C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

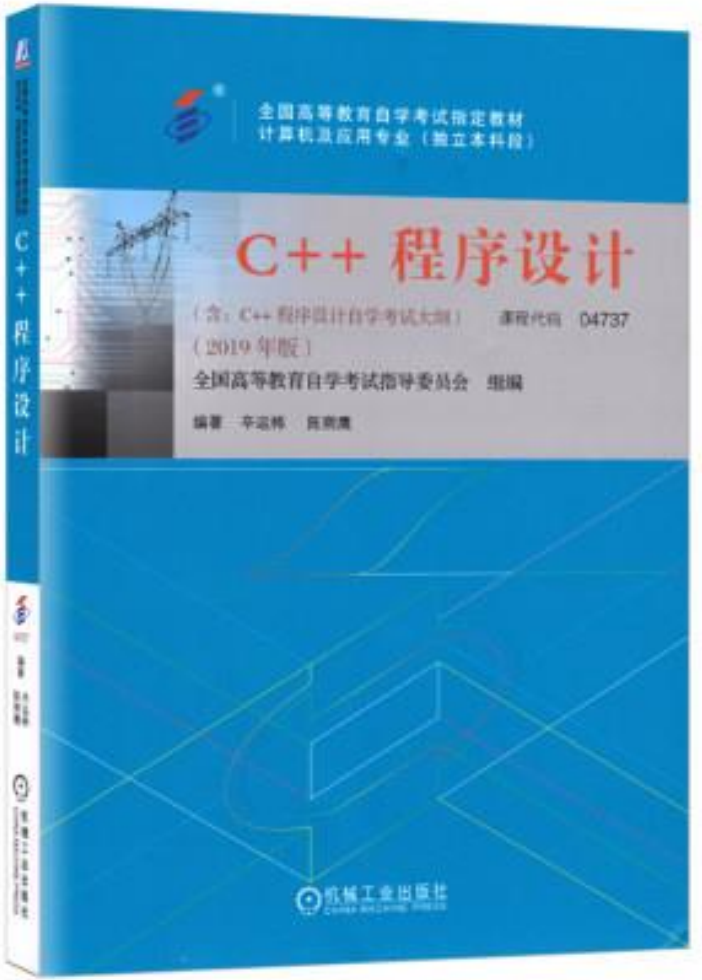
教材介绍

C++程序设计

（2019年版）

编著：辛运帷 陈朔鹰

机械工业出版社



考试题型

单选题　　1分×20题 = 20分

填空题　　1分×15题 = 15分

程序填空题　4分×5题 = 20分

程序分析题　6分×5题 = 30分

程序设计题　　2题 = 15分

（第一题5分，第二题10分）

第五章　类的继承与派生

本章主要内容

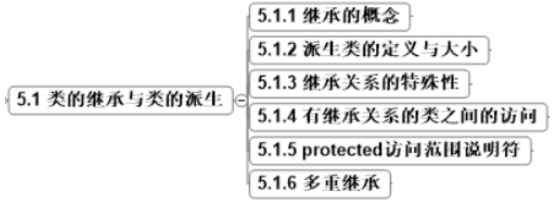
- 类的继承与类的派生
- 访问控制
- 派生类的构造函数和析构函数
- 类之间的关系
- 多层次的派生
- 基类与派生类指针的互相转换

5.1 类的继承与类的派生

继承和派生是人们认识客观世界的过程。在程序设计方法中，人们追求代码复用（这是提高软件开发效率的重要手段），将继承和派生用于程序设计方法中，从而有了面向对象程序设计的重要特点。C++对代码复用有很强的支持，“**继承**”就是支持代码复用的机制之一。

通过已有的类建立新类的过程，叫作类的派生。原来的类称为基类，也称为父类或一般类；新类称为派生类，也称为子类或特殊类。派生类派生自基类，或继承于基类，也可以说基类派生了派生类。派生机制是C++语言及面向对象程序设计方法的重要特征之一。派生类可以再作为基类派生新的派生类，由此基类和派生类的集合称作类继承层次结构。

5.1.1 继承的概念



使用基类派生新类时，**除构造函数和析构函数外，基类的所有成员自动成为派生类的成员，包括基类的成员变量和成员函数。**同时，派生类可以增加基类中没有的成员，这同样是指成员变量和成员函数。可以重新定义或修改基类中已有的成员，包括**可以改变**基类中成员的访问权限。当然派生类需要定义自己的构造函数和析构函数。

使用基类成员是一个重用的过程，在基类之上进行调整，不论是添加新成员还是改造已有的，都是扩充的过程。

5.1.1 继承的概念

若派生类中定义了一个与基类中同名的成员，则会出现基类与派生类有同名成员的情况，这是允许的。同名的成员既可以是成员变量，也可以是成员函数。这种情况下，若在派生类的成员函数中访问这个同名成员，或通过派生类对象访问这个同名成员时，除非特别指明，访问的就是派生类中的成员，这种情况叫“**覆盖**”，即派生类的成员覆盖基类的同名成员。覆盖也称为重定义或是重写。对于成员函数来说，派生类既继承了基类的同名成员函数，又在派生类中重写了这个成员函数。这称为函数重定义，也称为同名隐藏。“隐藏”的意思是指，使用派生类对象调用这个名字的成员函数时，调用的是派生类中定义的成员函数，即隐藏了基类中的成员函数。

真题演练

在C++中，可以被派生类继承的函数是（ ）。

A:成员函数

B:构造函数

C:析构函数

D:友元函数

真题演练

在C++中，可以被派生类继承的函数是（ ）。

A:成员函数

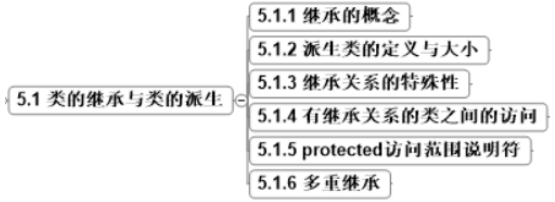
B:构造函数

C:析构函数

D:友元函数

答案：A

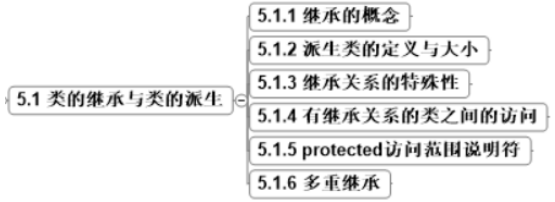
5.1.2 派生类的定义与大小



例5-1基类与派生类的定义

```
class BaseClass                                    //基类
{
        int v1,v2;
};
class DerivedClass : public BaseClass              //派生类
{
         int v3;
}
```

5.1.2 派生类的定义与大小



空类也可以作为基类，也就是说，空类可以派生子类。例如，下列语句定义了空基类的派生类：

```
class emptyClass{ };                                    //空基类
class subemptyClass : public emptyClass{ };             //派生类
```

派生类可以改变基类中成员的访问权限

5.1.2 派生类的定义与大小

2. 类的大小

派生类对象中包含基类成员变量，而且基类成员变量的存储位置位于派生类对象新增的成员变量之前。派生类对象占用的存储空间大小，等于基类成员变量占用的存储空间大小加上派生类对象自身成员变量占用的存储空间大小。

对象占用的存储空间包含对象中各成员变量占用的存储空间。出于计算机内部处理效率的考虑，为变量分配内存时，会根据其对应的数据类型，在存储空间内对变量的起始地址进行边界对齐。可以使用sizeof( )函数计算对象占用的字节数。

对象的大小与普通成员变量有关，与成员函数和类中的静态成员变量无关，即普通成员函数、静态成员函数、静态成员变量、静态常量成员变量等均对类对象的大小没有影响。

【程序5-2】基类与子类占用空间及字节对齐

```
#include<iostream>
using namespace std;
class BaseClass       //基类
{     int v1,v2;
      char v4;
public:
      int templ(){}
};
class DerivedClass : public BaseClass      //派生类
{     int v3;
      int*p;
public:
      int temp(){}
};
int main()
{     cout<<"Base="<<sizeof(BaseClass)<<endl;           //输出Base=12
      cout<<"Derived="<<sizeof(DerivedClass)<<endl;  //输出Derived=20
      return 0;
}
```

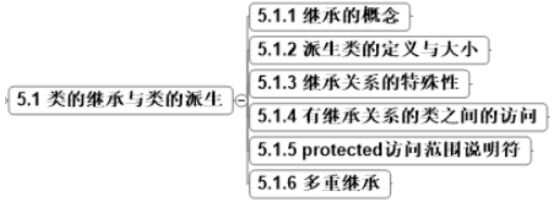
5.1.3 继承关系的特殊性

如果基类有友元类或友元函数，则其派生类**不会**因继承关系而也有此友元类或友元函数。如果基类是某类的友元，则这种友元关系是被继承的。即被派生类继承过来的成员函数，如果原来是某类的友元函数，那么它作为派生类的成员函数仍然是某类的友元函数。总之，基类的友元不一定是派生类的友元；基类的成员函数是某类的友元函数，则其作为派生类继承的成员函数仍是某类的友元函数。

```
#include<iostream>
using namespace std;
class another;     //前向引用声明
class Base         // 基类
{
private:
      float x;
public:
      void print(const another &K);
};
class Derived:public Base     //派生类
{private:
      float y;
};
class another                 //其他类
{private:
      int aaa;
public:
      another()
      {
            aaa=100;
      }
      friend void Base::print(const another &K);//基类的成员函数声明为本类的友元
};
void Base::print(const another &K)
{
      cout<<"Base:"<<K.aaa<<endl;   //可以访问私有成员变量
}
int main( )
{
      Base a;
      Derived d;
      another ano;    //aaa 初始化为100
      a.print(ano);    //输出为：Base：100
      d.print(ano);    //输出为：Base：100
      return 0;
}
```

```
Base:100
Base:100
```

5.1.3 继承关系的特殊性



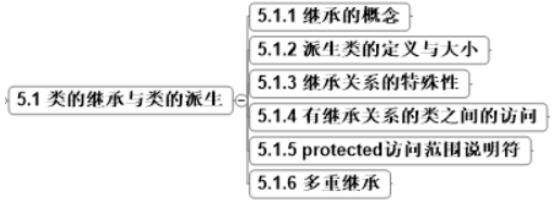
如果基类中的成员是静态的，则在其派生类中，被继承的成员也是静态的，即其静态属性随静态成员被继承。

如果基类的静态成员是公有的或是保护的，则它们被其派生类继承为派生类的静态成员。访问这些成员时，通常用“<类名>::<成员名>”的方式引用或调用。无论有多少个对象被创建，这些成员都只有一个拷贝，它为基类和派生类的所有对象所共享。

```
#include<iostream>
using namespace std;
class Base          // 基类
{
private:
      float x;
public:
      static int staV;
      Base( )
      { staV++;}
};
int Base::staV=0;
class Derived: public Base  //派生类
{
private:
      float y;
public:
     Derived( )
     {  staV++;}
};
```

```
int main()
{
      Base a;
      cout<<a.staV<<endl;  //输出1
      Derived d;
      cout<<d.staV<<endl;  //输出3
      return 0;
}
```

5.1.3 继承关系的特殊性



例5-2单项选择题

C++的继承性允许派生类继承基类的【 】

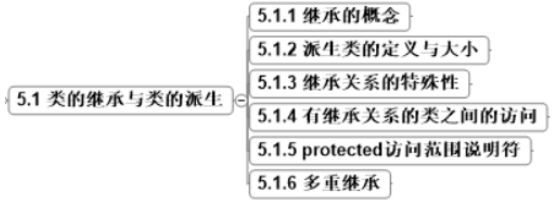
A.部分成员，并允许增加新的成员或重定义基类的成员

B.部分成员，但不允许增加新的成员或重定义基类的成员

C.所有成员，并允许增加新的成员或重定义基类的成员

D.所有成员，但不允许增加新的成员或重定义基类的成员

5.1.3 继承关系的特殊性



例5-2单项选择题

C++的继承性允许派生类继承基类的【 】

A.部分成员，并允许增加新的成员或重定义基类的成员

B.部分成员，但不允许增加新的成员或重定义基类的成员

C.所有成员，并允许增加新的成员或重定义基类的成员

D.所有成员，但不允许增加新的成员或重定义基类的成员

答案：A。

【分析】除基类的构造函数和析构函数外，派生类可以继承基类的全部成员变量和成员函数。构造函数和析构函数属于类的成员函数部分。另外，派生类也不能继承基类的友元函数，虽然友元函数不是基类的成员函数。但基类中的成员函数是其他类的友元函数时，是可以被继承的。

真题演练

f1(int)是类A的公有成员函数，p是指向成员函数f1()的指针，正确的语句是（　）。

A:p=f1;

B:p=A::f1;

C:p=A::f1();

D:p=f1();

真题演练

f1(int)是类A的公有成员函数，p是指向成员函数f1()的指针，正确的语句是（　）。

A:p=f1;

B:p=A::f1;

C:p=A::f1();

D:p=f1();

答案：B

5.1.4 有继承关系的类之间的访问

派生类和基类中都可以定义自己的成员变量和成员函数，派生类中的成员函数可以访问基类中的公有成员变量，但不能直接访问基类中的私有成员变量。也就是说，不能在派生类的函数中，使用“基类对象名.基类私有成员函数(实参)”，或是“基类对象名.基类私有成员变量”，或是“基类名::基类私有成员”的形式访问基类中的私有成员。

在类的派生层次结构中，基类的成员和派生类新增的成员都具有类作用域。二者的作用范围不同，是相互包含的两个层，派生类在内层，基类在外层。如果派生类声明了一个和基类某个成员同名的新成员，派生的新成员就隐藏了外层同名成员，直接使用成员名只能访问到派生类的成员。如果派生类中声明了与基类成员函数同名的新函数，即使函数的参数表不同，从基类继承的同名函数的所有重载形式也都会被隐藏。如果要访问被隐藏的成员，就需要使用基类名和作用域分辨符来限定。

```
#include<iostream>
using namespace std;
class CB
{
public:
   int a;
   CB(int x)
   {
       a=x;
   }
   void showa( )
   {
       cout<<"Class CB--a="<<a<<endl;
   }
};
```

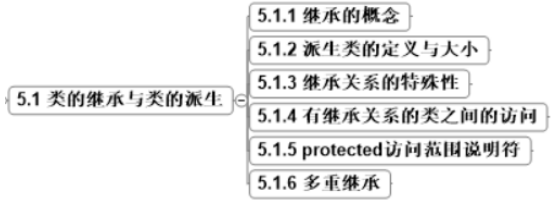
```
Class CB--a=12
Class CD--a=999
Class CB--a=48
CDobj.a=999
CDobj.CB::a=48
```

```
class CD:public CB
{
public:
       int a;                          //与基类a同名
       CD(int x,int y):CB(x)   //x用来初始化基类的成员变量a
       {   a=y;          }

        void showa()          //与基类showa同名
        {   cout<<“Class CD--a=”<<a<<endl;   }

        void print2a( )
        {   cout<<“a=”<<a<<endl;                  //访问派生类a
            cout<<“CB::a=”<<CB::a<<endl;    //访问基类a
        }
};
int main()
{
        CB CBobj(12);
        CBobj.showa();
        CD CDobj(48,999);
        CDobj.showa();          //访问派生类的showa ()
        CDobj.CB::showa();    //访问基类的showa ()
        cout<<"CDobj.a="<<CDobj.a<<endl;
        cout<<"CDobj.CB::a="<<CDobj.CB::a<<endl;
}
```

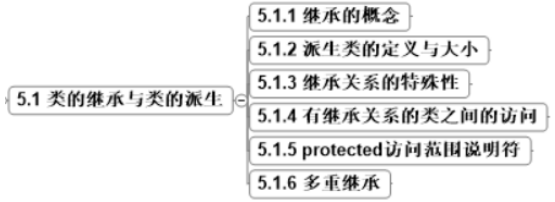
5.1.5 protected访问范围说明符



定义类时，类成员可以使用**protected**访问范围说明符进行修饰，从而成为“保护成员”。保护成员的访问范围比私有成员的访问范围大，能访问私有成员的地方都能访问保护成员。此外，基类中的保护成员可以在派生类的成员函数中被访问。

在基类中，一般都将需要隐藏的成员说明为保护成员而非私有成员。将基类中成员变量的访问方式修改为protected后，在派生类中可以直接访问。

5.1.6 多重继承

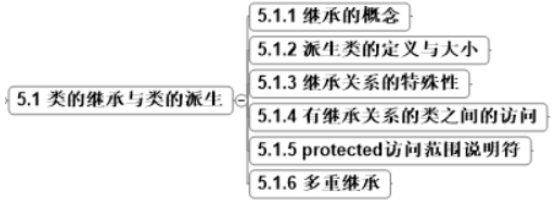


C++允许从多个类派生一个类，即一个派生类可以同时有多个基类。这称为多重继承。相应地，从一个基类派生一个派生类的情况，称为单继承或单重继承。一个类从多个基类派生的一般格式如下：

class派生类名：继承方式说明符 基类名1，继承方式说明符 基类名2,...,继承方式说明符 基类名n

```
{
    类体
};
```

5.1.6 多重继承



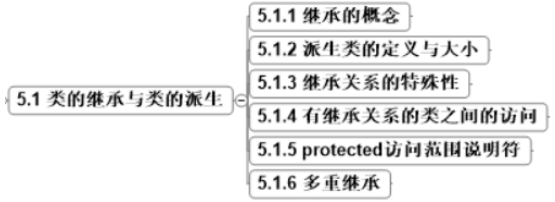
- 派生类继承了基类名1、基类名2、……、基类名n的所有成员变量和成员函数，各基类名前面的继承方式说明符用于限制派生类中的成员对该基类名中成员的访问权限，其规则与单继承情况一样。
- 多重继承情况下如果多个基类间成员重名时，按如下方式进行处理：对派生类而言，不加类名限定时默认访问的是派生类的成员；而要访问基类重名成员时，要通过类名加以限定。

```
#include<iostream>
using namespace std;
class CB1
{public:
    int a;                  //重名
    CB1 (int x)
    { a=x;  }
    void showa( )       //重名
    {   cout<<"Class CB1==>a="<<a<<endl;  }
};
class CB2
{public:
    int a;                  //重名
    CB2 (int x)
    {  a=x;  }
    void showa( )      //重名
    {
        cout<<"Class CB2==>a="<<a<<endl;
    }
};
```

```
class CD:public CB1,public CB2        //多重继承，两个基类
{
public:
      int a;          //与两个基类成员变量a重名
      CD(int x,int y,int z):CB1(x) ,CB2(y)
      {      a=z;            }
      void showa()       //与两个基类成员函数showa（）重名
      {      cout<<“Class CD==>a=”<<a<<endl;          }
      void print3a()
      {      cout<<"a="<<a<<endl;
             cout<<"CB1::a="<<CB1::a<<endl;
             cout<<"CB2::a="<<CB2::a<<endl;
      }
};
int main()
{
      CB1 CB1obj(11);
      CB1obj.showa( );
      CD CDobj(101,202,909);
      CDobj.showa( );        //调用派生类的showa（）
      CDobj.CB1::showa( );    //调用基类的showa（）
      cout<<"CDobj.a="<<CDobj.a<<endl;//访问派生类成员a
      cout<<"CDobj.CB2::a="<<CDobj.CB2::a<<endl;
}
```

```
Class CB1==>a=11
Class CD==>a=909
Class CB1==>a=101
CDobj.a=909
CDobj.CB2::a=202
```

5.1.6 多重继承



例5-3单项选择题

下列关于多重继承的描述中，错误的是【 】

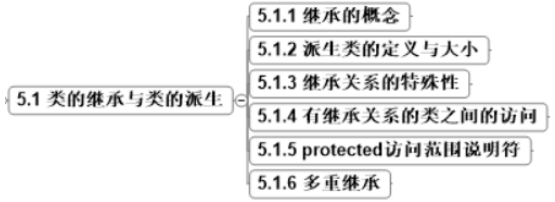
A.一个派生类对象可以拥有多个直接或间接基类的成员

B.在多重继承时不同的基类可以有同名成员

C.对于不同基类的同名成员，派生类对象访问它们时不会出现二义性

D.对于不同基类的不同名成员，派生类对象访问它们时不会出现二义性

5.1.6 多重继承



例5-3单项选择题

下列关于多重继承的描述中，错误的是【 】

A.一个派生类对象可以拥有多个直接或间接基类的成员

B.在多重继承时不同的基类可以有同名成员

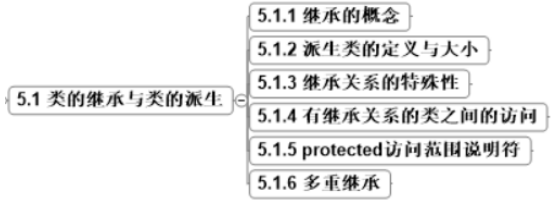
C.对于不同基类的同名成员，派生类对象访问它们时不会出现二义性

D.对于不同基类的不同名成员，派生类对象访问它们时不会出现二义性

答案：C。

【分析】多重继承是指一个派生类同时有一个以上的基类，多个基类中的所有成员除构造函数和析构函数外都被派生类继承。如果这些基类中有同名的成员，则派生类中也有同名的成员，它们来自于不同的基类。在派生类中还可以新增这样的同名成员。

5.1.6 多重继承



- 如果派生类中新增了同名成员，则派生类成员将隐藏所有基类的同名成员。使用“派生类对象名.成员名”或“派生类对象指针->成员名”的方式可以唯一标识和访问派生类新增成员。这种情况下，不会产生二义性。
- 如果派生类中没有新增同名成员，当满足访问权限时，使用“派生类对象名.成员名”或“派生类对象指针->成员名”方式时，系统无法判断到底是调用哪个基类的成员，从而产生二义性。为了避免二义性，必须通过基类名和作用域分辨符来标识成员。
- 当要访问派生类对象中的某个变量时，添加“基类::”作为前缀，指明需要访问从哪个基类继承来的，从而可以排除二义性。

课堂练习

下列关于类层次中重名成员的描述中，错误的是（　）。

A:C++允许派生类的成员与基类成员重名

B:在派生类中访问重名成员时，屏蔽基类的同名成员

C:在派生类中不能访问基类的同名成员

D:如果要在派生类中访问基类的同名成员，可以显式地使用作用域符指定

课堂练习

下列关于类层次中重名成员的描述中，错误的是（　）。

A:C++允许派生类的成员与基类成员重名

B:在派生类中访问重名成员时，屏蔽基类的同名成员

C:在派生类中不能访问基类的同名成员

D:如果要在派生类中访问基类的同名成员，可以显式地使用作用域符指定

答案：C

课堂练习

下列对派生类的描述中，错误的是（　）。

A:一个派生类可以作为另一个派生类的基类

B:派生类至少应有一个基类

C:派生类的成员除了自己定义的成员外，还包含了它的基类成员

D:基类中成员访问权限继承到派生类中都保持不变

课堂练习

下列对派生类的描述中，错误的是（　）。

A:一个派生类可以作为另一个派生类的基类

B:派生类至少应有一个基类

C:派生类的成员除了自己定义的成员外，还包含了它的基类成员

D:基类中成员访问权限继承到派生类中都保持不变

答案：D

课堂练习

C++类体系中，不能被派生类继承的有（　）。

A:静态数据成员

B:构造函数

C:常数据成员

D:静态成员函数

课堂练习

C++类体系中，不能被派生类继承的有（　）。

A:静态数据成员

B:构造函数

C:常数据成员

D:静态成员函数

答案：B

5.2 访问控制

设计继承类时，需要使用继承方式说明符指明派生类的继承方式。继承方式说明符可以是public(公有继承）、private(私有继承）或protected(保护继承）。

5.2.1 公有继承

各成员	派生类中	基类与派生类外
基类的公有成员	直接访问	直接访问
基类的保护成员	直接访问	调用公有函数访问
基类的私有成员	调用公有函数访问	调用公有函数访问
从基类继承的公有成员	直接访问	直接访问
从维类继承的保护成员	直接访问	调用公有函数访问
从基类继承的私有成员	调用公有函数访问	调用公有函数访问
派生类中定义的公有成员	直接访问	直接访问
派生类中定义的保护成员	直接访问	调用公有函数访问
派生类中定义的私有成员	直接访问	调用公有函数访问

课堂练习

以下派生方式中，能让派生类访问基类中的protected成员的是（　　）

A:public 和 private

B:public 和 protected

C:protected 和 private

D:仅protected

课堂练习

以下派生方式中，能让派生类访问基类中的protected成员的是（　　）

A:public 和 private

B:public 和 protected

C:protected 和 private

D:仅protected

答案：B

课堂练习

基类private成员，通过public派生，其在派生类中为（ ）。

A:private

B:protected

C:public

D:不可访问

课堂练习

基类private成员，通过public派生，其在派生类中为（ ）。

A:private

B:protected

C:public

D:不可访问

答案：D

课堂练习

下列对派生类的描述中，错误的是（ ）。

A:对基类成员的访问必须是无二义性的

B:派生类至少有一个基类

C:基类的公有成员在派生类中仍然是公有的

D:派生类的成员除了它自己的成员外，还包含了它的基类的成员

课堂练习

下列对派生类的描述中，错误的是（ ）。

A:对基类成员的访问必须是无二义性的

B:派生类至少有一个基类

C:基类的公有成员在派生类中仍然是公有的

D:派生类的成员除了它自己的成员外，还包含了它的基类的成员

答案：C

课堂练习

派生类的对象可以访问以下那种情况继承的基类成员（　）。

A:私有继承的私有成员

B:公有继承的私有成员

C:私有继承的保护成员

D:公有继承的公有成员

课堂练习

派生类的对象可以访问以下那种情况继承的基类成员（　）。

A:私有继承的私有成员

B:公有继承的私有成员

C:私有继承的保护成员

D:公有继承的公有成员

答案：D

课堂练习

例5-4单项选择题

当派生类公有继承基类时，基类中的所有公有成员成为派生类的【】

A.public成员　　B.private成员

C.protected成员　　D.友元

课堂练习

例5-4单项选择题

当派生类公有继承基类时，基类中的所有公有成员成为派生类的【】

A.public成员　　B.private成员

C.protected成员　　D.友元

答案：A。

【分析】公有继承时，基类中的所有公有成员成为派生类中的公有成员。

课堂练习

例5-5单项选择题

在公有派生的情况下，基类的公有或保护成员在派生类中的访问权限【】

A.受限制　　B.保持不变

C.受保护　　D.不受保护

课堂练习

例5-5单项选择题

在公有派生的情况下，基类的公有或保护成员在派生类中的访问权限【】

A.受限制　　B.保持不变

C.受保护　　D.不受保护

答案：B。

【分析】继承方式不同，基类中的成员在派生类中的访问权限可能也不同。在公有派生情况下，基类的公有及保护成员在派生类中保持不变，私有成员不能访问。

5.2.2 类型兼容规则

类型兼容规则是指在需要基类对象的任何地方，都可以使用公有派生类的对象来替代，也称为赋值兼容规则。

在公有派生的情况下，有以下3条类型兼容规则。

1)派生类的对象可以赋值给基类对象。

2)派生类对象可以用来初始化基类引用。

3)派生类对象的地址可以赋值给基类指针，即派生类的指针可以赋值给基类的指针。

上述3条规则反过来是不成立的。例如，不能把基类对象赋值给派生类对象。

在进行替代之后，派生类对象就可以作为基类的对象使用了，但只能使用从基类继承的成员。

5.2.2 类型兼容规则

如果类B为基类，类D为类B的公有派生类，则类D中包含了基类B中除构造函数、析构函数之外的所有成员。这时，根据类型兼容规则，在基类B的对象可以出现的任何地方，都可以用派生类D的对象来替代。假设有以下的声明：

```
class B{…}
class D : public B{…}
B b1, *pb1;
D d1;
```

这时，派生类对象可以隐含转换为基类对象，即用派生类对象中从基类继承来的成员变量的值，逐个为基类对象的成员变量的值进行赋值。

```
b1=d1;
```

派生类的对象也可以用来初始化基类对象的引用，即

```
B &rb=d1;
```

派生类对象的地址可以隐含转换为指向基类的指针，即派生类对象的地址赋给基类指针：

```
pb1=&d1;
```

由于类型兼容规则的引入，对于基类及其公有派生类的对象，可以使用相同的函数统一进行处理。因为当函数的形参为基类的对象（或引用、指针）时，实参可以是派生类的对象(或指针)，从而没有必要为每一个类设计单独的模块，大大提高了程序的效率。

课堂练习

下列选项中，不被允许的赋值是（　）

A:基类对象=派生类对象

B:指向基类对象的指针=派生类对象的地址

C:指向派生类类型的指针=基类对象的地址

D:基类的引用=派生类对象

课堂练习

下列选项中，不被允许的赋值是（　）

A:基类对象=派生类对象

B:指向基类对象的指针=派生类对象的地址

C:指向派生类类型的指针=基类对象的地址

D:基类的引用=派生类对象

答案：C

```
#include<iostream>
using namespace std;
class A
{       int an;
public:
        A( ){ }
        A(int n)
        {   an=n; }
};
class B:public A   //公有派生
{       int bn;
public:
        B(int n):A(2*n)
        {    bn=n; }
};
int main( )
{       A a(10);
        B b(20);
        a=b;  //派生类对象赋值给基类对象
        return 0;
}
```

```
#include<iostream>
using namespace std;
class A
{
    int an;
public:
    A( ){ }
    A(int n)
    {
        an=n;
    }
    void print( )
    {
        cout<<"A的对象:";
        cout<<"an:"<<an;
    }
    void print(int k)  //不同的输出格式
    {
        cout<<"an:"<<an;
    }
};
class B:public A   //公有派生
{
    int bn;
public:
    B(int n):A(2*n)
    {   bn=n;  }
    void print( )
    {
        cout<<"\nB的对象:";
        A::print(1);
        cout<<",bn="<<bn<<endl;
    }
};
int main()
{
    A a(10);
    B b(20);
    a.print( );
    b.print( );
    a=b;        //派生类对象赋值给基类对象
    a.print( );
    b.print( );
    return 0;
}
```

```
A的对象:an:10
B的对象:an:40,bn=20
A的对象:an:40
B的对象:an:40,bn=20
```

5.2.3 私有继承

	第一级派生类中	第二级派生类中	基类与派生类外
基类的公有成员	直接访问	不可访问	不可访问
基类的保护成员	直接访问	不可访问	不可访问
基类的私有成员	调用公有函数访问	不可访问	不可访问

5.2.4 保护继承

保护继承中，基类的公有成员和保护成员都以保护成员的身份出现在派生类中，而基类的私有成员不可以直接访问。这样，派生类的其他成员可以直接访问从基类继承来的公有和保护成员，但在类外通过派生类的对象无法直接访问它们。

5.3 派生类的构造函数和析构函数

派生类并不继承基类的构造函数，所以需要在派生类的构造函数中调用基类的构造函数，以完成对从基类继承的成员变量的初始化工作。具体来说，派生类对象在创建时，除了要调用自身的构造函数进行初始化外，还要调用基类的构造函数初始化其包含的基类成员变量。

在执行一个派生类的构造函数之前，总是先执行基类的构造函数。派生类对象消亡时，先执行派生类的析构函数，再执行基类的析构函数。

5.3.1 构造函数和析构函数

定义派生类构造函数的一般格式如下：

派生类名::派生类名(参数表)：基类名1(基类1 初始化参数表),...,基类名m(基类m初始化参数表),成员对象名1(成员对象1 初始化参数表),...,成员对象名n(成员对象n 初始化参数表)

```
{
    类构造函数函数体                                    //其他初始化操作
}
```

课堂练习

例5-9单项选择题

下列关于派生类对象的初始化叙述中，正确的是【 】

A.是由派生类的构造函数实现的

B.是由基类的构造函数实现的

C.是由基类和派生类的构造函数实现的

D.是系统自动完成的，完全不需要程序设计者干预

课堂练习

例5-9单项选择题

下列关于派生类对象的初始化叙述中，正确的是【 】

A.是由派生类的构造函数实现的

B.是由基类的构造函数实现的

C.是由基类和派生类的构造函数实现的

D.是系统自动完成的，完全不需要程序设计者干预

答案：C

【分析】创建派生类的对象时，需要调用派生类的构造函数，同时隐含或显式调用基类的构造函数。所以，是由基类和派生类的构造函数共同完成的（选项A和B都不正确）。而编写派生类的构造函数时，需要指明从基类中继承的成员变量的初始化方式，即基类对象的初始化方式，这是由程序员定义的（选项D不正确）。

课堂练习

例5-10单项选择题

在C++中，可以被派生类继承的函数是【 】

A.成员函数　　B.构造函数

C.析构函数　　D.友元函数

课堂练习

例5-10单项选择题

在C++中，可以被派生类继承的函数是【 】

A.成员函数　　B.构造函数

C.析构函数　　D.友元函数

答案：A

【分析】派生类从基类中可以继承成员变量、普通的成员函数，但不能继承构造函数和析构函数。友元函数不是类的成员函数，所以也不能继承。

```
#include<iostream>
using namespace std;
class BaseClass      //基类
{protected:
      int v1,v2;
public:
     BaseClass();
     BaseClass(int,int);
     ~BaseClass();};
BaseClass::BaseClass()
{    cout<<"BaseClass 无参构造函数"<<endl;}
BaseClass::BaseClass(int m,int n)
{    v1=m;
     v2=n;
     cout<<"BaseClass 2个参数构造函数"<<endl;}
BaseClass::~BaseClass()
{    cout<<"BaseClass析构函数"<<endl;   }
class DerivedClass:public BaseClass   //公有继承的派生类
{    int v3;
public:
     DerivedClass();
     DerivedClass(int);
     DerivedClass(int,int,int);
     ~DerivedClass();};
DerivedClass::DerivedClass()
{
      cout<<"DerivedClass无参构造函数"<<endl;
}
DerivedClass::DerivedClass(int k):v3(k)
{
      cout<<"DerivedClass 带1个参数构造函数"<<endl;
}
DerivedClass::DerivedClass(int m,int n,int k):BaseClass(m,n),v3(k)
{
      cout<<"DerivedClass 带3个参数构造函数"<<endl;
}
DerivedClass::~DerivedClass()
{
      cout<<  “DerivedClass析构函数"<<endl;
}
int main()
{
      cout<<"无参对象的创建"<<endl;
      BaseClass baseCla;  //基类对象
      DerivedClass derivedCla; //派生类对象
      return 0;
}
```

```
无参对象的创建
BaseClass 无参构造函数
BaseClass 无参构造函数
DerivedClass无参构造函数
DerivedClass析构函数
BaseClass析构函数
BaseClass析构函数
```

5.3.1 构造函数和析构函数

派生类构造函数执行的一般次序如下：

1)调用基类构造函数，调用顺序按照它们被继承时声明的顺序（从左向右）。

2)对派生类新增的成员变量初始化，调用顺序按照它们在类中声明的顺序。

3)执行派生类的构造函数体中的内容。

构造函数初始化列表中基类名、对象名之间的次序无关紧要，它们各自出现的顺序可以是任意的，无论它们的顺序怎样安排，基类构造函数的调用和各个成员变量的初始化顺序都是确定的。

```
#include<iostream>
using namespace std;
class Base
{
private:
        int Y;
public:
        Base(int y=0)
        {   Y=y;
            cout<<"Base("<<y<<")"<<endl;
        }
        ~Base()
        {
            cout<<"~Base()"<<endl;
        }
         void print()
        {    cout<<Y<<  “   ";
        }
};
class Derived:public Base
{private:
        int Z;
public:
    Derived(int y,int z):Base(y)
    {     Z=z;
          cout<<"Derived("<<y<<","<<z<<")"<<endl;
    }
    ~Derived()
    {     cout<<"~Derived()"<<endl;
    }
    void print()
    {     Base::print();
          cout<<Z<<endl;
    }
};
int main()
{       Derived d(10,20);
        d.print();
        return 0;
}
```

```
Base(10)
Derived(10,20)
10 20
~Derived()
~Base()
```

课堂练习

在创建派生类对象时，类层次中构造函数的执行顺序是（　　）。

A:由派生类的参数初始化列表的顺序决定的

B:由类定义中的定义顺序决定的

C:由系统规定的

D:任意的

课堂练习

在创建派生类对象时，类层次中构造函数的执行顺序是（　　）。

A:由派生类的参数初始化列表的顺序决定的

B:由类定义中的定义顺序决定的

C:由系统规定的

D:任意的

答案：B

调用基类构造函数，调用顺序按照它们被继承时声明的顺序（从左向右）；

对派生类新增的成员变量初始化，调用顺序按照它们在类中声明的顺序。

5.3.2 复制构造函数

对于一个类，如果程序中没有定义复制构造函数，则编译器会自动生成一个隐含的复制构造函数，这个隐含的复制构造函数会自动调用基类的复制构造函数，对派生类新增的成员对象执行复制。

如果要为派生类编写复制构造函数，一般也需要为基类相应的复制构造函数传递参数，但并不是必须的。

```
#include<iostream>
using namespace std;
class A
{public:
   A( )                    //默认构造函数
  {    i=100;
       cout<<"类A默认构造函数"<<endl;
  }
   A(const A &s)  //复制构造函数
   {   i=s.i;
       cout<<"类A复制构造函数"<<endl;
   }
   int  getValue();          //取值
   void setValue(int);     //设置值
private:
   int i;
};
int A::getValue()  {    return i;    }
void A::setValue(int k)   {  i=k; }
class B:public A          //公有派生类
{private:
   float f;
public:
   B( ){  f=20.1;
         cout<<"类B默认构造函数"<<endl;}
   B(const B &v):A(v),f(v.f)
   { cout<<"类B复制构造函数"<<endl;}
   float getValue();
   int   getValue1()   {    return A::getValue();      }
};
float B::getValue() //重写基类函数，改变了返回值类型
{   return f;  }
int main(   )
{    A a;       //调用类A默认构造函数
     B b;       //调用类A默认构造函数、类B默认构造函数
     B bb(b); //调用类A复制构造函数、类B复制构造函数
     return 0;
}
```

```
类A默认构造函数
类A默认构造函数
类B默认构造函数
类A复制构造函数
类B复制构造函数
```

```
#include<iostream>
using namespace std;
class CBase
{
public:
        CBase( ){ }
        CBase(CBase & c)
        {
            cout<<"CBase::复制构造函数"<<endl;
        }
        CBase & operator=(const CBase &b)
        {
            cout<<"CBase::operator="<<endl;
            return *this;
        }
};
class CDerived:public CBase
{
 public:
        CDerived( )
        {
            cout<<"CDerived::复制构造函数"<<endl;
        }
 };
 int main()
 {
        CDerived d1,d2;
        CDerived d3(d1);
        //d3初始化过程中会调用类CBase的复制构造函数
        d2=d1; //会调用类CBase重载的"="运算符
        return 0;
 }
```

```
CDerived::复制构造函数
CDerived::复制构造函数
CBase::复制构造函数
CBase::operator=
```

5.3.3 多重继承的构造函数与析构函数

当创建有多个基类的派生类的对象时，按照类定义中给出的基类的顺序，依次调用它们的构造函数，再调用派生类的构造函数。对象消亡时，按照构造函数调用的次序的逆，调用析构函数。

5.3.3 多重继承的构造函数与析构函数

在派生类构造函数执行之前，要先执行两个基类的构造函数，执行次序依定义派生类DerivedClass时所列基类的次序而定。

程序5-19中，定义派生类时最前面的语句是：

class DerivedClass : public BaseClass1, public BaseClass2

所以，先执行基类BaseClass1的构造函数，再执行基类BaSeClass2的构造函数，然后执行派生类DerivedClass的构造函数。

课堂练习

设有基类CB和CC，共同派生子类CD。类定义示意如下： class CD : public CB, public CC{…}; ，则创建类CD的对象时，构造函数的调用次序依次是（ ）。

A:CD、CB、CC

B:CC、CB、CD

C:CB、CC、CD

D:CD、CC、CB

课堂练习

设有基类CB和CC，共同派生子类CD。类定义示意如下： class CD : public CB, public CC{…}; ，则创建类CD的对象时，构造函数的调用次序依次是（ ）。

A:CD、CB、CC

B:CC、CB、CD

C:CB、CC、CD

D:CD、CC、CB

答案：C

5.4 类之间的关系

- 使用已有类编写新的类有两种方式：继承和组合。这也形成类和类之间的两种基本关系：继承关系和组合关系（组合关系也就是第三章第六节提到的包含关系）。
- 继承关系也称为“is a”关系或“是”关系。
- 组合关系也称为“has a”关系或“有”关系，表现为封闭类，即一个类以另一个类的对象作为成员变量。

课堂练习

例5-14单项选择题

下列关于类的描述中，正确的是【 】

A.基类具有派生类的特征

B.一个类只能从一个类继承

C.“is a”关系具有传递性

D.“has a”关系表示类的继承机制

课堂练习

例5-14单项选择题

下列关于类的描述中，正确的是【 】

A.基类具有派生类的特征

B.一个类只能从一个类继承

C.“is a”关系具有传递性

D.“has a”关系表示类的继承机制

答案：C

【分析】在类的继承机制中，派生类继承了基类的成员，也可以添加自己的成员，所以派生类中具有基类的特征，但基类中可能并不具有派生类中的特征。基类具有一般性，而派生类具有特性。选项A是错误的。

C++中有多重继承机制，一个派生类可以有多个基类。所以选项B也不正确。

继承可以是多级的，类A可以是类B的基类，类B可以是类C的基类。可以说类C的对象是类B中的一员，也是类A中的一员。即“is a”关系具有传递性。

而“has a”关系表示的是类的包含关系，并不表示继承关系。所以选项D也不正确。

课堂练习

在下列派生中，子类和父类间是is a关系的是（　）。

A:private

B:protected

C:public

D:publish

课堂练习

在下列派生中，子类和父类间是is a关系的是（　）。

A:private

B:protected

C:public

D:publish

答案：C

5.4.2 封闭类的派生

如果一个类的成员变量是另一个类的对象，则为封闭类。定义封闭类构造函数的一般形式如下：

```
类名::类名(形参表)：内嵌对象1(形参表)，内嵌对象2(形参表),...
{
    类体
}
```

其中，“内嵌对象1(形参表)，内嵌对象2(形参表),...”是初始化列表，其作用是对内嵌对象进行初始化。

课堂练习

已知基类是封闭类，在具有继承关系的类层次体系中，析构函数执行的顺序是（　）。

A:对象成员析构函数——基类析构函数——派生类本身的析构函数

B:派生类本身的析构函数——对象成员析构函数——基类析构函数

C:派生类本身的析构函数——基类析构函数——对象成员析构函数

D:基类析构函数——对象成员析构函数——派生类本身的析构函数

课堂练习

已知基类是封闭类，在具有继承关系的类层次体系中，析构函数执行的顺序是（　）。

A:对象成员析构函数——基类析构函数——派生类本身的析构函数

B:派生类本身的析构函数——对象成员析构函数——基类析构函数

C:派生类本身的析构函数——基类析构函数——对象成员析构函数

D:基类析构函数——对象成员析构函数——派生类本身的析构函数

答案：C

课堂练习

已知基类是封闭类，在创建派生类对象时，构造函数的执行顺序是（　）。

A:对象成员构造函数——基类构造函数——派生类本身的构造函数

B:派生类本身的构造函数——基类构造函数——对象成员构造函数

C:基类构造函数——派生类本身的构造函数——对象成员构造函数

D:基类构造函数——对象成员构造函数——派生类本身的构造函数

课堂练习

已知基类是封闭类，在创建派生类对象时，构造函数的执行顺序是（　）。

A:对象成员构造函数——基类构造函数——派生类本身的构造函数

B:派生类本身的构造函数——基类构造函数——对象成员构造函数

C:基类构造函数——派生类本身的构造函数——对象成员构造函数

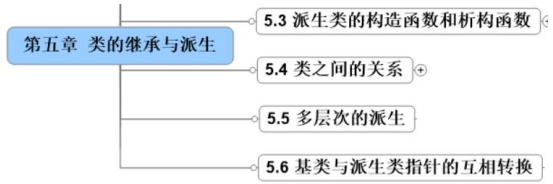
D:基类构造函数——对象成员构造函数——派生类本身的构造函数

答案：A

5.4.3 互包含关系的类

在处理相对复杂的问题而需要考虑类的组合时，很可能遇到两个类相互引用的情况，这种情况称为循环依赖。举例如下：

```
class A                                    //类 A 的定义
{
public:
        void f(B b);                       //以类 B 对象为形参的成员函数
};
class B                                    //类 B 的定义
{
public:
        void g(A a);                       //以类 A 对象为形参的成员函数
}
```



5.5 多层次的派生

- 在C++中，派生可以是多层次的。例如，类CStudent派生类CGraduatedStudent，而后者又可以派生CDoctorStudent等。总之，类A派生类B，类B可以再派生类C，类C又能够派生类D，以此类推。在这种情况下，称类A是类B的直接基类，类B是类C的直接基类，类A是类C的间接基类。当然，类A也是类D的间接基类。在定义派生类时，只需写直接基类，不需写间接基类。派生类沿着类的层次自动向上继承它所有的直接和间接基类的成员。在C++中，类之间的继承关系具有**传递性**。
- 派生类的成员包括派生类自己定义的成员、直接基类中定义的成员及所有间接基类中定义的全部成员。
- 当生成派生类的对象时，会从最顶层的基类开始逐层往下执行所有基类的构造函数，最后执行派生类自身的构造函数；当派生类对象消亡时，会先执行自身的析构函数，然后自底向上依次执行各个基类的析构函数。

课堂练习

例5-16单项选择题

下面关于基类和派生类的描述中，正确的是【 】

A.一个类可以被多次说明为某个派生类的直接基类，可以不止一次地成为间接基类

B.一个类不能被多次说明为某个派生类的直接基类，可以不止一次地成为间接基类

C.一个类不能被多次说明为某个派生类的直接基类，且只能成为一次间接基类

D.一个类可以被多次说明为某个派生类的直接基类，但只能成为一次间接基类

课堂练习

例5-16单项选择题

下面关于基类和派生类的描述中，正确的是【 】

A.一个类可以被多次说明为某个派生类的直接基类，可以不止一次地成为间接基类

B.一个类不能被多次说明为某个派生类的直接基类，可以不止一次地成为间接基类

C.一个类不能被多次说明为某个派生类的直接基类，且只能成为一次间接基类

D.一个类可以被多次说明为某个派生类的直接基类，但只能成为一次间接基类

答案：B

【分析】一个类可以是另一个或多个类的基类，但说明为某个派生类的基类时，只能说明一次，不能重复说明（选项A和D不正确）。一个类A可以是多个类的直接基类，这些类又可以各自派生自己的派生类，从而类A就是这些派生类的间接基类。特别地，一个类可以多次成为某个派生类的间接基类（选项C是错误的）。

课堂练习

在C++中类之间的继承关系具有（　）。

A:自反性

B:对称性

C:传递性

D:反对称性

课堂练习

在C++中类之间的继承关系具有（ ）。

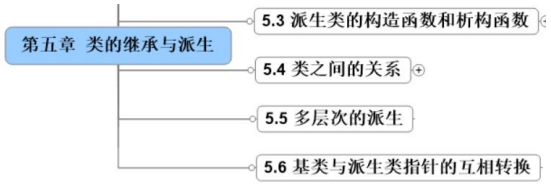
A:自反性

B:对称性

C:传递性

D:反对称性

答案：C



5.6 基类与派生类指针的互相转换

在公有派生的情况下，因为派生类对象也是基类对象，所以派生类对象可以赋给基类对象。对于指针类型，可以使用基类指针指向派生类对象，也可以将派生类的指针直接赋值给基类指针。但即使基类指针指向的是一个派生类的对象，也不能通过基类指针访问基类中没有而仅在派生类中定义的成员函数。

课堂练习

例5-17单项选择题

基类指针与派生类指针可以分别指向基类对象或派生类对象，从而形成4种组合情形。在这4种情形中，需要进行强制类型转换的是【 】

A.基类指针指向基类对象

B.基类指针指向派生类对象

C.派生类指针指向基类对象

D.派生类指针指向派生类对象

课堂练习

例5-17单项选择题

基类指针与派生类指针可以分别指向基类对象或派生类对象，从而形成4种组合情形。在这4种情形中，需要进行强制类型转换的是【 】

A.基类指针指向基类对象

B.基类指针指向派生类对象

C.派生类指针指向基类对象

D.派生类指针指向派生类对象

答案：C

【分析】基类指针指向基类对象（选项A)，或是派生类指针指向派生类对象（选项D)，都是正常的情况。所以选项A和D都不需要进行类型转换。

通过基类指针指向派生类对象符合赋值兼容规则，故选项B也不需要进行类型转换。

而使用派生类指针指向基类对象时，必须进行强制类型转换。

课堂练习

当基类指针指向派生类对象时，以下说法正确的是（　）。

A:会发生语法错误

B:只能调用基类自己定义的成员函数

C:可以调用派生类的全部成员函数

D:以上说法全部错误

课堂练习

当基类指针指向派生类对象时，以下说法正确的是（　）。

A:会发生语法错误

B:只能调用基类自己定义的成员函数

C:可以调用派生类的全部成员函数

D:以上说法全部错误

答案：B

课堂练习

当基类指针指向派生类对象时，利用基类指针调用派生类中与基类同名但被派生类重写后的成员函数时，调用的是（　）。

A:基类的成员函数

B:派生类的成员函数

C:不确定

D:随机调用

课堂练习

当基类指针指向派生类对象时，利用基类指针调用派生类中与基类同名但被派生类重写后的成员函数时，调用的是（　）。

A:基类的成员函数

B:派生类的成员函数

C:不确定

D:随机调用

答案：A

当使用指针调用类中的函数时，需要根据指针的类型来决定可调用的函数。编译器看到哪个类的指针，就会认为要通过它访问哪个类的成员。

（1）如果一个指针是基类类型的，则不管它指向的是基类对象还是派生类对象，都仅能调用基类中声明的函数，而不能调用基类中没有声明而仅在派生类中声明的函数。

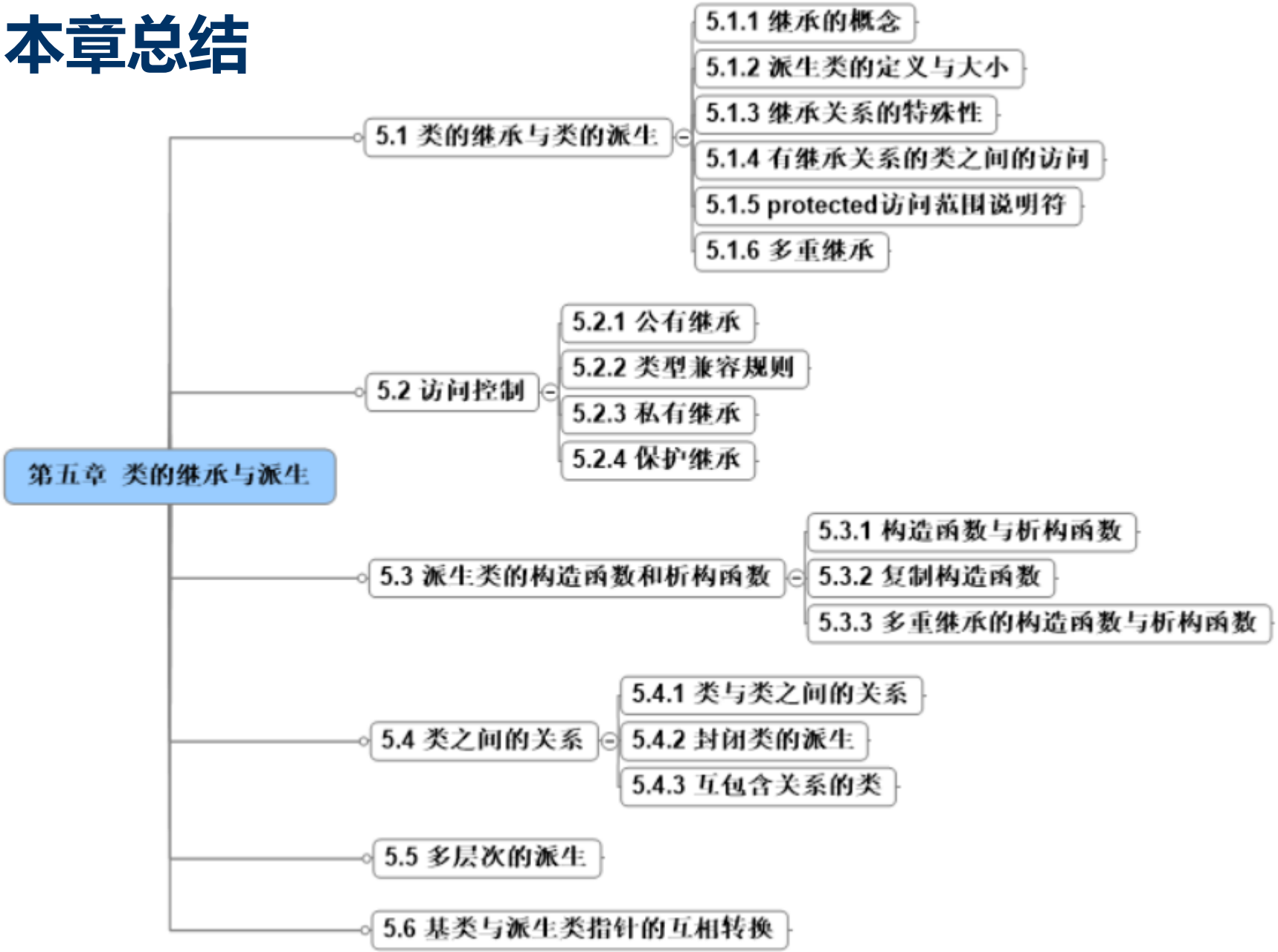
（2）如果一个指针是派生类类型的，则调用的是派生类中的函数，如果派生类中没有声明这个函数，则调用从基类继承的同名函数。

5.6 基类与派生类指针的互相转换

```
#include<iostream>
using namespace std;
class CBase
{
protected:
    int n;
public:
    CBase(int i):n(i){}
    void print() { cout<<"CBase:n="<<n<<endl; }
};
class CDerived:public CBase
{public:
    int v;
    CDerived(int i):CBase(i),v(2*i){}
    void Func( ){};
    void print( )
    {
        cout<<"CDerived:n="<<n<<endl;
        cout<<"CDerived:v="<<v<<endl;
    }
};
int main( )
{
    CDerived objDerived(3);
    CBase objBase(5);
    CBase*pBase=&objDerived; //使用基类指针指向派生类对象
    CDerived *pDerived;
    pDerived= &objDerived;
    cout<<"使用派生类指针调用函数"<<endl;
    pDerived->print( );          //调用的是派生类中的函数
    pBase=pDerived;              //基类指针=派生类指针，正确
    cout<<"使用基类指针调用函数"<<endl;
    pBase->print( );             //调用是基类中的函数
    //pBase->Func( ); //错误，通过基类指针不能调用派生类函数
    //pDerived=pBase; //错误 派生类指针=基类指针
     pDerived=(CDerived*)pBase; //强制类型转换，派生类指针=基类指针
     cout<<"使用派生类指针调用函数"<<endl;
     pDerived->print();  //调用的是派生类中的函数
     return 0;
}
```

```
使用派生类指针调用函数
CDerived:n=3
CDerived:v=6
使用基类指针调用函数
CBase:n=3
使用派生类指针调用函数
CDerived:n=3
CDerived:v=6
```

本章总结


第五章 类的继承与派生
5.1 类的继承与类的派生
5.1.1 继承的概念
5.1.2 派生类的定义与大小
5.1.3 继承关系的特殊性
5.1.4 有继承关系的类之间的访问
5.1.5 protected访问范围说明符
5.1.6 多重继承
5.2 访问控制
5.2.1 公有继承
5.2.2 类型兼容规则
5.2.3 私有继承
5.2.4 保护继承
5.3 派生类的构造函数和析构函数
5.3.1 构造函数与析构函数
5.3.2 复制构造函数
5.3.3 多重继承的构造函数与析构函数
5.4 类之间的关系
5.4.1 类与类之间的关系
5.4.2 封闭类的派生
5.4.3 互包含关系的类
5.5 多层次的派生
5.6 基类与派生类指针的互相转换

祝大家顺利通过考试！