C++代码编写规范

C++代码编写规范

1 引言

1.1 本文目的

随着越来越多的项目需要使用C++来编写，为了便于公司内部各开发人员之间交流项目源程序、部门经理对软件工程师编写的代码进行代码审核、保证源程序的可读性，故制定本《C++代码编写规范》，本文的读者为公司内全部软件研发人员，以期在编码过程之中，保持一致的风格，有利于软件工程项目的推行。本文所提汲的内容若不明确指出，皆为强制遵守的风格。本规范也是各开发部经理及公司技术主管对软件工程师编写的代码进行审查的依据。

1.2 背景

在软件工程领域，源程序的风格统一标志着可维护性、可读性，是软件项目的一个重要组成部分。而目前还没有成文的编码风格文档，以致于很多时候，程序员没有一个共同的标准可以遵守，编码风格各异，程序可维护性差、可读性也很差。

目前在编码上也有许多相关的风格约定，包括匈牙利命名法（一种变量的取名办法）、类取名方式，但是内容大多比较少、零散。本规范从命名规则、程序版式等几个代码编写的主要环节出发，对C++代码的编写方法、风格作出了明确的说明，力求公司所有研发部门编写的产品代码都能形成一致的风格。

1.3 术语

系统：指一个软件工程项目，是一个系统；

项目：指一个Visual C++项目；

MFC：Microsoft Foundation Class Library

1.4 参考资料

《高质量程序设计指南-C++/C语言》

《软件开发的科学与艺术》

《计算机软件工程规范国家标准汇编2000》

2 概述

每位程序员都有自己的编程风格，因为每位程序员都有自己的学习过程，就象每个人的个性一样，所以编程风格是风彩各异、百花齐放；而从软件工程理论、实践来看，现代软件是多人合作的结晶，编程风格是否统一，直接关系到软件项目的可读性、可维护性、培训，继而对软件开发成本有着直接的关系，编程风格一致，软件项目易培训，其它人员接手老项目的时间缩短，便于程序员之间的交流。编程风格混乱，则其它人员接手老项目时间增长，同时随着项目的不断开发，项目或者单个源程序文件内有着多种编程风格，这样不利于整个项目的开展以及程序员之间的交流。

本规范在参考业界已有的编码风格的基础上，描述了一个基于Visual C++编译器的项目风格，力求一种统一的编程风格，并从代码文件风格、函数编写风格、变量风格、注释风格几个方面进行阐述。

3 文件结构

3.1 头文件结构

头文件由三部分组成：

（1）、版权和版本声明。

（2）、预处理块。

（3）、函数和结构声明。

示例：

```
/**Copyright (c) 2000 ~ 2002,北京XXXX科技有限公司
* All rights reserved.
*
* 文件名称：myheader.h
* 文件标识：无

* 摘　　要：……
*
* 当前版本：2.6
* 作　　者：XXXX
* 完成日期：2002年7月10日
*
* 取代版本：2.2
* 原作者　：XXX
* 完成日期：2001年9月10日
*/


#if !defined(_MYHEADER_H)  //防止myheader.h被重复引用
#define _MYHEADER_H


#include <math.h>              //引用标准库
#include "otherheader.h"       //引用非标准库


void MyFunction1(…);           //全局函数声明

class CMyClass
{
   …
}
#endif
```

注意事项：

（1）、头文件尽可能只存放“声明”，不存放“定义”。

（2）、头文件中尽量不使用全局变量。

3.2 文件结构

文件由三部分组成：

（1）、版权和版本声明。（见3.1内容）

（2）、头文件引用。

（3）、程序实现体。

示例：

```
//版权和版本声明。（见3.1内容）

#include "myheader.h"          //引用非标准库

void MyFunction1(…)            //全局函数实现体
{
   …
}

void CMyClass::CMyClass()      //类成员函数的实现体
{
   …
}
```

3.3 文件生成（推荐风格）

对于规范的VC派生类，尽量用Class Wizard生成文件格式，避免用手工制作的头文件/实现文件。

无论是MFC源文件还是由App Wizard生成的文件，会发现在这些类中有以下注释；

```
// Constructors

// Attributes

// Operations

// Overridables

// Implementation
```

每一次类都至少有一个//Implementation，在不同的位置MFC做不同的处理，在编写代码时最好与MFC这种风格一致。

3.4 文件目录结构

当软件项目较大，文件较多时，**建议**头文件(".h")保存于include目录，定义文件(".cpp")保存于source目录，资源文件保存于res目录，工程文件保存于根目录。

4 程序版式

4.1 空行

文件之中不得存在无规则的空行（比如说连续十个空行），函数与函数之前的空行为1行。在函数体内部，在逻辑上独立的两个函数块可适当空行，一般为 1 行。

空行使用规则如下：

（1）、在每个类声明之后、每个函数定义结束之后都要加空行。

（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他地方应加空行分隔。

示例：

函数之间的空行	函数内部的空行

// 空行
void Function1(…)
{
…
}
// 空行
void Function2(…)
{
…
}
// 空行
void Function3(…)
{
…
}
	// 空行
	while (condition)
	{
	statement1;
	// 空行
	if (bCondition)
	{
	statement2;
	}
	else
	{
	statement3;
	}
	// 空行
	statement4;
	}

4.2 代码行

代码行使用规则如下：

（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。

（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{}。

示例：

不良风格的代码行	良好风格的代码行

Int width, height, depth; // 宽度高度深度	Int nWidth; // 宽度
	int nHeight;// 高度
	Int nDepth;// 深度

x = a + b; y = c + d; z = e + f;	x = a + b;//此处仅仅说明格式，一般
	y = c + d;//变量不可起名如此简单
	z = e + f;

if (width<height) DoSomething ();	if (nWidth < nHeight)
	{
	DoSomething();
	}

for (initialization; condition; update)
dosomething();
other();
	for (initialization; condition; update)
	{
	DoSomething();
	}
	//空行
	Other();

（3）、尽可能在定义变量的同时初始化该变量（就近原则）。禁止引用未被初始化的变量。

示例：

int nWidth = 10; // 定义并初始化 width

int nHeight = 10; // 定义并初始化 height

int nDepth = 10; // 定义并初始化 depth

4.3 代码行内的空格

（1）、关键字之后要留空格。像 const、virtual、line、case 等关键字之后至少要留一个空格。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。

（2）、函数名之后不要留空格，紧跟左括号“（”以与关键字区别。

（3）、“（”向后紧跟，“）”、“，”、“；”向前紧跟，紧跟处不留空格。

（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)。

（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另空格。

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

（7）、像“[ ]”、“.”、“->”这类操作符前后不加空格。

（8）、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))

示例：

void Funcl (int nX, int nY, int nZ);	//良好的风格
void Funcl (int x,int y,int z);	//不良的风格
if (nYear >= 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; 或 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.Function();	//不要写成 a . Function();
b->Function();	//不要写成 b -> Function();

4.4 对齐

（1）、每一个嵌套的函数块，使用一个TAB缩进，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。

（2）、{}之内的代码块在‘{’右边数格处对齐。

示例：

不良风格的代码行	良好风格的代码行

Void Function(int x){
… // program code
}
 	
if (condition){
… // program code
else {
… // program code
}
 	
for (initialization; condition; update){
… // program code
}
 	
while (condition){
… // program code
}
	Void Function(int nX)
	{
	… // program code
	}
	if (condition)
	{
	…// program code
	}
	else
	{
	… // program code
	}
	for (initialization; condition; update)
	{
	… // program code
	}
	while (condition)
	{
	… // program code
	}
	如果出现嵌套的{}，则使用缩进对齐，如:
	{
	…
	{
	…
	}

```
    …
}
```

4.5 长行拆分

（1）、代码行最大长度宜控制在70至80个字符以内。

（2）、长表达式要在低优先级操作符处拆分成为新行，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。

示例：

```
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();
}
```

4.6 修饰符的位置

将修饰符 * 和 & 紧靠变量名。例如：

```
char   *name;
int    *x, y;          // 此处y不会被误解为指针，但是一般变量不应该这样定义要分做两行
```

4.7 注释

单行注释用双斜杠进行注释；多行注释用/* */进行注释；在封存的某一版本的源代码之中不得存在由于调试而留下的大篇的注释。

注释一行不要太多，一般60个字符以内（保证VC集成编辑环境的可见区域之内），如有超过，则换行处理。

注释通常用于：版本、版权声明；函数接口说明；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。注意遵守以下规则：

（1）、注释是对代码的“提示”，而不是文档。程序中的注释不可喧宾夺主，注释太多了会让人眼花缭乱。注释的花样要少。

（2）、如果代码本来就是清楚的，则不必另注释。否则多此一举，令人厌烦。例如：

i++; // i 加 1。多余的注释

（3）、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掉。

（4）、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）、尽量避免在注释中使用缩写，特别是不常用的缩写。

（6）、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。

（7）、当代码比较长，特别是有多重嵌套时，应当在一些段落的结束处加注释，便于阅读。

示例1

```
/*
* 函数介绍：
* 输入参数：
* 输出参数：
* 返回值：
* /
void Function(float x,float y,float z)
{
…
}
```

示例2

```
if (…)
{
…
while(…)
{
…
}//end of while
…
}//end of if
```

4.8 类的版式

本规范要求类的版式采用如下方式：

本规范使用以行为为主的模式，即先写函数的定义后写变量的定义。

本规范遵循先公有后私有，即将public类型的函数写在前面，然后是protected而将private类型的数据写在最后。

示例：

```
class A
{
   public:
      void Func1(void);
   protected:
      void Func2(void);
   private:
      void Func3 (void);
   public:        //先函数后变量，使得函数和变量分开便于查找，否则混在一起眼花缭乱
      int m_nWidth;
   protected:
      int m_nSize;
   private:
      int m_nStudentNum;
 }
```

5 命名规则

5.1 基本规则

因为公司各研发部门使用的主要C++工具是VC++，因此命名规则采用匈牙利命名法结合VC++的原则。其规则为：<scope><BaseTag><Name>。

scope表示变量的作用域，全局变量用g_开始，类成员变量用m_，**局部变量一般不加scope**，如果函数较大则可考虑用l_用以显示说明其是局部变量。如下表所示：

scope	类型	例子
g_	Global Variable	g_Servers
m_	Member variable	m_pDoc, m_nCustomers
l_	Local variable	l_nValue

BaseTag代表数据类型，常用数据类型所对应的基本标记如下表所示：

BaseTag	类型	描述	例子
v	void	void	void* pvBuffer;

bt	BYTE	unsigned char	BYTE btData;
ch	char	8-bit character	char chGrade;
ch	TCHAR	16-bit character if _UNICODE is defined	TCHAR chName;
wch	WCHAR	16-bit Unicode character	WCHAR wchName;
b	BOOL	Boolean value	BOOL bEnabled;
n	int	Integer (size dependent on operating system)	int nLength;
un	UINT	Unsigned value (size dependent on operating system)	UINT unLength;
w	WORD, USHORT,	16-bit unsigned value	WORD wPos;
sh	SHORT	16-bit signed value	short shNum;
dw	DWORD	32-bit unsigned integer	DWORD dwRange;
l	LONG	32-bit signed integer	LONG lOffset;
ul	ULONG	32-bit unsigned integer	ULONG ulData;
f	float	32-bit floating-point number	float fData;
db	double	64-bit floating-point number	double dbData;
p	*	Ambient memory model pointer	CDocument* pDoc;
lp	FAR*	Far pointer	FAR* lpDoc;
psz	LPSTR	32-bit pointer to character string	LPSTR pszName;
psz	LPCSTR	32-bit pointer to constant character string	LPCSTR pszName;
psz	LPCTSTR	32-bit pointer to constant character string if _UNICODE is defined	LPCTSTR pszName;
h	handle	Handle to Windows object	handle hWnd;
lpfn	(*fn)()	callbackFar pointer to CALLBACK function	lpfnAbort
hr	HRESULT	LONG	HRESULT hr;
C	Class	Class or structure	CDocument, CPrintInfo

Windows对象名称缩写：

Windows 对象	**例子变量**	**MFC类**	**例子对象**
HWND	hWnd;	CWnd*	pWnd;

HDLG	hDlg;	CDialog*	pDlg;
HDC	hDC;	CDC*	pDC;
HGDIOBJ	hGdiObj	CGdiObject*	pGdiObj
HPEN	hPen;	CPen*	pPen;
HBRUSH	hBrush;	CBrush*	pBrush;
HFONT	hFont;	CFont*	pFont;
HBITMAP	hBitmap;	CBitmap*	pBitmap;
HPALETTE	hPalette;	CPalette*	pPalette;
HRGN	hRgn;	CRgn*	pRgn;
HMENU	hMenu;	CMenu*	pMenu;
HWND	hCtl;	CStatic*	pStatic;
HWND	hCtl;	CButton*	pBtn;
HWND	hCtl;	CEdit*	pEdit;
HWND	hCtl;	CListBox*	pListBox;
HWND	hCtl;	CComboBox*	pComboBox;

其他自定义类变量的定义没有BaseTag第一个字母即大写。

例:

CMyButton m_ColorButton;

Visual C++常用宏定义命名列表：

前缀	**符号类型**	**符号例子**	**范围**
IDR_	标识多个资源共享的类型	IDR_MAINFRAME	1 to 0x6FFF
IDD_	对话框资源(Dialog)	IDD_SPELL_CHECK	1 to 0x6FFF
HIDD_	基于对话框的上下文帮助(Context Help)	HIDD_SPELL_CHECK	0x20001 to 0x26FF
IDB_	位图资源(Bitmap)	IDB_COMPANY_LOGO	1 to 0x6FFF
IDC_	光标资源(Cursor)	IDC_PENCIL	1 to 0x6FFF
IDI_	图标资源(Icon)	IDI_NOTEPAD	1 to 0x6FFF
ID_ IDM_	工具栏或菜单栏的命令项	ID_TOOLS_SPELLING	0x8000 to 0xDFFF
HID_	命令上下文帮助(Command Help context)	HID_TOOLS_SPELLING	0x18000 to 0x1DFFF

IDP_	消息框提示文字资源	IDP_INVALID_PARTNO	8 to 0xDFFF
HIDP_	消息框上下文帮助(Message-box Help context)	HIDP_INVALID_PARTNO	0x30008 to 0x3DFFF
IDS_	字符串资源(String)	IDS_COPYRIGHT	1 to 0x7FFF
IDC_	对话框内的控制资源(Control)	IDC_RECALC	8 to 0xDFFF

5.1 其他规则

（1）、标识符采用英文单词或其组合。严禁使用汉语拼音。

（2）、标识符采用“大小写”混排方式，如AddChild。

（3）、程序不能出现仅靠大小写区分的相似的标识符。如：int x, X;

（4）、程序不能出现标识符完全相同的局部变量和全局变量。

（5）、变量的名字应当使用“名词”或者“形容词+名词”。如：

```
float fValue;
float fOldValue;
```

（6）、全局函数的名字应当使用“动词”或者“动词+名词”（动宾词组）。类的成员函数应当只使用“动词”，被省略的名词就是对象本身。如：

```
DrawBox();      //全局函数
pBox->Draw(); //类的成员函数
```

（7）、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。如：

```
int nMinValue;
int nMaxValue;
int SetValue();
int GetValue();
```

（8）、除非逻辑上的确需要编号，其余情况严禁在名字中出现数字编号，如：Value1,Value2等。

6 表达式和基本语句

6.1 运算符优先级

如果代码行中的运算符比较多，要用括号确定表达式的操作顺序，避免使用默认的优先级。如：

```
word = (high << 8) | low
if ((a|b) && (a&c))
```

6.2 复合表达式

本规范并不禁止使用复合表达式，如：a = b = c = 0;

但破坏可读性的复合表达式则禁止使用。下表列举了一些典型情况：

禁止使用的情况	实例
太复杂的复合表达式	l = a >= b && c < d && c + f <= g + h

多用途的表达式	d = (a = b + c) + r;应拆成：
	a = b + c;
	d = a + r;

与数学表达式混淆	if (a < b < c)并不表示if ((a<b)&&(b<c))而是if((a<b)<c)

6.3 if语句

（1）、布尔变量的比较

正确的用法	不良的用法

if (bFlag)　//如果bFlag为真
if (!bFlag)　//如果bFlag为假若
	if (bFlag == TRUE)
	if (bFlag == 1)
	if (bFlag == FALSE)
	if (bFlag == 0)

（2）、整型变量的比较

正确的用法	不良的用法
if (nValue == 0) if (nValue != 0)	if (nValue) //容易误解nValue为布尔 if (!nValue)

（3）、浮点变量的比较

不可将浮点变量用“==”或“!=”与任何数字比较。而要采用“>=”或“<=”形式。

正确的用法	错误的用法
if ((fValue >= -EPSINON) && (fValue <= EPSINON) //EPSINON是允许的误差（精度）	if (fValue == 0.0)

（5）、指针变量的比较

正确的用法	不良的用法

if (p == NULL)
if (p != NULL)
	if (p == 0) //容易误解为整数
	if (p != 0)
	if (p) //容易误解为布尔
	if (!p)

（6）、补充说明

对于if/else/return的组合，要注意采用下面的书写风格：

正确的用法	不良的用法

if (condition)
{
return x;
}
else
{
return y;
}
或者
return (condition ? x : y);
	if (condition)
	return x;
	return y;

6.4 循环语句

（1）在多重循环中，为提高效率，尽可能将最长的循环放在最内层，最短的循环放在最外层。

低效率	高效率

for (row = 0; row < 100; row ++)
{
for (col = 0; col < 5; col ++)
{
sum = sum + a[row][col];
}
	for (int nCol=0; nCol<5; nCol++)
	{
	for (int nRow=0; nRow<5; nRow++)
	{
	nSum = nSum + a[nRow][nCol];
	}

}	}

（2）如果循环体内存在逻辑判断，并且循环次数很大，宜将逻辑判断移到循环体的外面。

低效率程序简洁	高效率程序不简洁

for (i = 0; i < N; i ++)
{
  if (condition)
  DoSomething();
  else
  DoOtherthing();
}
	if (condition)
	{
	  for (I=0; I < N; I++)
	  DoSomething();
	}
	else
	{
	  for (I=0; I < N; I++)
	  DoOtherthing();
	}

（3）以上两条需要灵活处理，如果程序循环次数很多占用CPU不可忽略，可以参照以上两条。否则应该以简洁容易理解为主。

（4）不可在for循环体内修改循环变量，防止循环失去控制。

6.5 switch语句

switch的标准书写格式如下：

```
switch (variable)
{
        case value1 :
                …
                break;

        case value2 :
                …
                break;
```

```
        …
        default:
                …
                break;
}
```

6.6 goto语句

本规范要求慎用goto语句，而不禁用。

示例：

```
{…
  {…
    {…
      goto ERROR;
    }
  }
}
ERROR:
…
```

7 常量

7.1 常量定义规则

（1）、常量定义采用const形式，禁止使用#define。

（2）、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。

（3）、如果某一常量与其他常量密切相关，应在定义中包含这种关系，而不应给出一些孤立的值。如：

```
const float RADIUS = 100;
const float DIAMETER = RADIUS * 2;
```

8 函数

8.1 函数注释

对于自行编写的函数，若是系统关键函数，则必须在函数实现部分的上方标明该函数的信息，格式如下：

```
//////////////////////////////////////////////////////////////////////////////
// 编写者：
// 参考资料：
// 功能：
// 输入参数：
// 输出参数：
// 备注：(对使用的关键、复杂性算法进行说明)
//////////////////////////////////////////////////////////////////////////////
```

8.2 参数规则

（1）、参数书写要完整。

不良风格	良好风格
void SetValue(int, int);	void SetValue(int nWidth, int nHeight);

（2）、参数命名要恰当，顺序要合理。目的参数放在前面，源参数放在后面。如：

```
void StringCopy (char *str1, char *str2);                    //不良风格
void StringCopy (char *pszSource, char *pszDestination); //良好风格
```

（3）、如果参数是指针，且仅做输入用，要在类型前加const。如：

```
void StringCopy (char *pszSource, const char *pszDestination);
```

（4）、如果输入参数以值传递的方式传递对象，建议采用“const &”方式来传递。

（5）、设计函数时，参数个数尽量控制在5个以内。

（6）、禁止使用类型和数目不确定的参数。设计函数时，参数个数尽量控制在5个以内。

8.3 返回值规则

（1）、不要将正常和错误标志混在一起返回。正常值用输出参数获得，而错误标志用return语句返回。

8.4 函数实现规则

（1）、在函数的“入口处”要加强对参数有效性的检查。

（2）、在函数的“出口处”对return语句的正确性和效率要加强检查。

8.5 其他

（1）、函数功能要单一，不要设计多用途的函数。

（2）、函数体的规模要小，尽量控制在50行以内。

（3）、尽量避免函数有“记忆”功能。相同的输入应当产生相同的输出。尽量不使用static局部变量，除非必需。

（4）、不仅要检查输入参数的有效性，还要检查通过其他途径进入函数体内的变量的有效性，例如全局变量、文件句柄等。

（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。