1. 非常量引用的初始化必须是一个左值
2. 可以将变量赋值给常量指针，但是不允许通过该指针修改该变量的值，常量指针本身是一个变量，在定义的时候可以不用初始化。
3. Void\* 不能间接引用，不能增量

Void\* pv=pi;

\*pi;

1. 要使用void\*指针就必须将它转换到某个指向特定类型的指针
2. 不允许两个结构体之间直接复制
3. New->delete, new int[ ]->delete[ ]
4. 函数形参的类型要匹配
5. Void update(float& i)

Void g(double d, float r)

{

Update(3.0f)//×;

Update(r);

Update(d);//×

}

对非const引用参数不允许做类型转换，也不允许是一个数，必须是左值。

1. 只能对排在最后的那些参数提供默认值
2. 不要返回局部变量的指针或者引用,除非该局部变量是静态的static
3. Using namespace std; using std::cout;
4. 在所有函数之外定义的变量应该在main函数之前，完成初始化
5. 非成员函数禁止访问类的私有成员
6. 构造函数没有返回类型
7. 静态成员函数可以直接通过类名引用，而不通过类的对象。
8. 在const函数里，不能修改成员值
9. 非const 成员函数，只能对非const对象调用。
10. Const或非const对象都可以调用const成员函数

Void f(const Date& cd)

{cd.add\_year(1);//错误}

1. 函数在类的内部定义，称为在线成员函数
2. 构造函数不允许有返回值及类型
3. 没有默认构造函数的类，在定义时，必须实例化，参数的形式与构造函数相同
4. 静态数据成员必须在类的外部进行定义

Class date{

Static int n;

};

Int Date::n=0;//静态数据成员的定义

1. 静态数据成员的访问方法：通过对象，通过类型名，在类的方法中可以直接访问
2. 静态函数成员只允许访问类中的静态数据成员，静态函数成员
3. ~X（）；{ }
4. 包含const或者引用成员的类不能进行默认构造，除非显示提供默认构造函数
5. 析构函数的析构顺序和构造函数的构造顺序相反
6. Class X{

X const X &）//复制构造函数

简写形式：

X (const X& c): re(c.re), in(c.im) {}

X &operator =(const X &)//复制赋值};

X::X(const X& t)

{}

X& X::operator=(const X& t)//把返回类型放到最前面

{}

1. 成员初始式列表：： club::club(const string& n, Date fd):name(n), members(),officers(),founded(fd) { }
2. 当对象的某个数据成员为指针时（对象后定义先释放内存），在对象生成时赋值，用复制构造函数；在对象生成后赋值，用复制赋值
3. 二元运算符可以定义为取一个参数的非静态成员函数，第一个参数由类的默认对象（\*this）传入；或者定义为取两个参数的非成员函数
4. 在操作符重载中，不允许所有的参数全是基本类型；第一个操作数是基本类型的，不能定义为成员函数
5. Operator= operator[] operator() operator->只能定义为非静态成员函数，保证其第一个参数一定是一个左值
6. 某些运算符的重载可能是私有的，这就不允许，在类的外面不通过类的成员函数直接使用
7. 如果某个类型显示地声明了构造函数，就不能再用初始式成员列表
8. 构造函数、析构函数、虚函数定义为成员函数
9. 带初始值的成员初始式列表可以解决参数的缺省构造问题

X（int a=0, int b=0）: c(a),d(b) {}

1. 在重载操作符时，采用传引用，不传地址
2. 子类包括的父类的所有成员，再加上自己特殊的成员
3. 当通过指针或者引用操作时，派生类的对象可以被当做基类来看待
4. 将子类赋值给基类不需要显示类型转换
5. 基类必须定义之后，才能去声明子类
6. 子类不能使用父类的私有成员变量，可以使用父类的public and protected object.
7. 在子类中重新定义了父类中的一些成员函数时，在子类中再调用父类的同名成员函数要加上父类的类名，避免混淆。
8. 在调用子类的构造函数时，不可以直接给父类的私有成员赋值，应该利用父类的构造函数，给父类中相应的私有变量赋初值

Class X

{

Int a,int b;

Public:  
X(int a1, int b1):a(a1),b(b1) {}

};

Class Y: public X{

Int c,int d;

Public:

Y(int a1,int b1,int c1,int d1): X(a1,b1), c(c1), d(d1) {}

};

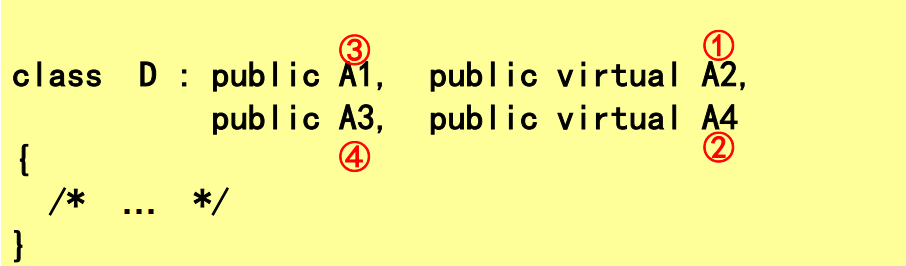
1. 父类中的某个成员函数时virtual函数，那么在子类中，对该成员函数的描述的参数类型（函数名，形参表，返回值类型）必须和父类严格保持一致，只允许在参数类型上有小小的改变。
2. 虚函数如果不是纯虚的就必须在这个类中必须定义。子类并不一定要重置父类中的虚函数
3. Virtual void draw() =0;//纯虚函数所在的类是抽象类，不能实例化，只能作为其他类的父类。可以定义指向抽象类的指针。
4. 派生类构造时需要调用父类的构造函数，沿着继承路径，上朔到类层次结构的最顶层（即一个没有基类的类），然后自顶向下地沿该路径逐一调用对应类的constructor。

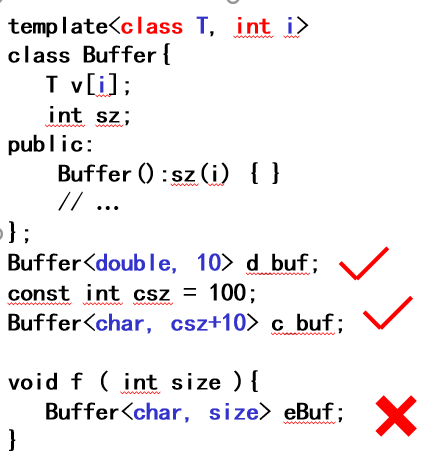
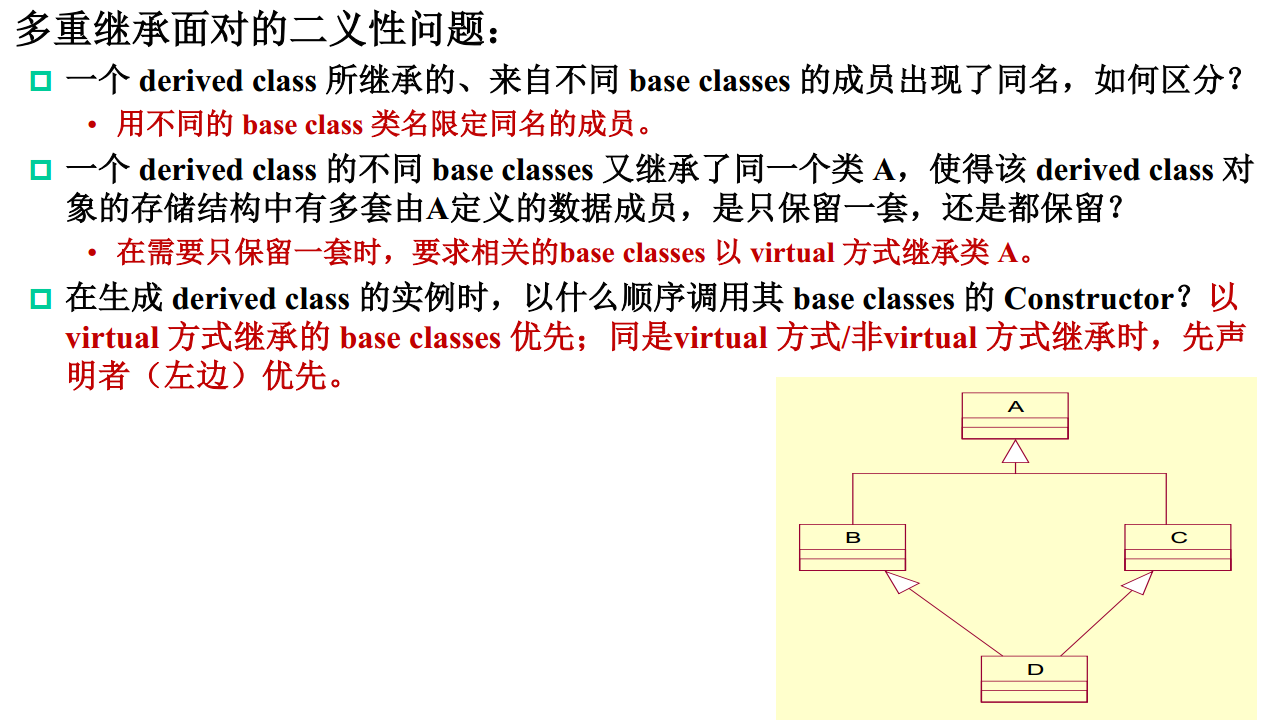
沿着继承路径，上朔到类层次结构的最顶层（即一个没有基类的类），然后自顶向下地沿该路径逐一调用对应类的constructor。若基类定义了多个Constructor，则派生类的 Constructor 按重载解析规则调用基类的对应Constructor。子类对象销毁时，自动调用父类的析构函数，析构的顺序与构造相反

1. 一个子类继承了多个父类，在继承的时候，用父类的名字来区别对待。
2. 派生类所继承的基类成员的能见度：

外部能见度：子类的对象不允许调用父类的任何成员，除非子类的继承方式是public，也只能调用父类的public成员。内部能见度：在子类中，不管继承方式是什么，只允许在子类中，调用父类的public成员。

1. 25、31在类的多态继承中，仍然适用
2. 多个直接父类的继承方式，虚继承优先，然后按照继承声明的顺序



1. ：的位置在子类名字的后面，继承方式的前面
2. 在一个类模板的参数列表中，先声明的类型，可以立即用于定于后续的形参
3. 合法的模板实参：一个类型，或者常量表达式
4. 类模板在声明对象之前，必须先声明参数列表中的所有类型，也就是将其实例化
5. 在异常处理中，如果throw一个错误就必须catch它。
6. 类的实例是对象，模板类的对象是类
7. 类在编译时生成
8. 函数模板的类型参数在实例化的时候，对实参不进行隐式类型转换
9. 一个类模板可以继承了另一个类模板，一个类模板可以继承另一个类。
10. 一个队列实例：

template<class T, unsigned int maxsize>

class Queue{

T arr [ maxsize ]; // 队列存储结构，声明一个数组用来存储队列中的元素

int curr\_number; // 当前元素个数， 判断队列是否空或者满

public:

Queue() { curr\_number = 0; }

bool empty() { return curr\_number <= 0; }

bool full() { return curr\_number >= maxsize; }

void push (const T& v); // 入队

T pop (); // 出队

};

template<class T, unsigned int maxsize>

void Queue<T, maxsize>::push(const T& v){

if( full() ){

cout << "\*\*\* Queue is full." << endl;

return;

}

// 入队：队尾

arr[curr\_number++] = v;

}

template<class T, unsigned int maxsize>

T Queue<T, maxsize>::pop() {

if( empty() ){

T dummuy; //???

cout << "\*\*\* Queue is empty." << endl;

return dummuy;

}

T v = arr[0]; // 队头元素 出队

for(int i = 0; i < curr\_number-1; i++)

arr[i] = arr[i+1];

curr\_number--;

return v;

}

**int main() {**

**Queue q;**

**--q; // Display：The queue is empty!**

**q = q + 5 + 6 + 3;**

**cout << q; // Display：5 6 3**

**cout << --q << endl; // Display：5**

**cout << q; // Display：6 3**

**q = 9 + q;**

**cout << q; // Display：6 3 9**

**q = 2 + q; // Display：The queue is full!**

**cout << q; // Display：6 3 9**

**return 0;**

**}**

**class Queue {**

**public:**

**enum { MAXSIZE = 3 };**

**Queue() { rear = 0; }**

**bool isEmpty() { return rear == 0 ? true : false; }**

**bool isFull() { return rear == MAXSIZE ? true : false; }**

**Queue operator+(int d)**

**{ if (isFull()) cout << " The Queue is full.\n";**

**else elem[rear++] = d;**

**return \*this;**

**}**

**friend Queue operator+(int d, Queue q)**

**{ if (q.isFull()) cout << " The Queue is full.\n";**

**else q.elem[q.rear++] = d;**

**return q;**

**}**

**int operator--();**

**friend ostream& operator<< (ostream&, Queue q);**

**private:**

**int elem[MAXSIZE];**

**int rear;**

**};**

**int Queue::operator--() {**

**if(isEmpty()) {**

**cout << "The queue is empty!\n";**

**return -1;**

**}**

**int value = elem[0];**

**for(int i = 0; i < rear; i++)**

**elem[i] = elem[i+1];**

**rear--;**

**return value;**

**}**

**ostream& operator<<(ostream& out, Queue q)**

**{**

**for(int i = 0; i < q.rear; i++)**

**out << q.elem[i] << '\t';**

**out << endl;**

**return out;**

**}**

1. 字符用单引号，字符串用双引号
2. 数组名不能作为左值
3. 复制构造函数不要忘记在类型的后面加上&
4. []重载，要有参数，用来说明是哪一个位置的变量。