广州大学学生实验报告

2018年4月7日

**开课学院及实验室：**计算机科学与工程实验室电子楼412A室

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | 计算机 | **专业班级** | 软件171 | **姓名** | 谢金宏 | **学号** | 1706300001 |
| **实验课程名称** | | 面向对象程序设计实验 | | **成绩** | |  | |
| **实验项目名称** | | 继承与多态性 | | **指导老师** | | 陶文正 | |

# 实验目的及要求

1. 掌握继承、基类和派生类的概念。
2. 掌握初始化基类成员的方法。
3. 掌握派生类对基类的继承。
4. 学习虚函数和纯虚函数的定义与使用方式。
5. 理解抽象类的概念，学习如何用指针指向其他的派生类，实现多态性。
6. 掌握抽象类的定义与使用方式，并注意指针的用法。
7. 学习如何使用虚函数、纯虚函数、抽象类和实现类的多态性。

# 实验设备及平台

1. 硬件环境：计算机
2. 软件环境：G++ 7.2.0, Visual Studio Code

# 实验内容及步骤

1. **定义派生类的实例**

#include <string>

#include <iostream>

using namespace std;

class Student {

public:

Student() = default;

Student(string serial, string name, int sex, int year, int month, int day) :

serial\_(serial), name\_(name), sex\_(sex),

year\_(year), month\_(month), day\_(day) {};

void set\_serial(string serial) { serial\_ = serial; }

void set\_name(string name) { name\_ = name; }

void set\_sex(int sex) { sex\_ = sex; }

void set\_year(int year) { year\_ = year; }

void set\_month(int month) { month\_ = month; }

void set\_day(int day) { day\_ = day; }

string serial() { return serial\_; }

string name() { return name\_; }

int sex() { return sex\_; }

int year() { return year\_; }

int month() { return month\_; }

int day() { return day\_; }

protected:

string serial\_, name\_;

int sex\_, year\_, month\_, day\_;

};

class ExtendedStudent: public Student {

public:

ExtendedStudent() = default;

ExtendedStudent(

string serial, string name, int sex, int year,

int month, int day, int C, int English

) : Student(serial, name, sex, year, month, day) {

C\_ = C, English\_ = English;

}

void set\_C(int C) { C\_ = C; }

void set\_English(int English) { English\_ = English; }

int C() { return C\_; }

int English() { return English\_; }

void List() {

cout << "学号：" << serial() << endl;

cout << "姓名：" << name() << endl;

cout << "性别：" << (sex() == 0 ? "女" : "男") << endl;

cout << "出生日期：" << year() << "年" << day() << "月" << day() << "日" << endl;

cout << "C++：" << C() << endl;

cout << "英语：" << English() << endl;

}

protected:

int C\_, English\_;

};

template <typename T>

void input(T &t, const string hint)

{

cout << hint; cin >> t;

}

int main() {

int n;

cout << "请输入学生人数：" << endl;

cin >> n;

ExtendedStudent \*s = new ExtendedStudent[n];

for (int i=0; i<n; ++i) {

cout << "正在输入第" << i+1 << "位学生" << endl;

string serial, name;

int sex, year, month, day, C, English;

input(serial, string("学号："));

input(name, string("姓名："));

input(sex, string("性别（非零数字表示男性）："));

cout << "出生年月日：" << endl;

cin >> year >> month >> day;

input(C, string("C语言成绩："));

input(English, string("英语成绩："));

s[i] = ExtendedStudent(serial, name, sex, year, month, day, C, English);

}

cout << endl;

cout << "下面列出已输入学生的信息：" << endl;

for (int i=0; i<n; ++i) {

if (i) cout << endl;

cout << "第" << i+1 << "位学生" << endl;

s[i].List();

}

delete[] s;

}

以上程序提醒输入学生的数量和对应数量学生的信息；输入完毕后将列出输入学生的信息。程序使用了面向对象编程中的继承特性。

1. **多重继承的实例**

#include <iostream>

using namespace std;

class Base1 {

public:

    void show\_i() { cout << i << endl; }

protected:

    int i;

};

class Base2 {

public:

    void show\_j() { cout << j << endl; }

protected:

    int j;

};

class Derived : public Base1, public Base2 {

public:

    void set(int x, int y) { i = x, j = y; }

};

int main() {

    Derived obj;

    obj.set(5, 7);

    obj.show\_i();

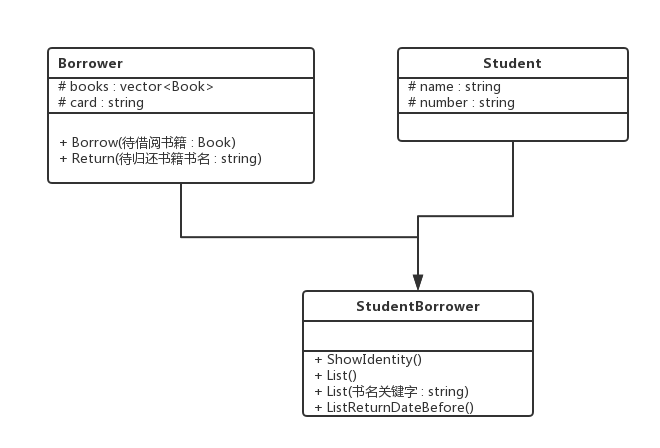
    obj.show\_j();

}

以上程序是多重继承的一个例子。

1. **抽象并编程现实生活中图书馆借阅的相关内容**

设有“借阅者”类，主要功能是记录借书证号以及借阅书籍，并记录借还书的日期等；“学生”类，记录学生的姓名和学号；设计“学生借阅者”类，它是“借阅者”类和“学生”类的派生类。在此基础上设计函数，显示借阅者的相关信息。



#include <algorithm>

#include <numeric>

#include <iostream>

#include <iomanip>

#include <string>

#include <sstream>

#include <vector>

using namespace std;

// Date结构 - 表示特定日期

// 构造器：

// Date(int day, int month, int year) - 以指定的年月日构造日期

// 方法：

// Show() - 从标准输出流输出日期

// toString() - 返回代表当前日期的字符串

// 重载等于号操作符以及小于号操作符

struct Date {

static const int kDaysInMonth[13];

    int day, month, year;

Date() {

year = 2018;

month = rand()%12+1;

day = rand()%kDaysInMonth[month]+1;

}

Date(int day, int month, int year) {

month = max(1, month), month = min(12, month); // 保证日期合法，下同

day = max(1, day), day = min(kDaysInMonth[month], day);

this->year = year, this->month = month, this->day = day;

}

    void Show() {

        cout.fill('0');

        cout << setw(4) << year << '-' << setw(2) << month << '-' << setw(2) << day;

    }

string toString() {

stringstream ss; ss.fill('0');

ss << setw(4) << year << '-' << setw(2) << month << '-' << setw(2) << day;

string ret; ss >> ret;

return ret;

}

bool operator == (const Date d) const {

return (day == d.day) && (month == d.month) && (year == d.year);

}

bool operator < (const Date d) const {

if (year != d.year) return year < d.year;

if (month != d.month) return month < d.month;

return day < d.day;

}

bool operator <= (const Date d) const {

return \*this < d || \*this == d;

}

};

const int Date::kDaysInMonth[13] = {

0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,

};

// Book结构 - 表示借阅的特定书籍

// 数据成员：

// name - 书籍名称

// borrow\_date, return\_date - 借出日期和应还日期

struct Book {

    string name;

    Date borrow\_date, return\_date;

};

// Borrower类 - 借阅者

// 数据成员：

// card - 借书证号

// books - 借阅的书籍

// 方法：

// Borrow(Book&) - 借阅给出的书籍

// Return(string) - 归还指定名称的书籍

class Borrower {

public:

void Borrow(const Book &);

bool Return(const string);

protected:

string card\_;

vector<Book> books\_;

};

void Borrower::Borrow(const Book &book) {

books\_.push\_back(book);

}

bool Borrower::Return(const string name) {

bool already\_returned = false;

for (auto it=books\_.begin(); it!=books\_.end(); ++it)

{

if (it->name == name) {

books\_.erase(it);

already\_returned = true;

}

}

return already\_returned;

}

// Student类 - 学生

// 数据成员：

// name - 姓名

// number - 学号

class Student {

public:

Student(string name, string number) {

name\_ = name, number\_ = number;

}

protected:

string name\_;

string number\_;

};

// StudentBorrower - 学生借阅者

// 继承自Borrower, Student

// 方法：

// ShowIdentity() - 显示借阅者的身份信息

// List() - 列出借阅的所有书籍

// List(string contain\_string) - 列出借阅书籍中书名包含contain\_string的书籍

// ListReturnDateBefore(Date date) - 列出借阅书籍中归还日期在指定日期之前的书籍

class StudentBorrower : public Borrower, public Student {

public:

StudentBorrower(string name, string number, string card) : Student(name, number) {

card\_ = card;

}

void ShowIdentity();

void List();

void List(string book\_name);

void ListReturnDateBefore(Date);

private:

void List(const vector<unsigned>&);

};

void StudentBorrower::ShowIdentity() {

cout << "借阅者：" << name\_ << " (" << number\_ << ")" << endl;

cout << "借阅证号：" << card\_ << endl;

}

void StudentBorrower::List(const vector<unsigned> &idx) {

unsigned len = 0;

for (int i : idx) {

len = max(unsigned(books\_[i].name.size()), len);

} // 统计待列出的书籍中最长书名的长度

cout << "#" << " | " << "标题" << " | " << "借书日期" << " | " << "应还日期" << endl;

for (unsigned i=0; i<idx.size(); ++i) {

cout.fill('0'); cout << right << setw(2) << i+1 << " | ";

cout.fill(' '); cout << left << setw(len) << books\_[idx[i]].name << " | ";

cout << books\_[idx[i]].borrow\_date.toString() << " | ";

cout << books\_[idx[i]].return\_date.toString() << endl;

}

}

void StudentBorrower::List() {

if (!books\_.size()) return;

vector<unsigned> idx(books\_.size());

iota(idx.begin(), idx.end(), 0);

List(idx);

}

void StudentBorrower::List(string contain\_string) {

vector<unsigned> idx;

for (unsigned i=0; i<books\_.size(); ++i) {

if (books\_[i].name.find(contain\_string) != string::npos) idx.push\_back(i);

}

if (idx.size()) List(idx);

else cout << "借阅的书籍中书名没有包含“" << contain\_string << "”的书籍" << endl;

}

void StudentBorrower::ListReturnDateBefore(Date date) {

vector<unsigned> idx;

for (unsigned i=0; i<books\_.size(); ++i) {

if (books\_[i].return\_date < date) idx.push\_back(i);

}

if (idx.size()) List(idx);

else cout << "借阅的书籍中没有应还日期在" << date.toString() << "之前的书籍" << endl;

}

int main()

{

// 随机生成书籍信息

Book books[10] {

{"钢铁是怎样炼成的", Date(), Date()},

{"青年近卫军", Date(), Date()},

{"大卫科波菲尔", Date(), Date()},

{"活着", Date(), Date()},

{"人性的弱点", Date(), Date()},

{"静静的顿河", Date(), Date()},

{"黄金时代", Date(), Date()},

{"再见，冥王星", Date(), Date()},

{"你好，乔安", Date(), Date()},

{"CLR via C#", Date(), Date()},

};

// 模拟借书过程

StudentBorrower borrower("Lightyears", "1706300001", "GZDX-1706300001");

borrower.ShowIdentity();

for (int i=0; i<10; ++i) {

borrower.Borrow(books[i]);

}

cout << endl;

cout << "列出借阅的所有书籍：" << endl;

borrower.List();

cout << endl;

cout << "列出借阅书籍中书名包含“的”字的书籍：" << endl;

borrower.List("的");

cout << endl;

cout << "列出借阅书籍中还书日期在10月30日之前的书籍：" << endl;

borrower.ListReturnDateBefore(Date(30, 10, 2018));

}

以上程序模拟借阅者从图书馆借书的过程，并根据预设指令打印借阅者的相关信息。程序产生的输出示例如下：

借阅者：Lightyears (1706300001)

借阅证号：GZDX-1706300001

列出借阅的所有书籍：

# | 标题 | 借书日期 | 应还日期

01 | 钢铁是怎样炼成的 | 2018-06-18 | 2018-11-11

02 | 青年近卫军 | 2018-06-05 | 2018-07-02

03 | 大卫科波菲尔 | 2018-11-15 | 2018-06-06

04 | 活着 | 2018-02-28 | 2018-02-16

05 | 人性的弱点 | 2018-08-08 | 2018-04-07

06 | 静静的顿河 | 2018-04-25 | 2018-03-30

07 | 黄金时代 | 2018-05-14 | 2018-10-24

08 | 再见，冥王星 | 2018-03-25 | 2018-12-27

09 | 你好，乔安 | 2018-12-07 | 2018-10-11

10 | CLR via C# | 2018-12-12 | 2018-08-06

列出借阅书籍中书名包含“的”字的书籍：

# | 标题 | 借书日期 | 应还日期

01 | 钢铁是怎样炼成的 | 2018-06-18 | 2018-11-11

02 | 人性的弱点 | 2018-08-08 | 2018-04-07

03 | 静静的顿河 | 2018-04-25 | 2018-03-30

列出借阅书籍中还书日期在10月30日之前的书籍：

# | 标题 | 借书日期 | 应还日期

01 | 青年近卫军 | 2018-06-05 | 2018-07-02

02 | 大卫科波菲尔 | 2018-11-15 | 2018-06-06

03 | 活着 | 2018-02-28 | 2018-02-16

04 | 人性的弱点 | 2018-08-08 | 2018-04-07

05 | 静静的顿河 | 2018-04-25 | 2018-03-30

06 | 黄金时代 | 2018-05-14 | 2018-10-24

07 | 你好，乔安 | 2018-12-07 | 2018-10-11

08 | CLR via C# | 2018-12-12 | 2018-08-06

1. 虚函数的简单使用

#include <iostream>

#include <exception>

using namespace std;

class Base {

public:

    virtual void func() { cout << "Invoke Base function.\n" ; }

};

class Child : public Base {

public:

    virtual void func() { cout << "Invoke Child function.\n" ; }

};

void invoke(Base &b) { b.func(); }

int main() {

    Base base; Child child;

    cout << "Passing Base class reference: "; invoke(base);

    cout << "Passing Child class reference: "; invoke(child);

    cout << "Passing casted Child class reference: "; invoke((Base&)child);

    cout << "Passing casted Child class reference: "; invoke(dynamic\_cast<Base&>(child));

}

程序输出：

Passing Base class reference: Invoke Base function.

Passing Child class reference: Invoke Child function.

Passing casted Child class reference: Invoke Child function.

Passing casted Child class reference: Invoke Child function.

从上述实验结果不难发现，在使用引用调用类对象的虚成员函数时，被调用的成员函数取决于对象的实际类型（而不是所使用引用的类型）。

1. 使用实现菜单程序

#include <iostream>

using namespace std;

class Menu {

public:

    virtual void action() = 0;

};

class Item1: public Menu {

public:

    virtual void action() { cout << "新建文件" << endl; }

};

class Item2 : public Menu {

public:

    virtual void action() { cout << "打开文件" << endl; }

};

class Item3 : public Menu {

public:

    virtual void action() { cout << "保存文件" << endl; }

};

class Item4 : public Menu {

public:

    virtual void action() { cout << "关闭文件" << endl; }

};

int main()

{

    Menu \*Fptr[4] {new Item1(), new Item2(), new Item3(), new Item4()};

    int select;

    do {

        cout << "1| New File" << endl;

        cout << "2| Open File" << endl;

        cout << "3| Save File" << endl;

        cout << "4| Close File" << endl;

        cout << "0| Exit" << endl;

        cin >> select;

        if (select>=1 && select<=4) Fptr[select-1]->action();

    } while (select != 0 && select != EOF);

}

通过虚函数机制在不同派生类实现不同功能，并通过父类指针数组统一管理派生类对象是一种实用的编程方式。

1. 定义一个从正方体、球体和圆柱体中抽象出来的公共基类Geometry类，在其中定义求表面积和体积的纯虚函数，在抽象类中定义一个公共成员radius，此数据可作为球体的半径、正方体的边长以及圆柱体底面的半径。
   1. 定义派生正方体类Cube
   2. 定义派生球体类Sphere
   3. 定义派生圆柱体类Cylinder

#include <iostream>

#include <cmath>

using namespace std;

const double PI = acos(-1);

class Geometry {

public:

    virtual double SurfaceArea() = 0;

    virtual double Volume() = 0;

protected:

    double radius\_;

};

class Cube : public Geometry {

public:

    Cube(double radius) {

        radius\_ = radius;

    }

    virtual double SurfaceArea() {

        return radius\_\*radius\_\*6;

    }

    virtual double Volume() {

        return radius\_\*radius\_\*radius\_;

    }

};

class Sphere : public Geometry {

public:

    Sphere(double radius, double height) {

        radius\_ = radius, height\_ = height;

    }

    virtual double SurfaceArea() {

        return PI\*radius\_\*radius\_\*4;

    }

    virtual double Volume() {

        return PI\*radius\_\*radius\_\*radius\_\*4/3;

    }

private:

    double height\_;

};

class Cylinder : public Geometry {

public:

    Cylinder(double radius, double height) {

        radius\_ = radius, height\_ = height;

    }

    virtual double SurfaceArea() {

        return 2\*PI\*radius\_\*radius\_ + 2\*PI\*radius\_\*height\_;

    }

    virtual double Volume() {

        return PI\*radius\_\*radius\_\*height\_;

    }

private:

    double height\_;

};

int main()

{

    Cube cube(1);

    Sphere sphere(1, 1);

    Cylinder cylinder(1, 1);

    cout << "Type | SurfaceArea | Volume" << endl;

    cout << "Cube: " << cube.SurfaceArea() << ' ' << cube.Volume() << endl;

    cout << "Sphere: " << sphere.SurfaceArea() << ' ' << sphere.Volume() << endl;

    cout << "Cylinder: " << cylinder.SurfaceArea() << ' ' << cylinder.Volume() << endl;

}

1. 编写人员信息管理系统，实现交互式的校园人员信息录入和显示。

学校里主要有四类人员：本科学生，教师、研究生和助教。

大学本科生每周有固定的学时数。教师除了固定的学时数外，还有每周的教学时数。研究生除了固定的学时数外，每周还可以自由做一定的研究。助教除了上课以外，还要做研究和一定的教学工作。人员的基本信息包括姓名、编号、性别、身份证号、总学时数以及每周固定学时数。

各类人员之间的关系如下：People类派生出Student类和Teacher类，Student类派生出Graduate类，Graduate类派生出TA类。

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class People {

public:

    void Read();

    virtual void Print();

    virtual ~People() = default;

protected:

    bool ismale\_;

    string name\_, serial\_;

    string id\_number\_;

    int total\_learning\_hours\_, fixed\_learning\_hours\_;

};

void People::Print() {

    cout << "姓名：" << name\_ << " (" << serial\_ <<")" << endl;

    cout << "性别：" << (ismale\_ ? "男" : "女") << endl;

    cout << "总学时：" << total\_learning\_hours\_ << endl;

    cout << "身份证号：" << id\_number\_ << endl;

    cout << "固定学时：" << fixed\_learning\_hours\_ << endl;

    cout << "合计学时：" << total\_learning\_hours\_ << endl;

}

template<typename T>

void read(T &t, const string &hint) {

    cout << hint << "> "; cin >> t;

    cin.clear(); while (getchar() != '\n') continue;

}

void People::Read() {

    read(name\_, "姓名");

    char sex; read(sex, "性别(M/F)");

    ismale\_ = (sex=='M');

    read(serial\_, "编号");

    read(id\_number\_, "身份证号");

    read(total\_learning\_hours\_, "总学时");

    read(fixed\_learning\_hours\_, "固定学时");

}

class Student : virtual public People {

public:

    void Read();

    virtual void Print();

    virtual ~Student() = default;

};

void Student::Read() {

    cout << "请录入学生信息" << endl;

    People::Read();

}

void Student::Print() {

    cout << "类别 - 本校学生" << endl;

    People::Print();

}

class Teacher : virtual public People {

public:

    virtual ~Teacher() = default;

    void Read();

    void Print();

protected:

    int total\_teaching\_hours\_;

};

void Teacher::Read() {

    cout << "请录入教师信息" << endl;

    People::Read();

    read(total\_teaching\_hours\_, "教学时长");

}

void Teacher::Print()

{

    cout << "类别 - 本校教师" << endl;

    People::Print();

    cout << "教学时数：" << total\_teaching\_hours\_ << endl;

}

class Graduate : virtual public Student {

public:

    ~Graduate() = default;

    void Read();

    virtual void Print();

protected:

    int free\_reserching\_hours\_;

};

void Graduate::Read() {

    cout << "请录入研究生信息" << endl;

    People::Read();

    read(free\_reserching\_hours\_, "研究时长");

}

void Graduate::Print()

{

    cout << "类别 - 研究生" << endl;

    People::Print();

    cout << "研究时长：" << free\_reserching\_hours\_ << endl;

}

class TeacherAssistant : virtual public Graduate, virtual public Teacher {

public:

    ~TeacherAssistant() = default;

    void Read();

    virtual void Print();

};

void TeacherAssistant::Read() {

    cout << "请录入助教信息" << endl;

    People::Read();

    read(total\_teaching\_hours\_, "教学时长");

    read(free\_reserching\_hours\_, "研究时长");

}

void TeacherAssistant::Print()

{

    cout << "类别 - 助教" << endl;

    People::Print();

    cout << "研究时数" << free\_reserching\_hours\_ << endl;

    cout << "助教时数" << total\_teaching\_hours\_ << endl;

}

void ShowMenu() {

    cout << "==============" << endl;

    cout << "1| 录入学生信息" << endl;

    cout << "2| 录入教师信息" << endl;

    cout << "3| 录入研究生信息" << endl;

    cout << "4| 录入助教信息" << endl;

    cout << "5| 显示已登记人员" << endl;

    cout << "0| 退出系统" << endl;

}

template<typename T>

void NewMember(vector<People\*> &storage)

{

    T \* ptr = new T;

    (\*ptr).Read();

    storage.push\_back(dynamic\_cast<People\*>(ptr));

}

void ShowMember(vector<People\*> &storage)

{

    for (People \* ptr : storage) {

        cout << "==============" << endl;

        ptr->Print();

    }

}

void Clear(vector<People\*> &storage) {

    for (People \* ptr : storage) {

        delete ptr;

    }

    storage.clear();

}

int main()

{

    cout << "交互式人员信息管理系统 v0.0" << endl;

    vector<People\*> storage;

    int selection;

    while (true) {

        ShowMenu(); read(selection, "请选择操作");

        switch (selection) {

            case 0: Clear(storage); return 0;

            case 1: NewMember<Student>(storage); break;

            case 2: NewMember<Teacher>(storage); break;

            case 3: NewMember<Graduate>(storage); break;

            case 4: NewMember<TeacherAssistant>(storage); break;

            case 5: ShowMember(storage); break;

        }

    }

}

上述程序利用C++虚函数多态的特性，基本完成了题目的要求。

# 思考问题及课后练习

1. 编写一个程序，定义一个汽车类Vehicle，它具有一个需传递参数的构造函数，类中的数据成员包括车轮个数和车的重量，并放到保护段中；定义轿车类Car是汽车类Vehicle的私有派生类，其中包含载人数；再定义卡车类Truck是汽车类Vehicle的私有派生类，其中包含载人数和载重量。每个类都有相应的数据输出。

#include <iostream>

using namespace std;

class Vehicle

{

public:

    Vehicle(int tire\_number, int vehicle\_weight)

    {

        tire\_number\_ = tire\_number, vehicle\_weight\_ = vehicle\_weight;

    }

    virtual ~Vehicle() {}

    virtual void speak() {

        cout << "This vehicle has " << tire\_number\_ << " tires, it weights " << vehicle\_weight\_ << "kg.\n";

    }

protected:

    int tire\_number\_;

    int vehicle\_weight\_;

};

class Car : Vehicle

{

public:

    Car(int tire\_number, int vehicle\_weight, int people\_number)

        : Vehicle(tire\_number, vehicle\_weight) {

        people\_number\_ = people\_number;

    }

    virtual ~Car() {}

    virtual void speak() {

        cout << "This car has " << tire\_number\_ << " tires, it weights " << vehicle\_weight\_ << "kg. ";

        cout << "There " << (people\_number\_>1 ? "are " : "is ") << people\_number\_ << " people on board.\n";

    }

private:

    int people\_number\_;

};

class Truck : Vehicle

{

public:

    Truck(int tire\_number, int vehicle\_weight, int people\_number, int load\_weight)

        : Vehicle(tire\_number, vehicle\_weight) {

        people\_number\_ = people\_number, load\_weight\_ = load\_weight;

    }

    virtual ~Truck() {}

    virtual void speak() {

        cout << "This truck has " << tire\_number\_ << " tires, it weights " << vehicle\_weight\_ << "kg. ";

        cout << "There " << (people\_number\_>1 ? "are " : "is ") << people\_number\_ << " people on board, ";

        cout << "and it carrys " << load\_weight\_ << "kg load.\n";

    }

private:

    int people\_number\_;

    int load\_weight\_;

};

int main()

{

    Vehicle vehicle(4, 10);

    Car car(4, 10, 2);

    Truck truck(12, 20, 1, 50);

    vehicle.speak();

    car.speak();

    truck.speak();

}

程序运行结果如下：

This vehicle has 4 tires, it weights 10kg.

This car has 4 tires, it weights 10kg. There are 2 people on board.

This truck has 12 tires, it weights 20kg. There is 1 people on board, and it carrys 50kg load.

1. 设有系类，它的数据包括系编号、系名，功能包括置数据、修改数据和取数据。设有班类，它继承系类，另数据包括学号、姓名、性别，功能包括置数据、修改数据和取数据。主函数默认设置一位同学数据，键盘输入一位同学数据，然后显示二位同学数据。

#include <iostream>

#include <string>

using namespace std;

class Xi

{

public:

    virtual ~Xi() {};

    string serial() { return serial\_; }

    string name() { return name\_; }

    void set\_serial(const string &serial) { serial\_ = serial; }

    void set\_name(const string &name) { name\_ = name; }

protected:

    string serial\_, name\_;

};

class Ban : public Xi

{

public:

    virtual ~Ban() {};

    string serial() { return serial\_; }

    string name() { return name\_; }

    bool ismale() { return ismale\_; }

    void set\_serial(const string &serial) { serial\_ = serial; }

    void set\_name(const string &name) { name\_ = name; }

    void set\_ismale(bool ismale) { ismale\_ = ismale; }

private:

    string serial\_, name\_;

    bool ismale\_;

};

template<typename T>

void input(T &val, const string &hint)

{

    cout << hint; cin >> val;

    cin.clear(); while (getchar() != '\n') continue;

}

template<> void input(string &val, const string &hint)

{

    cout << hint; getline(cin, val);

}

void read(Ban &info) // 从键盘录入学生信息

{

    string name, serial, name\_of\_school, serial\_of\_school;

    bool ismale;

    cout << "Please specify student information: " << endl;

    input(name, "Name: ");

    input(serial, "Serial: ");

    input(ismale, "Gender(0 for female, other number for male): ");

    input(name\_of\_school, "Name of School: ");

    input(serial\_of\_school, "Serial of School: ");

    cout << "Done. Thank you." << endl << endl;

    info.set\_name(name);

    info.set\_serial(serial);

    info.set\_ismale(ismale);

    info.Xi::set\_name(name\_of\_school);

    info.Xi::set\_serial(serial\_of\_school);

}

void print(Ban &info) // 向屏幕输出

{

    cout << "Name: " << info.Ban::name() << ' ' << (info.ismale() ? "(Male)" : "(Female)") << endl;

    cout << "Serial: " << info.Ban::serial() << endl;

    cout << "School: " << info.Xi::name() << '(' << info.Xi::serial() << ')' << endl;

    cout << "==========" << endl;

}

int main()

{

    Ban info1, info2;

    info1.Xi::set\_serial("1706300");

    info1.Xi::set\_name("School of Computer Science");

    info1.Ban::set\_serial("001");

    info1.Ban::set\_name("Xie jinhong");

    info1.Ban::set\_ismale(true);

    read(info2);

    print(info1); print(info2);

}

程序的一个运行结果如下：

Name: Xie Jinhong (Male)

Serial: 001

School: School of Computer Science(1706300)

==========

Name: Bu Cunzai (Female)

Serial: 999

School: School of Computer Science(1706300)

# 总结

继承是C++面向对象编程中的最重要的特性之一。个人认为如果没有继承，面向对象编程也不过是把面向过程编程的思维包装在对象里而已。

C++中的继承分三种：公有继承、保护继承和私有继承。其中保护继承和私有继承子在实际的开发中是较少使用的。从面向对象的观点看，公有继承建立了一种is-a关系，即派生类是基类的特化。私有继承建立了一种has-a关系，表示派生类持有基类。实际上，has-a关系可以通过类对象成员变量的方式来表达，并且成员变量的表达方式更加清晰和优雅，因此私有继承极少被使用。保护继承可以理解为一种语法糖，适用于特定条件下的开发工作。在良好抽象的面向对象编程中，私有继承和保护继承极少使用。可以认为，一般谈及“继承”时，是指“公有继承”。

C++的运行时多态性部分建立在继承的基础上。通过虚函数特性，程序员可以通过基类指针或引用操纵派生类对象，在统一接口的前提下使派生类表现出自有的特性。

C++是允许多重继承的语言，一个子类可以派生自多个父类。这种继承方式为程序员提供便利的同时也带来了一定的问题。在子类继承多个父类，多个父类继承自一个祖类的情况下，程序员主要面临两个问题？“哪个祖类？”和“哪个方法？”，由此引入虚继承和作用域解析操作符等相关编程特性。在抽象良好的面向对象编程中，多重继承应该较少使用或者不被使用。