3.类

3.1 通常

3.1.1 按访问权限组织类的定义，顺序如下：public, protect, private。

3.1.2 定义类类型变量时用直接初始化而不要用拷贝初始化。

3.1.3 为管理资源的类定义拷贝构造函数，拷贝赋值运算符和析构函数。

3.1.4 用一个原子的、不抛出异常的swap()来实现拷贝赋值操作符。

3.1.5 确保operator=() 的实现正确，即便在自赋值，继承或有资源管理的情况下，并且其表现行为应该与内置的一致。

Scott Meyers 推荐框架如下：

A& A::operator= (const A& rhs)

{

if(this != &rhs)

{

Release\_All\_Resource\_Of(this);

Base::operator=(rhs);

Copy\_Member\_Data(this,rhs);

}

return \*this;

}

3.1.6 不要内联虚函数。

3.1.7 不要用inline方法的定义放在类内部，这样可以使类定义更加紧凑。

class C{

public:

int bar() {return 0;} //prefer

inline int car() {return 0} //avoid

inline int foo();

};

inline int C::bar() //avoid{

return 0;

}

3.1.8 将任何不改变类的逻辑状态的成员函数声明为const。

3.1.10 禁止用类型转换运算符进行类型转换。因为类型转换能在程序员不察觉到的情况下偷偷发生。所以，不要为类编写类型转换操作符，这样还不行，构造函数也可能偷偷进行类型转换，因此，凡是单形参的构造函数，都要声明为explicit，这样才能把全部源头堵住。

3.1.12 为每个类提供一个输出操作符（operator<<），方便调试输出。

3.1.13 所有的类都要有四个函数：default constructor, copy constructor, copy assignment operator, destructor。

3.2 构造函数和析构函数

3.2.1 确保所有构造函数为每个虚继承基类、非虚继承基类、和所有非静态数据成员提供了一个初始值。在初始化列表中，各个成员排列顺序如下：

1. 基类按继承深度从左到右排列。

2. 非静态数据成员按声明的顺序排列。

下面是一个例子：

class B {};

class VB:public virtual B {};

class C {};

class DC :public VB,public C

{

public:

DC():B(),VB(),C(),i(0),c() //correct order of initialization

{}

private:

int i;

C c;

};

3.2.6 不要把构造函数和析构函数定义成inline的。

因为如果构造函数是内联函数，每次声明一个对象的地方都会插入构造函数的代码，会造成代码膨胀。析构函数的道理类似。

3.3 继承

3.3.1 只使用public继承。

3.3.2 基类的析构函数应该为virtual。

如果一个delete 一个继承类指针，如果基类的析构函数不是virtual的，将只会调用基类的析构函数，而不会调用子类的析构函数。一般情况下，类的析构函数都应该是virtual的，以防将来维护或重用代码时添加继承类，那时还是需要将基类的析构函数改为virtual的。

3.3.5 一定要重写基类所有的重载函数。

class Base

{

public:

virtual void foo(short){

std::cout << "Base::foo(short)" << std::endl;

}

virtual void foo(double){

std::cout << "Base::foo(double)" << std::endl;

};

};

class Derived:public Base{

public:

virtual void foo(short){

std::cout << "Derived::foo(short)" << std::endl;

}

void test(void){

return foo(3.14);//calls Derived::foo(short) !

}

};

3.3.6 如果基类的某个函数没有被子类重写，说明它不需要为virtual，因此不要把它定义为virual的，减少不必要的开销。

3.3.7 基类的某个函数如果对于子类来说也是有效的操作，才可以把它定义为virtual的。

3.3.8 如果子类继承了基类的某个virtual函数，并且每个子类都重写了这个函数，实现了自己特定的操作，那么应该把基类的这个函数声明为纯虚函数。

3.3.9如果子类继承了基类的某个virtual函数，并且每个子类都重写了这个函数，实现的代码互相完全相同，则应该把基类的这个函数声明为非virtual的。

3.3.11 不要重载或覆盖基类的非虚函数。

3.3.12 当重新声明或override一个函数时，要使用相同的参数默认值。

参数默认值是静态决定的。

class Base

{

public:

virtual void goodvFn(int a = 0);

virtual void badvFn(int a = 0);

};

class Derived:public Base

{

public:

virtual void goodvFn(int a = 0);

virtual void badvFn(int a = 10);

};

void foo(Derived& obj){

Base& baseObj = obj;

// Ok - derived and base have the save default value

//

baseObj.goodvFn(); // calls Derived::goodvFn with a = 0

obj.goodvFn(); // calls Derived::goodvFn with a = 0

//Uses default value from base even though calls derived function

//

baseObj.badvFn(); // calls Derived::badvFn with a = 0

obj.badvFn(); // calls Derived::badvFn with a = 10

}

3.3.13 不要在构造函数或析构函数里调用虚函数。

在构造函数或西沟函数里调用虚函数，调用的总是当前类或当前类的基类的函数。

3.3.15 确保基类在子类中只有一个实例。

3.3.16 在子类中一定将多态函数声明为虚函数。

虽然不是必要的，但这是为了不造成不必要的误解，增强可读性。

class A{

public:

virtual void f();

virtual void operator+(A const&);

virtual ~A();

};

class B:public A{

public:

virtual void f(); //virtual: make explicit

virtual void operator+(A const&); //virtual: make explicit

virtual ~B(); //virtual: make explicit

};

3.4 面向对象设计

3.4.1 数据成员应全部为private。

3.4.2 const成员方法不应返回指向数据成员的非const指针或引用。

3.4.3 成员方法不应返回比自己访问权限更高的数据成员的指针或引用。

3.4.4 尽量避免friend函数，除非有一个设计上合理的考虑。

3.4.5 基类都应该是抽象类。

3.4.7 基类的所有成员对于子类来说必须是有意义的。

3.5 操作符重载

3.5.1 不要重载逗号运算符（,），逻辑与（&&）和逻辑或（||）。

逗号运算符是从左到有求值的，当它重载后，就变成了函数，而函数的参数求值顺序是未定义的。逻辑与和逻辑或有短路语义，而重载后变成了函数，就没有这个特性了。

3.5.2 确保重载后的运算符与原来有一致的表现。

3.5.4 二元操作符不应该是类的成员，这样可以允许左操作数进行隐式转换。

class complex{

public:

complex(float r, float i = 0);

complex operator+(const complex& rhs);

};

void Add(){

complex a(1,0);

a = a + 2; // fine:2 is converted to complex

a = 2 + a; // error: no applicable operator +

}

3.5.5 如果重载[]，一定要实现const 和非const两个版本。

4. 复杂度

4.3 函数不应有太多形参。

5 流程控制

5.1 流程控制关键词（if, else, while, for, do, switch）后必须有花括号，哪怕只有一行代码。

5.2 布尔表达式必须显示的用等于或不等于。

5.4 switch的每个case后都必须跟有break。

5.5 不要在for循环体内修改循环变量。

5.7 循环变量应该与一个常量相比较。

//avoid

//

for(int i = 0; i < xxx.size(); ++i)

{}

//prefer

//

const int list\_size = xxx.size();

for(int i = 0; i < list\_size ++i)

{}

5.8 禁止使用goto。

5.9 确保每个复合语句（复合语句当然包括函数）只有一个入口，一个出口。

5.12 在for语句内定义循环变量（C语言不适用）。

6 常量

6.1 long, unsigned int, unsigned long常量要有后缀L, U, UL；float, double常量要有后缀F, L。

6.2 字符串字面量不要和宽字符串字面量相邻，这是未定义的。

7 类型转换

7.1 禁用饮食转换，只使用 cast：static\_cast, const\_cast, dynamic\_cast和reinterpret\_cast或显式调用构造函数。

7.2 尽量减少使用cast。

8 声明和定义

8.1 结构体

8.1.1 头文件中应只有下列内容：非内联函数，全局变量，枚举，结构体，类的声明，inline函数和模板的定义。

8.2 域

8.2.1 禁止覆盖外层域中的名字。

8.2.2 尽量减少使用全局变量。

class Application{

//

};

Application& theApp(){

static Application app;

return app;

}

8.2.3 禁止使用using指示，用名字空间前缀。

8.3 语言限制

8.3.1 声明对象时避免使用static，将其放在名字空间内。

8.3.5 在函数风格的类型转换和声明之间不要使用有歧义的语法（即声明时不要使用不必要的括号）。

class A {};

A a; // ok

A (b); //ambiguous

8.4 对象声明和定义

8.4.2 一行只声明一个变量，如果变量不是自注释的，要添加注释。

8.4.3 定义对象时即要初始化。

8.4.8 不要typedef数组类型。

如果使用了，会不合直觉，容易出错。

typedef int ARRAY\_TYPE[10];

int \*array = new ARRAY\_TYPE;

delete array; //incorrect, should be delete [].

8.4.10 禁止指向成员的指针。

8.4.11 尽可能的使用const（use const whenever possible）。

8.4.12 \*, &要靠近标志符，不要靠近类型。

8.4.13 使用有符号数值类型，不要使用无符号数值类型。

9 异常

9.1 不要使用异常。

10 表达式

10.1 禁止使用魔幻数。

10.3 不要假设子表达式的求值顺序。

10.4 使用圆括号来指明表达式的意图，不要依赖操作符的优先级。

10.5 不要使用赋值运算符的结果，仅使用它的副作用。

int main(int argc, char\*\* argv){

int i = 1;

int j = 2;

// confusing use of assignment operator, always discard the result

//

if((j = i) == 1){

std::cout << "hit" << std::endl;

}

// prefer to write

//

j = i;

if(1 == j){

std::cout << "hit" << std::endl;

}

return j;

}

10.6 当变量与常量比较相等时，将常量放在==的左边。

10.9 &&, ||, sizeof 和typeid 的右边不要有副作用。

10.19 不要使用逗号运算符（,）。

10.20 不要使用条件运算符（?:）。

11 函数

11.2 将所有内连接的全局函数放入未命名空间，不要使用static。

11.3 在函数的声明和定义处都要指定参数名字，不要省略。

11.4 优先使用传引用调用，传值调用和传

11.6 不要使用…来定义参数个数不定函数。

11.9 不要重载数值和指针两个类型的函数。

void f(char);

void f(class X\*);

void test(){

f(0); // ambiguous

f(1); // calls f(char)

f('1'); // calls f(char)

}

12 内存管理

12.1 重载函数不要使用默认参数值，容易引起歧义。

12.2 使用new 和 delete，不要使用 malloc 和 free。

12.5 不要返回指针指向函数内动态分配的对象（容易造成内存泄漏）。

12.6 如果重载了new，一定重载delete。

12.7 将重载操作符new 和delete显式声明为static的，虽然它们隐式就是static的。

12.8 当使用delete后一定要立即将指针设为NULL，如果delete后对指针立即赋值或在析构函数中，则不必要。

13 可移植性

14 预处理

14.1 不要在源文件中使用Tab键。

14.5最好使用是否存在某个符号定义来控制条件编译（即使用#ifdef,#ifndef而不要使用#if）。

14.6 使用\_cplusplus来区分C和C++代码。

14.7 在宏中不要包含C注释。

14.9 使用<>来include系统和标准库，使用””来include其他库。

14.13 编写头文件时确保其需要的其他头文件都被包含进来。

14.14 宏的参数和宏体都要用小括号包起来。

14.17 不要用#define来定义常量，用const或枚举常量。

14.19 不要使用函数宏，用inline函数代替之。

15 结构体，联合和枚举

16 模板

17 STL

17.5 不要将试图将子类插入晕本存放基类的容器。

17.6 用empty()而不是判断size()是否为零。

17.7 不要继承STL容器，因为它们没有虚析构函数。

17.8 不要创建包含auto\_ptr的容器。

17.9 用vector和string来代替需要动态分配内存的数组。

17.10 尽可能预先分配足够内存，减少容器重新分配内存的次数。

17.11 将vector传递给C风格的函数时使用&v[0]。

17.13 禁止使用vector<bool>。

17.14 当等价时间谓词应返回FALSE。

// 大部分排序算法用如下方式判断两个对象是否等价

// 如果等价时返回TRUE，那么排序算法永远无法探测到两个对象等价

bool areElementsEqual(T &a, T &b){

if(!pred(a,b) && !pred(b,a)){

return true;

}

}

17.15 不要修改set或multiset的key部分，否则会破坏其内部有序性。

17.17 谓词结果应只依赖它的形参。

17.19 优先使用容器的成员函数，通用算法其次。