Chapter 11

Object-Oriented Programming: Inheritance

OBJECTIVES

- To create classes by **inheriting** from existing classes.
- How inheritance promotes software reuse.
- The notions of **base classes** and **derived classes** and the relationships between them.
- The **protected** member access specifier.
- The use of constructors and destructors in inheritance hierarchies.
- The differences between **public**, **protected** and **private inheritance**.

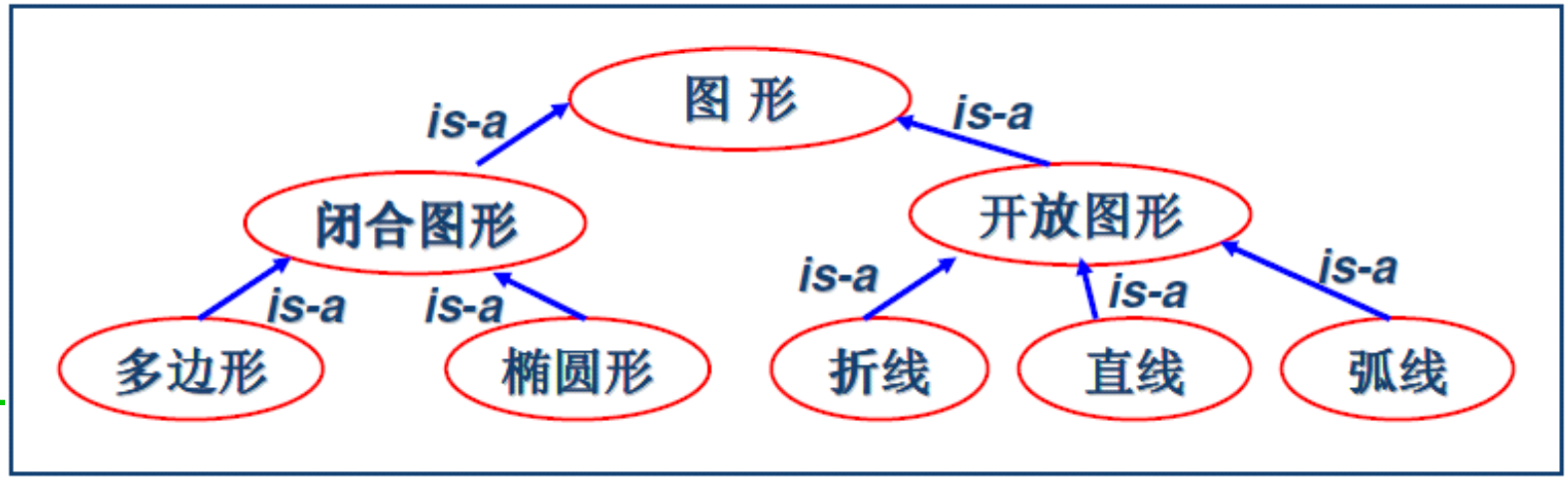
Topics

- **11.1 Introduction**
- 11.2 Base Classes and Derived Classes
- 11.3 protected Members
- 11.4 Relationship between Base Classes and Derived Classes
- 11.5 Constructors and Destructors in Derived Classes
- 11.6 public, protected and private Inheritance

11.1 Introduction
--类之间的关系

- *has-a* relation
  Composition 组合
- *is-a* relation
  Inheritance 继承

- base class / derived class
  (基类 / 派生类)
- direct / indirect base class
  (直接 / 间接基类)
- single / multiple inheritance
- 3 kinds of inheritance

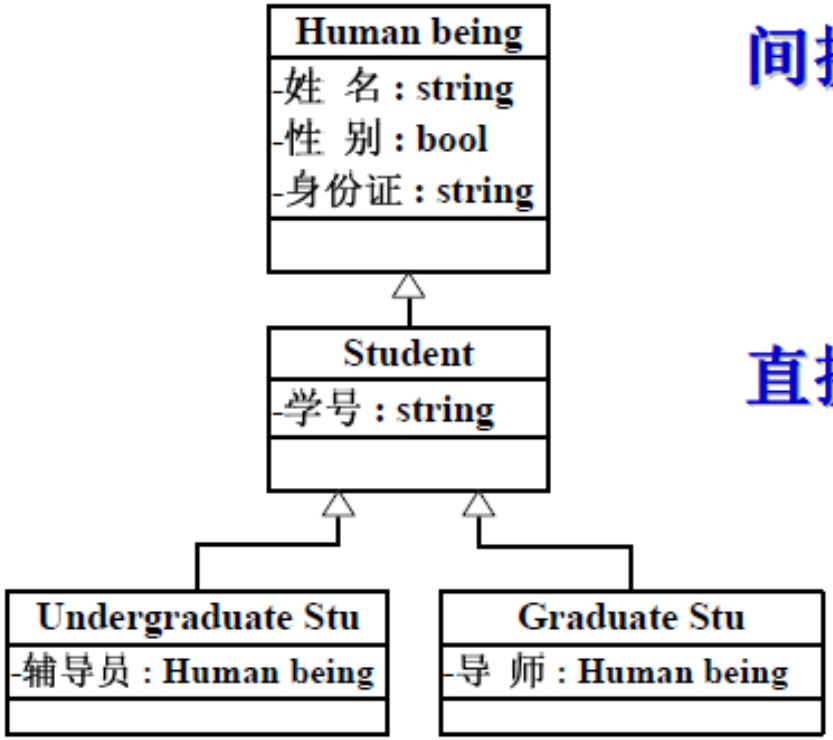


Topics

- 11.1 Introduction
- **11.2 Base Classes and Derived Classes**
- 11.3 protected Members
- 11.4 Relationship between Base Classes and Derived Classes
- 11.5 Constructors and Destructors in Derived Classes
- 11.6 public, protected and private Inheritance

11.2 Base Classes and Derived Classes

- **Base class**: 基类, 被继承的类
- **Derived class**: 派生类, 继承后得到类



间接基类

直接基类

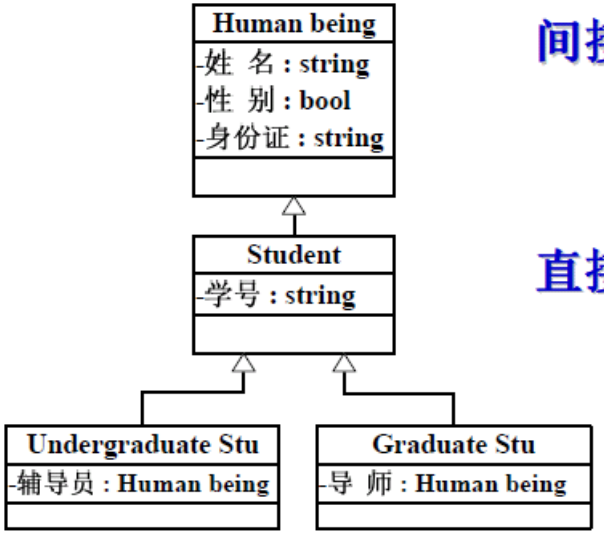
派生类

继承机制作用:
- 软件复用
- 支持软件的增量开发

Inheritance hierarchy 树形的层次关系图

11.2 Base Derived

- 类继承的语法:



Inheritance hierarchy 树形的层次关系图

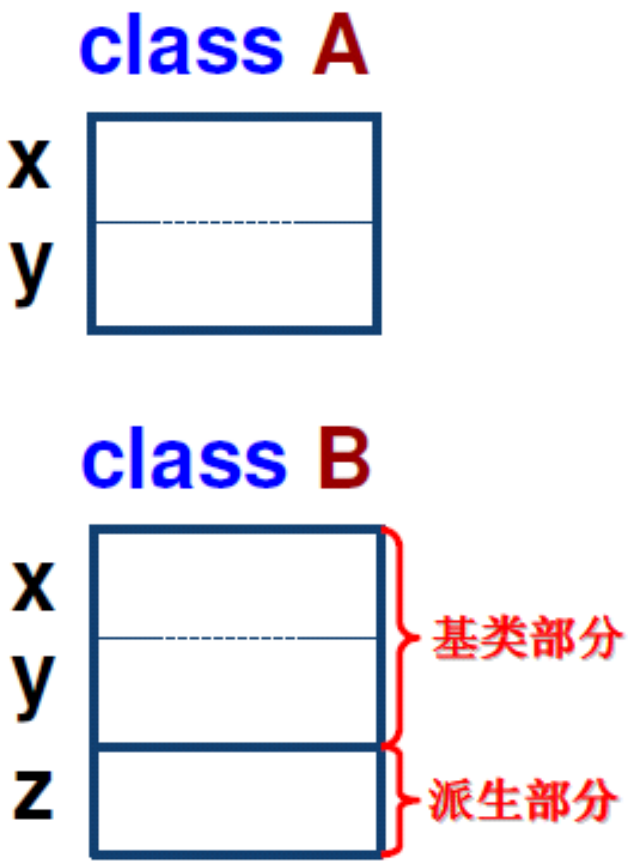
- class Student: public HumanBeing
  - With public inheritance(公有继承), all other base-class members retain their original member access when they become members of the derived class.
  - Note: friend, constructor, destructor functions are not inherited.

11.2 Base Classes and Derived Classes

```
class A{
public:
   int x, y;
};
class B : public A{
public:
   int z;
};

int main()
{
   cout << "Size of A is " << sizeof(A) << endl
        << "Size of B is " << sizeof(B) << endl;
   return 0;
}
```

```
Size of A is 8
Size of B is 12
```



Topics

- 11.1 Introduction
- 11.2 Base Classes and Derived Classes
- **11.3 protected Members**
- 11.4 Relationship between Base Classes and Derived Classes
- 11.5 Constructors and Destructors in Derived Classes
- 11.6 public, protected and private Inheritance

11.3 protected Members

- 解决问题: 派生类访问基类成员的权限控制
- 类有两种用户:
  - 对象(句柄): 对类进行实例化
  - 派生类: 在该类的基础上派生并设计新类

	类对象	派生类
public	/	/
protected	X	/
private	X	X

11.3 protected Members

- 基类的public成员能够被程序中所有函数访问
- 基类的private成员只能被基类的成员和友元函数访问
- 基类的protected成员只能被基类的成员和友元函数+ 派生类的成员和友元函数访问
- • 注意: 不能被类的对象访问

11.3 protected Members

- 派生类如何使用基类的成员?
- • 派生类可以直接通过成员名来使用基类的public成员和protected成员
- • 派生类可以重定义(redefine)基类的成员, 并且依然可以通过以下方式访问基类的public /protected成员:

  base-class :: 成员名
- • 重定义(redefine): 在派生类中给出基类的同名成员

Topics

- 11.1 Introduction
- 11.2 Base Classes and Derived Classes
- 11.3 protected Members
- **11.4 Relationship between Base Classes and Derived Classes**
- 11.5 Constructors and Destructors in Derived Classes
- 11.6 public, protected and private Inheritance

11.4 Relationship between Base Classes and Derived Classes

- Commission Employee 佣金制雇员
- Base-salaried commission employees 带底薪的佣金制雇员

11.4.1 CommissionEmployee类

11.4.2 完全重写的BasePlusCommissionEmployee类

11.4.3 继承+ 访问基类private成员

11.4.4 继承+ 访问基类protected成员

11.4.5 继承+ 通过public函数访问private数据

11.4 Relationship between Base Classes and Derived Classes

- class CommissionEmployee

  (first name, last name, social security number, commission rate and gross sales amount)

```
 9  class CommissionEmployee
10  {
11  public:
12     CommissionEmployee( const string &, const string &, const string &,
13        double = 0.0, double = 0.0 );
14
15     void setFirstName( const string & ); // set first name
16     string getFirstName() const; // return first name
17
18     void setLastName( const string & ); // set last name
19     string getLastName() const; // return last name
20
21     void setSocialSecurityNumber( const string & ); // set SSN
22     string getSocialSecurityNumber() const; // return SSN
23
24     void setGrossSales( double ); // set gross sales amount
25     double getGrossSales() const; // return gross sales amount
26
27     void setCommissionRate( double ); // set commission rate (percentage)
28     double getCommissionRate() const; // return commission rate
29
30     double earnings() const; // calculate earnings
31     void print() const; // print CommissionEmployee object
32  private:
33     string firstName;
34     string lastName;
35     string socialSecurityNumber;
36     double grossSales; // gross weekly sales
37     double commissionRate; // commission percentage
38  }; // end class CommissionEmployee
```

```
7  // constructor
8  CommissionEmployee::CommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate )
11 {
12    firstName = first; // should validate
13    lastName = last; // should validate
14    socialSecurityNumber = ssn; // should validate
15    setGrossSales( sales ); // validate and store gross sales
16    setCommissionRate( rate ); // validate and store commission rate
17 } // end CommissionEmployee constructor
```

```
85 // calculate earnings
86 double CommissionEmployee::earnings() const
87 {
88    return commissionRate * grossSales;
89 } // end function earnings
90
91 // print CommissionEmployee object
92 void CommissionEmployee::print() const
93 {
94    cout << "commission employee: " << firstName << ' ' << lastName
95       << "\nsocial security number: " << socialSecurityNumber
96       << "\ngross sales: " << grossSales
97       << "\ncommission rate: " << commissionRate;
98 } // end function print
```

11.4 Relationship between Base Classes and Derived Classes

- class BasePlusCommissionEmployee

  (first name, last name, social security number, commission rate, gross sales amount and base salary)

- class CommissionEmployee

  (first name, last name, social security number, commission rate, gross sales amount)

```
10 class BasePlusCommissionEmployee
11 {
12 public:
13    BasePlusCommissionEmployee( const string &, const string &,
14       const string &, double = 0.0, double = 0.0, double = 0.0 );
15
16    void setFirstName( const string & ); // set first name
17    string getFirstName() const; // return first name
18
19    void setLastName( const string & ); // set last name
20    string getLastName() const; // return last name
21
22    void setSocialSecurityNumber( const string & ); // set SSN
23    string getSocialSecurityNumber() const; // return SSN
24
25    void setGrossSales( double ); // set gross sales amount
26    double getGrossSales() const; // return gross sales amount
27
28    void setCommissionRate( double ); // set commission rate
29    double getCommissionRate() const; // return commission rate
30
31    void setBaseSalary( double ); // set base salary
32    double getBaseSalary() const; // return base salary
33
34    double earnings() const; // calculate earnings
35    void print() const; // print BasePlusCommissionEmployee object
36 private:
37    string firstName;
38    string lastName;
39    string socialSecurityNumber;
40    double grossSales; // gross weekly sales
41    double commissionRate; // commission percentage
42    double baseSalary; // base salary
43 }; // end class BasePlusCommissionEmployee
44
45 #endif
```

11.4 Relationship between Base Classes and Derived Classes

- Define a new version of

  BasePlusCommissionEmployee class that **inherits directly** from class **CommissionEmployee**

11.4 Relationship between Base Classes and Derived Classes

```
 8  #include "CommissionEmployee.h" // CommissionEmployee class declaration
 9  using namespace std;
10
11  class BasePlusCommissionEmployee : public CommissionEmployee
12  {
13  public:
14     BasePlusCommissionEmployee( const string &, const string &,
15        const string &, double = 0.0, double = 0.0, double = 0.0 );
16
17     void setBaseSalary( double ); // set base salary
18     double getBaseSalary() const; // return base salary
19
20     double earnings() const; // calculate earnings
21     void print() const; // print BasePlusCommissionEmployee object
22  private:
23     double baseSalary; // base salary
24  }; // end class BasePlusCommissionEmployee
```

注意点

- 使用#include包含基类的头文件

- • 告诉编译器基类的存在(基类名)
- • 让编译器根据类定义确定对象大小以分配内存: 派生类的对象大小取决于派生类显式定义的数据成员和继承自基类(直接+间接)的数据成员
- • 让编译器能够判断派生类是否正确地使用了基类的成员

注意点

- 继承的语法

  class BasePlusCommissionEmployee :

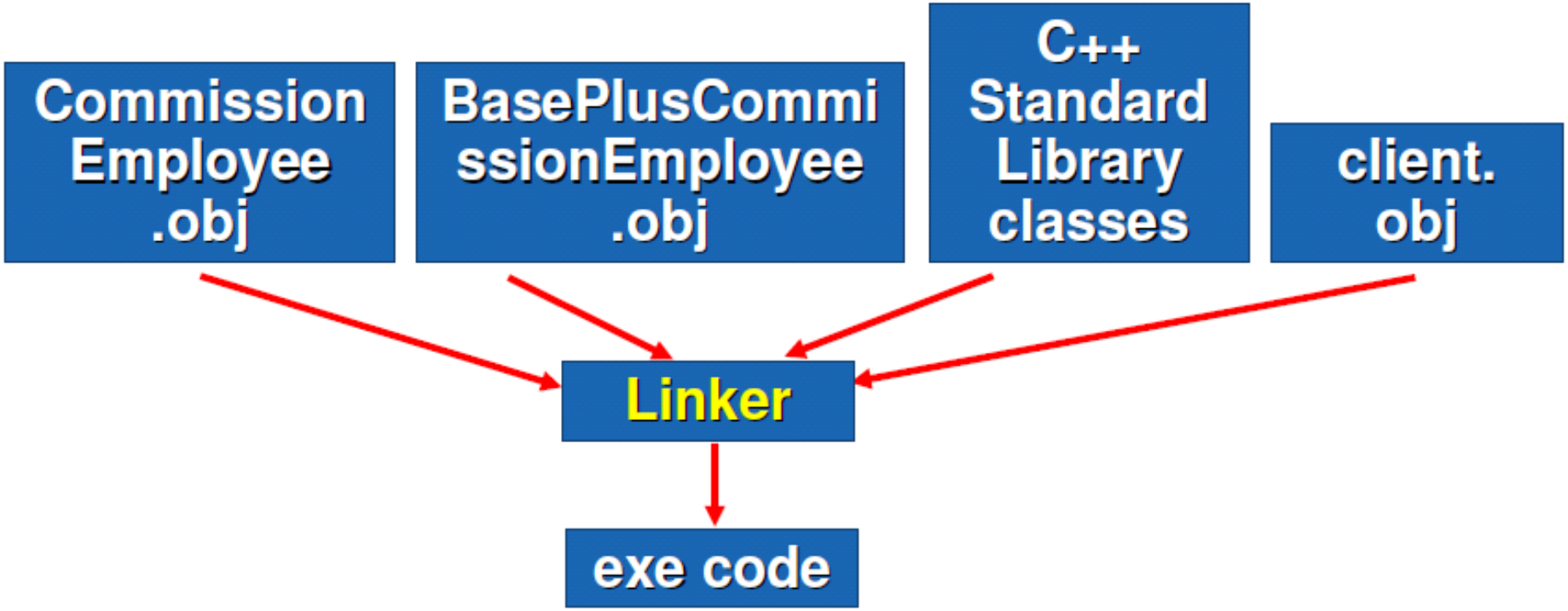
  public CommissionEmployee

- 基类在派生类构造函数初始化列表中初始化
  - 一般应显式调用构造函数进行初始化
  - 如果未显式初始化, 则编译器隐性调用缺省构造函数

```
 7  // constructor
 8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
 9     const string &first, const string &last, const string &ssn,
10     double sales, double rate, double salary )
11     // explicitly call base-class constructor
12     : CommissionEmployee( first, last, ssn, sales, rate )
13  {
14     setBaseSalary( salary ); // validate and store base salary
15  } // end BasePlusCommissionEmployee constructor
33  double BasePlusCommissionEmployee::earnings() const
34  {
35     // derived class cannot access the base class's private data
36     return baseSalary + ( commissionRate * grossSales );
37  } // end function earnings
38
39  // print BasePlusCommissionEmployee object
40  void BasePlusCommissionEmployee::print() const
41  {
42     // derived class cannot access the base class's private data
43     cout << "base-salaried commission employee: " << firstName << ' '
44        << lastName << "\nsocial security number: " << socialSecurityNumber
45        << "\ngross sales: " << grossSales
46        << "\ncommission rate: " << commissionRate
47        << "\nbase salary: " << baseSalary;
48  } // end function print
```

注意点

- 链接过程


Commission Employee .obj
BasePlusCommissionEmployee .obj
C++ Standard Library classes
client. obj
Linker
exe code

```
7  // constructor
8  BasePlusCommiss
9     const string
10    double sales
11    // explicitl
12    : Commission
13 {
14    setBaseSalar
15 } // end BasePl
33 double BasePlusC
34 {
35    // derived cl
36    return baseSa
37 } // end functio
38
39 // print BasePlu
40 void BasePlusCom
41 {
42    // derived class cannot access the base class's private data
43    cout << "base-salaried commission employee: " << firstName << ' '
44       << lastName << "\nsocial security number: " << socialSecurityNumber
45       << "\ngross sales: " << grossSales
46       << "\ncommission rate: " << commissionRate
47       << "\nbase salary: " << baseSalary;
48 } // end function print
```

```
C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(36) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(36) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(43) :
   error C2248: 'CommissionEmployee::firstName' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(44) :
   error C2248: 'CommissionEmployee::lastName' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(44) :
   error C2248: 'CommissionEmployee::socialSecurityNumber' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(45) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'
```

11.4 Relationship between Base Classes and Derived Classes

- 使用protected数据成员替代private数据成员

```
class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
      double = 0.0, double = 0.0 );

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate

   double earnings() const; // calculate earnings
   void print() const; // print CommissionEmployee object
protected:
   string firstName;
   string lastName;
   string socialSecurityNumber;
   double grossSales; // gross weekly sales
   double commissionRate; // commission percentage
}; // end class CommissionEmployee
```

11.4 Relationship between Base Classes and Derived Classes

- 影响数据的有效性检查
- • 在派生类中可以直接修改基类的数据成员

- 派生类依赖于基类的实现
- • 基类的数据成员发生改变有可能影响派生类的实现
- • 软件健壮性差

11.4 Relationship between Base Classes and Derived Classes

- 私有数据成员+ public函数接口

```
30
31 // calculate earnings
32 double BasePlusCommissionEmployee::earnings() const
33 {
34    return getBaseSalary() + CommissionEmployee::earnings();
35 } // end function earnings
36
37 // print BasePlusCommissionEmployee object
38 void BasePlusCommissionEmployee::print() const
39 {
40    cout << "base-salaried ";
41
42    // invoke CommissionEmployee's print function
43    CommissionEmployee::print();
44
45    cout << "\nbase salary: " << getBaseSalary();
46 } // end function print
```

注意点

- ① 通过调用基类的public成员函数来访问基类的私有数据成员
- ② 当功能相同时, 尽量调用成员函数, 以避免代码拷贝
- ③ 注意print()的重定义: 调用基类的print()成员函数时, 一定要使用“基类名::”, 否则会引起无限递归
- ④ 符合软件工程要求: 使用继承, 通过调用成员函数隐藏了数据, 保证了数据的一致性

redefine与overload的区别

```
1.  class A
2.  {
3.  public:
4.      void f(){ cout << "A::f()" << endl; }
5.  };
6.  class B: public A
7.  {
8.  public:
9.      void f(){ cout << "B::f()" << endl; }
10.     void h(){
11.         f();
12.         A::f();
13.     }
14. };
15.
16. int main()
17. {
18.     B b;
19.     b.f();
20.     b.A::f();
21.     return 0;
22. }
```

```
B::f()
A::f()
```

redefine与overload的区别

```
1.  class A
2.  {
3.  public:
4.      void f(){ cout << "A::f()" << endl; }
5.  };
6.  class B: public A
7.  {
8.  public:
9.      void f( int n){ cout << "B::f()" << endl; }
10.     void h(){
11.         f(0);
12.         A::f();
13.     }
14. };
15.
16. int main()
17. {
18.     B b;
19.     b.f();
20.
21.     return 0;
22. }
```

b.A::f();

error C2660: 'f' : function does not take 0 parameters

Topics

- 11.1 Introduction
- 11.2 Base Classes and Derived Classes
- 11.3 protected Members
- 11.4 Relationship between Base Classes and Derived Classes
- **11.5 Constructors and Destructors in Derived Classes**
- 11.6 public, protected and private Inheritance

11.5 Constructors and Destructors in Derived Classes

- 基类先构造, 派生类后构造

A → B

- 派生类先析构, 基类后析构

B → A

11.5 Constructors and Destructors in Derived Classes

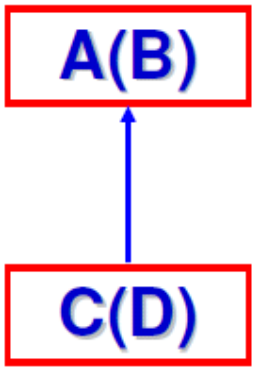
- 若类中含有其他类的对象(组合)

  构造: 先基类后派生类, 先被包含类后宿主类

  B → A → D → C

  析构: 与构造顺序相反

  C → D → A → B



11.5 Constructors and Destructors in Derived Classes

- 全局对象: 在任何函数(含**main**)执行前, 构造; 在程序结束时, 析构.
- 局部对象
  - 自动变量: 对象定义时, 构造; 块结束时, 析构.
  - 静态变量: 首次定义时, 构造; 程序结束时, 析构.
- 多个全局和静态对象(均为静态存储类别)析构顺序恰好与构造顺序相反.
- 特例1: 调用**exit**函数退出程序执行时, 不调用剩余自动对象的析构函数.
- 特例2: 调用**abort**函数退出程序执行时, 不调用任何剩余对象的析构函数.

```
1.  int main()
2.  {
3.      cout << fixed << setprecision( 2 );
4.      { // begin new scope
5.        CommissionEmployee employee1
6.           "Bob", "Lewis", "333-33-3333",
7.      } // end scope

8.      cout << endl;
9.      BasePlusCommissionEmployee
10.        employee2( "Lisa", "Jones", "555-55-5555",
11.
12.     cout << endl;
13.     BasePlusCommissionEmployee
14.        employee3( "Mark", "Sands", "888-88-8888"
15.     cout << endl;
16.     return 0;
17. } // end main
```

```
CommissionEmployee constructor:
CommissionEmployee destructor:
CommissionEmployee constructor:
BasePlueCommissionEmployee constructor:
CommissionEmployee constructor:
BasePlueCommissionEmployee constructor:
BasePlueCommissionEmployee destructor:
CommissionEmployee destructor:
BasePlueCommissionEmployee destructor:
CommissionEmployee destructor:
```

Design and implement a hierarchical class structure, according to the following requirements.

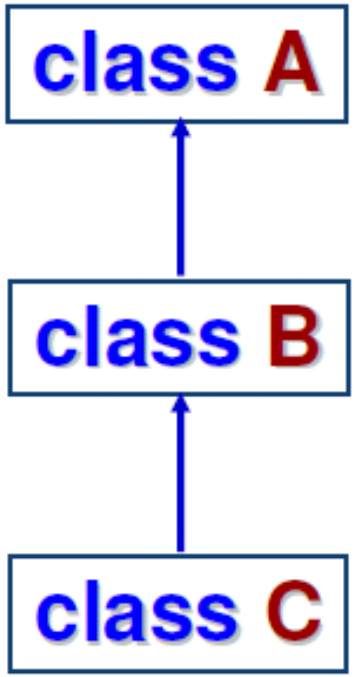
- Shape is a base class.
- Classes Circle and Rectangle are directly inherited from shape.
- Square is directly inherited from Rectangle.
- Each object must include at least one data member named id (string).
- Objects of derived classes should contain some necessary data members to determine their area, such as radius for Circle etc.
- Objects of class Square have one special method named incircle. This method can create and return the inscribed circle object(circle) of the corresponding Square object.
- Each object provides area() function to calculate the area of an shape object and print() function to display all information of an object such as radius, width, length, area and incircle.

Topics

- 11.1 Introduction
- 11.2 Base Classes and Derived Classes
- 11.3 protected Members
- 11.4 Relationship between Base Classes and Derived Classes
- 11.5 Constructors and Destructors in Derived Classes
- **11.6 public, protected and private Inheritance**

11.6 public, protected and private Inheritance

- B继承A, C继承B



- A的对象和B能否访问A的成员

  取决于A的成员的访问权限设置

- B的对象和C能否访问B中继承自A的成员

  取决于A的成员的访问权限设置和B继承A的类型, 即public / protected / private inheritance

11.6 public, protected and private Inheritance

继承方式	基 类(A)	派生类(B)
public	public成员 →	public成员
	protected成员 →	protected成员
	private成员 →	不可见
private	public成员 →	private成员
	protected成员 →	private成员
	private成员 →	不可见
protected	public成员 →	protedted成员
	protected成员 →	protected成员
	private成员 →	不可见

```
1.  class A{
2.  public:
3.      int x;
4.  protected:
5.      int y;
6.  private:
7.      int z;
8.  };
9.  class B : public A{
10. public:
11.     int m;
12.     int f(){ x=1;   public
13.              y=2;   protected
14.          X  z=3; }  private
15. };
16. int main()
17. {
18.     B obj;
19.     obj.x = 10;
20.     obj.y = 20;   X
21.     obj.z = 30;   X
22.     obj.m = 40;
23. }
```

```
1.  class A{
2.  public:
3.      int x;
4.  protected:
5.      int y;
6.  private:
7.      int z;
8.  };
9.  class B : protected A{
10. protected:
11.     int m;
12.     int f(){ x=1;   protected
13.              y=2;   protected
14.          X  z=3; }  private
15. };
16. int main()
17. {
18.     B obj;
19.     obj.x = 10;   X
20.     obj.y = 20;   X
21.     obj.z = 30;   X
22.     obj.m = 40;   X
23. }
```

```
1.  class A{
2.  public:
3.      int x;
4.  protected:
5.      int y;
6.  private:
7.      int z;
8.  };
9.  class B : private A{
10. private:
11.     int m;
12.     int f(){ x=1;   private
13.              y=2;   private
14.          X  z=3; }  private
15. };
16. int main()
17. {
18.     B obj;
19.     obj.x = 10;   X
20.     obj.y = 20;   X
21.     obj.z = 30;   X
22.     obj.m = 40;   X
23. }
```

Summary

- 基类和派生类的定义
- protected成员, 派生类如何访问基类成员
- 继承关系中构造函数和析构函数顺序
- 三种继承

Homework

- 实验必做题目：Ex1, Ex3 (5), Ex4, Ex5
- 实验选做题目：11.10

其中，Ex4 主函数修改如下：

```
int main()
{
    MyBase a(2), *p = &a;
    MyDerived b(4), *q= &b;
    MyBase &c = a;
    MyBase &d = b;
    cout << a.getX() << " " << p-> getX();
    cout << b.getY() << " " << q-> getY() << b.getX() << " " << q-> getX();
    a = b;
    cout << a.getX() << " "; // << a.getY() << endl;
    p = q;
    cout << p->getX() << " "; // << p->getY() << endl;
    cout << c.getX() << " " << d.getX() << " "; // << d.getY() << endl;
   // b = a;
    cout << b.getX() << " " << b.getY() << endl;
}
```