# 实验六 继承与派生

## 题目1 图形类

### 实验描述：

设计一个图形基类：Shape，在此基础上派生Rectangle、Circle、Triangle 类。

1、Cirlce 类基本信息：圆心坐标、半径；

Rectangle 类基本信息：长、宽；

Triangle 类基本信息：三个顶点坐标；

其中：成员变量为 private 属性，成员函数为public 属性；

2、每个图形类有多个构造函数：缺省构造函数、带参数的构造函数；

3、每个图形类有计算图形的面积GetArea()，显示图形的基本信息函数Show()，修改基

本信息的函数Set(形参)。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\6\1picPlus.cpp

 \* @Description: 设计一个图形基类：Shape，在此基础上派生Rectangle、Circle、Triangle 类。

1、Cirlce 类基本信息：圆心坐标、半径；

Rectangle 类基本信息：长、宽；

Triangle 类基本信息：三个顶点坐标；

其中：成员变量为 private 属性，成员函数为public 属性；

2、每个图形类有多个构造函数：缺省构造函数、带参数的构造函数；

3、每个图形类有计算图形的面积GetArea()，显示图形的基本信息函数Show()，修改基

本信息的函数Set(形参)。

 \*/

#include <iostream>

#include <cmath>

using namespace std;

#define M\_PI 3.14159265358979323846 // 定义圆周率常量

class Shape // 形状基类

{

protected:

    double area; // 面积

public:

    Shape() {}                             // 默认构造函数

    virtual double GetArea() { return 0; } // 虚函数，计算面积

    virtual void Show() {}                 // 虚函数，显示形状信息

};

class Circle : public Shape // 圆形派生类，继承自Shape

{

private:

    double x, y, r; // 圆心坐标和半径

public:

    Circle() : x(0), y(0), r(0) {}                                   // 默认构造函数

    Circle(double x\_, double y\_, double r\_) : x(x\_), y(y\_), r(r\_) {} // 带参数的构造函数

    double GetArea() override { return M\_PI \* r \* r; }               // 计算圆形的面积

    void Show() override                                             // 显示圆形的信息

    {

        cout << "Shape: Circle\n";

        cout << "Center: (" << x << ", " << y << ")\n";

        cout << "Radius: " << r << "\n";

        cout << "Diameter: " << r \* 2 << "\n";

        cout << "Circumference: " << 2 \* M\_PI \* r << "\n";

        cout << "Area: " << GetArea() << "\n";

    }

};

class Rectangle : public Shape // 矩形派生类，继承自Shape

{

private:

    double length, width; // 长度和宽度

public:

    Rectangle() : length(0), width(0) {}                                         // 默认构造函数

    Rectangle(double length\_, double width\_) : length(length\_), width(width\_) {} // 带参数的构造函数

    double GetArea() override { return length \* width; }                         // 计算矩形的面积

    void Show() override                                                         // 显示矩形的信息

    {

        cout << "Shape: Rectangle\n";

        cout << "Length: " << length << "\n";

        cout << "Width: " << width << "\n";

        cout << "Perimeter: " << 2 \* (length + width) << "\n";

        cout << "Area: " << GetArea() << "\n";

    }

};

class Triangle : public Shape // 三角形派生类，继承自Shape

{

private:

    double x1, y1, x2, y2, x3, y3; // 三个顶点的坐标

public:

    Triangle() : x1(0), y1(0), x2(0), y2(0), x3(0), y3(0) {} // 默认构造函数

    Triangle(double x1\_, double y1\_, double x2\_, double y2\_, double x3\_, double y3\_)

        : x1(x1\_), y1(y1\_), x2(x2\_), y2(y2\_), x3(x3\_), y3(y3\_) {} // 带参数的构造函数

    double GetArea() override                                     // 计算三角形的面积

    {

        double a = sqrt((x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1));

        double b = sqrt((x3 - x2) \* (x3 - x2) + (y3 - y2) \* (y3 - y2));

        double c = sqrt((x1 - x3) \* (x1 - x3) + (y1 - y3) \* (y1 - y3));

        double p = (a + b + c) / 2;

        return sqrt(p \* (p - a) \* (p - b) \* (p - c));

    }

    void Show() override // 显示三角形的信息

    {

        cout << "Shape: Triangle\n";

        cout << "Vertex 1: (" << x1 << ", " << y1 << ")\n";

        cout << "Vertex 2: (" << x2 << ", " << y2 << ")\n";

        cout << "Vertex 3: (" << x3 << ", " << y3 << ")\n";

        cout << "Perimeter: " << sqrt((x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1)) + sqrt((x3 - x2) \* (x3 - x2) + (y3 - y2) \* (y3 - y2)) + sqrt((x1 - x3) \* (x1 - x3) + (y1 - y3) \* (y1 - y3)) << "\n";

        cout << "Area: " << GetArea() << "\n";

    }

};

int main()

{

    Shape \*s1 = new Circle(0, 0, 5);            // 创建一个圆形对象

    Shape \*s2 = new Rectangle(3, 4);            // 创建一个矩形对象

    Shape \*s3 = new Triangle(0, 0, 3, 0, 0, 4); // 创建一个三角形对象

    s1->Show(); // 显示圆形信息

    s2->Show(); // 显示矩形信息

    s3->Show(); // 显示三角形信息

    delete s1; // 释放内存

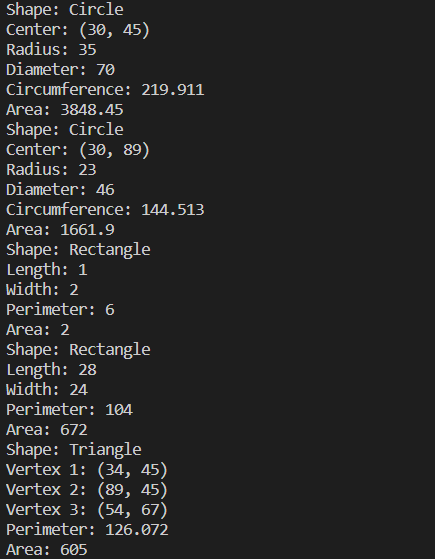
    delete s2;

    delete s3;

    return 0;

}

### 实验结果：



### 思考与扩展：

无

## 题目2 教师干部类

### 实验描述：

设计两个基类：教师类（Teacher）、干部类（Cadre），采用多重继承派生出新类：

Teacher\_Cadre

1、两个基类都包含：姓名、年龄、性别、地址、电话等；

在 Teacher 类中包含：title（职称），在Cadre 类中包含：post（职务），在Teacher\_Cadre

中包含：wages（工资）；

2、两个基类都包含：display()函数，分别输出本类中的成员信息；

3、在两个基类中的姓名、年龄、性别、地址、电话等数据成员使用相同名字，在派生

类中引用这些数据成员时，采用指定作用域的方式；

4、在派生类Teacher\_Cadre 的成员函数Show（）中调用Teacher 类中的display（）函

数，输出姓名、年龄、性别、地址、电话；

5、使用对象数组保存输入的对象；

### 实验代码：

#include <iostream>

#include <string>

using namespace std;

class Teacher

{

protected:

    string name;     // 姓名

    int age;         // 年龄

    string gender;   // 性别

    string address;  // 地址

    string phone;    // 手机号

    string title;    // 职称

public:

    Teacher() {} // 默认构造函数

    Teacher(string n, int a, string g, string add, string p, string t)

        : name(n), age(a), gender(g), address(add), phone(p), title(t)

    {

    }

    void display() // 显示教师信息

    {

        cout << "Name:" << name << endl;     // 输出姓名

        cout << "Age:" << age << endl;       // 输出年龄

        cout << "Gender:" << gender << endl; // 输出性别

        cout << "Address:" << address << endl;   // 输出地址

        cout << "Phone:" << phone << endl;   // 输出手机号

        cout << "Title:" << title << endl;   // 输出职称

    }

};

class Cadre

{

protected:

    string name;     // 姓名

    int age;         // 年龄

    string gender;   // 性别

    string address;  // 地址

    string phone;    // 手机号

    string post;     // 岗位

public:

    Cadre() {} // 默认构造函数

    Cadre(string n, int a, string g, string add, string p, string post)

        : name(n), age(a), gender(g), address(add), phone(p), post(post)

    {

    }

    void display() // 显示干部信息

    {

        cout << "Name:" << name << endl;     // 输出姓名

        cout << "Age:" << age << endl;       // 输出年龄

        cout << "Gender:" << gender << endl; // 输出性别

        cout << "Address:" << address << endl;   // 输出地址

        cout << "Phone:" << phone << endl;   // 输出手机号

        cout << "岗位:" << post << endl;        // 输出岗位

    }

};

class Teacher\_Cadre : public Teacher, public Cadre

{

protected:

    int wages; // 工资

public:

    Teacher\_Cadre() {} // 默认构造函数

    Teacher\_Cadre(string n, int a, string g, string add, string p, string t, string post, int w)

        : Teacher(n, a, g, add, p, t), Cadre(n, a, g, add, p, post), wages(w)

    {

    }

    void Show() // 显示教师兼干部的信息

    {

        Teacher::display();                 // 调用Teacher类的display函数，显示教师信息

        cout << "Wages:" << wages << endl;   // 输出工资

    }

};

int main()

{

    Teacher\_Cadre tc[20]; // 声明一个Teacher\_Cadre类型的数组，用于存储教师兼干部的信息

    int choice;          // 用户选择

    string n, g, add, p, t, post; // 临时变量，用于存储输入的姓名、性别、地址、手机号、职称、岗位

    int a, w;            // 临时变量，用于存储输入的年龄、工资

    int index = 0;       // 数组索引，表示当前已经存储的教师兼干部的数量

    while (true)

    {

        cout << "1.Add\n2.show\n3.exit" << endl;  // 显示菜单选项

        cin >> choice;                             // 获取用户选择

        getchar();                                 // 处理换行符

        switch (choice)

        {

        case 1:

            cout << "Add NO." << index + 1 << " info" << endl;  // 输入要添加的教师兼干部信息编号

            cout << "Input Name:";                             // 输入姓名

            getline(cin, n);

            cout << "Input Age:";                              // 输入年龄

            cin >> a;

            getchar();

            cout << "Input gender:";                           // 输入性别

            getline(cin, g);

            cout << "Input address:";                          // 输入地址

            getline(cin, add);

            cout << "Input phone:";                            // 输入手机号

            getline(cin, p);

            cout << "Input title:";                            // 输入职称

            getline(cin, t);

            cout << "Input post:";                             // 输入岗位

            getline(cin, post);

            cout << "Input wages:";                            // 输入工资

            cin >> w;

            tc[index++] = Teacher\_Cadre(n, a, g, add, p, t, post, w); // 在数组中添加教师兼干部信息

            break;

        case 2:

            for (int i = 0; i < index; i++)

            {

                cout << "No." << i + 1 << " info" << endl; // 显示第i个教师兼干部的信息

                tc[i].Show();

            }

            break;

        case 3:

            return 0;

        default:

            break;

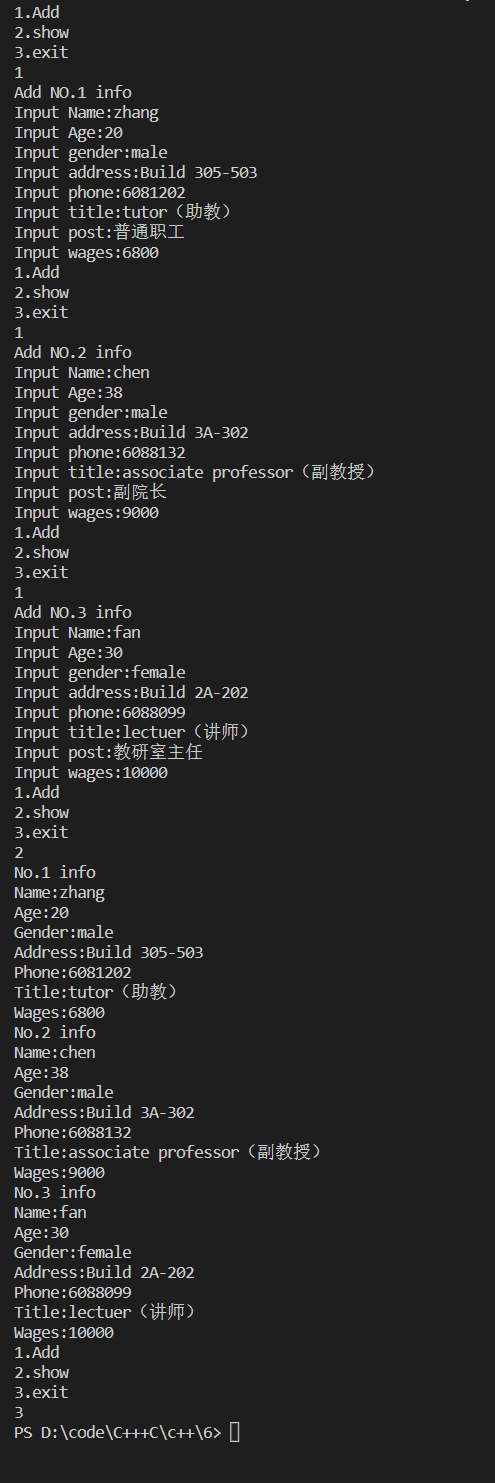
        }

    }

    return 0;

}

### 实验结果：



### 思考与扩展：

无

## 题目3 小型公司人员管理

### 实验描述：

某小型公司有四类人员：总经理、技术人员、销售经理、推销员。设计一个基类employee，

派生出manager（总经理）、technician（技术人员）、salesmanager（销售经理）、saleman（推销员）。销售经理既是经理又是销售人员，兼具两类人员的特点，因此同时继承manager 和salesman 两个类。

### 实验代码：

Main.cpp

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\6\employee\Main.cpp

 \* @Description:某小型公司有四类人员：总经理、技术人员、销售经理、推销员。设计一个基类employee，

派生出manager（总经理）、technician（技术人员）、salesmanager（销售经理）、saleman（推销员）。销售经理既是经理又是销售人员，兼具两类人员的特点，因此同时继承manager 和salesman 两个类。

 \*/

// g++ -std=c++11 -o Main Main.cpp employee.cpp Date.cpp

#include <iostream>

#include <string>

#include <vector>

#include <ctime>

#include <map>

#include "employee.h"

#include "Date.h"

using namespace std;

map<string, int> department\_map; // 存储部门销售额的映射

// manager 类

class manager : virtual public employee

{

public:

    manager() {}

    manager(int id, const string &name, string gender, const Date &birth) : employee(id, name, gender, birth, "Manager", GetMonthlySalary()) {}

    double GetMonthlySalary() const override

    {

        return 8000.0; // 经理的月薪固定为8000元

    }

};

// technician 类

class technician : public employee

{

public:

    technician() {}

    technician(int id, const string &name, string gender, const Date &birth, double work\_time) : work\_time\_(work\_time)

    {

        this->id\_ = id;

        this->name\_ = name;

        this->gender\_ = gender;

        this->birth\_ = birth;

        this->position\_ = "technician";

        this->salary\_ = GetMonthlySalary();

    }

    double GetMonthlySalary() const override

    {

        return work\_time\_ \* hourly\_wage\_; // 技术员的月薪为工作时间乘以时薪

    }

private:

    double hourly\_wage\_ = 25; // 技术员的时薪为25元

    double work\_time\_; // 工作时间

};

// salesman 类

class salesman : virtual public employee

{

public:

    salesman() {}

    salesman(int id, const string &name, string gender, const Date &birth, const string &department, double sales) : department\_(department), sales\_(sales)

    {

        this->id\_ = id;

        this->name\_ = name;

        this->gender\_ = gender;

        this->birth\_ = birth;

        this->position\_ = "salesman";

        this->salary\_ = GetMonthlySalary();

        department\_map[department] += sales; // 将销售额累加到对应部门的销售额中

    }

    double GetMonthlySalary() const override

    {

        return sales\_ \* 0.04; // 销售员的月薪为销售额的4%

    }

protected:

    string department\_; // 部门

    double sales\_; // 销售额

};

// salesmanager 类

class salesmanager : public manager, public salesman

{

public:

    salesmanager() {}

    salesmanager(int id, const string &name, string gender, const Date &birth, const string &department)

    {

        this->id\_ = id;

        this->name\_ = name;

        this->birth\_ = birth;

        this->gender\_ = gender;

        this->department\_ = department;

        this->position\_ = "salesmanager";

        this->salary\_ = GetMonthlySalary();

    }

    double GetMonthlySalary() const override

    {

        return 5000 + department\_map[department\_] \* 0.05; // 销售经理的月薪为5000元加上部门销售额的5%

    }

};

int main()

{

    vector<employee \*> employees; // 员工指针的向量

    int choice, year, month, day, work\_time, sales;

    int id;

    string name;

    string gender;

    string department, position;

    while (true)

    {

        // 显示菜单选项

        cout << "1.添加经理" << endl

             << "2.添加技术员" << endl

             << "3.添加销售员" << endl

             << "4.添加销售经理" << endl

             << "5.显示所有员工" << endl

             << "6.退出" << endl;

        cin >> choice; // 获取用户选择

        switch (choice)

        {

        case 1:

            cout << "添加经理（id、姓名、性别、出生年份、月份、日期）" << endl;

            cin >> id >> name >> gender >> year >> month >> day;

            employees.push\_back(new manager(id, name, gender, Date(year, month, day))); // 创建并添加经理对象到员工指针向量中

            break;

        case 2:

            cout << "添加技术员（id、姓名、性别、出生年份、月份、日期、工作时间）" << endl;

            cin >> id >> name >> gender >> year >> month >> day >> work\_time;

            employees.push\_back(new technician(id, name, gender, Date(year, month, day), work\_time)); // 创建并添加技术员对象到员工指针向量中

            break;

        case 3:

            cout << "添加销售员（id、姓名、性别、出生年份、月份、日期、部门、销售额）" << endl;

            cin >> id >> name >> gender >> year >> month >> day >> department >> sales;

            employees.push\_back(new salesman(id, name, gender, Date(year, month, day), department, sales)); // 创建并添加销售员对象到员工指针向量中

            break;

        case 4:

            cout << "添加销售经理（id、姓名、性别、出生年份、月份、日期、部门）" << endl;

            cin >> id >> name >> gender >> year >> month >> day >> department;

            employees.push\_back(new salesmanager(id, name, gender, Date(year, month, day), department)); // 创建并添加销售经理对象到员工指针向量中

            break;

        case 5:

            for (int i = 0; i < employees.size(); i++)

            {

                DisplayEmployee(\*employees[i]); // 显示所有员工信息

            }

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

            break;

        case 6:

            exit(0); // 退出程序

            break;

        default:

            cout << "输入错误！" << endl;

            break;

        }

    }

    system("pause");

    return 0;

}

Employee.h

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\6\employee\employee.h

 \* @Description: employee类的头文件，包含友元函数DisplayEmployee的声明，以及employee类的声明

 \*/

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

#include <string>

#include "Date.h" // 包含"Date"头文件

#include <iostream>

using namespace std;

class employee

{

public:

    employee() {} // 默认构造函数

    // 带参数的构造函数

    employee(int id, const string &name, string gender, const Date &birth, const string &position, double salary);

    // 友元函数声明，用于显示员工信息

    friend void DisplayEmployee(const employee &e);

    // 获取员工薪资的虚函数

    virtual double GetMonthlySalary() const = 0;

protected:

    int id\_;          // 员工ID

    string name\_;     // 员工姓名

    string gender\_;   // 员工性别

    Date birth\_;      // 员工生日

    string position\_; // 员工职位

    double salary\_;   // 员工薪水

};

#endif

Employee.cpp

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\6\employee\employee.cpp

 \* @Description: employee类的实现文件，包含友元函数DisplayEmployee的实现

 \*/

#include "employee.h"

using namespace std;

// 参数化构造函数的定义

employee::employee(int id, const string &name, string gender, const Date &birth, const string &position, double salary)

    : id\_(id), name\_(name), gender\_(gender), birth\_(birth), position\_(position), salary\_(salary) {}

// 友元函数用于显示员工信息的定义

void DisplayEmployee(const employee &e)

{

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

    cout << "ID:" << e.id\_ << endl;

    cout << "Name: " << e.name\_ << endl;

    cout << "Gender: " << e.gender\_ << endl;

    cout << "Birthday: " << e.birth\_.GetYear() << "." << e.birth\_.GetMonth() << "." << e.birth\_.GetDay() << endl;

    cout << "Position: " << e.position\_ << endl;

    cout << "Salary: " << e.salary\_ << endl;

}

#ifndef DATE\_H

#define DATE\_H

class Date

{

public:

    Date() {}

    /\*\*

     \* @brief 构造函数，用于初始化日期对象

     \* @param year 年份

     \* @param month 月份

     \* @param day 日期

     \*/

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

    /\*\*

     \* @brief 设置年份

     \* @param year 年份

     \*/

    void SetYear(int year);

    /\*\*

     \* @brief 设置月份

     \* @param month 月份

     \*/

    void SetMonth(int month);

    /\*\*

     \* @brief 设置日期

     \* @param day 日期

     \*/

    void SetDay(int day);

    /\*\*

     \* @brief 获取年份

     \* @return 年份

     \*/

    int GetYear() const;

    /\*\*

     \* @brief 获取月份

     \* @return 月份

     \*/

    int GetMonth() const;

    /\*\*

     \* @brief 获取日期

     \* @return 日期

     \*/

    int GetDay() const;

private:

    int year\_; // 年份

    int month\_; // 月份

    int day\_; // 日期

};

#endif

Date.h

#include "date.h"

Date::Date(int year, int month, int day) : year\_(year), month\_(month), day\_(day)

{

    // 构造函数，用于初始化日期对象

    // 参数：year 年份，month 月份，day 日期

    // 将传入的参数赋值给对应的成员变量

}

void Date::SetYear(int year)

{

    // 设置年份

    // 参数：year 年份

    // 将传入的年份设置为对象的年份

    year\_ = year;

}

void Date::SetMonth(int month)

{

    // 设置月份

    // 参数：month 月份

    // 将传入的月份设置为对象的月份

    month\_ = month;

}

void Date::SetDay(int day)

{

    // 设置日期

    // 参数：day 日期

    // 将传入的日期设置为对象的日期

    day\_ = day;

}

int Date::GetYear() const

{

    // 获取年份

    // 返回值：年份

    // 返回对象的年份

    return year\_;

}

int Date::GetMonth() const

{

    // 获取月份

    // 返回值：月份

    // 返回对象的月份

    return month\_;

}

int Date::GetDay() const

{

    // 获取日期

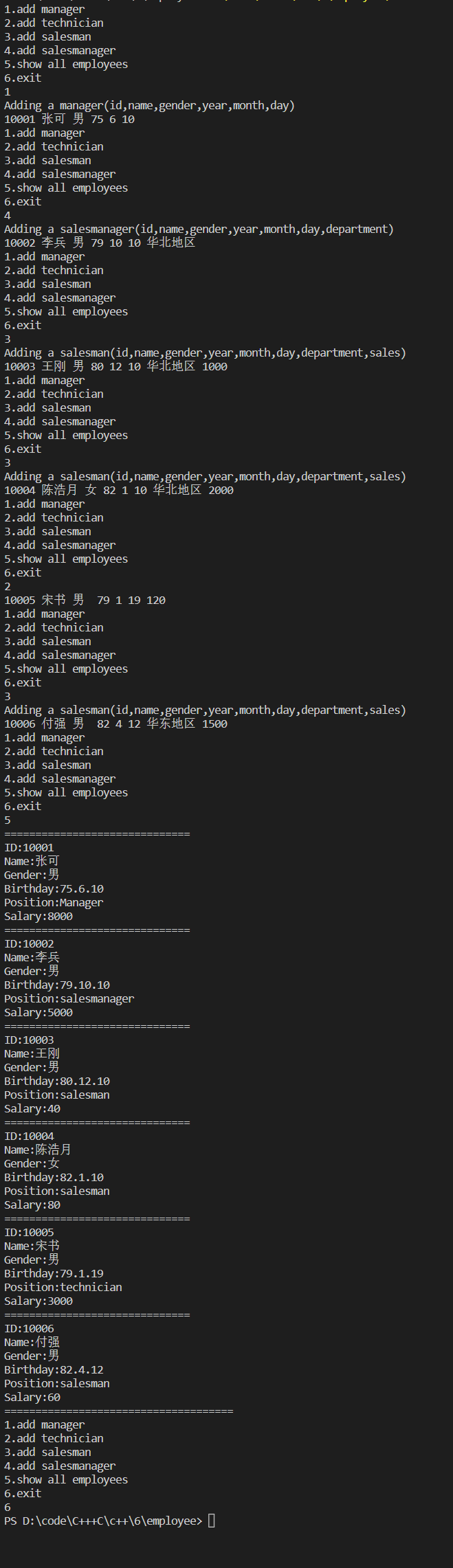
    // 返回值：日期

    // 返回对象的日期

    return day\_;

}

### 实验结果：



### 思考与扩展：

无