C++编程命名规则

一、程序风格：

1、严格采用阶梯层次组织程序代码：

各层次缩进的分格采用VC的缺省风格，即每层次缩进为4格，括号位于下一行。

正在装载数据……

要求相匹配的大括号在同一列，对继行则要求再缩进4格。例如：

2、提示信息字符串的位置

在程序中需要给出的提示字符串，为了支持多种语言的开发，除了一些给调试用的临时信息外，其他所有的提示信息必须定义在资源中。

3、对变量的定义，尽量位于函数的开始位置。

二、命名规则：

1、变量名的命名规则

①、变量的命名规则要求用“匈牙利法则”。即开头字母用变量的类型，其余部分用变量的英文意思或其英文意思的缩写,尽量避免用中文的拼音,要求单词的第一个字母应大写。

即： 变量名=变量类型+变量的英文意思（或缩写）

对非通用的变量，在定义时加入注释说明，变量定义尽量可能放在函数的开始处。

见下表：

bool(BOOL) 用b开头 bIsParent

byte(BYTE) 用by开头 byFlag

short(int) 用n开头 nStepCount

long(LONG) 用l开头 lSum

char(CHAR) 用c开头 cCount

float(FLOAT) 用f开头 fAvg

double(DOUBLE) 用d开头 dDeta

void(VOID) 用v开头 vVariant

unsigned int（WORD） 用w开头 wCount

unsigned long(DWORD) 用dw开头 dwBroad

HANDLE（HINSTANCE） 用h开头 hHandle

DWORD 用dw开头 dwWord

LPCSTR(LPCTSTR) 用str开头 strString

用0结尾的字符串 用sz开头 szFileName

对未给出的变量类型要求提出并给出命名建议给技术委员会。

②、指针变量命名的基本原则为：

对一重指针变量的基本原则为：

“p”+变量类型前缀+命名

如一个float\*型应该表示为pfStat

对多重指针变量的基本规则为：

二重指针： “pp”+变量类型前缀+命名

三重指针： “ppp”+变量类型前缀+命名

......

③、全局变量用g\_开头,如一个全局的长型变量定义为g\_lFailCount,即：变量名=g\_+变量类型+变量的英文意思（或缩写）

④、静态变量用s\_开头,如一个静态的指针变量定义为s\_plPerv\_Inst,即： 变量名=s\_+变量类型+变量的英文意思（或缩写）

⑤、成员变量用m\_开头,如一个长型成员变量定义为m\_lCount;即：变量名=m\_+变量类型+变量的英文意思（或缩写）

⑥、对枚举类型（enum）中的变量，要求用枚举变量或其缩写做前缀。并且要求用大写。

如：enum cmEMDAYS

{

EMDAYS\_MONDAY;

EMDAYS\_TUESDAY;

……

};

⑦、对struct、union、class变量的命名要求定义的类型用大写。并要加上前缀，其内部变量的命名规则与变量命名规则一致。

结构一般用S开头

如：struct ScmNPoint

{

int nX;//点的X位置

int nY; //点的Y位置

};

联合体一般用U开头

如: union UcmLPoint

{

long lX;

long lY;

}

类一般用C开头

如：

class CcmFPoint

{

public:

float fPoint;

};

对一般的结构应该定义为类模板，为以后的扩展性考虑

如：

template

class CcmTVector3d

{

public:

TYPE x,y,z;

};

⑧、对常量（包括错误的编码）命名，要求常量名用大写，常量名用英文表达其意思。

如：#define CM\_FILE\_NOT\_FOUND CMMAKEHR(0X20B) 其中CM表示类别。

⑨、对const 的变量要求在变量的命名规则前加入c\_,即：c\_+变量命名规则；例如：

const char\* c\_szFileName;

2、 函数的命名规范：

函数的命名应该尽量用英文表达出函数完成的功能。遵循动宾结构的命名法则，函数名中动词在前,并在命名前加入函数的前缀，函数名的长度不得少于8个字母。

例如：

long cmGetDeviceCount(……);

3、函数参数规范：

①、 参数名称的命名参照变量命名规范。

②、 为了提高程序的运行效率，减少参数占用的堆栈，传递大结构的参数，一律采用指针或引用方式传递。

③、 为了便于其他程序员识别某个指针参数是入口参数还是出口参数，同时便于编译器检查错误，应该在入口参数前加入const标志。如：

……cmCopyString(const char \* c\_szSource, char \* szDest)

4、引出函数规范：

对于从动态库引出作为二次开发函数公开的函数，为了能与其他函数以及Windows的函数区分，采用类别前缀+基本命名规则的方法命名。例如：在对动态库中引出的一个图象编辑的函数定义为 imgFunctionname(其中img为image缩写)。

现给出三种库的命名前缀：

①、 对通用函数库，采用cm为前缀。

②、 对三维函数库，采用vr为前缀。

③、 对图象函数库，采用img为前缀。

对宏定义，结果代码用同样的前缀。

5、文件名(包括动态库、组件、控件、工程文件等)的命名规范：

文件名的命名要求表达出文件的内容，要求文件名的长度不得少于5个字母，严禁使用象file1,myfile之类的文件名。

三、注释规范：

1、函数头的注释

对于函数，应该从“功能”，“参数”，“返回值”、“主要思路”、“调用方法”、“日期”六个方面用如下格式注释：

//程序说明开始

//================================================================//

// 功能： 从一个String 中删除另一个String。

// 参数： strByDelete,strToDelete

// （入口） strByDelete: 被删除的字符串（原来的字符串）

// （出口） strToDelete: 要从上个字符串中删除的字符串。

// 返回： 找到并删除返回1，否则返回0。（对返回值有错误编码的要// 求列出错误编码）。

// 主要思路：本算法主要采用循环比较的方法来从strByDelete中找到

// 与strToDelete相匹配的字符串，对多匹配strByDelete

// 中有多个strToDelete子串）的情况没有处理。请参阅：

// 书名......

// 调用方法：......

// 日期：起始日期，如：2000/8/21.9:40--2000/8/23.21:45

//================================================================//

函数名(……)

//程序说明结束

①、 对于某些函数，其部分参数为传入值，而部分参数为传出值，所以对参数要详细说明该参数是入口参数，还是出口参数，对于某些意义不明确的参数还要做详细说明（例如：以角度作为参数时，要说明该角度参数是以弧度（PI）,还是以度为单位）,对既是入口又是出口的变量应该在入口和出口处同时标明。等等。

②、 函数的注释应该放置在函数的头文件中，在实现文件中的该函数的实现部分应该同时放置该注释。

③、 在注释中应该详细说明函数的主要实现思路、特别要注明自己的一些想法，如果有必要则应该写明对想法产生的来由。对一些模仿的函数应该注释上函数的出处。

④、 在注释中详细注明函数的适当调用方法，对于返回值的处理方法等。在注释中要强调调用时的危险方面，可能出错的地方。

⑤、 对日期的注释要求记录从开始写函数到结束函数的测试之间的日期。

⑥、 对函数注释开始到函数命名之间应该有一组用来标识的特殊字符串。

如果算法比较复杂，或算法中的变量定义与位置有关，则要求对变量的定义进行图解。对难以理解的算法能图解尽量图解。

2、变量的注释：

对于变量的注释紧跟在变量的后面说明变量的作用。原则上对于每个变量应该注释，但对于意义非常明显的变量，如：i,j等循环变量可以不注释。

例如： long lLineCount //线的根数。

　3、文件的注释：

文件应该在文件开头加入以下注释：

/////////////////////////////////////////////////////////////////////

// 工程: 文件所在的项目名。

// 作者：\*\*，修改者：\*\*

// 描述:说明文件的功能。

// 主要函数：…………

// 版本: 说明文件的版本，完成日期。

// 修改: 说明对文件的修改内容、修改原因以及修改日期。

// 参考文献： ......

/////////////////////////////////////////////////////////////////////

为了头文件被重复包含要求对头文件进行定义如下:

#ifndef \_\_FILENAME\_H\_\_

#define \_\_FILENAME\_H\_\_

其中FILENAME为头文件的名字。

　　　4、其他注释：

在函数内我们不需要注释每一行语句。但必须在各功能模块的每一主要部分之前添加块注释，注释每一组语句，在循环、流程的各分支等，尽可能多加以注释。

其中的循环、条件、选择等位置必须注释。

对于前后顺序不能颠倒的情况，建议在注释中增加序号。

例如：

在其他顺序执行的程序中，每隔3—5行语句，必须加一个注释，注明这一段语句所组成的小模块的作用。对于自己的一些比较独特的思想要求在注释中标明。

四、程序健壮性：

1、函数的返回值规范：

对于函数的返回位置，尽量保持单一性，即一个函数尽量做到只有一个返回位置。(单入口单出口)。

要求大家统一函数的返回值，所有的函数的返回值都将以编码的方式返回。

例如编码定义如下：

#define CM\_POINT\_IS\_NULL CMMAKEHR(0X200)

:

:

建议函数实现如下：

long 函数名(参数,……)

{

long lResult; //保持错误号

lResult=CM\_OK;

//如果参数有错误则返回错误号

if(参数==NULL)

{

lResult=CM\_POINT\_IS\_NULL;

goto END;

}

……

END:

return lResult;

}

2、关于goto的应用：

对goto语句的应用，我们要求尽量少用goto语句。对一定要用的地方要求只能向后转移。

3、资源变量的处理（资源变量是指消耗系统资源的变量）：

对资源变量一定赋初值。分配的资源在用完后必须马上释放，并重新赋值。

4、对复杂的条件判断，为了程序的可读性，应该尽量使用括号。

例：if(((szFileName!=NULL)&&(lCount>=0)))||(bIsRead==TRUE))

五、可移植性：

1、高质量的代码要求能够跨平台，所以我们的代码应该考虑到对不同的平台的支持，特别是对windows98和windowsnt的支持。

2、由于C语言的移植性比较好，所以对算法函数要求用C代码，不能用C++代码。

3、对不同的硬件与软件的函数要做不同的处理