1. constructor (non-copy)

a. for the member that is a kind of class

1) there is a default constructor in the member(including synthesized constructor)

If the constructor does not init the member explicitly, the default constructor of the member will be inserted to the constructor of the class.

2) there is no default constructor in the member(but have other constructor)

It should init the member explicitly in the memberlist or an compiler error will be arised.

3) there is no constructor in the member

synthesized constructor of the member will be insert to constructor: If a class does not have any constructor it will synthesize a constructor for us, or not.

\* synthesized constructor calls all the default constructor of each class member. So class member must have default constructor

\* it will not init the raw data member (int, long...)

b. for the member of raw data

The data should init it by the programmer in the constructor

2. memberinit list

1) init the member should follow orders of variables definition in the list

2) must init in the member init list: const member, reference member, non-default constructor class member/base member

3. copy type: copy constructor/copy assignment(operator=)

a. if the class do not have this copy type function, it will synthesize the function. For the base item or member item

\* the item is a class: if the item has the copy function, the compiler inserts the copy function . If not synthesize one for this item and insert it

\* the item is a raw type: the compiler insert it in the bitwise copy ways.

b. There is a copy type function defined by ourselves for the class

It should explicitly copy all the item (base, member[class/raw] item).

For base:

\* Copy constructor:Base(rhs) in the member list

\* Copy assignment: Base::operator=(rhs)

c. notice:

1) for copy assignment: must check whether the parameter is itself. If (this == &rhs) return \*this;

2) if a class has const or reference member: must define its own copy type function or error “can't use default assignment operator”;

4. destructor

a. A class always synthesizes a destructor if it doesn’t have one.

1). The destructor should only care about the raw data destruct of it’smembers.

2). For the base class and class member.

If the base class and member do not have the destructor, the compiler will not synthesize it in to the destructor. Or the destructor of the member will be inserted to the destructor of the class.

b. a pure virtual destruction must have a definition. If it needs, every derived class destructor will be expanded by the compiler that calls the destructor of each base class.

c. declare destructors as a virtual function in the polymorphic base class. A heap object of Derive class is pointed by a Base class pointer such as Base \*b = new Derive(); Then delete it delete b; It will call the destructor of Derive class firstly if the destructor is a virtual function or not.

5. Vptr initialization

a) constructor: it will be initialized after the base class initial and before the memberlist.

b) destructor: it will be uninitialized or point to base class one after all the member destructor and Its own destructor are called. Before the base class destructor is called.

c) It’s not a good idea to call virtual function in the constructor or destructor with this to show polymorphic feature.

1) in the structors: the vptr always point to its own type. Even the Derive class constructor initialize the Base class in the member list( with Base constructor), the vptr in the calling Base constructor also point to the Base.

2) usually the function will access the member data. But in the constructor the member data maybe not be initialized before the virtual function is called.