河北师范大学软件学院
Software College of Hebei Normal University

C++

第十讲 类和对象（六）

C++备课组 丁盟

自我介绍

丁盟

qq：2622885094

上一讲教学目标

- 掌握C++中对象数组的使用
- 理解C++中this指针的含义及使用
- 掌握C++中对象引用的使用

本讲教学目标

- 了解C++中嵌套类的使用
- 了解C++中局部类的使用

1 指向成员的指针

2 嵌套类

3 局部类

指向成员的指针

成员指针的分类

- **指向数据成员的指针**
- 指向成员函数的指针

指向数据成员的指针

- 格式：类型　类名::*指针变量名；

```
int Type::*p = NULL;
```

- 数据成员的地址：

&类名::数据成员；

```
&Type::ival;
```

- 指向：int Type::*p = &Type::ival;

指向数据成员的指针

```
class Box {
public:
    Box(int aLen=0);
    int get() const;
    int m_iLen;
private:
};
Box::Box(int aLen) {
    m_iLen = aLen;
}
int Box::get() const {
    return m_iLen;
}
```

```
int main(void)
{
    Box box1;
    Box *q = &box1;
    int Box::*p
        = &Box::m_iLen;
    cout << box1.*p
         << endl;
    cout << q->*p
         << endl;

    return 0;
}
```

指向成员的指针

成员指针的分类

- 指向数据成员的指针
- **指向成员函数的指针**

指向成员函数的指针

- 回忆指向函数指针的定义方法:

```
void max(int aFirst, int aSec)
{
    ……
}
void (*p)( int, int ) = &max;
```

- 函数指针在两个方面和指向的函数匹配：
  ① 函数形参的类型和数目，包括形参是否为const
  ② 函数返回值类型

指向成员函数的指针

- 声明：

指向成员函数的指针要从三个方面与他指向的函数匹配

① 函数形参的类型和数目，包括成员是否为const

② 函数返回值类型

③ 所属类类型

指向成员函数的指针

- 格式：

返回类型 (类名::*指针变量名)（参数列表）

```
void (Type::*p)();
```

- 成员函数地址的获取：

&类名::成员函数名

不推荐省略&

```
&Type::display
```

指向成员函数的指针

- 指向：

返回类型 (类名::*指针变量名)(参数列表)；
类名::指针变量名 = &类名::成员函数名;

```
void (Atest ::*p1)() ;
Atest::p1 = &Atest::display;
```

返回类型 (类名::*指针变量名)(参数列表)
= &类名::成员函数名;

```
void (Type ::*p1)() = &Type::display;
```

指向成员函数的指针

```
class Type {
public:
    Type(int arg = 0);
    int get() const;
    void set(int arg);
    int m_iVal;
};
Type::Type(int arg):m_iVal(arg)
{}
int Type::get() const {
    return m_iVal;
}
void Type::set(int arg) {
    m_iVal = arg;
}
```

```
int main(void) {
    Type  obj;
    int (Type::*p)() const
        = &Type::get;
    cout << obj.get() << " ";
    cout << (obj.*p)() << endl;
    Type *p1 = &obj;
    cout << p1->get() << " ";
    cout << (p1->*p)() << endl;
    cout << (*p1).get() << " ";
    cout << ((*p1).*p)() <<
endl;

    return 0;
}
```

指向成员函数的指针

- 总结：

  通过成员指针引用成员必须借助两个运算符

  - 成员指针解引用操作符: ".*"

  功能：从对象或引用获取成员

  - 成员指针箭头操作符: "->*"

  功能：通过对象的指针获取成员

	对象	对象指针及"->"	对象指针及"."
引用数据成员	obj.*p	p_obj->*p	(*p_obj).*p
引用成员函数	(obj.*p)()	(p_obj->*p)()	((*p_obj).*p)()

1 指向成员的指针

2 嵌套类

3 局部类

嵌套类

- 嵌套类的概念
- 嵌套类的定义方法
- 嵌套类的访问权限
- 成员互访
- 嵌套类域中的名字解析

嵌套类的概念

1. 概念：

一个类可以在另一个类中定义，这样的类成为嵌套类

```
class Tree // 外围类
{
public:
    class Node // 嵌套类是外围类的"类型"成员
    { };
};
```

嵌套类的定义方法

2. 定义：定义在另一个类内

[注意] 嵌套类的成员函数（或静态成员）可以在嵌套类外定义，但不能在外围类中实现，只能在外围类之外实现。

```
class List {
public:
    class Item {
    public:
        void mf(const List &r);
        static int value;
        int memb;
    };
};
```

```
int List::Item::value;
void List::Item::mf(const List
&r)
{ }
int main(void)
{
    return 0;
}
```

嵌套类的定义方法

- 嵌套类可以先在外围类中声明，然后在外围类之外定义

```
class List {
public:
    int init(int);
private:
    class ListItem;
};
class List::ListItem {
public:
    static int value;
    void  mf( const  List  &r);
    int  memb;
};
```

```
int List::ListItem::value;
void List::ListItem
    ::mf(const List &r)
{ }

int main(void)
{
    return 0;
}
```

嵌套类的定义方法

- 嵌套类可以先在外围类中声明，然后在外围类中定义

```
class List {
private:
    class ListItem; // 声明
    class Red {
        // 在看到类体前只能定义该类的指针或引用
        ListItem *pli;
    };
    class ListItem { // 定义
        Red *pref;
    };
};
```

嵌套类的定义方法

- 嵌套类本质是另一个类的"类型成员"，两个类作为两个命名空间，成员名可相同

```
class Node {};
class Tree {
public:
    int m_iVal;
    class Node    // Tree::Node隐藏了::Node
    {
        int m_iVal;
    };
    Node * tree; //ok:被解析为嵌套类:Tree::Node
};
```

嵌套类的访问权限

3. 嵌套类的访问权限

嵌套类作为"类型成员"和普通成员一样可以用访问标号进行限定

访问标号	使用权限
private类型成员	只能在本类内使用
protected类型成员	在本类或者派生类中使用
public:类型成员	任何位置

嵌套类的访问权限

```
class Tree {
public:
      class Node1 { };
private:
      class Node2 { };
      Node1 obj1; // OK
      Node2 obj2; // OK
};
int main(void) {
     Tree::Node1 objNode1;
     // Tree::Node2 objNode2;
     return 0;
}
```

通常嵌套类被限定为private

类外访问类型成员时需类型限定！

private类型成员不能在类外使用！

成员互访

4. 嵌套类访问内层类的成员

回忆一下豪不相关的两个类之间成员的相互访问

- 类A不能访问类B的非公有成员，如果非要访问可将类A定义为类B的友元类
- 类A不能直接访问类B的公有成员，只能通过类B的对象、引用、指针访问

成员互访

- 本质上，外围类和内层类是无关的两个类，因此他们的成员彼此无关，不能直接访问

```
class Outer {
public:
    int getInner() const {
        // return m_dVal;
    }
private:
    int m_iVal;
    class Inner;
};
```

```
class Outer::Inner {
public:
    int getOuter() const {
        // return m_iVAl;
    }
private:
    int m_dVal;
};
```

成员互访

- 外部类与嵌套类可以通过彼此的对象、指针、引用间接访问彼此的公有成员

```
class List {
public:
   int init( int );
private:
   class ListItem {
   public:
      ListTtem(int val = 0);
      void mf(List &r);
      int  memb;
   };
};
```

```
List::ListItem::ListItem(int  val) {
    // List::init()是类List的非静态成员
    // 必须通过List类型的对象或者指针使用
    value = init( val ); // 错误
}
List::ListItem::mf(const List &r) {
    // OK,通过引用调用init( )
    memb = r.init();
}
```

成员互访

- 嵌套类可以直接访问外围类的公有静态成员、类型名、枚举值，这些成员可以不加限定修饰符

（类型名指一个typedef名字，枚举类型名、或一个类名）

成员互访

```
class List {
public:
    static int s_iSize;
    typedef int (*pFunc)();
    enum ListStatus {
        Good,Empty,Corrupted };
private:
    class ListItem {
    public:
        void  check_status();
        ListStatus status;
        pFunc action;
    };
};
```

```
int List::s_iSize;

void List::ListItem::check_status()
{
    s_iSize = 5;
    ListStatus s = status;
    switch(s)
    {
    case Empty: break;
    case Corrupted: break;
    case Good: break;
    }
}
```

成员互访

- 在外围类作用域之外引用外围类的静态成员，类型名和枚举名都要求有解析操作符

```
List::pFunc myAction; // OK
List::ListStatus stat = List::Empty; // OK
```

- 引用枚举值时不能写成下面形式，因为枚举定义并不像类定义一样，他不产生名字空间

```
List::ListListStatus::Empty // Error
List::Empty; // OK
```

嵌套类域中的名字解析

5. 嵌套类域中的名字解析

类体之中而成员函数定义之外的名字解析

嵌套类成员函数体之中的名字解析

嵌套类域中的名字解析

- 解析过程

① 名字使用点之前的嵌套类的成员声明

② 名字使用点之前的外围类的成员声明

③ 名字使用点之前的全局作用域中的声明

嵌套类域中的名字解析

```
enum ListStatus { Good ,Empty , Corrupted };
class List {
public:
    //......
private:
    class ListItem {
        // 查找: 1) ListItem中
        //       2) 在List中
        //       3) 在全局域中
        ListStatus status; // 引用全局枚举
    };
    //......
};
```

嵌套类域中的名字解析

```
enum ListStatus {Good, Empty, Corrupted};
class List {
private:
    class ListItem;
    //......
public:
    enum  ListStatus {Good, Empty, Corrupted};
    //......
};
class List::ListItem{
public:
    ListStatus status; // List::ListStatus
};
```

嵌套类域中的名字解析

- 类体之中而成员函数定义之外的名字解析
- **嵌套类成员函数体之中的名字解析**

嵌套类域中的名字解析

- 解析过程
  - ① 从成员函数局部域中到名字使用点前的声明
  - ② 嵌套类作用域中
  - ③ 外围类作用域中
  - ④ 成员函数定义之前的名字空间域中出现的声明

嵌套类域中的名字解析

```
class List {
public:
     int list;
     enum ListStatus { Good,Empty,Corrupted };
private:
    class ListItem {
    public:
        void check_staus();
        ListStatus status;
    };
};
int list = 0; // 全局List
void List::ListItem::check_status() {
    int value = list; // 哪个list?
}
```

为了访问全局的list必须加::

```
void List::ListItem::check_status()
{
    int value = ::list;
}
```

1 指向成员的指针

2 嵌套类

3 局部类

局部类

- 概念：

  在函数体内定义的类

- 定义：

  局部类的数据成员和成员函数必须完全定义在类体内，因此局部类不允许声明static成员

```
void foo() {
    class Bar {
    public:
        void fn() { }
    private:
        int ival;
        static int count;
    };
}
```

局部类

- 局部类和外围函数中元素的互访

① 局部类中不能使用函数中的变量

```
void foo() {
    int iival;
    class Bar {
    public:
        void fun() {  iival; //Error }
    };
}
```

不能访问外围函数的变量

局部类

② 外围函数不能访问内部类的非公有成员，可通过类的对象、指针、引用访问公有成员

```
void foo() {
    int iival;
    class Bar {
    public:
        void display() {cout << "Class Bar";}
    };
    Bar obj;
    obj.display(); // right
}
```

局部类

③ 外围函数不能访问局部类的公有静态成员、类型名、枚举值(类型名是一个typedef名字，枚举类型名、或一个类名）

局部类

```
void foo() {
    class Bar {
    public:
        enum test{one,two};
        typedef int INTE;
        Bar(int n=0)
        {  m_iVal = n;}
        int get() const
        {  return m_iVal;}
    private:
        int m_iVal;
    };
    Bar obj(100);
    // INTE a; // ERROR
    // cout << one; // ERROR
    // test en; // ERROR
    cout << obj.get() << endl;
}
```

局部类

④ 局部类被完全限定在了函数体这个名字空间中，函数外无法使用这个类定义对象！

```
void foo() {
    class Bar { ... };
}
int main(void) {
    Bar obj; // Error

    return 0;
}
```

本质上讲局部类是就是函数的内部"类型成员"，只能在函数体内使用这个类型定义对象，而不能在函数外面使用这个类型

局部类

⑤ 局部类的名字解析过程和嵌套类解析的原理相同

成员函数定义外的名字解析

```
typedef int COUNT;
void foo() {
    class Bar {
    public :
        void test(){}
    private:
        COUNT a;
    };
}
```

成员函数定义中的名字解析

```
void foo() {
    class Bar {
    public :
        void test(){COUNT a;}
    private:
        int ival;
        typedef int COUNT;
    };
}
```

本讲教学目标

- 了解C++中嵌套类的使用
- 了解C++中局部类的使用


THANKS