REAL研究团队C++编程规范

version 0.1

REAL研究团队C++编程规范

李泽光

November 9, 2012

Contents

文档修改记录

编号	修改者	修改时间	修改内容
v0.1	李泽光	November 9, 2012	格式建立，初稿完成

1　文档编写目的

为了保证应用程序的结构和代码风格标准化，使REAL研究团队其他成员可以协同开发或共享开发成果，特制定《REAL研究团队C++编程规范》。本规范注重于代码的物理结构和外观，而不是代码的逻辑结构，使得代码更加容易阅读、维护、管理以及修订。

本规范于2012年11月由李泽光起草制定，经REAL团队讨论后，于同月实施。从实施之日起，所有REAL团队中新增及修订的C++语言源代码都必须遵从本规范进行书写。在条件允许的情况下，团队各成员应尽量遵从本规范修订在该日期之前已有的源代码。

需要对本规范进行修订增删时，须由REAL团队进行集体讨论，先达成共识，再进行修改。

2　命名规则

通则：

1. 所有命名都应使用标准的英文单词或缩写，不得使用拼音或拼音缩写，除非该名字描述的是中文特有的内容，如半角、全角，声母、韵母等。

2. 所有命名都应遵循达意原则，即名称应含义清晰、明确。

3. 所有命名都不宜过长，应控制在规定的最大长度以内。

5. 命名中，每个单词的第一个字母应该大写，单词与单词之间直接连接，用大写字母加以区别。

6. 所有命名都应尽量使用全称，如果使用缩写，则应该使用《通用缩写表》（见附录）中的缩写。原则上不推荐使用《通用缩写表》以外的缩写，如果使用，则必须对其进行注释和说明。

7. 命名的长度应当符合“min-length && max-information”原则。一般来说，长名字能够更好地表达含义。单字符的名字也是有用的，常见如i、j、k、n、x、y、z等，它们通常可用作函数内的局部变量。

2.1　工程命名

工程项目的意义名称根据REAL团队讨论决定，在此工程意义名称的前面添加大写的“TR”（Tsinghua REAL）作为此工程项目的工程命名。如，工程项目的意义名称为Model，工程名称为TRModel。

2.2　类(CLASS)的命名

所有的类(CLASS)的名字都必须以“CTR”开头，“CTR”后第一个字母应该大写，例如CTRImage。所有对于类(CLASS)的头文件和代码文件，必须遵循VC的命名规则，去掉字母C即可，例如TRImage.h，TRImage.cpp。

2.3 函数的命名

2.3.1 全局函数

全局函数的命名必须符合：限定名词 + 动词 [+ 名词] 的原则，例如：long ModelGetCenter(double &x, double &y) 的Model为限定名词、Get为动词、Center为名词。限定名词需要根据工程名称进行规定。

2.3.2 类成员函数

类成员函数的命名必须符合：动词 [+ 名词] 的原则，例如CTRImage class中void GetWidth(double *Width)中Get为动词、Width为名词。类的成员函数可以只使用“动词”，被省略掉的名词就是对象本身。例如

```
ModelDrawImage ();      // 全局函数
Image->Draw ();      // 类的成员函数
```

2.4 宏

宏的命名必须全部采用大写字母，并且用下划线分割单词，例如：

```
#define MAX_IMAGE_LENGTH 1024
```

2.5 文件的命名

文件名长度不限于8.3格式，建议不多于30个字符。

2.5.1 头文件(H)、代码文件(CPP)

* 对于类(CLASS)的头文件和代码文件的命名遵循VC的命名规则，在第2.2中已有规定。

* 对于全局函数的头文件和代码文件的命名，应该遵循：TR+限定名词+[其他]+cpp/H的格式，例如：TRRmodelCreat.h和TRRmodelCreat.cpp。其中TR为文件名前缀，Rmodel为限定词，Creat表明此代码的内容。

2.5.2 资源文件和def文件

资源文件和def文件应该与相应的工程文件同名，只不过后缀不同。

2.6 变量的命名

变量的命名应该遵循匈牙利命名法，即：[限定词+‘_’+] 类型缩写+意义名词。最终的变量名总长不得超过32个英文字符。

2.6.1 限定词和类型缩写

限定词标明一个变量的可见范围，一般包括以下类型：

Table 1: 变量限定词

限定词	说明	例子
无	局部变量	
m_	类的成员变量	int m_Width
ms_	类的静态成员变量	static int ms_InitValue
s_	静态全局变量	static int s_InitValue
g_	普通全局变量	int g_HowManyPeople

类型前缀标示的是一个变量的类型，一般来讲我们作如下定义。

Table 2: 变量类型前缀

前缀	说明
c	char, TCHAR
s	字符串
n	int, _int16, _int32, _int64
u	unsigned
l	long
ul	unsigned long
f	double, float
by	BYTE
w	WORD
dw	DWORD
fn	function
p	pointer

2.6.2 全局变量的命名

全局变量的限定词为“g”，所以全局变量必须以小写字母“g_”开始，例如int g_nImageNumber中“g”表示全局变量，“n”表示此变量为int型，“ImageNumber”表示此变量的意义。

2.6.3 局部变量的命名

局部变量不必要加限定词，即：类型缩写+意义名词。例如int nImageNumber，其中“n”表示此变量为int型，“ImageNumber”表示此变量的意义。

2.6.4 类中的成员变量的命名

类中的成员变量命名的限定词为字母“m”，所以类中的成员变量命名必须以小写字母“m_”开始。例如：int m_nImageNumber中，“m”表示类中变量，”n”表示此变量为int型，“ImageNumber”表示此变量的意义。

2.6.5 静态变量的命名

静态变量加限定词为“s”，所以对于类中的成员静态变量命名必须以小写字母”ms_”开始。例如：int ms_nImageNumber中，“ms”表示类中成员静态变量，“n”表示此变量为int型，“ImageNumber”表示此变量的意义。

对于局部的静态变量命名必须以小写字母”s_”开始。例如int s_nImageNumber，其中“s”表示静态变量，“n”表示此变量为int型，“ImageNumber”表示此变量的意义。

2.6.6 指针变量的命名

所有的指针的类型缩写前都必须加前缀“p”，如对于类中成员指针变量必须以小写字母“m_p”开始。例如int *m_pnImageNumber中，“m”表示类中成员变量，“pn”表示变量为int *型，“ImageNumber”表示此变量的意义。

2.6.7 常量的命名

常量必须全部用大写字母，不需要加前缀。当常量由多个单词组成时，各单词由“_”加以分割，并且用const来定义常量。例如：

```
const int MAX_NUMBER = 100; // C++语言的const常量
const float PI = 3.14159; // C++语言的const常量
```

在程序中，需要对外公开的常量放在头文件中，不需要对外公开的常量放在定义文件的头部。

2.6.8 参数的命名

参数的命名和局部变量的命名相同，即：类型缩写+意义名词。例如：long GetImageNumber(int &nImageNumber)之中，“n”表示参数为int型，“ImageNumber”表示参数的意义。

2.7 其他

2.7.1 类型(TYPEDEF)命名

类型(typedef)的命名不作具体要求，但是其第一个字母应该用大写字母。例如：

```
typedef PWORD IMGPTR; // 16 bit image buffer pointer
或
typedef PWORD Imgptr; // 16 bit image buffer pointer
```

2.7.2 枚举(enum)

枚举类型的命名中，单词与单词之间直接连接，用大写字母加以区别，并且加大写字母“EM”作为前缀。枚举列表之的命名必须遵循敞亮的命名规则，全部用大写字母，当有多个单词组成时，各单词由“_”加以分割，例如：

```
enum EMClockDirecter{CLOCKWISE = 1, ANTICLOCKWISE = -1};
```

2.7.3 联合(union)

联合类型的命名中，单词与单词之间直接连接，用大写字母加以区别，并且加大写字母“UN”作为前缀。联合包含的类型数据的命名必须遵循局部变量的命名规则，例如：

```
union UNKnow    // Declare union type
{
   char cType;   // Declare member types
   int nNumber;
   float fValue;
}
```

2.7.4 结构(struct)

结构类型的命名中，单词与单词之间直接连接，用大写字母加以区别，并且加大写字母“ST”作为前缀。结构包含的类型数据的命名必须遵循局部变量的命名规则，例如：

```
struct STPerson    // Declare struct type
{
   int nAge;   // Declare member types
   float fWeight;
```

```
}
```

3 注释规则

使用代码注释的目的主要有：
1. 用文字说明代码的作用（即为什么编写该代码，而不是如何编写）；
2. 明确指出该代码的编写思路和逻辑方法；
3. 使人们注意到代码中的重要转折点；
4. 使代码的阅读者不必在他们的头脑中仿真运行代码的执行过程。
需要注意的是，空行和空白字符也是一种特殊的注释。
注释可以与语句在同一行，也可以在上行，禁止在下面注释。
规定在所有的注释中都以“//”开始，“/*”和“*/”之间的代码仅表示此段代码暂时不用。在注释中所有的标示符都必须用窄字符。

注释的两种方式如下：
* 简单的说明，必须与代码同一行或上行，用“//”开始，例如：

```
if(fValue > 0) // if value greater than 0. execute ...
```

* 详细的说明，必须在代码上方，用“//@Begin Description:”标示开始进行说明，用“//@End Description”标示结束。例如：

```
//@Begin Description:
//if value is greater than 0. execute ...
//@End Description
if(value > 0)
{
    ......
}
```

3.1 工程的注释

工程的注释说明应该写入工程的说明文档，包括：
1. 项目的委托单位、开发单位或部门（或涉及人员）；
2. 该软件系统与其他系统的关系；
3. 定义列出用到的专门属于的定义和缩写词。

3.2 代码文件注释

代码文件主要是指h文件和cpp文件。每个代码文件赋予一个注释标头。代码文件的注释标头内容包括：单位、版权、作者名称、时间等。注意一个cpp必须有至少一个人来完成（属于Modfier）。
实际书写时，请按照下例写注释标头：

```
//************************************************
//@File Name: RAngle.h: general description, one line
//@version : V1.0
//@Copyright: (c) 1998-1999 THU, DEP, REAL
//@Designer : ***
//@Creater : ***
//@Begin Date: 2001.10.9
//@End Date: 2001.11.9
//***************** First Modify******************
//@Modifier: //修改人:
//@Modify Date: 2001.11.9
//@Modify Context: xxx
//************************************************
```

3.3 类的注释

类的说明注释应该放在头文件中（H文件中，CPP文件中无），其注释的模板为:

```
//@Begin Description:
//Descriptions of the class
//@End Description

//@Begin Example:
//Examples of the class
//@End Example

class CTRImage
{
   ......
}
```

如果对类中的内容进行注释，就按照通用注释方法进行。

3.4 函数注释

函数的简单说明应该放在头文件中，详细的说明应放在CPP文件中。
在头文件(H文件)中的注释模板为:

```
//@Begin Description:
```

```
//Get the bits depth of the tray image. the default value
//is 8;
@End Description
   size_t GetBits() const;
```

或者为

```
   size_t GetBits() const; // Get the bits depth
```

在CPP文件中，函数的注释模板为(放在文件头)：

```
//*************************************************************
//@Function Name:
//@Parameter [in] : int a—
//@Parameter [out] : int b—
//@Parameter [in,out] : int c—
//@Description:
//@Return Value:
//       为1，表示：
//       为0，表示：
//@Example: Demo about the function
//*************************************************************
```

3.5 变量注释

变量的注释根据需要在同一行加以说明，例如：

```
int nImageNumber //explain the number of image
```

3.6 常量注释

常量的注释同变量的注释，例如：

```
const int COLOR_NUMBER = 255 // description
```

4 代码的格式化

基本要求如下：
* 程序结构清晰，简单易懂，单个函数的程序函数推荐在60行以内。
* 不要随意定义全局变量，尽量使用局部变量。

* 循环、分支层次不要超过五层。
* 用 IF 语句来强调只执行两组语句中的一组，禁止 ELSE GOTO 和 ELSE RETURN。
* 用 CASE 实现多路分支。
* 函数只有一个出口。

4.1 类(CLASS)的书写顺序

规定类(CLASS)中内容的书写顺序如下：

Public:
    1. Type Define 类型定义
Public:
    2. Constructor Function
    3. Destroy Function
Public:
    4. Property or Data Member
Public:
    5. SetParalmeter Method
    6. Execute Method
    7. GetPalameter Mehod
Protected:
    8. Type Define 类型定义
Protected:
    9. Data Member
Protected:
    10. Method
Privated:
    11. Type Define 类型定义
Privated:
    12. Data Member
Privated:
    13. Method

4.2 变量的初始化

所有变量必须初始化，尤其是指针必须初始化为 NULL 。

```
pSrc = NULL // 良好的风格
pSrc = 0 // 不良的风格
```

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。如果引用了未初始化的变量，可能会导致程序错误。例如：

```
int nWidth = 10; // 定义并初始化Width
```

4.3 对齐方式

在软件中，所有的对齐都用TAB进行对齐（虽然TAB=4个空格，但是为了保证一致，不采用空格）

```
while(……)
{
    if(……)
    {
        for(……)
        {
            switch(……)
            {
                case 1:
                ……
            } // switch(……)
        } // for(……)
    } // if(……)
} // while(……)
```

4.3.1 条件编译宏

所有的条件编译宏都必须靠最左边。

4.3.2 括号“{”、“}”的对齐方式

程序的分界符“{”、“}”应独占一行并且位于同一列，同时与引用它们的语句对齐。{ }之内的代码块在“{”右边一个TAB处左对齐。例如：

```
for ( ; ; )
{ //
    ......
} // 良好的风格

for ( ; ; ) { //
```

```
..... //
} // 不良的风格
```

4.3.3 循环对齐方式

循环体必须另起一行，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{ }。例如：

```
for ( ; ; )
{ // ok
    do something (); // 良好的风格
} // ok

for ( ; ; ) dosomething (); // 不良的风格
```

4.3.4 if对齐方式

if语句独占一行，执行语句不得紧跟其后。不论执行语句有多少都需要加{ }，并且if要和其后程序体的“{”、“}”和“else”在同一列。例如：

```
if (...)
{
    .....
}
else
{
    if (...)
    {
        .....
    }
    else
    {
        .....
    }
}
```

4.4 关于goto的使用

程序中尽量避免goto的使用，虽然使用goto可以带来程序结构等许多方面的好处，但是它也会带来许多灾难性的后果，而这些错误都是很难查出来的。在必须使用goto的情况下，尽量使用do {.....} while(false)代替。

4.5 空行的使用

程序中，空行起着分隔程序段落的作用。空行得体将使程序的布局更加清晰。

首先，在每个类声明、每个函数定义结束之后都要加空行，例如：

```
// 空行
void Function1(......)
{
    ......
}
// 空行
void Function2(......)
{
    ......
}
// 空行
void Function3(......)
{
    ......
}
```

在一个函数体内，逻辑上密切相关的语句之间不加空行，其他地方应加空行分隔。例如：

```
// 空行
while (condition)
{
    statement1;
    // 空行
    if (condition)
    {
        statement2;
    }
    else
    {
        statement3;
    }
    // 空行
    statement4;
}
```

4.6 空格的使用

空格的使用需要注意以下内容:

* 关键字之后要留空格。类似const、virtual、inline、case等关键字之后至少要留一个空格，否则无法辨析关键字。类似if、佛如、while等关键字之后应留一个空格再跟左括号“(”，以突出关键字。

* 函数名之后不要留空格，紧跟左括号“(”，以与关键字区别。

* “(”向后紧跟，“)”、“,”、“;”向前紧跟，紧跟处不留空格。

* “,”之后要留空格，例如 Function(x, y, z)。如果“;”不是一行的结束符号，其后要留空格，例如 for (initialisation; condition; update)。

* 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“=”、“+=”、“>二元操作符的前后都应当加空格。

* 一元操作符如“!”、“～”、“++”、“－”、“&”（地址运算符）等前后不加空格。

* 类似“[]”、“.”、“->”这类操作符前后不加空格。

* 对于表达式比较长的for语句或if语句，为了紧凑起见可以适当去掉一些空格，例如 for (i=0; i<10; i++)和 if((a<=b) && (c<=d))等。

下面是一些典型的空格使用规范例子:

```
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
switch (Value)           // 良好的风格
switch(Value)            //不良的风格
while (Value)            // 良好的风格
while(Value)             //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
x = a < b ? a : b;       // 良好的风格
x=a<b?a:b;               // 不良的风格
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.Func();                // 不要写成 a . Func();
b->Func();               // 不要写成 b -> Func();
```

4.7 其他

4.7.1 长行拆分

程序中，代码行的最大长度宜控制在70至80个字符以内。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，新行的起始位置应该与“(”或操作符之后对齐，使排版整齐，语句可读。例如：

```
for (very_long_initialization;
      very_long_condition;
      very_long_update)
{
       dosomething();
}
if ((very_long_variable1 >= very_long_variable2)
   && (very_long_variable3 >= very_long_variable4)
   && (very_long_variable5 >= very_long_variable6))
{
       dosomething();
}
variable = long_variable1 + long_variable2
             + long_variable3 + long_variable4;
```

4.7.2 修饰符的位置

修饰符 * 和 & 应该紧靠变量名。例如：

```
char *name;
int *x, y;       // 此处y不会被误解为指针
```

5 附录：通用缩写表

说明：

1. 本缩写表中列出的都是通用性缩写,不提供标准缩写,如:Win9x、COM 等。
2. 使用本缩写表里的缩写时,请对其进行必要的注释说明。
3. 除少数情况以外,大部分缩写与大小写无关。

Table 3: 通用缩写表

#	缩写	全称
1	addr	Address
2	adm	Administrator
3	app	Application
4	arg	Argument
5	asm	assemble
6	asyn	asynchronization
7	avg	average
8	DB	Database
9	bk	back
10	bmp	Bitmap
11	btn	Button
12	buf	Buffer
13	calc	Calculate
14	char	Character
15	chg	Change
16	clk	Click
17	clr	color
18	cmd	Command
19	cmp	Compare
20	col	Column
21	coord	coordinates
22	cpy	copy
23	ctl / ctrl	Control
24	cur	Current
25	cyl	Cylinder
26	dbg	Debug

续表：

27	dbl	Double
28	dec	Decrease
29	def	default
30	del	Delete
31	dest / dst	Destination
32	dev	Device
33	dict	dictionary
34	diff	different
35	dir	directory
36	disp	Display
37	div	Divide
38	dlg	Dialog
39	doc	Document
40	drv	Driver
41	dyna	Dynamic
42	env	Environment
43	err	error
44	ex/ext	Extend
45	exec	execute
46	flg	flag
47	frm	Frame
48	func / fn	Function
49	grp	group
50	horz	Horizontal
51	idx / ndx	Index
52	img	Image
53	impl	Implement
54	inc	Increase
55	info	Information

续表：

56	init	Initial/Initialize/Initialization
57	ins	Insert
58	inst	Instance
59	INT / intr	Interrupt
60	len	Length
61	lib	Library
62	lnk	Link
63	log	logical
64	lst	List
65	max	maximum
66	mem	Memory
67	mgr / man	Manage / Manager
68	mid	middle
69	min	minimum
70	msg	Message
71	mul	Multiply
72	num	Number
73	obj	Object
74	ofs	Offset
75	org	Origin / Original
76	param	Parameter
77	pic	picture
78	pkg	package
79	pkt	packect
80	pnt / pt	Point
81	pos	Position
82	pre / prev	previous
83	prg	program
84	prn	Print

续表：

85	proc	Process / Procedure
86	prop	Properties
87	psw	Password
88	ptr	Pointer
89	pub	Public
90	rc	rect
91	ref	Reference
92	reg	Register
93	req	request
94	res	Resource
95	ret	return
96	rgn	region
97	scr	screen
98	sec	Second
99	seg	Segment
100	sel	Select
101	src	Source
102	std	Standard
103	stg	Storage
104	stm	Stream
105	str	String
106	sub	Subtract
107	sum	summation
108	svr	Server
109	sync	Synchronization
110	sys	System
111	tbl	Table
112	temp / tmp	Temporary
113	tran / trans	translate/transation/transparent

续表：

114	tst	Test
115	txt	text
116	unk	Unknown
117	upd	Update
118	upg	Upgrade
119	util	Utility
120	var	Variable
121	ver	Version
122	vert	Vertical
123	vir	Virus
124	wnd	Window