|  |  |
| --- | --- |
| **编号:** | **5-1** |

****

信息科学与工程学院实验报告

《面向对象程序设计》

**Object-Oriented Programming**

|  |  |
| --- | --- |
| 姓名： | 张泽浩 |
| 学号： | 202111000212 |
| 班级： | 计工本2102 |
| 导师: | 张庆科 |
| 时间： | 2022年12月10日 |

**《面向对象程序设计》实验报告**

**基本要求：**实验报告包含实验目的、实验内容、实验过程（详细操作流程）、实验结果（程序运行结果高清截图）、实验分析总结五个部分。报告中若涉及代码程序，请在附录部分提供完整程序源码及源码托管地址(基于Highlight软件导入源码)。报告撰写完毕后请提交PDF格式版本报告到课程云班课系统。

1. **实验目的**
2. 理解类的三种不同关系（组合，依赖，继承）；
3. 掌握复合类构造函数、析构函数的定义方法与使用方法；
4. 熟练掌握类继承的定义方式（单继承，多继承）；
5. 理解三种不同继承方式间的区别（公有，私有，保护）；
6. 掌握派生类同名覆盖原理及相应同名冲突解决方法；
7. 掌握赋值兼容性基本原理（左基 = 右派）；
8. 熟练掌握复杂类的设计方法（三构一析+普函）。
9. **实验内容**
10. **任务一：类继承的设计**

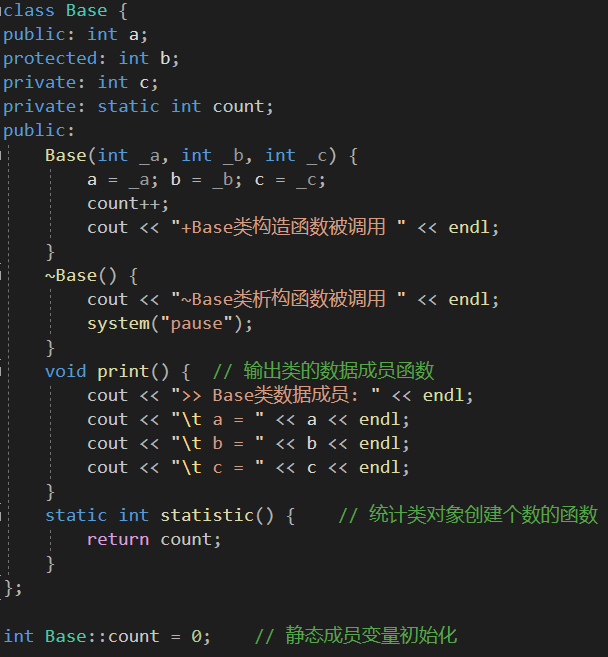
设计一个基类base，其内含有数据成员（public: int a, protected: int b, private: int c, private: static int count）和函数成员（输出类的数据成员函数print( )，统计类对象创建个数的函数 static int statistic( )），然后请采用三种不同的继承方式由base类分别派生出三个子类：derived1, derived2, derived3, 请根据上述基类和派生类尝试编程论证下面的三个问题。（可参考课堂演示程序）

1. 派生类全盘接受基类的所有本类成员，其中包括基类的普通公有成员，保护成员和私有成员。
2. 根据继承类数据成员能否在类内或类外被访问的问题，探索分析三种不同继承方式各自的特点（参考课程ppt）。
3. 派生类对象被建立时派生类是如何调用构造函数的，给出构造函数调用的次序，析构函数析构次序，并分析其中规律。
4. **任务二：类继承的设计**

定义一个二维空间点类Location，采用数据成员x, y表示该类对象在二维坐标系中的坐标位置，类中函数成员函数move( )可以实现移动该类对象的坐标位置，show( )函数可以输出当前类对象的信息。然后，以Location为基类，派生出三维空间坐标点类Point，接着，再利用三维空间点类Point派生出一个三维空间下的球体类Sphere,定义Point点类和球体类Sphere中各自特有的move( )函数和show( )函数。要求设计并实现上述类，并在主函数中定义各个类的对象，通过各自对象调用上述成员函数。

1. **实验过程**
2. **任务一：类继承的设计**
3. 基类设计

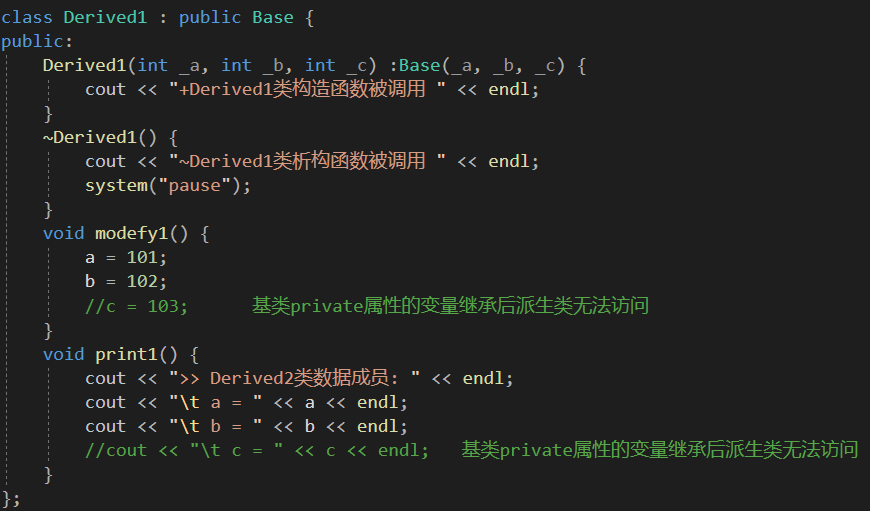
基类Base中包含成员变量a、b、c和静态成员变量count，以及成员函数print( )和静态成员函数statistic( )（如图1）。

****

**图1 Base类的设计**

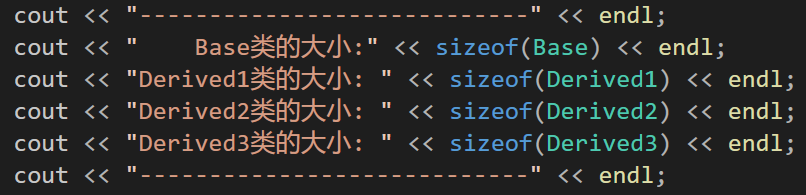
1. 派生类设计

Derived1类、Derived2类、Derived3类分别以公有public、私有private、保护protected的形式继承Base类。在每个派生类中设计一个修改成员变量的函数和输出成员变量函数，分别对继承来的成员变量进行修改操作和输出操作。（派生类Derived1如图2）

****

**图2 Derived1派生类的设计**

1. 主函数设计
2. 通过使用sizeof( )运算符计算基类和派生类所占空间大小，判断派生类是否全盘接受基类的所有本类成员（公有成员、私有成员、保护成员）。（如图3）

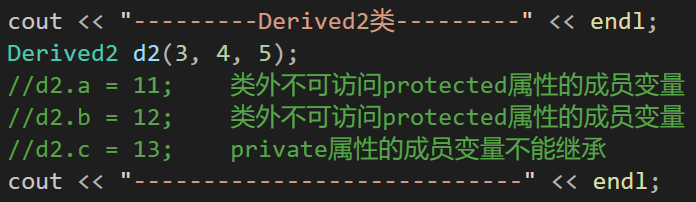
****

**图3 计算基类和派生类所占空间大小**

1. 在每一个派生类中，设计一个修改函数，用于修改继承的成员变量的值，判断类内是否能够访问继承类数据成员；在主函数中，分别对每个派生类的继承数据成员进行修改，判断类外是否能够访问继承类数据成员。（如图4、图5）

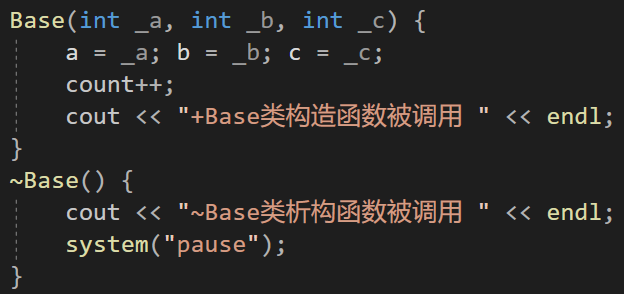
****

**图4 类内尝试修改继承类数据成员**

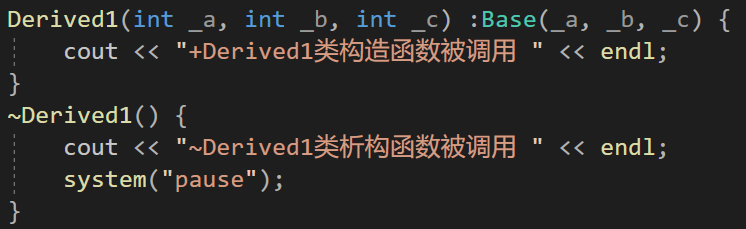
****

**图5 类外尝试修改继承类数据成员**

1. 在基类和所有派生类的构造函数和析构函数中，添加输出标志语句，在执行构造与析构时，能够实时反馈输出，从而能够更好的判断构造函数和析构函数调用的次序。（如图6、图7）

****

**图6 Base类构造函数与析构函数设计**

****

**图7 派生类构造函数与析构函数设计**

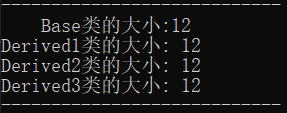
1. **任务二：类继承的设计**

本程序设计了三个类，分别为Location类、Point类、Sphere类，分别表示二维坐标系中的点类、三维坐标系中的点类和球体类。在每个类中，分贝设计了move( )函数和show( )函数，用于移动点和输出点的坐标。三个类之间的继承关系为：Point类继承Location类、Sphere类继承Point类。（具体代码见附录）

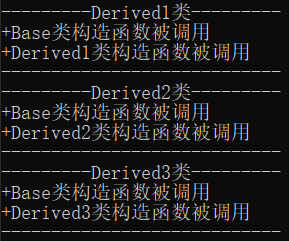
1. **实验结果**
2. **任务一：类继承的设计**

基于对上述程序的实现，可得Base类、Derived1类、Derived2类、Derived3类所占空间大小均为12字节（如图8），由此可得结论：派生类全盘接受基类的所有本类成员。

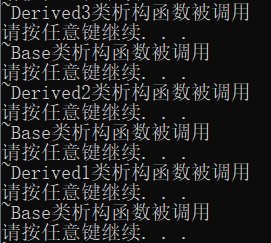
构造函数和析构函数的调用顺序是：构造时，先调用基类的构造函数，再调用派生类的构造函数；析构时，先调用派生类的析构函数，再调用基类的析构函数（如图9、图10）。

****

**图8 sizeof( )运算符计算类所占空间大小运行结果**

****

**图9 构造函数调用顺序运行结果**

****

**图10 析构函数调用顺序运行结果**

通过对程序代码的修改与运行，验证了继承类数据成员在类内类外的访问规律，即：公有继承public方式所继承的父类的public属性的成员变量既可以在类内访问也可以在类外访问，protected属性的成员变量只能在类内访问，private属性的成员变量不可访问；保护继承protected方式所继承的父类的public属性的成员变量和protected属性的成员变量只能在类内访问，private属性的成员变量不可访问；私有继承private方式所继承的父类的public属性的成员变量和protected属性的成员变量只能在类内访问，private属性的成员变量无法访问。

综上所述，三种不同继承方式各自的特点可以总结为：公有继承一一对应、保护继承被保护、私有继承被私有（如图11、图12、图13）。

****

**图11 共有继承一一对应 图12 保护继承被保护**

****

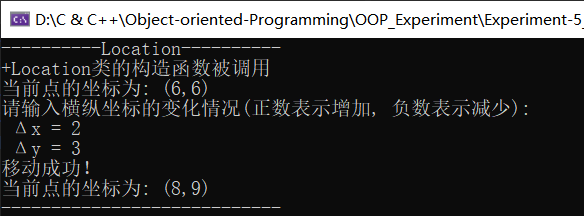
**图13 私有继承被私有**

1. **任务二：类继承的设计**
2. 测试案例

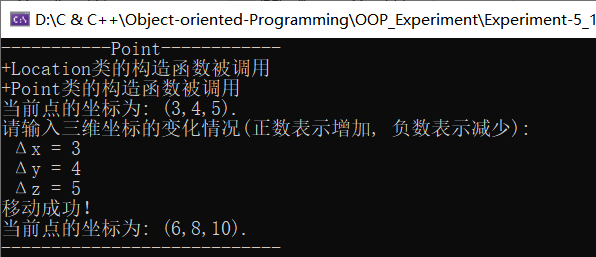
**表1 坐标类的测试案例表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **类** | **坐标及球体半径** | **Δx** | **Δy** | **Δz** | **修改后坐标** |
| Location类 | (6, 6) | 2 | 3 |  | (8, 9) |
| Point类 | (3, 4, 5) | 3 | 4 | 5 | (6, 8, 10) |
| Sphere类 | (0, 0, 0) 5 | 7 | 8 | 9 | (7, 8, 9) |

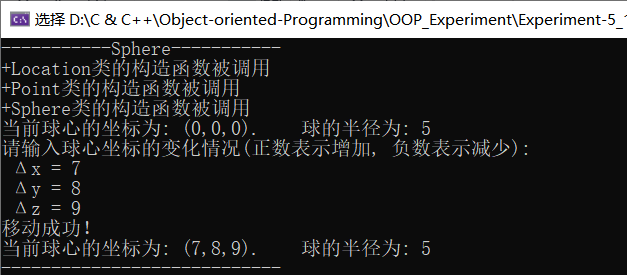
1. 运行结果

****

**图14 Location类的运行结果**

****

**图15 Point类的运行结果**

****

**图16 Sphere类的运行结果**

1. **实验总结**

通过本次实验，掌握了类继承的定义方式，并通过对类继承的实现，理解了三种不同继承方式（公有继承、保护继承、私有继承）之间的区别。通过本实验的任务二，对于基类与派生类分别设计了相同的函数move( )函数和show( )函数，深刻理解了派生类的同名覆盖原理及其解决同名冲突的方法。

在本实验中，通过不断修改程序与调试运行程序，体会了不同继承方式对派生类访问基类成员变量的差别，也深刻理解了“公有继承一一对应”、“保护继承被保护”、“私有继承被私有”的继承的特点。另外，本实验通过使用visio作图，也巩固了visio的使用方法。

* **附录：实验源代码（基于Highlight软件粘贴带有行号的源码）**
* **代码托管地址：**
* [Object-oriented-Programming/OOP\_Experiment/Experiment-5\_1 at master · keepIHDR/Object-oriented-Programming (github.com)](https://github.com/keepIHDR/Object-oriented-Programming/tree/master/OOP_Experiment/Experiment-5_1)
* **任务一**
* 01 #include<iostream>
* 02 using namespace std;
* 03
* 04 class Base {
* 05 public:
* 06 int a;
* 07 protected:
* 08 int b;
* 09 private:
* 10 int c;
* 11 private:
* 12 static int count;
* 13 public:
* 14 Base(int \_a, int \_b, int \_c) {
* 15 a = \_a;
* 16 b = \_b;
* 17 c = \_c;
* 18 count++;
* 19 cout << "+Base类构造函数被调用 " << endl;
* 20 }
* 21 ~Base() {
* 22 cout << "~Base类析构函数被调用 " << endl;
* 23 system("pause");
* 24 }
* 25 void print() { // 输出类的数据成员函数
* 26 cout << ">> Base类数据成员: " << endl;
* 27 cout << "\t a = " << a << endl;
* 28 cout << "\t b = " << b << endl;
* 29 cout << "\t c = " << c << endl;
* 30 }
* 31 static int statistic() { //
* 32 统计类对象创建个数的函数
* 33 return count;
* 34 }
* 35 };
* 36
* 37 int Base::count = 0; // 静态成员变量初始化
* 38
* 39 class Derived1 : public Base {
* 40 public:
* 41 Derived1(int \_a, int \_b, int \_c) :Base(\_a, \_b, \_c) {
* 42 cout << "+Derived1类构造函数被调用 " << endl;
* 43 }
* 44 ~Derived1() {
* 45 cout << "~Derived1类析构函数被调用 " << endl;
* 46 system("pause");
* 47 }
* 48 void modefy1() {
* 49 a = 101;
* 50 b = 102;
* 51 //c = 103;
* 52 //
* 53 基类private属性的变量继承后派生类无法
* 54 访问
* 55 }
* 56 void print1() {
* 57 cout << ">> Derived2类数据成员: " << endl;
* 58 cout << "\t a = " << a << endl;
* 59 cout << "\t b = " << b << endl;
* 60 //cout << "\t c = " << c << endl;
* 61 //
* 62 基类private属性的变量继承后派生类无法
* 63 访问
* 64 }
* 65 };
* 66
* 67 class Derived2 : private Base {
* 68 public:
* 69 Derived2(int \_a, int \_b, int \_c) :Base(\_a, \_b, \_c) {
* 70 cout << "+Derived2类构造函数被调用 " << endl;
* 71 }
* 72 ~Derived2() {
* 73 cout << "~Derived2类析构函数被调用 " << endl;
* 74 system("pause");
* 75 }
* 76 void modefy2() {
* 77 a = 201;
* 78 b = 202;
* 79 //c = 203;
* 80 //
* 81 基类private属性的变量继承后派生类无法
* 82 访问
* 83 }
* 84 void print2() {
* 85 cout << ">> Derived2类数据成员: " << endl;
* 86 cout << "\t a = " << a << endl;
* 87 cout << "\t b = " << b << endl;
* 88 //cout << "\t c = " << c << endl;
* 89 //
* 90 基类private属性的变量继承后派生类无法
* 91 访问
* 92 }
* 93 };
* 94
* 95 class Derived3 : protected Base {
* 96 public:
* 97 Derived3(int \_a, int \_b, int \_c) :Base(\_a, \_b, \_c) {
* 98 cout << "+Derived3类构造函数被调用 " << endl;
* 99 }
* 100 ~Derived3() {
* 101 cout << "~Derived3类析构函数被调用 " << endl;
* 102 system("pause");
* 103 }
* 104 void modefy1() {
* 105 a = 301;
* 106 b = 302;
* 107 //c = 303;
* 108 基类private属性的变量继承后派生类无法
* 109 访问
* 110 }
* 111 void print3() {
* 112 cout << ">> Derived3类数据成员: " << endl;
* 113 cout << "\t a = " << a << endl;
* 114 cout << "\t b = " << b << endl;
* 115 //cout << "\t c = " << c << endl;
* 116 基类private属性的变量继承后派生类无法
* 117 访问
* 118 }
* 119 };
* 120
* 121 int main() {
* 122 cout << "----------------------------" << endl;
* 123 cout << " Base类的大小:" << sizeof(Base) << endl;
* 124 cout << "Derived1类的大小: " << sizeof(Derived1) <<
* 125 endl;
* 126 cout << "Derived2类的大小: " << sizeof(Derived2) <<
* 127 endl;
* 128 cout << "Derived3类的大小: " << sizeof(Derived3) <<
* 129 endl;
* 130 cout << "----------------------------" << endl;
* 131
* 132 cout << "-----------Base类-----------" << endl;
* 133 Base bb(1, 2, 3);
* 134 bb.print();
* 135 cout << ">> 基类对象的数量: " << Base::statistic()
* 136 << endl;
* 137 cout << "----------------------------" << endl;
* 138
* 139
* 140 cout << "---------Derived1类---------" << endl;
* 141 Derived1 d1(2, 3, 4);
* 142 d1.a = 11;
* 143 //d1.b = 12;
* 144 类外不可访问protected属性的成员变量
* 145 //d1.c = 13; private属性的成员变量不能继承
* 146 cout << "----------------------------" << endl;
* 147
* 148
* 149 cout << "---------Derived2类---------" << endl;
* 150 Derived2 d2(3, 4, 5);
* 151 //d2.a = 11;
* 152 类外不可访问protected属性的成员变量
* 153 //d2.b = 12;
* 154 类外不可访问protected属性的成员变量
* 155 //d2.c = 13; private属性的成员变量不能继承
* 156 cout << "----------------------------" << endl;
* 157
* 158
* 159 cout << "---------Derived3类---------" << endl;
* 160 Derived3 d3(4, 5, 6);
* 161 //d2.a = 11;
* 162 类外不可访问private属性的成员变量
* 163 //d2.b = 12;
* 164 类外不可访问private属性的成员变量
* 165 //d2.c = 13; private属性的成员变量不能继承
* 166 cout << "----------------------------" << endl;
* 167
* 168
* 169 system("pause");
* 170 return 0;
* 171 }
* 任务二
* 01 #include<iostream>
* 02 using namespace std;
* 03
* 04 class Location {
* 05 protected:
* 06 int x, y;
* 07 public:
* 08 Location(int \_x, int \_y) {
* 09 x = \_x;
* 10 y = \_y;
* 11 cout << "+Location类的构造函数被调用" <<
* 12 endl;
* 13 }
* 14 ~Location() {
* 15 cout << "~Location类的析构函数被调用" <<
* 16 endl;
* 17 system("pause");
* 18 }
* 19 void move() {
* 20 int \_x, \_y;
* 21 cout << "请输入横纵坐标的变化情况(
* 22 正数表示增加, 负数表示减少): " << endl;
* 23 cout << " Δx = ";
* 24 cin >> \_x;
* 25 cout << " Δy = ";
* 26 cin >> \_y;
* 27 x += \_x;
* 28 y += \_y;
* 29 cout << "移动成功！" << endl;
* 30 }
* 31 void show() {
* 32 cout << "当前点的坐标为: (" << x << "," << y <
* 33 < ")" << endl;
* 34 }
* 35 };
* 36
* 37 class Point : public Location {
* 38 public:
* 39 int z;
* 40 public:
* 41 Point(int \_x, int \_y, int \_z) : Location(\_x, \_y) {
* 42 z = \_z;
* 43 cout << "+Point类的构造函数被调用" << endl;
* 44 }
* 45 ~Point() {
* 46 cout << "~Point类的析构函数被调用" << endl;
* 47 system("pause");
* 48 }
* 49 void move() {
* 50 int \_x, \_y, \_z;
* 51 cout << "请输入三维坐标的变化情况(
* 52 正数表示增加, 负数表示减少): " << endl;
* 53 cout << " Δx = ";
* 54 cin >> \_x;
* 55 cout << " Δy = ";
* 56 cin >> \_y;
* 57 cout << " Δz = ";
* 58 cin >> \_z;
* 59 x += \_x;
* 60 y += \_y;
* 61 z += \_z;
* 62 cout << "移动成功！" << endl;
* 63 }
* 64 void show() {
* 65 cout << "当前点的坐标为: (" << x << "," << y <
* 66 < "," << z << ")." << endl;
* 67 }
* 68 };
* 69
* 70 class Sphere : public Point {
* 71 public:
* 72 double R;
* 73 public:
* 74 Sphere(int \_x, int \_y, int \_z, double \_r) : Point(\_x, \_y,
* 75 \_z) {
* 76 R = \_r;
* 77 cout << "+Sphere类的构造函数被调用" << endl;
* 78 }
* 79 ~Sphere() {
* 80 cout << "~Sphere类的析构函数被调用" << endl;
* 81 system("pause");
* 82 }
* 83 void move() {
* 84 int \_x, \_y, \_z;
* 85 cout << "请输入球心坐标的变化情况(
* 86 正数表示增加, 负数表示减少): " << endl;
* 87 cout << " Δx = ";
* 88 cin >> \_x;
* 89 cout << " Δy = ";
* 90 cin >> \_y;
* 91 cout << " Δz = ";
* 92 cin >> \_z;
* 93 x += \_x;
* 94 y += \_y;
* 95 z += \_z;
* 96 cout << "移动成功！" << endl;
* 97 }
* 98 void show() {
* 99 cout << "当前球心的坐标为: (" << x << "," <<
* 100 y << "," << z << ").";
* 101 cout << " 球的半径为: " << R << endl;
* 102 }
* 103 };
* 104
* 105 int main() {
* 106 cout << "----------Location----------" << endl;
* 107 Location L1(6, 6);
* 108 L1.show();
* 109 L1.move();
* 110 L1.show();
* 111 cout << "----------------------------" << endl;
* 112
* 113 cout << "-----------Point------------" << endl;
* 114 Point p1(3, 4, 5);
* 115 p1.show();
* 116 p1.move();
* 117 p1.show();
* 118 cout << "----------------------------" << endl;
* 119
* 120 cout << "-----------Sphere-----------" << endl;
* 121 Sphere s1(0, 0, 0, 5);
* 122 s1.show();
* 123 s1.move();
* 124 s1.show();
* 125 cout << "----------------------------" << endl;
* 126
* 127 system("pause");
* 128 return 0;
* 129 }