
 {float zongfen = 0; int d;
 for( d = 0; d < 10; d ++){
 if(c[d] > 0)
 zongfen += c[b];
 b = zongfen /10;
 }
```

这段程序在做什么？
这段程序是否有错误？
这程序中存在哪些不良的书写风格？它们可能会引起什么后果？

CNASIC

```
float  b, c[10];
void abc(void)
{float zongfen = 0; int d;
for( d = 0; d < 10; d ++){
if(c[d] > 0)
zongfen += c[b];
b = zongfen /10;}
```



有了哪些改进？

你认为还有什么地方需要改进？

```
#define STUDENT_NUM     10      //学生总数
float fAvgScore;                //平均分
float fScore[STUDENT_NUM ];     //分数
/*平均分计算函数*/
void AvgScore(void)
{
      int i;
      float total_score = 0;            //总分
      for(i = 0; i < STUDENT_NUM; i++ )
      {                                 //累加计算总分
            if(fScore[i] > 0)           //遇到负分,记为0分
                  total_score += fScore[i];
      }                                 //计算平均分
      if(STUDENT_NUM >0)
            fAvgScore = total_score / STUDENT_NUM;
}
```

CNASIC

语言规则－基础

- 编写清晰表达设计思路和意图的代码
- 针对易读来优化代码, 效率的优化留给编译器去做.
- 编写可大声朗读的代码.
- 利用注释阐述和解释代码,并进行总结.
- 使用有意义且无歧义的命名方法.（推荐使用全英文的命名）
- 尽可能使用标准C 函数.
- 不要将同样的代码使用三次以上, 编写相应的函数.
- 让程序自己检查运行中的错误－编写调试代码

CNASIC

语言规则－基础

- 谨慎使用GOTO语句.
- 不要修补那些风格差的代码,重写他们.
- 不要比较两个浮点数是否相等.
- 优化代码或调试一旧版本前,备份并记录所做的修改.
- 避免机器及编译器相关的代码,如必需,隔离相关代码.
- 将编译器设为最高警告水平,把每一个警告视为错误来处理.
- 不要直接在程序中直接书写常量，应该使用常量的宏定义

CNASIC

语言规则－避免使用

- #include 的头文件没有被引用.
- 在同一个编译单元内(一般是一C文件)重复引用同一头文件.
- 在头文件内说明却仅仅在一个C文件中引用.
- 全局量仅仅在一个C文件中引用. (应该使用static 量).
- 在付值中,左右两边的数据类型不一样. (如确实必须,应该显式地进行类型转换)
- 函数返回指向函数内说明的自动变量的指针. (应该使用指向static 变量的指针).
- 删除switch case 语句中的break语句（除非两个或多个case的处理代码是完全一致的， 这时应该加以注释。）

CNASIC

语言规则—避免使用

- 两个不同类型指针间的运算.
- 隐含的数据类型转换.
- 隐含的对于变量是否为0的测试.　　(比如: “if (a = b)” ; 正确的写法是　　“if ((a = b) != 0)”)
- 缺少default 的switch 语句.
- 表达式中假设了运算顺序. (不要怕写括号)
- 忽略函数的返回值, 如果函数不需要返回值可使用 (void) f(); 但如果程序中无 返回值函数的数量太多, 则系统的设计可能有问题.

CNASIC

语言规则—依赖关系

- 模块间的依赖关系对于开发效率,可测性,可维护性都有很大的影响. 良好的依赖关系应该是简单的,层次化的, 和非循环的.
- 函数间传递的参数越少越好，减少模块件的依赖关系和耦合程度,最大程度上实现对模块的封装,将模块内的复杂性屏蔽,而对外提供简洁的数据接口。
- 尽量减少全局量的使用，局限在一个c文件中的全局量应该说明为 static。
- 对于一组在逻辑上相关的变量，应该尽量将他们封装在结构中。

CNASIC

语言规则－类型

- 推荐使用 typedef 来进行数据类型的说明。
- 所有不同类型变量间的运算，必须显式地进行类型转换。（这一点对于不同类型的指针间运算尤其重要）
- 对于没有加 unsigned 修饰的类型, 应该小心处理可能的数据溢出

CNASIC

语言规则－变量

- 在程序(函数或c文件)的开始处对变量进行说明, 将相关的变量说明放在相邻的行
- 变量的说明应该遵循一个变量一行的原则, 除非所说明的变量是紧密相关的
- 将不变的变量说明为 const
- 尽量在变量的说明行中对变量进行初始化
- 避免不必要的全局变量

CNASIC

语言规则－指针

- 什么时候使用指针?
  - 该变量在其有效期内可能表示不同的对象.
  - 该变量表示一个任意的关系, 也即其可能为空.
  - 使用指针可能有更高的效率(关键代码中)或更好的实现
- 避免无效的指针
- *假设任何指针都可能为空*
- *使用NULL来比较指针，　而不是0．仅有指针才会拥有NULL值*、、
- 使用NULL 来表示指针不指向任何对象;使用 0x0 表示数值零; 使用 '\0'表示字符串的结束.

CNASIC

语言规则－表达式

- 只有在没有更好的变通情况下使用GOTO语句
- 确保数组的存取没有越界
- 假设所有的临时变量再使用完毕后就被立刻清除

CNASIC

语言规则－函数

- 所有函数的入口参数都必须进行合法性检查
- 函数间的接口越简洁越好, 参数传递应该尽可能的简单
- 理想的函数应该仅有一个统一的返回点(出口)
- 对于某个具体的项目而言, 函数应该拥有尽量统一的返回值约定
- 函数的调用者应该检查函数的返回值
- 过深层次的嵌套调用应该充分考虑系统或该进程的堆栈大小, 防止堆栈溢出
- 每个函数前必须有相应的说明
- 所有函数的返回类型必须显式的定义， 没有返回值的函数应该说明为void

CNASIC

语言规则－函数

```
/************************************************************
*FUNCTION NAME :                  test_func
*
*ARGUMENT:
*in_arg1: brief description of the argument
*in_arg2: brief description of the argument
*in_arg3: brief description of the argument
*
*FUNCTION(S) CALLED
*function1
*function2
*
*GLOBAL VAR REFERENCED: g_var1, g_var2
*GLOBAL VAR MODIFIED:         g_var2
*
*DESCRIPTION: A detailed description of the function should be list here
*
*NOTE:  The information should be noted list here
*
************************************************************
*MODIFICATION HISTORY
mm.dd.yy  Lingming  Description of the changes made to this func
Changes should be list in reverse order
************************************************************/
```

www.cnasic.com

CNASIC

语言规则－内存及资源

- 程序申请内存或资源时, 必须检查返回值是否有效
- 所申请的资源或内存在使用完毕后, 必须显式地及时地进行释放
- 函数内部的局部量中, 不应该使用大的数组. (此时应该使用static 说明 ）

CNASIC

语言规则－源文件

```
/************************************************************
*
*Copyright © 2000 National ASIC Center, All right Reserved
*
* FILE NAME:                    test.c
* PROGRAMMER:     Lingming
* Date of Creation        2000/11/1
*
*DESCRIPTION:        Describe the function of the file
*NOTE:                 the information that should be noted of this file
*FUNCTIONS LIST:
*fun1(arg1,arg2)                    description of func1
*fun2(arg1,arg2,arg3)   description of func2
*
*GLOBAL VARS LIST:
*gVar1       description of gvar
*gVar2       description of gvar
*gVar3       description of gvar
*
************************************************************
*MODIFICATION HISTORY
*
*
* 2000/11/2 by Lingming  Description of the changes mode to the file
*changes should be listed in the reverse order
************************************************************/
```

CNASIC

语言规则－源文件

- 引用头文件的顺序
  - 系统头文件
  - 厂商头文件
  - 结构头文件
  - 应用程序头文件

CNASIC

语言规则－源文件

- 头文件多重引用检查(见 6.4)
- 文件说明(见 6.1)
- 依赖头文件引用
- #define 语句
- 全局结构与枚举定义
- 全局变量引用说明
- 全局函数引用说明

CNASIC

头文件重复引用检查

```
#ifndef     _SAMPLE_H
#define    _SAMPLE_H
…

(include file contents)

…
#endif              /*_SAMPLE_H*/
```


CNASIC

C源文件中顺序

- 文件说明(见 6.1)
- 头文件的引用
- #define 语句
- 结构及枚举说明
- 全局变量的说明
- 本地函数(static)说明
- 全局函数说明及代码 (与头文件中的顺序一致)

CNASIC

风格－程序书写

- **使用合适的注释**
- **在需要的地方写注释**
  - 预编译宏定义语句
  - 每个函数前， 说明函数的功能， 参数， 返回值，出错处理等。
  - 较为复杂的 if …else …语句。
  - 大块的，逻辑上独立的代码段， 用以说明该会代码的作用。
  - 循环语句， 说明循环的功能及跳出循环的条件。
  - 全局变量的定义处。
  - 结构定义中的分量。
  - 其它需要注释处。
- **利用括号来表示运算的优先顺序**
- **不要使用过长的语句, 必要时可以换行写**

CNASIC

风格－程序书写

- **操作符空格**
  - 逗号及分号：前面没有空格， 后面一个空格。
  - 对象引用符（., ->, []）：前后都没有空格。
  - 二进制操作符： 前后都有一个空格。
  - 逻辑运算符：前后都有一个空格。
  - ++, --, *, & : 前后都没有空格。
  - 括号：在函数调用或宏调用时，前后都没有空格；其他时， 前后都加空格。

CNASIC

风格－程序书写

- **每行语句单独占用一行**
- 使用大括号

```
for ( I = 0; I <= max; I++ )
{
   codes here
   codes here
}
```

- **在 if else while for do 语句中使用括号**

CNASIC

风格－命名

- 函数命名
- 全局变量命名
- 局部变量命名
- 宏的命名
- 文件的命名

CNASIC

标识符

标识符的命名要清晰、明了，有明确含义。使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。

标识符应当采用英文单词或其组合，切忌使用汉语拼音来命名。

坏的命名：**int a / Age1 / XueshengAge;**

好的命名：**int StudentAge;**

1.1 变量名

1.1.1 不同作用域变量的命名

- 局部变量以小写字母命名；
- 全局变量以首字母大写方式命名（骆驼式）；
- 定义类型和宏定义常数以大写字母命名；
- 变量的作用域越大，它的名字所带有的信息就应该越多。

局部变量： **int student_age;**

全局变量： **int StudentAge;**

宏定义常数：**#define STUDENT_NUM 10**

类型定义： **typedef INT16S int;**

CNASIC

标识符

1.1.2 不同类型变量的命名（匈牙利命名法）

匈牙利命名法是一种命名约定。匈牙利命名法把变量的类型 (或者它的预期使用) 等信息编码在变量名中。

一些常用的匈牙利命名法前缀

数据类型	前缀	例子
char	c	cInChar
unsigned char	uc	ucOutChar
int	i	iReturnValue
unsigned int	ui	uiControlWord
long	l	lNumRecs
float	f	fLength
double	d	dArea

1.1.3 指针变量的命名

对于指针的定义，名称大小写根据指针为全局/局部变量来定，但指针名必须以小写的“p”开头。

如：

```
int  *pDay;      //全局指针
int  *pday;      //局部指针
int **ppDay;     //指针的指针
```

1.1.4 在某一模块中使用的变量，变量名的开始需有模块名。

如：

模块 KEY.C 中的变量：

```
int  iKeyNum;
int  iKeyNumBuff[10];
```


CNASIC
标识符

1.2 函数名

- 函数名的命名应象全局变量一样采用首字母大写方式（骆驼式）。
- 函数名的开始应以“ 模块名_ ”的格式注明函数所属模块。

例如：

（1）KEY.C模块的函数

```
void KEY_Init(void);
void KEY_StartScan(void);
void KEY_StopScan(void);
```

（2）TMR.C模块的函数

```
void TMR_Init(void);
void TMR_Start(TMR_ID tmr);
void TMR_Stop(TMR_ID tmr);
```

二、排版

名字的合理选择可以帮助理解程序。同样，也应该以尽可能一目了然的形式书写语句。这就像保持书桌整洁可以使你容易找到东西一样。

2.1 用缩进格式书写代码

- 函数或过程的开始、结构定义及循环、判断等语句中的代码都要采用缩进；
- 缩进的空格数为4个；
- 使用空格键，不使用TAB键；
- 程序块的分界符（‘{’和‘}’）应各独占一行。

例：
```
for(i = 0; i < STUDENT_NUM; i++ )
    {                                   //累加计算总分
        if(fScore[i] > 0)
        {                               //遇到负分,记为0分
            total_score += fScore[i];
        }
    }
```


CNASIC

二、排版

2.2 其他书写格式

2.2.1 长语句

- 较长的语句（多于80字符）要分成多行书写；
- 长表达式要在低优先级操作符处划分新行，操作符放在新行之首；
- 划分出的新行要进行适当的缩进，使排版整齐，语句可读；
- 不允许把多个短语句写在一行中，即一行只写一条语句。

例：

修改前：
```
MeasData.TransT[dir]=TransT[dir]*SetData.Filter+TransT[dir]*(1-SetData.Filter);
```
修改后：
```
MeasData.TransT[dir] = TransT[dir] * SetData.Filter
                     + TransT[dir] * (1 - SetData.Filter);
```

www.cnasic.com

CNASIC

二、排版

2.2.2 空行和空格的使用

- 相对独立的程序块之间、变量定义之后语句开始以前必须加空行 ；
- 逗号、分号只在后面加空格 ；
- 比较操作符、赋值操作符、算术操作符、逻辑操作符、位域操作符等双目操作符的前后加空格；
- “!”、“~”、“++”、“--”、“&”等单目操作符前后不加空格；
- "->"、"."前后不加空格 。

例：

```
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

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
```


CNASIC

三、表达式和基本语句

3.1 运算表达式

- 不要编写太复杂的复合表达式；

例如：

```
i = a >= b && c < d && c + f <= g + h;  //复合表达式过于复杂
```

- 不要有多用途的复合表达式；

例如：

```
d = (a = b + c) + r ;                   //应拆分为两个语句：
a = b + c;
d = a + r;
```

- 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。

例如：

```
if(a | b && a & c)                      //不良的风格
if((a | b) && (a & c))                  //良好的风格
```

注意：

只需记住加减运算的优先级低于乘除运算，其它地方一律加上括号。

3.2 if 语句

3.2.1 布尔变量与零值比较

➢ 不可将布尔变量直接与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Visual C++ 将TRUE 定义为1，而Visual Basic 则将TRUE 定义为-1。

例：假设布尔变量名字为flag，它与零值比较的标准if 语句如下：

```
if (flag)                  // 表示flag为真时满足条件
if (!flag)                 // 表示flag为假时满足条件
```

其它的用法都属于不良风格，例如：

```
if (flag == TRUE)
if (flag == 1 )
if (flag == FALSE)
if (flag == 0)
```


CNASIC
三、表达式和基本语句

3.2.2 整型变量与零值比较

➢ 应当将整型变量用“==”或“！=”直接与0比较。

例：假设整型变量为value，它与零值比较的标准if 语句如下：

```
if (value == 0)
if (value != 0)
```

不可模仿布尔变量的风格而写成

```
if (value)                          // 会让人误解 value 是布尔变量
if (!value)
```

3.2.3 浮点变量与零值比较

➢ 不可将浮点变量用“==”或“！=”与任何数字比较。

千万要留意，无论float 还是double 类型变量，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，应当将

```
if (x == 0.0)                          // 隐含错误的比较
```

转化为

```
if ((x>=-EPSINON) && (x<=EPSINON))  //EPSINON 是精度
```

三、表达式和基本语句

3.2.3 对if 语句的补充说明

有时候我们可能会看到这样古怪的格式：

```
if (NULL == p)
```

不是程序写错了，是有经验的程序员为了防止将 `if (p == NULL)` 误写成 `if (p = NULL)`，而有意把p 和NULL 颠倒。

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。

CNASIC

三、表达式和基本语句

3.3 switch 语句

- 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）

- 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。

switch 语句的标准格式是：

```
switch (variable)
{
    case value1 : ...
        break;
    case value2 : ...
        break;
    ...
    default : ...
        break;
}
```

www.cnasic.com

四、常量

这是一个根据LCD的列来计算像素X坐标的函数：

```
INT16U LCD_GetStX(INT8U col)
{
    INT16U x;

    if(col > 29)
        col = 29;
#if LCD_MODE == _VGA
    x = 150 + (INT16U)col * 8;
#elif
    x = (INT16U)col * 8;
#endif
    return(x);
}
```

代码中的数（29，8，150）都是什么意义？
这些神秘的数给程序的阅读和维护增加了很大的难度。

四、常量

```
#define LCD_MAX_COL    29          //LCD最大列数
#define LCD_START_X    150         //LCD起始X坐标
#define LCD_COL_WIDTH  8           //LCD列宽

INT16U LCD_GetStX(INT8U col)
{
    INT16U x;

    if(col > LCD_MAX_COL )
        col = LCD_MAX_COL ;
#if LCD_MODE == _VGA
    x = LCD_START_X + (INT16U)col * LCD_COL_WIDTH ;
#elif
    x = (INT16U)col * LCD_COL_WIDTH ;
#endif
    return(x);
}
```

CNASIC

四、常量

4.1 为什么要用常量

如果不使用常量，直接在程序中填写数字或字符串，将会有什么麻烦？

（1） 程序的可读性（可理解性）变差。程序员自己会忘记那些数字或字符串是什么意思，用户则更加不知它们从何处来、表示什么。

（2） 在程序的很多地方输入同样的数字或字符串，难保不发生书写错误。

（3） 如果要修改数字或字符串，则会在很多地方改动，既麻烦又容易出错。

4.2 定义常量的方法

（1）#define 宏定义

（2）const 常量

（3）enum 枚举

CNASIC

四、常量

4.3 #define 宏定义

使用最广泛，如：

```
#define MAX_TEACHER  100
```

缺点：宏定义的常量没有类型，只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误。

4.4 const 常量

如：

```
const int MAX_STUDENT = 100;
```

优点：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。

缺点： const 常量无法在数组定义时作为数组长度。如：

```
float StudentHeight[MAX_STUDENT];          //错误
```

四、常量

4.5 枚举

```
enum{RED, BLUE, GREEN, YELLOW, WHITE, BLACK, COLOR_NUM};
```

优点：(1) 适合一次定义批量常数，尤其是在数值连续时尤为方便；

(2) 枚举常量可以作为数组长度，如：

```
float BallSize[COLOR_NUM];
```

缺点：不能定义除整型外的其他类型常量，如float和double。

五、注释

- 一个运行正常但没有注释的程序如同一个等待爆炸的定时炸弹；而在有注释，但注释不正确时，炸弹的当量更大。
- 注释应当准确、易懂，防止注释有二义性。错误的注释不但无益反而有害。

5.1 注释的基本概念

C 语言的注释符为"/*…*/"。C++语言中，程序块的注释常采用"/*…*/"，行注释一般采用"//…"。注释通常用于：

（1）版本、版权声明；

（2）函数接口说明；

（3）重要的代码行或段落提示。

虽然注释有助于理解代码，但注意不可过多地使用注释。

CNASIC

五、注释

5.2 注释的准确性

- 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。
- 注释应当准确、易懂，防止注释有二义性。错误的注释不但无益反而有害。
- 尽量避免在注释中使用缩写，特别是不常用缩写。
- 注释的位置应与被描述的代码相邻，可以放在代码的上方或右方，不可放在下方。

5.3 变量、常量的注释

- 对于所有有物理含义的变量、常量，在定义和声明时都必须加以注释，说明其物理含义。变量、常量、宏的注释应放在其上方相邻位置或右方。
- 数据结构定义和声明(包括数组、结构、类、枚举等)，必须加以注释。对数据结构的注释应放在其上方相邻位置，不可放在下面；对结构中的每个成员的注释放在此成员的右方；同一结构中不同成员的注释要对齐。
- 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。

CNASIC

五、注释

如：

```
#define  GUI_TXT_DISP_MOD      0x0C    //文本框内文字显示模式
#define  GUI_TXT_RIGHT_DISP    0x00    //文本框内文字左对齐显示
#define  GUI_TXT_LEFT_DISP     0x04    //文本框内文字左对齐显示
#define  GUI_TXT_CENTRE_DISP   0x08    //文本框内文字居中显示
typedef struct
{
    CTR_ASPECT      Aspect;           //外观
    INT8U          *pTxt;             //文字
    INT8U           TxtFont;          //字体
    INT8U           Sta;              //文本框状态模式
    INT8U           WorkMod;          //工作模式
}TXT_BOX;
```

五、注释

5.4 函数的注释

```
/*****************************************************************************
 *                          可编辑文本框选项左移                               *
 *                                                                            *
 * 功能描述    :   按向左键时，可编辑文本框的编辑位向左移一位。                *
 * 输入参数    : pbox              指向可编辑文本框的指针                     *
 * 返回参数    : 无                                                           *
 * 作    者    : Liy-tj                                                       *
 *****************************************************************************/
void GUI_EditTxtBox_Left(EDIT_TXT_BOX *pbox)
{
    … …
}
```


CNASIC

五、注释

5.5 /* */ 和 //

➢ 文件头、函数头注释使用“/* */”，函数内部注释“//”，如：

```
/*****************************************************************************
*                     GET HOW LONG KEY HAS BEEN PRESSED                      *
* Description : Function return the time the key has been pressed.           *
* Arguments   : none                                                         *
* Returns     : key down time in 'milliseconds'                              *
* Author      : Liy-tj                                                       *
*****************************************************************************/
INT32U  KEY_GetKeyDownTime (void)
{
    INT32U ctr;
    OS_ENTER_CRITICAL();            // Start of critical section of code
    ctr = KeyDownCtr;               // Get key down count
    OS_EXIT_CRITICAL();             // End of critical section of code
    return (ctr * KEY_SCAN_DLY);
}
```

www.cnasic.com