**第七次实验报告**

1. 程序分析

1. 预期的输出结果

constructing A...

constructing A...

Constructing B...

(1,2)(1,1)(3,4)

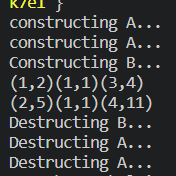
(2,5)(1,1)(4,11)

Destructing B...

Destructing A...

Destructing A...

程序运行输出结果截图



分析：

这段程序定义了两个类A和B，其中B是从A派生而来的私有派生类。类A和B都包含构造函数，析构函数和成员函数add()和display()。类A包含私有数据成员x和y，构造函数初始化x和y，析构函数输出"destructing A..."，成员函数add()对x和y进行加法操作，成员函数display()输出(x,y)的值。类B包含私有数据成员i和j，成员函数add()对i和j进行加法操作，并通过调用A::add()对类A的成员变量x和y进行加法操作。类B还包含A类型的成员变量Aobj，该成员变量在构造函数中初始化，并在display()函数中调用其display()函数输出(1,1)。类B的构造函数在初始化列表中调用了类A的构造函数和Aobj的构造函数，并输出"constructing B..."，析构函数输出"destructing B..."。

在main函数中，创建了一个名为b的类B对象，其构造函数中传递了参数1、2、3和4。然后调用b的display()函数，输出了(1,2)(1,1)(3,4)。然后调用b的add()函数，传递了参数1、3、5和7。最后再次调用b的display()函数，输出了(2,5)(1,1)(4,11)。

总的来说，这个程序演示了如何创建和使用类A和类B，以及如何使用类的构造函数和析构函数。它还演示了如何在类B中调用类A的成员函数和访问类A的私有数据成员，以及如何创建和使用类A的对象作为类B的成员变量。

1. 预期的输出结果

Constructing A...9

Constructing A...9

Constructing A...

Constructing C...

contructing D...

9 9 11

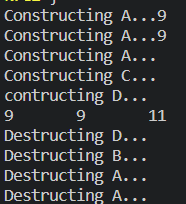
Destructing D...

Destructing B...

Destructing A...

Destructing A...

程序运行输出结果截图



分析

这段代码定义了四个类 A、B、C 和 D。类 A 有一个私有成员变量 x 和公有成员函数 fun()，构造函数和析构函数。类 B 是 A 的公有派生类，同时 B 也有一个私有成员变量 y 和一个 A 类型的对象 Aobj，构造函数和析构函数。类 C 是 B 的公有派生类，构造函数和析构函数。类 D 是 C 和 A 的公有派生类，构造函数和析构函数。

在 main() 函数中，创建了一个 D 类型的对象 d，并调用了其成员函数 fun() 和 display()。

在类的继承中，如果需要使用多继承，要注意虚继承（virtual inheritance），避免出现菱形继承（diamond inheritance）问题。

根据代码的输出，可以得出以下结论：

在创建 D 类型的对象 d 时，首先创建了 A 类型的对象，然后创建了 B 类型的对象，再创建了 C 类型的对象，最后创建了 D 类型的对象。在调用成员函数 fun() 时，先调用了 A 类型的对象的 fun() 函数，再调用了 Aobj 的 fun() 函数，最后返回了 y+1 的值。在调用成员函数 display() 时，先输出了 A 类型的对象的 fun() 函数的返回值，然后输出了 Aobj 的 fun() 函数的返回值，最后输出了 y+1 的值。在程序结束时，先销毁 D 类型的对象，然后销毁 C 类型的对象，再销毁 B 类型的对象，最后销毁 A 类型的对象和 Aobj 对象。

(3) 预期的输出结果

-------------------------

-------------------------

Deri::g1()...2

Base::g2()...3

Base::g3()...4

Base::g4()...5

-------------------------

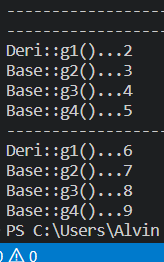
Deri::g1()...6

Base::g2()...7

Base::g3()...8

Base::g4()...9

程序运行输出结果截图



分析

这段代码主要定义了两个类 Base 和 Derive，其中 Derive 继承自 Base。代码中还定义了一些成员函数和变量，以及一个 main() 函数。

Base 类定义了一个构造函数 Base(int m) 和四个成员函数 g1()、g2()、g3() 和 g4()，其中 g1()、g2() 和 g4() 都输出一些信息，g3() 则只输出信息。Base 类中还有一个整型成员变量 n，用于存储传递给构造函数的参数 m 和在 g1()、g2() 和 g3() 中递增的值。

Derive 类定义了一个构造函数 Derive(int n1, int n2) 和两个成员函数 g1() 和 g3()，其中 g1() 和 g3() 分别覆盖了 Base 类中的同名函数，并且在其中调用了基类 Base 中的一些成员函数。Derive 类中还有一个整型成员变量 j，用于存储传递给构造函数的参数 n2。

在 main() 函数中，首先创建了一个 Derive 类对象 Dobj，并将其赋值给了一个 Base 类对象 Bobj，这是一个向上转型的例子。然后定义了一个指向 Base 类的指针 bp，并将其指向 Dobj 对象。最后定义了一个 Base 类的引用 bref，并将其绑定到 Dobj 对象上。

第一个分割线上下没有输出信息。第二个分割线上下也没有输出信息。在第三个分割线之前，调用了 bref.g1()，输出了一些信息，其中 Deri::g1()... 表示调用了 Derive 类中的 g1() 函数，++n 表示将 Base 类中的 n 值递增，并输出递增后的值，最后调用了 g4() 函数。在第三个分割线之后，调用了 Dobj.g1()，输出了一些信息，其中 Deri::g1()... 表示调用了 Derive 类中的 g1() 函数，++n 表示将 Base 类中的 n 值递增，并输出递增后的值，然后调用了 Base 类中的 g2() 函数，输出了一些信息，其中 Base::g2()... 表示调用了 Base 类中的 g2() 函数，++n 表示将 Base 类中的 n 值再次递增，并输出递增后的值，最后调用了 Base 类中的 g3() 和 g4() 函数，依次输出了一些信息。

(4) 预期的输出结果

A::A() called.

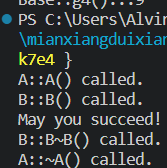
B::B() called.

May you succeed!

B::B~B() called.

A::~A() called.

程序运行输出结果截图



分析

这段代码主要演示了虚析构函数的作用和向上转型的概念。

首先，类 A 和类 B 之间存在继承关系，其中类 A 声明了虚析构函数，而类 B 在其析构函数中调用了 delete 来释放在构造函数中申请的内存空间。

在 main() 函数中，首先通过向上转型创建了一个 B 类对象，并将其地址赋值给了 A 类指针 p。然后，调用了 fun() 函数，并将 p 作为参数传递给了该函数。在 fun() 函数中，先输出了一行字符串 "May you succeed!"，然后调用了 delete 来释放 p 指向的内存空间，这将会调用 B 类的析构函数。

由于在类 A 中声明了虚析构函数，在删除 B 类对象时，将首先调用 B 类的析构函数，再调用 A 类的析构函数。因此，运行结果将会输出以下内容：

可以看到，程序先输出了 A::A() called. 和 B::B() called.，这是由于在创建 B 类对象时，先调用了 A 类的构造函数，再调用了 B 类的构造函数。然后，输出了 May you succeed!，这是 fun() 函数中的输出。接着，输出了 B::~B() called. 和 A::~A() called.，这是由于在删除 B 类对象时，先调用了 B 类的析构函数，再调用了 A 类的析构函数。

2. 源代码

#include <iostream>

#include <string>

using namespace std;

class Employee{

public:

    Employee(string name, int id, int level, double baseSalary):

        name(name), id(id), level(level), baseSalary(baseSalary){};

    virtual void calcSalary() = 0; // 纯虚函数

    virtual void promote() = 0; // 纯虚函数

    void display(){

        cout << "姓名：" << name << "，编号：" << id << "，级别：" << level

             << "，月薪：" << salary << endl;

    }

protected:

    string name;

    int id;

    int level;

    double baseSalary;

    double salary;

};

class Manager: public Employee{

public:

    Manager(string name, int id, int level, double baseSalary):

        Employee(name, id, level, baseSalary){};

    void calcSalary(){

        salary = 8000;

    }

    void promote(){

        level = 4;

    }

};

class Technician: public Employee{

public:

    Technician(string name, int id, int level, double baseSalary, int hours):

        Employee(name, id, level, baseSalary), hours(hours){};

    void calcSalary(){

        salary = hours \* 100;

    }

    void promote(){

        level = 3;

    }

private:

    int hours;

};

class Salesman: public Employee{

public:

    Salesman(string name, int id, int level, double baseSalary, double sales):

        Employee(name, id, level, baseSalary), sales(sales){};

    void calcSalary(){

        salary = baseSalary + sales \* 0.04;

    }

    void promote(){

        // 销售员不升级

    }

private:

    double sales;

};

class SalesManager: public Employee{

public:

    SalesManager(string name, int id, int level, double baseSalary, double sales, double commission):

        Employee(name, id, level, baseSalary), sales(sales), commission(commission){};

    void calcSalary(){

        salary = baseSalary + sales \* commission \* 0.001;

    }

    void promote(){

        level = 3;

    }

private:

    double sales;

    double commission;

};

int main(){

    Manager m("张三", 1001, 1, 8000);

    m.promote();

    m.calcSalary();

    m.display();

    Technician t("李四", 1002, 1, 100, 80);

    t.promote();

    t.calcSalary();

    t.display();

    Salesman s("王五", 1003, 1, 3000, 20000);

    s.promote();

    s.calcSalary();

    s.display();

    SalesManager sm("赵六", 1004, 1, 5000, 100000, 5);

    sm.promote();

    sm.calcSalary();

    sm.display();

    return 0;

}

---

运行结果截图

