本文作者：黄邦勇帅

学习本文首先你应熟悉 C++中的构造函数，基本的类的声明及怎样初始化类，关于这些问题，请参看本人所作的《C++构造函数,复制构造函数和析构函数》一文，在这篇文章中作了详细的介绍。
本文分两部分即继承和虚函数与多态性，本文第一部分详细讲解了继承时的构造函数和析构函数的问题，父类与子类的同名变量和函数问题，最后介绍了多重继承与虚基类。本文第二部分重点介绍了虚函数与多态性的问题，因此学习虚函数的基础是继承，因此在学习虚函数前应学好继承。本文详细易懂，内容全面，是学习 C++的不错的资料。
本文内容完全属于个人见解与参考文现的作者无关，其中难免有误解之处，望指出更正。

主要参考文献：
1、C++.Primer.Plus.第五版.中文版 [美]Stephen Prata 著 孙建春 韦强译 人民邮电出版社 2005 年 5 月
2、C++.Primer.Plus.第四版.中文版 Stanley B.Lippman、Barbara E.Moo 著 李师贤等译 人民邮电出版社 2006 年 3 月
3、C++.Primer.Plus.第三版.中文版 Stanley B.Lippman 等著 潘爱民 张丽译 中国电力出版社 2002 年 5 月
4、C++入门经典 第三版 [美]Ivor Horton 著 李予敏译 清华大学出版社 2006 年 1 月
5、C++参考大全 第四版 [美]Herbert Schidt 著 周志荣 朱德芳 于秀山等译 电子工业出版社 2003 年 9 月
6、21 天学通 第四版 C++ [美]Jesse Liberty 著 康博创作室 译 人民邮电出版社 2002 年 3 月

14 继承(基类，父类，超类)，派生类，子类

一：继承中的访问权限关系。

1．基类，父类，超类是指被继承的类，派生类，子类是指继承于基类的类.
2．在 C++中使用：冒号表示继承，如 class A:public B；表示派生类 A 从基类 B 继承而来
3．派生类包含基类的所有成员，而且还包括自己特有的成员，派生类和派生类对象访问基类中的成员就像访问自己的成员一样，可以直接使用，不需加任何操作符，但派生类仍然无法访问基类中的私有成员.
4．在 C++中派生类可以同时从多个基类继承，Java 不允许这种多重继承，当继承多个基类时，使用逗号将基类隔开.
5．基类访问控制符，class A:public B 基类以公有方式被继承，A:private B 基类以私有方式被继承，A:protected B 基类以受保护方式被继承，如果没有访问控制符则默认为私有继承。
6. protected 受保护的访问权限：使用 protected 保护权限表明这个成员是私有的，但在派生类中可以访问基类中的受保护成员。派生类的对象就不能访问受保护的成员了。
7. 如果基类以 public 公有方式被继承，则基类的所有公有成员都会成为派生类的公有成员．受保护的基类成员成为派生类的受保护成员
7．如果基类以 private 私有被继承，则基类的所有公有成员都会成为派生类的私有成员．基类的受保护成员成为派生类的私有成员.
8．如果基类以 protected 受保护方式被继承，那么基类的所有公有和受保护成员都会变成派生类的受保护成员.
9．不管基类以何种方式被继承，基类的私有成员，仍然保有其私有性，被派生的子类不能访问基类的私有成员.

例：继承中的访问权限关系

```
class A   {int a;    protected:int b;     public:int c;    A(){a=b=c=1;}   };
//类B以公有方式从基类A继承
class B:public A
{public: int d;
B(){//a=2; //错误，不能访问基类中的私有成员
    b=2;   //正确，可以在类中访问基类中的受保护成员，但类的对象不能访问，基类中的受保护成员b在类B中仍然是受保护成员
        c=d=2;} };  //基类中的公有成员c在类B中仍然是公有成员
//以受保护和私有方式从基类A继承。
class C:protected A    {public: int e;    C(){//a=3; //错误，不能访问基类中的私有成员
                                  b=c=e=3; }};//这里基类受保护成员b和公有成员c都成为类C中的受保护成员。
class D:private A  {public: D(){b=c=4;} };//基类中的公有和受保护成员都成为了类D中的私有成员。
//验证受保护和私有方式继承的访问权限。
class C1:public C
{public: C1(){b=c=e=4;} };//正确;类A中的成员b和c在类C中是以受保护方式被继承的，b和c都成为了类C中的受保护成员。
class D1:public D
{public:D1(){//b=5; //错误，在A中受保护的成员b在类D中是以私有方式继承的，这样b就成为了类D中的私有成员，所以无法访问。
```

```
//c=5; //错误，在A中公有的成员c在类D中是以私有方式继承的，这样c就成为了类D中的私有成员，所以无法访问。} };
int main()
{A m1; B m2; C m3; D m4;
//cout<<m1.b<<m2.b<<m3.b<<m4.b;  //错误；不能用类的对象访问受保护的成员，只有在类中才能访问。
cout<<m1.c;   cout<<m2.c;
//cout<<m3.c; //错误，类C是以受保护的方式从A继承的，基类中的变量c在类C中就是受保护的，所以类的对象不能访问
//cout<<m4.c; //错误，类C是以私有的方式从A继承的，基类中的变量c在类C中就是私有的，所以类的对象不能访问 }
```

二：覆盖和隐藏基类成员变量或成员函数。

1. **基类的成员变量或函数被覆盖：**如果派生类覆盖了基类中的成员函数或成员变量，则当派生类的对象调用该函数或变量时是调用的派生类中的版本，当用基类对象调用该函数或变量时是调用的基类中的版本。
2. **隐藏基类成员函数的情况：**如果在派生类中定义了一个与基类同名的函数，不管这个函数的参数列表是不是与基类中的函数相同，则这个同名的函数就会把基类中的所有这个同名的函数的所有重载版本都隐藏了，这时并不是在派生类中重载基类的同名成员函数，而是隐藏，比如类 A 中有函数 f(int i,intj)和 f(int i)两个版本，当在从 A 派生出的类 B 中定义了基类的 f()函数版本时，这时基类中的 f(int i)和 f(int i,int j)就被隐藏了，也就是说由类 B 创建的对象比如为 m，不能直接访问类 A 中的 f(int i)版本，即使用语句 m.f(2)时会发生错误。
3. **怎样使用派生类的对象访问基类中被派生类覆盖或隐藏了的函数或变量：**
   - 3.1. 方法 1 使用作用域运算符::，在使用对象调用基类中的函数或变量时使用作用域运算符即语句 m.A::f(2)，这时就能访问基类中的函数或变量版本。注意，访问基类中被派生类覆盖了的成员变量只能用这种方法
   - 3.2. 方法 2 使用 using：该方法只适用于被隐藏或覆盖的基类函数，在派生类的类定义中使用语句 using 把基类的名字包含进来，比如 using A::f；就是将基类中的函数 f()的所有重载版本包含进来，重载版本被包含到子类之后，这些重载的函数版本就相当于是子类的一部分，这时就可以用派生类的对象直接调用被派生类隐藏了的基类版本，比如 m.f(2)，但是使用这种语句还是没法调用基类在派生类中被覆盖了的基类的函数，比如 m.f()调用的是派生类中定义的函数 f，要调用被覆盖的基类中的版本要使用语句 m.A::f()才行。
4. **在派生类的函数中调用基类中的成员变量和函数的方法：**就是在函数中使用的被派生类覆盖的基类成员变量或函数前用作域解析符加上基类的类名，即 a::f()就是在派生类的函数中调用基类中被派生类覆盖了的函数 f()的方法。
5. **派生类以私有方式被继承时改变基类中的公有成员为公有的方法：**
   - 5.1. 使用::作用域运算符，不提倡用这种方法，在派生类的 public 后面用作用域运算符把基类的 公有成员包函进来，这样基类的成员就会成为派生类中的公有成员了，注意如果是函数的 话后面不能加括号 如 A::f；如果 f 是函数的 话不能有括号。
   - 5.2. 使用 using 语句，现在一般用这种方法，也是在派生类的 public 使用 using 把基类成员包函进来，如 using A::f。

例：隐藏或覆盖基类中的成员，使用：：作用域运算符访问

```
class A {int a;     protected:int b;     public:int c,d;    void f(int i){cout<<"class A"<<"\n";}    A(){a=b=c=d=1;}  };
class B:public A {public:int d;  //覆盖基类中的成员变量d。
B(){b=c=d=2;  //这里是给子类B中的成员变量d赋值，而不是基类中的d
A::d=3; } //给基类中被覆盖的成员d赋值，注意在类中访问的被覆盖成员的方式。
void f(){cout<<"class B"<<"\n";//在子类中重定义基类中的同名函数，虽然参数列表不一样，但同样会隐藏基类中的同名函数
        A::f(1);   //在函数中调用基类中被隐藏了的同名函数的方法，使用作用域解析运算符。
        //f(1);  //错误，因为基类中的函数被子类中的同名函数隐藏了，在这里子类不知道有一个带参数的函数f。} };
int main()
{B m;     cout<<m.d<<"\n";    //输出子类中的成员变量d的值，注意派生类中覆盖了基类成员d.
cout<<m.A::d<<"\n"; //输出基类中的成员变量d的值，注意这是使用对象访问被覆盖的基类成员的方式
m.f();             //调用子类中的不带参数的函数f。
//m.f(2);  //错误，因为基类中的带一个参数的函数f被子类中的同名函数隐藏掉了，不能这样访问，须用作用域解析运算符来访问。
m.A::f(1); }  //使用子类对象访问基类中被隐藏的函数的方法。
```

例：使用 using 语句以便访问基类中被隐藏的函数

```
class A{int a;   protected:int b;   public:int c,d;    void f(){cout<<"Amoren"<<"\n";}
 void f(int i){cout<<"class A"<<"\n";}       A(){a=b=c=d=1;}   };

class B:public A  {public:int d;  //覆盖基类中的成员变量d。
B(){b=c=d=2;  //这里是给类B中的成员变量d赋值，而不是基类中的d
A::d=3; } //给基类中被覆盖的成员d赋值，注意在类中访问的被覆盖成员的方式。
using A::f;  //使用语句using把类A中的函数f包含进来，以便以后可以直接访问基类被隐藏了的函数，注意函数f没有括号
void f(){cout<<"class B"<<"\n";//在子类中覆盖基类中的同名函数，注意这里是覆盖，同时会隐藏基类中的其他同名重载函数
        f(1);  //正确，因为使用了using语句，所以可以在类中直接使用基类中f函数的重载版本。
         A::f(2) ;//正确，虽然使用了using语句，但同样可以按这种方法访问基类中的函数。
   A ma;
   ma.f();  //正确，在子类中创建的基类对象，可以直接用对象名调用基类中被子类覆盖或隐藏了的函数，因为这时不会出现二义性。
   ma.f(1);}  //正确，在子类中创建的基类对象，可以直接用对象名调用基类中被子类覆盖或隐藏了的函数，因为这时不会出现二义性。
void g(){cout<<"this g"<<"\n";  f();  //正确，但该语句访问的是子类中的不带参数函数f，虽然在类中使用了using语句，但直接调用被子类覆盖了的基类函数时不能使用这种方法
A::f();  }};//正确，调用被子类覆盖了的基类中的函数f，注意，虽然使用了using但要访问被子类覆盖了的函数，只能这样访问。
int main()
{B m;     m.f();   //调用子类中的不带参数的函数，这里不会调用基类中的不带参数的被覆盖的函数f。
m.A::f(); //调用基类中被子类覆盖了的函数f，虽然子类使用了using语句，但要访问基类中被覆盖的方法只能像这样使用。
```

```
m.f(1);  //调用基类重载的f函数，注意这里可以不用::运算符，因为在子类中使用了using，只要子类没有覆盖基类中的方法，都可以这样直接调用。
m.A::f(2); } //当然，使用了 using 后，也可以使用这种方法
```

例：派生类以私有方式被继承时改变基类中的公有成员为公有的方法

```
class A   {public: int a,b;       void f(){cout<<"f"<<"\n";}        void g(){cout<<"g"<<"\n";}  };
class B:private A
{public:   A::f;   A::a;  //使用::运算符使基类中的成员成为公有的。注意函数名后不能有括号。
           using A::g;   }; //使用using语句使基类中的成员函数g成为类B中的公有成员，注意函数名后不能有括号。

int main()
{ B m;
 //m.b=1;  //错误，因为类B是以私有方式继承的，类A中的成员在类B中是私有的，这里不能访问私有成员。
m.f(); m.g(); m.a=1;}
```

三：继承时的构造函数和析构函数问题

1. 在继承中，基类的 构造函数构建对象的基类部分，派生类的 构造函数构建对象的派生类部分。
2. 当创建派生类对象时 先用派生类的构造函数调用基类的构造函数构建基类 然后再执行派生类构造函数构造派生类。即先构造基类再构造派生类的顺序。执行析构函数的顺序与此相反。
3. 调用基类 带参数的构造函数的方法：在派生类的 构造函数中使用初始化列表的形式就可以调用基类带参数的构造函数初始化基类成员，如 B():A(int i){}，类 B 是类 A 的派生类。
4. 派生类的 构造函数调用基类的构造函数的方法为：
   4.1 如果派生类没有 显示用初始化列表调用基类的构造函数时 这时就会用派生类的 构造函数调用基类的默认构造函数，构造完基类后，才会执行派生类的构造函数函数体，以保证先执行基类构造函数再执行派生类构造函数的顺序，如果基类没有 默认构造函数就会出错
   4.2 如果派生类用 显示的初始化列表调用基类的构造函数时 这时就会 检测派生类的初始化列表 当检测到显示调用基类的 构造函数时，就调用基类的 构造函数构造基类 然后再构造派生类，以保证先执行基类构造函数再执行派生类 构造函数的顺序，如果基类没有定义派生类 构造函数初始化列表调用的构造函数版本就会出错
6. 如果在基类中没有定义 默认构造函数 但定义了 其他构造函数版本，这时派生类中定义了 几个构造函数的不同版本 这时只要派生类有一个 构造函数没有显示调用基类中定义的构造函数版本就会发生错误 因为编译器会首先检查派生类构造函数调用基类构造函数的匹配情况，如果发 现不匹配就会出错，即使没有创建 任何类的对象都会出错 而不管这个派生类的对象有没有调用派生类的这个 构造函数比如：基类有一个 构造函数版本A(int i)而没有定义 默认构造函数，派生类 B，有这几个版本的构造函数 B():A(4){}，B(int i):A(5){}，再有语句 B(int i, int j){}没有 显示调用基类定义的 构造函数而是调用基类的默认构造函数 如果创建了 B m 和语句 B m(1)时都会提 示没有可用的基类默认构造函数可用的错误，虽然这时类 B 的对象 m 没有调用派生类 B 的带有两个形参的构造函数，但同样会出错。
7. 同样的道理，如果基类中定义了 默认构造函数 却没有其他版本的构造函数，而这时派生类 却显示调用了基类构造函数的 其他版本，这时就会出错，不管 你有没有创建类的对象，因为编译器会先在创建对象前就检查构造函数的匹配问题。
8. 派生类只能 初始化他的直接基类 比如类 C 是类 B 的子类，而类 B 又是类 A 的子类，这时 class C:public B{public: B():A(){} };将会出错，该语句试图显示调用类B 的基类类 A 的构造函数，这时会出 现类A 不是类 C 的基类的错误。
9. 继承中的复制构造函数和构造函数一样，基类的 复制构造函数复制基类部分，派生类的 复制构造函数复制派生类部分。
10. 派生类 复制构造函数调用基类复制构造函数的方法为：A(const A& m):B(m){}其中 B 是基类，A 是派生类。
11. 如果在派生类中定义了 复制构造函数而没有用初始化列表显示调用基类的复制构造函数 这时不管基类是 否定义了复制构造函数，这时出 现派生类对象的复制初始化情况时就将调用基类中的默认构造函数初始化基类的成员变量，注意是默认构造函数不是默认复制构造函数，如果基类没有 默认构造函数就会出错 也就是说派生类的 复制构造函数的默认隐藏形式是B(const B& j):A(){}这里 B 是 A 的派生类，也就是说如果不 显示用初始化列表形式调用基类的复制构告函数时，默认情况下是用初始化列表的形式调用的是基类的默认构造函数。
12. 当在派生类中定义了 复制构造函数且显示调用了基类的复制构造函数 而基类 却没有定义基类的复制构造函数时，这时出 现派生类对象的复制初始化情况就将调用基类中的默认复制构造函数初始化基类部分，调用派生类的 复制构造函数初始化派生类部分，因为复制构造函数只有一种形式，即 A(const A& m){}，比如当出 现调用时A(const A& m):B(m){}如果这时基类 B 没有定义 复制构造函数，则该语句将会调用派生类 A 的默认复制构造函数。
12. 如果基类定义了 复制构造函数 而派生类没有定义时，则会调用基类的 复制构造函数初始化基类部分 调用派生类的默认复制构造函数初始化派生类部分。

例：基类不定义默认构造函数，派生类不定义复制构造函数而基类定义复制构造函数的情形。

```
//类A不定义默认构造函数
class A {public:  int a,a1;   A(int i){a=a1=11;cout<<"goucaoA1"<<"\n";}   A(const A& J){a=4; a1=4;cout<<"fugouA"<<"\n";}
 ~A(){cout<<"hahaA"<<"\n";}   };

//类B不定义复制构造函数
class B:public A
{public: int b,b1;
//B(){b=b1=2; cout<<"goucaoB"<<"\n";}     //错误，此语句会调用基类中的默认构造函数，而基类没有默认构造函数。
```

```
//B(int i,int j){b=b1=3;}  //错误，同上，此语句没有显示调用基类构造函数，就会调用基类的默认构造函数，而这时基类没有默认构造函数。
B(int i):A(2){b=b1=3;cout<<"goucaoB1"<<"\n";} ~B(){cout<<"hahaB"<<"\n";}};//显示调用基类带一个形参的构造函数，注意语法
int main()
{ B m(0);  cout<<m.a<<m.b<<"\n"; //输出113
  B m1(m);  cout<<m1.a<<m1.b;} //输出43，注意，此语句将调用基类定义的复制构造函数，然后调用派生类的默认复制构造函数。
```

例：派生类B使用默认的B(const B& K):A(){}形式，而不使用初始化列表显示调用基类A中的复制构造函数的情况

```
class A
{public:  int a,a1;   A(){a=a1=2;cout<<"goucaoA"<<"\n";}  A(int i){a=a1=11;cout<<"goucaoA1"<<"\n";}
 A(const A& J){a=4; a1=4;cout<<"fugouA"<<"\n";}    ~A(){cout<<"hahaA"<<"\n";}  };
```

//类中B不使用初始化列表调用基类中的复制构造函数的情况

```
class B:public A
{public: int b,b1;   B(){b=b1=2; cout<<"goucaoB"<<"\n";}
//B(int i,int j):A(4,4){b=b1=3;}  //错误，调用了基类没有定义的带两个参数的构造函数。
B(int i):A(2){b=b1=3;cout<<"goucaoB1"<<"\n";}
B(const B& K){b=b1=5;cout<<"fugouB"<<"\n";} //注意，这里没有显示调用基类的复制构造函数，而是用默认的B(const B& K):A(){}的形式调用的基类中的默认构造函数，注意是默认构造函数而不是默认复制构造函数。
~B(){cout<<"hahaB"<<"\n";}};
int main()
{ B m(0);  cout<<m.a<<m.b<<"\n";// 输出113。
  B m1(m); cout<<m1.a<<m1.b;}//输出 25，注意此语句将调用基类的默认构造函数将基类中的成员初始化为，再调用派生类的复制构造函数将类 B 中的成员 b 初始化为。
```

14.1.6 多重继承与虚基类

1. C++允许一个派生类从多个基类继承，这种继承方式称为多重继承，当从多个基类继承时每个基类之间用逗号隔开，比如 class A:public B, public C{}就表示派生类 A 从基类 B 和 C 继承而来。
2. **多重继承的构造函数和析构函数：**多重继承中初始化的次序是按继承的次序来调用构造函数的而不是按初始化列表的次序，比如有 class A:public B, public C{}那么在定义类 A 的对象 A m 时将首先由类 A 的构造函数调用类 B 的构造函数初始化 B，然后调用类 C 的构造函数初始化 C，最后再初始化对象 A，这与在类 A 中的初始化列表次序无关。
3. **多重继承中的二义性问题：**
   - 3.1. 成员名重复：比如类 A 从类 B 和 C 继承而来，而类 B 和 C 中都包含有一个名字为 f 的成员函数，这时派生类 A 创建一个对象，比如 A m; 语句 m.f()将调用类 B 中的 f 函数呢还是类 C 中的 f 函数呢?
   - 3.2. 多个基类副本：比如类 C 和 B 都从类 D 继承而来，这时 class A:public B, public C{}类 A 从类 C 和类 B 同时继承而来，这时类 A 中就有两个类 D 的副本，一个是由类 B 继承而来的，一个是由类 C 继承而来的，当类 A 的对象比如 A m;要访问类 D 中的成员函数 f 时，语句 m.f()就会出现二义性，这个 f 函数是调用的类 B 继承来的 f 还是访问类 C 继承来的函数 f 呢。
   - 3.3. 在第 2 种情况下还有种情况，语句 classA:public B,public C{}，因为类 A 首先使用类 B 的构造函数调用共同基类 D 的构造函数构造第一个类 D 的副本，然后再使用类 C 的构造函数调用共同基类 D 的构造函数构造第二个类 D 的副本。类 A 的对象 m 总共有两个共享基类 D 的副本，这时如果类 D 中有一个 公共成员变量d，则语句 m.B::d 和 m.D::d 都是访问的同一变量，类 B 和类 D 都共享同一个副本，既如果有语句 m.D::d=3 则 m.B::d 也将是 3。这时 m.C::d 的值不受影响而是原来的值。为什么会这样呢？因为类 A 的对象 m 总共只有两个类 D 的副本，所以类 A 的对象 m 就会从 A 继承来的两个直接基类 B 和 C 中，把从共同基类 D 中最先构造的第一个副本作为类 A 的副本，即类 B 构造的 D 的副本。因为 class A:public B,public C{}最先使用 B 的构造函数调用共同基类类 D 创造 D 的第一个副本，所以类 B 和类 D 共享同一个副本。
   - **3.4. 解决方法：**对于第 1 和第 2 种情况都可以使用作用域运算符::来限定要访问的类名来解决二义性。但对于第二种情况一般不允许出现两个基类的副本，这时可以使用 虚基类来解决这个问题 一旦定义了虚基类，就只会有一个基类的 副本

例：多重继承及其二义性

```
class A {public:int a; A(int i){a=i;cout<<"A";}};  //共同的基类A
class B:public A {public: int b;   B():A(4){cout<<"B";}};
class C:public A{public: int c;    C():A(5){cout<<"C";}};
class D:public B,public C
{public: int d;D():C(),B(){cout<<"D";}}; //先调用类B的构造函数而不会先调用类C的构造函数，初始化的顺序与初始化列表顺序无关
int main()
{D m;  //输出ABACD，调用构造函数的顺序为类ABACD，注意这里构造了两个类A的副本，调用了两次类A的构造函数
 // m.a=1; //错误，出现二义性语句，因为类D的对象m有两个公共基类A的副本，这里不知道是调用由类B继承来的A的副本还是由类C继承来的A的副本
m.B::a=1;cout<<m.B::a;  //输出1。
m.A::a=3; cout<<m.B::a<<m.A::a;//输出33，类B和类A共享相同的副本，调用类A中的a和类B继承来的a是一样的，因此最后a的值为3。
m.C::a=2;  cout<<m.C::a;} //输出 2，类 C 和类 A 类 B 的副本是彼此独立的两个副本，因此，这里不会改变类 B 和类 A 的副本中的 a 的值。
```

4. **虚基类：**方法是使用 virtual 关见字，比如 class B:public virtual D{}，class C:virtual public D{}注意关见字 virtual 的次序不关紧要。类 B 和类 C 以虚基类的方式从类 D 继承，这样的话从类 B 和类 C 同时继承的类时就会只创见一个类

D 的副本，比如 class A:public B, public C{}这时类 A 的对象就只会有一个类 D 的副本，类 A 类 B 类 C 类 D 四个类都共享一个类 D 的副本，比如类 D 有一个 公有成员变量 d，则 m.d 和 m.A::d，m.B::d，m.C::d，m.D::d 都将访问的是同一个变量。这样类 A 的对象调用类 D 中的成员时就不会出 现二义性了。

5. **虚基类的构造函数：**比如 class B:public virtual D{}；class C:virtual public D{}; class A:public B,public C{}；这时当创建类 A 的对象 A m 时初始化虚基类 D 将会使用类 A 的构造函数直接调用虚基类的构造函数初始化虚基类部分，而不会使用类 B 或者类 C 的构造函数调用虚基类的构造函数初始化虚基类部分，这样就保证了只有一个虚基类的副本。但是当创建一个类 B 和类 C 的对象时 仍然会使用类 B 和类 C 中的构造函数调用虚基类的构造函数初始化虚基类。

例：虚基类及其构造函数

```
class A {public:int a;   A(){cout<<"moA"<<"\n";}     A(int i){a=i;cout<<"A";} };
class B:public virtual A {public: int b;   B(int i):A(4){cout<<"B";}  };  //以虚基类的方式继承
class C:public virtual A {public: int c;    C():A(5){cout<<"C";}   };
class D:public B, public C  {public: int d;   D():A(4),C(),B(2){cout<<"D";}};
//因为类D是虚基类，所以类D会直接调用虚基类的构告函数构造虚基类部分，而不会使用类B或者类C的构造函数来调用虚基类的构造函数初始化虚基类部分。要调用虚基类中的带参数的构造函数必须在这里显示调用，如果不显示调用就将调用虚基类的默认构造函数。
int main()
{D m;  //输出ABCD，注意这里没有重复的基类副本A。
C m1;  //输出AC，虽然D是虚基类，但当创建类C的对象时仍然会使用类C的构造函数调用虚基类的构造函数初始化虚基类部分。
cout<<m.a;  //输出4，因为使用了虚基类所以这里就不存在二义性问题。
 m.B::a=1;
 cout<<m.a<<m.A::a<<m.B::a<<m.C::a;} //输出 1111，类 A，类 B，类 C，类 D 共享的是同一个虚基类的副本，所以这里输出四个 1。
```

15 虚函数(virtual 关见字)和多态性

一：继承中的指针问题。

1. 指向基类的指针可以指向派生类对象，当基类 指针指向派生类对象时，这种指针只能访问派生对象从基类继承而来的那些成员，不能访问子类 特有的元素 除非应用强类型转换，例如有基类 B 和从 B 派生的子类 D，则 B *p;D dd; p=ⅆ是可以的，指针 p 只能访问从基类派生而来的成员，不能访问派生类 D 特有的成员．因为基类不知道派生类中的这些成员。
2. 不能使派生类 指针指向基类对象
3. 如果派生类中覆盖了基类中的成员变量或函数，则当声明一个基类指针指向派生类对象时，这个基类 指针只能访问基类中的成员变量或函数。例如：基类 B 和派生类 D 都定义了函数 f，则 B *p; D m; p=&m; m.f()将调用基类中的函数 f()而不会调用派生类中的函数 f()。
4. 如果基类 指针指向派生类对象，则当对 其进行增减运算时，它将指向它所认为的基类的下一个对象，而不会 指向派生类的 下一个对象，因此，应该认为对这种指针进行的增减操作是无效的.

二：虚函数

1. **为什么要使用虚函数：**正如上面第 1 和 3 点所讲的，当声明一个基类指针指向派生类对象时，这个基类 指针只能访问基类中的成员函数，不能访问派生类中 特有的成员变量或函数 如果使用 虚函数就能使这个指向派生类对象的基类指针访问派生类中的成员函数，而不是基类中的成员函数，基于这一 点派生类中的这个成员函数就必须和基类中的虚函数的形式完全相同，不然基类指针就找不到派生类中的这个成员函数。使用虚函数就实现了一个接口多种方法。
2. **注意不能把成员变量声明为虚有的，也就是说 virtual 关见字不能用在成员变量前面。**
3. 正如上面所介绍的，一般应使用基类指针来调用虚函数，如果用 点运算符来调用虚函数就 失去了它的意义.
4. 如果基类含有 虚函数则当声明了一个基类的指针时，当基类 指针指向不同的派生类时，它就会调用相应派生类中定义的虚函数版本．这种调用方法是在运行时 决定的 例如在类 B 中声明了虚函数，C,D,E 都从 B 继承而来且都实现了自己的虚函数版本，那么当定义了一个 B 类的指针 P 时，当 P 指向子类 C 时就会调用子类 C 中定义的 虚函数，当 P 指向子类 D 时就会调用子类 D 中定义的 虚函数，当 P 指向子类 E 时就会调用子类 E 中定义的 虚函数
5. 虚函数须在基类中用 virtual 关见字声明也可以在基类中定义虚函数，并在一个或 多个子类中重 新定义．重定义 虚函数时不需再使用 virtual 关见字，当然也可以继续标明 virtual 关见字，以便程序更好理解。
6. 包括虚函数的类被称为**多态类**．C++使用 虚函数支持多态性.
7. 在子类中重定义 虚函数时，虚函数必须有与基类虚函数的声明完全相同的参数类型和数量，这和重载是不同的．如果不相同，则是函数重载，就失去了虚函数的本质.

8. 虚函数不能是声明它的类的友元函数，必须是声明它的类的成员函数，不过虚函数可以是另一个类的友元.
9. 一旦将函数声明为虚函数，则不管它通过多少层继承，它都是虚函数，例如D和B继承，而E又从D继承，那么在B中声明的虚函数，在类E中仍然是虚函数.
10. **隐藏虚函数：**如果基类定义了一个虚函数，但派生类中却定义了一个虚函数的重载板本，则派生类的这个版本就会把基类的虚函数隐藏掉，当使用基类指针调用该函数时只能调用基类的虚函数，而不能调用派生类的重载版本，当用派生类的对象调用基类的虚函数时就会出现错误了，因为基类的虚函数被派生类的重载版本隐藏了。
11. **带默认形参的虚函数：**当基类的虚函数带有默认形参时，则派生类中对基类虚函数的重定义也必须有相同数量的形参，但形参可以有默认值也可以没有，如果派生类中的形参数量和基类中的不一样多，则是对基类的虚函数的重载，对虚函数的重定义也就意味着，当用指向派生类的基类指针调用该虚函数时就会调用基类中的虚函数版本。比如基类定义virtual void f(int i=1, int j=2){}则派生类中必须定义带有两个形参的函数f才是对基类虚函数f的重定义，不然就是函数f的重载版本，比如派生类中定义的void f()，void f(int i)，void f(int i=2)都是对函数f的重载，不是对f的重定义。而void f(int i, int j)，void f(int i, int j=3)，void f(int i=4, int j=5)都是对虚函数f的重定义。
12. 如果虚函数形参有默认值，那么派生类中的虚数的形参不论有无默认值，当用指针调用派生类中的虚函数时就会被基类的默认值覆盖，即派生类的默认值不起作用。但用派生类的对象调用该函数时，就不会出现这种情况。
13. **当用指向派生类的基类指针调用虚函数时是以基类中的虚函数的形参为标准的，也就是只要调用的形式符合基类中定义的虚函数的标准就行了。**比如基类中定义virtual void f(int i=1,int j=2){}派生类中重定义为void f(int i, int j=3){}这时如果用派生类的对象调用这个派生类中的虚函数时必须至少要有一个实参，但是用指向派生类的基类指针调用该虚函数时就可以不用任何形参就能调用派生类中的这个函数f，比如语句p->f()就会调用派生类中的虚函数版本当用指向派生类的基类指针调用虚函数时是以基类中的虚函数的形参为标准的，也就是只要调用的形式符合基类中定义的虚函数的标准就行了。
14. 析构函数可以是虚函数，但构造函数不能.
15. 纯虚函数声明形式为 **virtual 类型 函数名(参数列表)=0;**注意后面的等于0；
16. 如果类至少有一个纯虚函数，**则这个类就是抽象的**。
17. 如果基类只是声明虚函数而不定义虚函数则此虚函数是纯虚函数，任何派生类都必须实现纯虚函数的自己的版本.如果不实现纯虚函数那么该类也是抽象类。
18. 抽象类不能有对象，抽象类只能用作其它类的基类，因为抽象类中的一个或多个函数没有定义，所以不能用抽象类声明对象，
19. 仍然可以用抽象类声明一个指针，这个指针指向派生类对象.
20. 如果派生类中未定义虚函数，则会使用基类中定义的函数.
21. **虚函数虚拟特性是以层次结构的方式来继承的，**例如C从B派生而且C中重定义了B中的虚函数，而D又从C派生且未重定义B中的虚函数，这时声明一个基类指针P，当P指向类D，并调用D中的虚函数时，由于D中未重定义虚函数他会调用基类中的虚函数版本，这时他会调用类C中的虚函数而不是类B中的虚函数，因为类C比类B更接近于类D.

例：虚函数的应用

```
class A
{public:int a;     virtual void  f(){cout<<"jixu"<<"\n";}      virtual void h(int i=1,int j=2){cout<<"jixuH"<<"\n";}
 ~A(){cout<<"xiA"<<"\n";}//virtual int b;  //错误，不能把成员变量声明为虚有的。  };
class B:public A
{public: int b;
void f(int i){cout<<"paif()"<<"\n";}   //重载虚函数f。
void f(){cout<<"paixu"<<"\n";}       //在派生类中重定义虚函数f
void h(){int b;b=5; cout<<"paifu"<<b<<"\n";}   //重载虚函数h的版本。注意这里不是对基类虚函数的重定义。
void h(int i,int j=3){int b; b=j;cout<<"paixuH"<<b<<"\n";}//当基类中的虚函数有默认形参时，派生类中重定义基类中的虚函数的版本必须有相同数量的形参，形参可以有默认值，也可以没有。如果形参数量不一样多则是对虚函数的重载。
~B(){cout<<"xiB"<<"\n";}};
int main()
{B m;    A *p=&m;
//p->b=3; //错误，指向派生类的基类指针不能调用派生类中的成员，只能调用基类中的成员，除非该成员是虚函数。
p->f();   //调用派生类中的函数f，输出paixu
//p->f(4); //错误，注意这里不是在调用派生类中带一个形参的f函数，因为带一个参数的f函数不是虚函数，用指向派生类的基类指针时不会调用派生类中的函数，除非这个函数是虚函数。这里基类中没有定义这种带一个形参的f函数，所以这时会出现错误。
p->A::f(); //调用基类的虚函数f，输出jixu，可以用作用域运算符使用指向派生类的基类指针调用基类的虚函数
p->h();//调用派生类中的虚函数版本h输出paixuH2，用指向派生类的基类指针调用虚函数时派生类中的虚函数的默认值在这里不起作用。虽然派生类中的虚函数需要一个参数，但这里不给参数仍是调用的派生类的带两个参数的虚函数h，而不是调用派生类中的不带参数的h函数
//使用派生类对象调用成员
m.h();  //调用派生类中不带参数的h函数，如果要用对象调用派生类中带两个形参的h函数，在本例中必须使用一个实参值。
m.h(1); //调用派生类中带两个形参的h函数，输出paixuH3，用对象调用派生类中的虚函数时函数的默认值不受基类虚函数默认值的影响
```

```
m.A::h();} // 调用基类中的虚函数h.
```

13.1.8 虚析构函数

1. **为什么需要虚析构函数：**当使用 new 运算符动态分配内存时，基类的析构函数就应该定义为虚析构函数，不然就会出问题。比如类 B 由类 A 继承而来，则有语句 A *p= new A；delete p; 这时没有问题，调用类 A 的析构函数释放类 A 的资源。但如果再把类B 的内存动态分配给指针 p 时如 p= new B; delete p；如果基类的析构函数不是虚析构函数的话就会只调用基类 A 中的析构函数释放资源，而不会调用派生类 B 的析构函数，这时派生类 B 的资源没有被释放。
2. 解决这个问题的方法是把基类的析构函数声明为虚析构函数，即在析构函数前加 virtual 关见字，定义为虚析构函数时当用 delete 释放派生类的资源时就会根据基类的析构函数自动调用派生类中的析构函数释放派生类的资源。
3. 只要基类中的析构函数是虚析构函数，则该基类的派生类中的析构函数自动为虚析构函数，虽然派生类中的析构函数前没有 virtual 关见字，析构函数名字也不一样，但派生类中的析构函数被自动继承为虚析构函数
4. 如果要使用 new 运算符分配内存，最好将析构函数定义为虚析构函数。

例：使用 new 分配内存，但不定义为虚析构函数的情形

```
class A  {public:    int a;      ~A(){cout<<"xiA"<<"\n";   }};
class B:public A    {public: int b;      ~B(){cout<<"xiB"<<"\n";   }};
class C:public B{public: int c; ~C(){cout<<"xiC"<<"\n";}};
int main()
{A *p=new A;   delete p; //输出xiA;
//B m; p=&m;   //此语句没有错，但是将使指针p指向一个静态分配的内存地址，这时不能用delete语句释放指针p的资源。
//delete p;  //错误，指针p现在指向的内容不是动态分配的内存，而是静态内存，delete只能释放动态分配的内存。
p=new B;  //动态分配派生类B的内存，并把地址赋给指针p。
delete p; //输出 xiA;在这里没有调用派生类的析构函数释放动态分配的派生类的内存资源。
B *p1= new B;  delete p1; //输出xiB,xiA.
p1=new C;      delete p1; //输出xiB,xiA注意，这里没有释放掉子类C的资源。
}
```

例：使用 new 分配内存，且基类定义为虚析构函数的情形

```
class A {public:   int a;          virtual ~A(){cout<<"xiA"<<"\n";}   };//基类定义为虚析构函数
class B:public A {public: int b;   ~B(){cout<<"xiB"<<"\n";}  };  //派生类B自动继承为虚析构函数
class C:public B {public: int c;   ~C(){cout<<"xiC"<<"\n";}};    //派生类C也自动继承为虚析构函数

int main()
{A *p=new A;  delete p;  //输出xiA
p=new B;
delete p; //输出xiB,xiA因为基类定义的是虚析构函数，所以在这里调用派生类的析构函数释放动态分配的派生类的内存资源，并调用基类的析构函数释放基类的资源
B *p1= new B;  delete p1; //输出xiB,xiA
p1=new C;delete p1; } //输出 xiC,xiB,xiA 这里因为类 B 的析构函数被自动继承为虚析构函数，所以这里释放了子类 C 的释源。
```

作者：黄邦勇帅