**C++编码规范**

* 头文件尽可能依赖于前置申明

【优势】较小的文件依赖，节约编译时间，更加明确类的依赖关系

*#ifndef \_MYCLASS\_H\_*

*#define \_MYCLASS\_H\_*

class myClass1;

class myClass2;

class myClass

{

//TODO:

};

*#endif*

* 函数的参数申明顺序，输入参数在前，输出参数在后。   
  输入参数一般为传值和常数的引用，输出参数或输入输出参数一般为非常数指针。   
  【优势】通过位置明确函数参数的作用，有利于代码的合并。

class foo;

void func(int nInputNum,

foo& rInputFoo,

foo\* pInputFoo,

void\* output);

* 头文件包含的顺序（之间用空行）   
  类对应头文件   
  C系统库文件   
  C++系统库文件   
  其他系统库文件   
  本项目内头文件

【优势】增加代码可读性，利于代码的合并

class <foo/public/class.h>

class <stdio.h>

class <math.h>

class <iostream>

class <vector>

class <projClass1.h>

class <projClass2.h>

* 局部变量的初始化   
  尽可能小的作用域中申明变量，离第一次使用的位置越近越好。明确包含使用的顺序。

【优势】使代码易读

int nNum = 0;

float fNum;

fNum = 0;

* 局部类的初始化   
  一个类的局部变量会默认做一次构造函数和析构函数，应注意避免多次的初始化和析构。

class Foo;

Foo f1;

for (int i = 0 ; i < 1000; ++i)

{

Foo f2; //会降低效率，多次初始化和析构

f1.doSomething();

}

* 类的初始化   
  养成良好的类成员申明的习惯。

class CStudent

{

public:

CStudent() : m\_id(0) {}

private:

size\_t m\_id;

};

* 类的拷贝构造和=运算符重载   
  注意深浅拷贝

class CStudent

{

public:

CStudent(const CStudent& rStu)

{

m\_name = new char[strlen(rStu.m\_name) + 1];

strcpy(m\_name, rStu.m\_name);

}

CStudent& operator=(const CStudent& rStu)

{

if (this == &rStu)

{

return \*this;

}

delete[] m\_name;

m\_name = new char[strlen(rStu.m\_name) + 1];

strcpy(m\_name, rStu.m\_name);

return \*this;

}

private:

char\* m\_name;

}

* 结构体和类   
  仅当只有数据成员时才使用结构体，其余情况一般使用类。结构体默认是public，类默认是private。   
  同时结构体也支持初始化构造和析构。

struct stStudent

{

char name[128];

size\_t id;

stStudent()

{

memset(name, 0, sizeof(name));

id = 0;

}

};

*//也可以直接初始化*

struct stStudent stu = {0};

* 操作符重载   
  操作符重载能带给我们便利，但是同时会使我们代码变得不直观。因此我们应该尽可能定义多的方法来取代操作符重载。

class CString

{

public:

CString& equals(const CString& rStr);

CString& operator=(const CString& rStr);

};

int main()

{

CString str1("hello");

CString str2;

str2.equals(str1); *//instead of str2 = str1;*

return 0;

}

* 将类的成员进行私有化   
  应尽可能将类的成员定义为private，并通过set和get函数对外公开。

【优势】有利于代码的封装性和代码的调试。

class CStudent

{

public:

void set(const char\* name) { strcpy(m\_name, name); }

char\* get() { return m\_name; }

private:

char\* m\_name;

}

* 类中的申明顺序   
  声明的顺序一般为，public，protected，private   
  在每个块中，成员函数需申明在成员变量之前，其他的申明顺序如下：   
  typedef和enums   
  常量   
  构造函数   
  析构函数   
  成员函数，包括静态成员函数   
  数据成员，包括静态数据成员

class CText

{

public:

*//typedef*

typedef vector<int> vecInt;

*//常量*

const int kNum;

*//构造和析构*

CText();

~CText();

*//普通方法*

void doSome();

*//静态成员*

static int m\_sId;

*//静态方法*

static int getID（）;

protected:

private:

};

* 编写短小的函数   
  如果没有特殊的需求，函数代码的长度应该控制在40行左右，如果过长的情况下且不影响程序的运行时，可将长函数进行切割。

【优势】提取重复的代码，易于他人的阅读和修改，容易发现和定位bug

* 输入的引用参数需要添加const   
  如果传入的参数是引用型，建议添加const

【优势】约束了调用行为，自解释了参数的作用

bool equal(const CPoint& p)

{

return (x == p.x) && (y == p.y);}

* 不使用函数的重载，而直接使用函数的名称进行区分   
  没有特殊要求尽量不进行函数的重载，而直接使用函数的名称进行区分，过多的函数重载会使调用者无从选择。

class CStudent

{

CStudent\* GetStudent(const int nID);

CStudent\* GetStudent(const char\* chName);

CStudent\* GetStudentFromID(const int nID);

CStudent\* GetStudentFromName(const char\* chName);

};

* 禁止使用缺省参数   
  函数的参数应该添加缺省的参数

【优势】可以避免错误的理解

*//避免使用的函数方法*

CStudent\* GetStudentFromID(const int nID,

bool isFromLocale = true);

CStudent\* GetStudentFormName(const char\* chName,

bool isFromLocale = true);

* 禁止使用RTTI   
  不使用dynamic\_cast

【优势】利用类型号和virtual关键字可以实现运行时确定类本身的功能

* 自增自减操作符重载   
  前置的自增自减效率高于后置自增自减，特别是对于迭代器。

vector<int> vecInt(10, 10);

for (vector<int>::iteartor it = vecInt.begin();

it != vecInt.end();

++it)

{

*//TODO:*

}

* 尽可能的使用const   
  如果函数不会修改传入的引用后指针类型的参数，返回的参数应该为const；   
  不修改数据的函数应定义为const；   
  如果成员变量在构造函数初始化之后不会改变，应声明为const；

【优势】约束变量的操作行为

*//类作为函数参数的时候，应该使用引用或者指针，减少类的拷贝构造，增加效率*

bool isEqual(const Class& a, const Class\* b);

* 0，NULL和初始化   
  整数使用0   
  浮点数使用0.0   
  指针使用NULL   
  字符串使用”\0”

int nNum = 0;

double dNum = 0.0;

char\* ptr = NULL;

char array[128] = "\0";

* sizeof的用法   
  应尽可能使用sizeof（变量名）来代替sizeof（类型）

【优势】防止变量类型在运行时改变

struct stBook

{

int BookId;

char BookName[128];

};

int main()

{

stBook book;

memset(&book, 0, sizeof(book)); *//not sizeof(stBook)*

return 0;

}

* 变量的命名   
  不要定义一些无法解释的名称变量   
  尽量避免错误的拼写或者拼写不完整   
  类名称全部以大写开头   
  结构体、枚举全部以大写开头   
  变量命名一律以小写开头   
  类成员变量带m\_开头   
  常量以k开头   
  函数名称以大写开头，每个单词首字母第一个大写   
  类的存取函数，存函数以Set开头，取函数以Get开头   
  枚举类型的值全部大写，每个单词以下划线隔开   
  全局变量以g开头   
  宏定义全部字母大写，每个单词以下划线分割

*//无法解释的名称变量*

int a, b, c;

*//错误的拼写或者拼写不完整*

int err, nam, cot;

*//类名以大写开头*

class CStudent;

*//结构体和枚举大写开头*

struct StBook;

*//变量命名使用小写*

bool isTrue;

int nNum;

*//类成员变量*

private:

int m\_id;

*//常量*

const int kCount;

*//函数*

void GetMyName();

*//存取函数*

int GetIndex();

void SetIndex(int nIndex);

*//枚举类型的值*

enum Week

{

W\_MONDAY = 0,

W\_TUESDAY,

}

*//全局变量*

int g\_nCount = 0;

*//宏定义*

#define MAX\_LEN 100