# 编码的基本要求

n         程序结构清晰，逐步求精，单个函数的程序行数一般不超过100行。

n         代码精简，避免编码拖沓冗长。

n         尽量使用标准库函数和公共函数。

n         不要随意定义全局变量，尽量使用局部变量、类的成员变量。

n         表达式复杂时，多使用括号以避免二义性。

## 1可读性

n         可读性第一，效率第二。

n         保持注释与代码完全一致。

n         每个源程序文件，都有文件头说明，说明规格见规范。

n         每个函数，都有函数头说明，说明规格见规范。

n         主要变量（结构、联合、类或对象）定义或引用时，注释能反映其含义。

n         常量定义（DEFINE）有相应说明。

n         处理过程的每个阶段都有相关注释说明。

n         在典型算法前都有注释。

n         利用缩进来显示程序的逻辑结构，缩进量一致并以空格键为单位，定义为 4个  
字节，不建议用Tab键。

n         循环、分支层次不要超过五层。必要时，使用Goto 语句，减少层次。

n         注释可以与语句在同一行，也可以在上行。

n         空行和空白字符也是一种特殊注释。

n         一目了然的语句不加注释。

n         注释的作用范围可以为：定义、引用、条件分支以及一段代码。

n         注释行数（不包括程序头和函数头说明部份）应占总行数的 1/5 到 1/3 。

## 2结构化要求

n         代码严格缩进，缩进使用TAB键，设置TAB键等价于4个空格字符。

n         禁止出现两条等价的支路。

n         减少使用GOTO语句。为减少IF语句层次，也可以适当使用。

n         用 CASE 实现多路分支。减少使用 ELSE IF句型。

n         避免从循环引出多个出口。

n         函数一般只有一个出口。为减少判断语句层次，可适当使用GOTO 语句。

n         不使用条件赋值语句。 如：if (a=b) …

n         不要轻易用条件分支去替换逻辑表达式。

n         考虑到代码量和可阅读性，有如下编码建议（非强制性规范）：

1）减少括号的使用

if (!m\_bIsOpen ) return FALSE;

如果写成如下，将大大增代码量，函数体代码语句多的时候不利于阅读：

if (!m\_bIsOpen )

{

return FALSE;

}

else

{

// ......

}

2）使用GOTO语句减少语句层次

下面是函数体，使用GOTO语句减少代码层次的例子

该例子如使用IF语句做整体判断则明显增加代码缩进的层次。

{

if ( 条件不满足1 )

{

iret = -1;

goto label\_end;

}

if ( 条件不满足2 )

{

iret = -2;

goto label\_end;

}

label\_end:

// ... ...

return iret;

}

## 3正确性与容错性要求

n         程序首先是正确，其次是才是优美。

n         无法证明你的程序没有错误，因此在编写完一段程序后，应先回头检查。

n         改一个错误时可能产生新的错误，因此在修改前首先考虑对其它程序的影响。

n         所有变量在调用前必须被初始化。数组、结构体要ZeroMemory 初始化。

n         对所有的用户输入，必须进行合法性检查。

n         不要比较浮点数的相等，如： 10.0 \* 0.1 == 1.0 ， 不可靠

n         程序与环境或状态发生关系时，必须主动去处理发生的意外事件，如文件能否逻辑锁定、打印机是否联机等。

n         单元测试也是编程的一部份，提交联调测试的程序必须先由程序员经过单元测试。

## 4可重用性要求

n         重复使用的、相对独立功能的算法或代码应抽象为公共控件或类。

n         公共控件或类应考虑使用OO编程思想，减少外界联系，考虑独立性和封装性。

n         公共控件或类应建立使用模板。

n         一些涉及到商业机密或者公司核心技术的部件，可使用DLL或者COM组件的形式对公用模块进行封装。

# C++代码书写规范

## 1文件头的编写规范

每个文件开头部分至少应当包括下面几项内容：文件标题（Title ）、版权说明（Copyright）、修改记录（Modification History）。

n         Title 中包括文件的名称和文件的简单说明。Title 应该在一行内完成。

n         Copyright 包括一个版权说明。

例如：

// Copyright (c) 2005 EASTCOM LingTong.

// All rights reserved.

n         Modification History

记录文件修改的过程，只记录最主要的修改。当书写新的函数模块时，只需要使用

形如“add func1()”这样的说明；如果后面又对函数中的算法进行了修改，需要指出修改

的具体位置和修改方法。

Modification History 的具体格式为：

<版本号> <修改日期> <修改人和修改动作>

如：

// V0.9.0 20050823 创建初始版本

// V0.9.1 20050830 修改结构体项目对齐方式为默认

一个文件头的具体范例如下：

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

//

// Copyright (c) 2005 EASTCOM LingTong.

// All rights reserved.

//

// 文件名： CCFrmw.h

// 类型： HPP

// 版本号： 0.9.1

// 编制： Trueway Lee

// 描述： 用于实现 ABWOA 模块DLL

// 该模块内部实现了 EASTCOM-ABWOA 软件总线机制

//

//--------------------------------------------------------------------

// 版本历史：

//--------------------------------------------------------------------

// V0.9.0 20050823 创建初始版本

// V0.9.1 20050830 修改结构体项目对齐方式为默认

//

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

## 2变量命名

为了便于程序的阅读，变量命名必须具有一定的实际意义。

变量命名按照匈牙利命令风格：形式为xAbcFgh，x由变量类型确定，Abc、Fgh表示连续意义字符串，如果连续意义字符串仅两个，可都大写。如OK。

下面是常规数据类型的变量命名方法：

|  |  |  |
| --- | --- | --- |
| ****类型名**** | ****C/C++ 等价类型**** | ****定义实例**** |
| ****BOOL**** | int | BOOL fFlag； |
| ****LPSTR**** | char \* | LPSTR szText； |
| ****HANDLE**** | void \* | HANDLE hEvent1； |
| ****HWND**** | void \* | HWND hWnd； |
| ****INT**** | int | INT iIndex； |
| ****UINT**** | unsignedint | UINT nIndex； |
| ****LPVOID**** | void \* | LPVOID pData； |
| ****WORD**** | unsignedshort | WORD wData； |
| ****DWORD**** | unsignedlong | DWORD dwData； |
| ****BYTE**** | unsignedchar | BYTE bChar； |
| ****CHAR**** | char | CHAR cChar； |
| ****LONG**** | long | LONG lIndex； |
| ****ULONG**** | unsignedlong | ULONG ulData； |
| ****WPARAM**** | unsignedint | WPARAM wParam； |
| ****LPARAM**** | long | LPARAM lParam； |

这里只是列举了一部分，更详细的内容请查看 MSDN 文档。

## 3各作用域变量命名规则

n         全局变量（g\_类型、）

“g\_” ＋ “类型标识”＋“变量名”

例如：g\_nMsg、g\_bFlag

n         类成员变量（m\_xxxx）

“m\_” ＋ “类型标识”＋“变量名”

例如：m\_szText、m\_hEvent

n         局部变量

“类型标识”＋“变量名”

局部变量中可采用如下几个通用变量：nTemp，iRet，I，J（一般用于循环变量）。

n         函数参数变量，参数变量是局部变量的特例

“a\_” ＋“类型标识”＋“变量名”

例如：int test\_fun(long a\_lMode， LPSTR a\_szData)；

## 4常量命名和宏定义

下面是常量和宏定义方面的一般规则：

n         常量和宏定义必须具有一定的实际意义；

n         常量和宏定义在 #include 和函数定义之间；

n         常量和宏定义一般全部以大写字母来撰写，中间可根据意义的连续性用下划线连接，每一条定义的右侧必须有一简单的注释，说明其作用。

#define ****\_POSIX\_ARG\_MAX 4096****

#define ****\_POSIX\_CHILD\_MAX 6****

#define ****\_POSIX\_LINK\_MAX 8****

#define ****\_POSIX\_MAX\_CANON 255****

#define ****\_POSIX\_MAX\_INPUT 255****

#define ****\_POSIX\_NAME\_MAX 14****

考虑到阅读的方便性，也可以使用大小写混合的形式定义常量，但第一段必须是大写，标识该常量的使用访问或者类别。

#define ****ERROR\_SSIPC\_Name\_Error (-1)****

#define ****ERROR\_SSIPC\_Time\_Out (-2)****

#define ****ERROR\_SSIPC\_Call\_Failture (-3)****

#define ****ERROR\_SSIPC\_Pack\_Data (-4)****

n         资源名字定义格式：  
菜单：IDM\_XX或者CM\_XX  
位图：IDB\_XX  
对话框：IDD\_XX  
字符串：IDS\_XX  
DLGINIT：DIALOG\_XX  
ICON：IDR\_XX

## 5函数命名

n         函数原型说明包括引用外来函数及内部函数，外部引用须在右侧注明函数来源：模  
块名及文件名， 如是内部函数，不需要特殊说明。

n         函数名第一个字母大写，其他部分要求用大小写字母组合命名。

必要时可用下划线间隔。

示例如下：

extern void UpdateDB\_Tfgd (TRACK\_NAME)； //Module Name ：r01/sdw.c  
extern PrintTrackData (TRACK\_NAME)； //Module Name ：r04/tern.c  
extern ImportantPoint (void)； //Module Name ：r01/sdw.c  
void ShowChar (int ， int ，long)；   
void ScrollUp\_V (int ， int)；

## 6结构体

n         结构体类型命名使用大写字母，原则上前面以下划线开始；

n         结构体变量用大小写字母组合，第一个字母大写，必要时可用下划线间隔;

n         对于私有数据区，须注明其所属的进程。全局数据定义只需注明其用途。

示例如下：  
typedef struct  
{  
 char szProductName[20]；  
 char szAuthor[20]；  
 char szReleaseDate[16]；  
 char szVersion[10]；   
 unsigned long MaxTables；  
 unsigned long UsedTables；  
}****DBS\_DATABASE****, ****\* LPDBS\_DATABASE****；  
****DBS\_DATABASE****  g\_dataBase；

## 7控件的命名

用小写前缀表示类别  
fm 窗口  
cmd 按钮  
cob combo，下拉式列表框  
txt 文本输入框  
lab label，标签  
img image，图象  
pic picture  
grd Grid，网格  
scr 滚动条  
lst 列表框  
frm fram

控件命名可以为：

n         小写类别＋大小写混合的控件标识

n         小写类别＋“\_\_”＋ 控件标识

比如： labText、lab\_text 都可以。

## 8注释

n         原则上注释要求使用中文；

n         文件开始注释内容包括：公司名称、版权、作者名称、时间、模块用途、背景介绍等；

n         复杂的算法需要加上流程说明；

n         函数注释包括：输入、输出、函数描述、流程处理、全局变量、调用样例等；

n         复杂的函数需要加上变量用途说明；

n         程序中注释（阶段注释）包括：修改时间和作者、方便理解的注释等；

注释主要分为三类，文件头注释、函数体内阶段注释、函数注释

****引用一： 文件开头的注释模板****

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

//

// Copyright (c) 2005 EASTCOM LingTong.

// All rights reserved.

//

// 文件名: cccdmfw\_xpp.h

// 类型: XPP

// 版本号: 0.1

// 编制: jjt

// 描述: 用于实现 ABWOA 出钞机构模块定义

// 该模块内部实现了 EASTCOM-ABWOA 模块出钞机构的动作，

// 基于ABWOA模块的出钞机构CCCdmFW类定义

//

//--------------------------------------------------------------------

// 版本历史:

//--------------------------------------------------------------------

// V0.1 2005-9-22 10:30 创建初始版本

//

//

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

****引用二： 程序中的注释模板****

if ( ( pBuff->sCallType == SWBUS\_Asynchronization\_Mode ) &&

( pBuff->pJobState ) )

pBuff->pJobState->sJobState = CCFRMW\_JOB\_RUNNING;// 设置任务状态

//---------------------------------------------------------------------

// 获得模块信息

//---------------------------------------------------------------------

LPMYMODULENODE lpInfo = NULL;

int iret = g\_load.GetModuleInfo(pBuff->szFWName, &lpInfo);

if ( iret != CCFRMW\_RC\_OK ) return iret;

//---------------------------------------------------------------------

// 获得模块相关的对象信息（找到最后的子类，作为服务对象）

//---------------------------------------------------------------------

LPMYMODULEDETAIL lpDetail = NULL;

iret = g\_load.GetFirstObjectInfo(lpInfo, &lpDetail);

if ( iret != CCFRMW\_RC\_OK ) return iret;

****引用三： 函数开头的注释模板，考虑到编码的方便性，函数注释可以简单一点****

// 获得DD变量值

SHORT CCDataDicFW::GetDataItem( // get the data item - USHORT

CHAR \* pszDataItemName, // (in) property name

USHORT & rusVar, // (out) buffer for retrieved data

ULONG ulIndex) // (in) index (used for indexing arrays)

{

SHORT sRc = CCDATADICTFW\_OK;

char ownerFW[NODE\_VALUE\_LEN];

SHORT funcId;

TrcWritef(TRC\_ENTRY,"> CCDataDicFW::GetDataItem(%s)",pszDataItemName);

sRc = GetDataInfo(GET,ownerFW,funcId,pszDataItemName);

if(sRc == CCDATADICTFW\_OK)

{

sRc = FrmResolve(ownerFW,funcId, pszDataItemName, strlen(pszDataItemName)+1, (void \*)&rusVar, sizeof(USHORT), NULL, 0, ulIndex);

}

TrcWritef(TRC\_ENTRY,"< CCDataDicFW::GetDataItem(%s) Returns %d",pszDataItemName,sRc);

return sRc;

}

## 9 限制使用全局变量

考虑到代码移植方面和出现信息存贮冲突，在代码中，避免使用全局变量。

可以进行替代：

n         使用类的成员变量

n         使用类的静态变量

n         线程入口函数使用参数传递指针

n         一些不能避免的情况，如：WINPROC 中的变量访问，仍可以使用全局变量。

## 10 数据初始化

与很多语言不同，C/C++一般不提供对结构体变量、字符数组、整形等数据的初始化控制。比如 integer I; 此语句执行后，I 里面的值是一个随机数字。

char buff[100]; 执行后，buff 的每个单元的内容也是一个随机值。

为了避免出现问题，要求变量申明后，一定要初始化，这里还包括了类的成员变量。

比如：

char buff[100];

ZeroMemory(buff, sizeof(buff));

下面针对各类数据类型做数据初始化的规范说明：

n         简单类型数据，需要在变量标识后加“＝初始值；”

integer i = 0;

char c = 0;

double d = 12.0;

n         指针变量、类似指针的变量，在申请内存前和使用完成后，都要使值保持NULL

HANDLE p = NULL;

// ......

if (p )

{

CloseHandle(p);

p = NULL;

}

n         字符数组，注意{0} 的妙用

char buff[100];

ZeroMemory(buff, sizeof(buff));

也可以写成一句话：

char buff[100] ={0};

n         结构体变量，注意{0} 的妙用

MYSTRUCTA bb;

ZeroMemory(&bb, sizeof(MYSTRUCTA ));

也可以写成一句话：

MYSTRUCTA bb = {0};

n         C++类中，在类的构造子中完成对成员变量数据的初始化

# 调试规范

## 1 Trace跟踪

n         程序中应该包括足够的调试信息。调试信息一律使用Trace跟踪。

n         在函数入口处，推荐使用assert() 对于参数的有效性进行判断。

n         对于入口参数是指针的，必须使用Assert()进行判断。

n         assert()的使用方法为：assert(<逻辑表达式>)。assert()的具体使用方法请参考MSDN文档，这里不做详细说明。

n         在代码中易出现问题的地方，也应该使用assert（）、或者其他的机制进行相应的检测。

## 2调试记录、测试报告

调试记录是软件产品的一个重要组成部分。在软件的调试过程中，应做好调试记录，以便于软件的后续开发和维护工作。

调试的文档包括两个内容：

n         调试记录

即发现软件中的Bug 并进行修改的记录。

在软件编码过程中的修改很频繁，对每个发现的问题都记录下来是不现实的。在实际记录过程中可选择以下信息的部分作为记录内容，调试记录一般需要包括如下内容：

****（日期）****

[如：2000 年6 月26 日]

****（问题编号）****

[如：1]

****（问题征兆）****

[如：发现缓冲中的报文被无端修改]

****（问题根源及解决方法）****

[如：由于在多任务环境中对共享资源没有使用互斥机制，对于缓冲增加一个信号量进行保护（修改位于buffer.c 中）]

n         测试报告

在软件项目实现阶段，除了要编写好代码以外，经常需要对实现的子模块进行测试，尤其是对提供给他人使用的模块，必须进行测试。

在测试前，需要做测试计划，测试中做好零散的记录，测试完成后，整理成报告。测试报告是一笔宝贵的财富，当软件做到一定程度的功能，以后对软件的更新，可以参照历史测试报告进行回归测试。减少重新编制测试报告的工作量，同时能保存测试指标，使软件再测试具有延续性。

## 3测试用例、测试代码

为使测试自动化和更具全面性，需花费时间和精力制作测试用例和编写测试代码，比如测试类的功能，就需要编写测试代码。

测试用例需要按照软件工程学的相关规范来制作。最简单的原则是，测试用例要能覆盖问题可能出现的范围。详细测试用例的制作方法参考软件工程学资料。

测试代码需要实现对单个模块的模拟调用，在整个项目相关的软件模块并为一一齐备的情况下，也需要模拟实现未真正实现的功能模块。此外，测试代码还可以用于构建软件模块的压力测试、内存泄漏测试。

## 4调试、测试要求

n         对于测试代码和测试用例，要注意保存，并适当使用文档加以说明。

文档中应该说明测试代码的测试目的。如果测试代码和文档分别存放，还需要在文档中对测试代码的保存位置加以说明。

n         写好测试报告。例如：对某个模块的若干接口加以测试，要记录测试的目的、具体的测试方法和测试的结果（如：通过测试，或测试失败）。

n         出现问题，及时通知相关模块具体负责人。

测试人员在测出问题后，除完成测试报告外，还需要和模块负责人进行必要的沟通。

# 源码管理办法

****1.      修改及时提交****

源码完成编制、修改后，须提交到配置库。使用指定的版本管理软件实施控制。

当某功能模块完成编程后，使用软件管理工具提交。在进行提交之前，必须对新加入的功能进行一定的测试，在尽量保证正确性后在进行提交。

避免出现长时间修改而不提交的情况，因为可能出现多人修改同一模块的情况。长时间不提交将导致并发的修改方每次修改前不能获得最新的版本。

****2.      提交要完整****

每次提交，要注意完成性，把本次涉及到的修改都做提交，否则容易出现别处下载更新代码后不能编译、调试的情况。

****3.      先更新，再修改****

每次进行修改前，从配置库“Check out”最新版本到本地，以免在老版本上做修改。

****4.      写修改说明****

每次提交，须在提交报告中注明新加入功能的内容、修改的文件范围等内容。

****5.      定期更新本地版本****

需要定期对本地版本的代码、文档做全面更新，特别是在软件项目庞大或者项目组人员较多的情况下，需要定期更新本地版本，特别是非自身负责的模块，以便能在最新版本的代码上进行联合编译、调试和测试。

****6.      共用的.H、.CPP、.LIB、.DLL文件单独存放****

类似VC++的安装目录安排，需设置INCLUDE、LIB、SRC等共有目录，将共有代码放置其内，避免出现每个模块将共有代码放在各自的项目内的情况。

单独存放共用代码，利于统一更新。

****7.      配置库中除管理源代码外，还管理 BIN 内容****

阶段编译结果的EXE、DLL、OCX等内容，也作为配置项，减少不必要的重新编译。

****8.      阶段成果分开保存****

针对相对稳定的阶段成果，需和主配置分开保存。以避免最新的修改影响阶段成果。

**代码：**

#include<iostream>  
using namespace std;  
  
class Date  
{  
public:  
Date() {}  
Date(int y, int m, int d)  
{  
year=y;  
month=m;  
day=d;  
}  
int GetYear() { return year; }  
int GetMonth() { return month; }  
int GetDay() { return day; }  
  
void SetYear(int y) { year=y; }  
void SetMonth(int m) { month=m; }  
void SetDay(int d) { day=d; }  
  
private:  
int year;  
int month;  
int day;  
};  
  
class People  
{  
public:  
People() {}  
People(int n, int s, Date birth, int i)  
{  
number=n;  
sex=s;  
birthday=birth;  
id=i;  
}  
  
~People() {}  
  
inline int GetN() { return number; }  
int GetS() { return sex; }  
int GetI() { return id; }  
  
void set();  
void show();  
  
private:  
int number;  
int sex;  
Date birthday;  
int id;  
};  
  
void People::set()  
{  
cout<<"Please input the number of the people:";  
cin>>number;  
cout<<"Please input the sex of the people(1男，2女):";  
cin>>sex;  
cout<<"Please input the birthday of the people:";  
int y,m,d;  
cin>>y>>m>>d;  
birthday.SetYear(y);  
birthday.SetMonth(m);  
birthday.SetDay(d);  
cout<<"Please input the id of the people:";  
cin>>id;  
}  
  
void People::show()  
{  
cout<<"Number:"<<number<<endl;  
cout<<"Sex:"<<sex<<endl;  
cout<<"Birthday:"<<birthday.GetYear()<<birthday.GetMonth()<<birthday.GetDay()<<endl;  
cout<<"ID:"<<id<<endl;  
}  
  
void main()  
{   
Date a(1993,01,01);  
People p(001,1,a,142);  
p.set();  
p.show();