**西安电子科技大学**

**OOP课程设计**

**(2019年度)**

**实**

**验**

**报**

**告**

**实验名称：** **销售公司员工管理**

**班 级：**

**姓 名：**

**学 号：**

**目录**

[1 题目分析 1](#_Toc27086167)

[1.1 所需基本类 1](#_Toc27086168)

[1.2 将员工抽象成结构体Node 2](#_Toc27086169)

[1.3 用户管理类 2](#_Toc27086170)

[1.4 数据结构的选用 2](#_Toc27086171)

[2 类设计及功能说明 2](#_Toc27086172)

[2.1 Staff类 3](#_Toc27086173)

[2.2 Salesman类 4](#_Toc27086174)

[2.3 Manager类 5](#_Toc27086175)

[2.4 SalesManager类 6](#_Toc27086176)

[2.5 list结构体 7](#_Toc27086177)

[2.6 ManagementSystem类 7](#_Toc27086178)

[3 问题及解决 9](#_Toc27086179)

[3.1 纯虚函数的实现 9](#_Toc27086180)

[3.2 弃用静态变量sales 9](#_Toc27086181)

[3.3 vector中erase用法 10](#_Toc27086182)

[3.4 UI交互的bug消除 11](#_Toc27086183)

[4 结果演示 13](#_Toc27086184)

[4.0 刚开始进入页面 13](#_Toc27086185)

[4.1 查询所有员工 14](#_Toc27086186)

[4.2 按员工号或姓名查询 14](#_Toc27086187)

[4.3 查询人数 15](#_Toc27086188)

[4.4 标记待解雇员工 16](#_Toc27086189)

[4.5 解雇已标记员工 16](#_Toc27086190)

[4.6 添加员工 17](#_Toc27086191)

[4.7 修改员工信息 18](#_Toc27086192)

[5 程序源代码 19](#_Toc27086193)

# 

# 1 题目分析

目标:设计一个虚基类Staff(员工)，包含编号、姓名和年龄保护数据成员以及相关的成员函数；由Staff派生出销售员类Salesman，包含销售额保护数据成员以及相关的成员函数；再由Staff派生出经理类Manager；然后由Salesman和Manager类派生出销售经理类SalesManager。现要求设计一个利用文件处理方式实现对公司人员进行管理，具有增加数据、更新数据、查询数据、删除数据以及重组文件的功能。另外要求分别统计这三类员工的人数以及所有员工的总数。

## 1.1 所需基本类

从题目中可以看出, 需要四个基本类基类, **class Staff, class Salesman, class Manager, class SalesManager**.其中类Salesman, 类Manager虚继承类Staff; 类SalesManager多继承类Manager, 类SalesManager. 由于存在比较复杂的继承关系(包括数据成员继承和函数继承), 所以要格外注意各个类之间的关系, 同时要在功能较好实现和代码简洁上下功夫.

## 1.2 将员工抽象成结构体Node

为了方便多态,将员工同意抽象成抽象结构体struct Node,其中的属性有Staff \*staff和bool isEmployed.

## 1.3 用户管理类

题目要求设计一个利用文件处理方式实现对公司人员进行管理，具有增加数据、更新数据、查询数据、删除数据以及重组文件的功能。另外要求分别统计这三类员工的人数以及所有员工的总数。所以可以设计一ManagementSystem用来实现上述功能. 在类ManagementSystem中, 要分别实现题目中要求实现的功能, 这些功能放在SelectFunction()中由用户调用. 在类中要显示定义一个构造函数, 用于数据成员的初始化.

## 1.4 数据结构的选用

由于涉及到字符串, 为了方便操作, 选择使用封装好的**string类**.

由于员工的信息是顺序存放的, 这里有数组, 链表, 动态数组(vector)选择. 由于链表容易出现指向问题, 这里还是选择方便的**vector**.

# 2 类设计及功能说明

类的设计原则我尽量做到封装(用户仅能接触SelectFunction()函数、private和protected的运用等)、组合(将一个功能实现分割成各个小函数), 实现多态, 以及防止方式程序员本身做的误操作(比如使用private和const).

尽量做到功能很好地实现和代码的简介, 这点在基类的设计上应该可以体现出来…纯虚数, 虚函数, 常成员函数都用在里面了.

最后尽量把UI交互的bug排尽. 所以最终的代码就比较长,有大概550行, 其中防止用户误操作的大概就有150行…

## 2.1 Staff类

Staff虚基类结构如下所示:

class Staff{

    protected://这样派生类也能继承

    string no; //编号

    string name; //姓名

    int age; //年龄

    public:

    Staff(string no, string name, int age){

        this->no = no;

        this->name = name;

        this->age = age;

    }

    virtual ~Staff(){}

    //声明为public, 对外接口

    virtual void showData() const = 0; //输出员工信息(纯虚函数-派生类必须实现)

    virtual string getTitle() const = 0;//得到员工的职称(纯虚函数)

    string getNo() const{ return no;} //得到员工的编号(常函数)

    void setNo(string no){this->no = no;};//修改员工编号

    string getName() const{return name;}//得到员工的姓名

    void setName(string name){this->name = name;};//修改员工姓名

    int getAge() const{return age;}//得到员工的年龄

    void setAge(int age){this->age = age;};//修改员工年龄

    virtual double getSales()const{}//得到员工销售额(如果有的话)(虚函数-派生类可不实现-调用各自对应的同名函数)

    virtual void setSales(double sales){}//修改员工销售额(如果有的话)

};

编号no: 可长可短, 可能带任何字符, 所以声明为string类.(不使用const char\* 是容易出错且不方便…)

姓名name: 姓名可重复, 这在后面添加、修改和查询员工信息时都有考虑到.

年龄age: 需为整数

showData(): 声明为纯虚数, 因为派生类都需要重新实现(主要是因为Manager没有sales, 不然只用写一遍了…)

getTitle(): 纯虚数, 因为不同类的title不同. 由于Staff类不需要title, 所以就不声明数据成员, 只声明个函数了.

getXX()(no, name, age): 常函数, 派生类继承该函数, 返回各自对应的数据.而且由于只是读取数据, 所以声明为const常成员函数.

setXX()(no, name, age): 正常函数, 派生类继承该函数, 修改各自对应的数据.

virtual double getSales()const: 虚函数, 原因是Manager没有sales成员, 所以就继续使用基类的空函数体, Salesman和Salesmanager有sales数据, 所以需要返回sales. const声明是因为不修改数据.

virtual void setSales(double sales): 和getSales同样, 不过因为要修改数据所以没有const.

## 2.2 Salesman类

Salesman类结构如下所示:

class Salesman: virtual public Staff{

    protected:

    double sales;//销售额

    public:

    SalesManager(string no, string name, int age, double sales)

: Staff(no, name, age){

        this->no = no;

        this->name = name;

        this->age = age;

        this->sales = sales;

    }

    virtual ~Salesman(){}

    virtual void showData() const{ //纯虚函数重写格式需严格

        cout<<"编号:"<<no<<" ";

        cout<<"姓名:"<<name<<" ";

        cout<<"年龄:"<<age<<" ";

        cout<<"职称:"<<"Salesman"<<"     ";

        cout<<"销售额:"<<sales<<endl;

    }

    virtual string getTitle() const{//重写,返回各个类职称

        return "Salesman";

    }

    double getSales() const{return this->sales;}

    void setSales(double sales){this->sales = sales;};//修改员工Sales

};

大多函数都已经在Staff类中实现.

Sales：double型, 只Salesman中声明, 继承到SalesManager中.

showData():　输出员工信息，包括销售额．而且要注意到纯虚函数的重写非常严格，之前没写好，导致报错了也找不到原因．

virtual string getTitle() const: 虚函数重写, Manage类中不需要写, 直接继承基类的空函数体.

getSales()/setSales(double sales): 在Salesman中定义, 在SalesManager中再重写一遍, 防止歧义.

## 2.3 Manager类

class Manager: virtual public Staff{

    //数据成员都由基类继承来

    public:

    Manager(string no, string name, int age): Staff(no, name, age){

        this->no = no;

        this->name = name;

        this->age = age;

    }

    virtual ~Manager(){}

    virtual void showData() const{

        cout<<"编号:"<<no<<" ";

        cout<<"姓名:"<<name<<" ";

        cout<<"年龄:"<<age<<" ";

        cout<<"职称:"<<"Manager"<<endl;

    }

    virtual string getTitle() const{

        return "Manager";

    }

    //虽然继承了getSales()和setSales()但函数体为空

};

Manager由于没有sales数据成员, 所以得花好多心思设计类…

## 2.4 SalesManager类

SalesManager类结构如下所示。除构造函数外其它函数定义基本同Salesman类。

class SalesManager: public Salesman, public Manager{

    //sales由Salesman继承来

    public:

    SalesManager(string no, string name, int age, double　sales): Staff(no, name, age),

    Salesman(no, name, age, sales), Manager(no, name, age){

        this->no = no;

        this->name = name;

        this->age = age;

        this->sales = sales;

    }

    virtual ~SalesManager(){}

    virtual void showData() const{

        cout<<"编号:"<<no<<" ";

        cout<<"姓名:"<<name<<" ";

        cout<<"年龄:"<<age<<" ";

        cout<<"职称:"<<"SalesManager"<<" ";

        cout<<"销售额:"<<sales<<endl;

    }

    virtual string getTitle() const{

        return "SalesManager";

    }

    double getSales() const{return this->sales;}//防止歧义, 重写

    void setSales(double sales){this->sales = sales;};

};

由于其直接基类是虚继承自Staff, 所以数据成员只保存自父类的一份.

## 2.5 list结构体

对于公司中的每一个员工, 为了便于删改操作以及动态存储, 采用了如下带存储结构.每一个结点包含了一个指向员工类得指针和判断是否还在干活的bool型数据.

struct Node{ //vector中的元素

    Staff \*staff;//指向Staff的指针

    bool isEmployed;//待解雇用

};

## 2.6 ManagementSystem类

class ManagementSystem{

    private:

    vector<Node> vec;//动态数组

    int StaffNum;//员工总人数

    //private,将功能封装, 用户只能通过SelectFunction()接口调用

    void fileIn();//文件录入

    void showStaff();//显示所有信息

    void showCount();//显示所有员工人数

    void searchStaff();//显示单个人的信息

    void addData();//添加员工

    void markStaffToBeFired();//标记待解雇员工

    void updateStaff();//修改员工信息

    void fireStaffs();//解雇isEmployed为false的员工

    void recombineFile();//保存文件并退出

    //辅助函数

    void hire(string no, string name, int age, string title, double sales = 0);//往vector中新加结点

    bool hasRepeatted(string no);//员工号为码,判断vector中是否已存在相同员工号

    bool inTitles(string title){//判断输出title是否正确

        return (!title.compare("Salesman"))||(!title.compare("Manager"))||(!title.compare("SalesManager"));

    }

    bool hasSales(string title){//判断是否拥有sales这个数据成员

        return ((!title.compare("Salesman")) || (!title.compare("SalesManager")));}

    bool isSalesman(string title){return (!title.compare("Salesman"));}//判断是否是Manager

    bool isManager(string title){return (!title.compare("Manager"));}

    bool isSalesManager(string title){return (!title.compare("SalesManager"));}

    int getStaffNum(){return StaffNum;}

    public:

    ManagementSystem();

    ~ManagementSystem(){}

    void SelectFunction();//选择功能

};

ManagementSystem类主要负责整个系统的管理, 里面主要的角色是vector动态数组, 所有的操作都是对它进行的, 明确了这一点, 只要对着功能一个个实现就好了. 其中一点需要之一的是将各种功能函数声明为private类, 这样用户就不能直接调用这些函数而只能通过SelectFunction()选择功能了, 只有SelectFunction()是public的. 具体的函数功能已经在注释里了, 这里不再赘述.

# 3 问题及解决

实现的过程中遇到了许多问题, 现就仍记得的问题及其解决方案记录在下面.

## 3.1 纯虚函数的实现

一开始只是按照书上的纯虚函数写法写了基类的纯虚函数, 以为这样子类的就可以随便写其格式了, 后来报了一个看不懂的错误, 最后发现是派生类中纯虚函数的重写有问题.

**基类中纯虚函数的实现:**

virtual void showData() const = 0; //输出员工信息(纯虚函数-派生类必须实现)

virtual string getTitle() const = 0;//得到员工的职称(纯虚函数)

**派生类中的纯虚函数重写格式:**

virtual void showData() const{ //纯虚函数重写格式需严格

        //省略}

virtual string getTitle() const{//重写,返回各个类职称

        return "Salesman";}

## 3.2 弃用静态变量sales

一开始注意到每个类中的title是固定的, 所以就想在每个类中声明一个title类,并初始化成其对应的值, 就像这样:

static const string title;

const string Staff::title = "Salesman";//在类外初始化

但是随后发现静态变量sales也会被继承, 而且staff显然不需要title. 而且通过指针调用staff时, 由于staff同名且调用会优先选择父类的原因(子类没有覆盖其值), 最终弃用了这个方案.改为:

virtual string getTitle() const = 0;//得到员工的职称(纯虚函数)

virtual string getTitle() const{//重写,返回各个类职称

        return "Salesman";}

这样就能用很少的代码(基本能压缩到3行,在基类和两个派生类中)实现功能.了.

3.3 vector中erase用法

vector中erase的功能时删除当前指向的函数并指向删除元素下一个元素.

情况是这样的当我尝试删除vector中的最后一个元素时发现界面直接卡住了. 类似源代码如下:

vector<Node>::iterator it;

for(it = vec.begin(); it != vec.end();it++){

        if(\*it == 9){//9为数组中最后一个元素

           it = vec.erase(it);

        }

}

乍一看很正常, 如果指向的元素是9的话则删除, 再下一个嘛. 后来发现当删除最后一个元素后it还会+1, 即当it.erase(iter)之后，it就变成了一个野指针，对一个野指针进行 it ++ 是肯定会出错的。查看MSDN，对于erase的返回值是这样描述的：An iterator that designates the first element remaining beyond any elements removed, or a pointer to the end of the vector if no such element exists(翻译: 一个迭代器，它指像剩余的任何元素的第一个元素，或者如果没有这样的元素，指向向量的结尾的指针)，于是改代码为：

for(it = vec.begin(); it != vec.end();){

      if(\*it == 9)

          it = vec.erase(it); //返回值指向删除元素的下一个元素

      else

            it++;

}

这样当对最后一个元素进行erase操作后 it就不会再+1变成野指针了.

## 3.4 UI交互的bug消除

这样的较大型的UI交互(用C++写的原始的)不可避免地会在使用中产生大量的交互问题(比如用户不小心输错了啥导致一直循环某段文字或者是系统卡住然后退出).

所以这次我在设计伊始就比较注意这方面的问题. 在代码这块主要用以下语句防止交互BUG.

**检测输入类型:**

cin>>age;//age 需为int型

if (cin.good() == 1){cin>>title;}

else{cout<<"年龄需为整数."<<endl;cin.clear();cin.sync();continue;}

**忽略用户后面内容:**

cin.ignore(100, '\n');

当用户想退出这一步时:

cout<<"请输入员工号，姓名，年龄，职位，(营销额):(如果想退出请输入'#'"<<endl;

cin>>no; //判断输入是否为#

if(!no.compare("#")) return;

**直到用户输入正确为止:**

while(1){

cin >> sales;cin.ignore(100, '\n');//防止用户多输入数据

    if (cin.good() == 1);

    else{cout<<"销售额需为复数."<<endl;cin.clear();cin.sync();continue;}

    break;

}

最后以addData()的函数为例说明下如何实现UI交互的:

void ManagementSystem::addData(){//需考虑员工号重复情况

    int age;

    string no, name, title;

    double sales;

    while(1){

    cout<<"请输入员工号，姓名，年龄，职位，(营销额):(如果想退出请输入'#'"<<endl;

    cin>>no; //判断输入是否为#

    if(!no.compare("#")) return;

    cin>>name;

    cin>>age;//age 需为int型

    if (cin.good() == 1){cin>>title;}

    else{cout<<"年龄需为整数."<<endl;cin.clear();cin.sync();continue;}

    if(inTitles(title));// title只能为3种

    else{cout<<"职位输入错误."<<endl;cin.ignore(100, '\n');continue;}

    if(hasRepeatted(no)){cout<<"员工编号重复."<<endl;continue;}

    if (hasSales(title)){

        while(1){

            cin >> sales;cin.ignore(100, '\n');//防止用户多输入数据

            if (cin.good() == 1);

            else{cout<<"销售额需为复数."<<endl;in.clear();in.sync();continue;}

            break;

        }

    }

    else{

        if(!cin.eof()){

            cout<<"(该员工无需输入销售额."<<endl;

            cin.ignore(100, '\n');

        }

    }

    hire(no, name, age, title, sales);//录入并退出函数

    break;

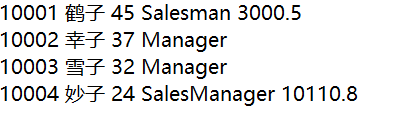
    }

    cout << "员工信息已录入" << endl;

}

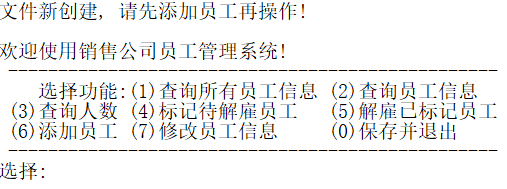
# 4 结果演示

Staff.txt中存储数据如下：

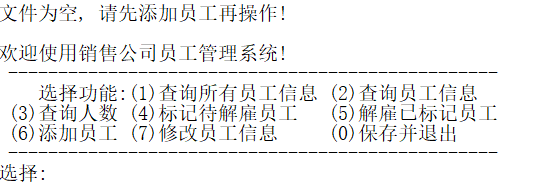
****

## 4.0 刚开始进入页面

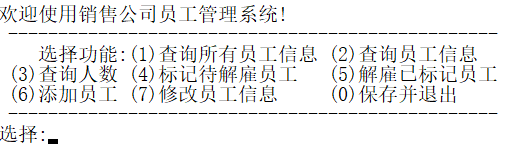
**当文件未创建:**



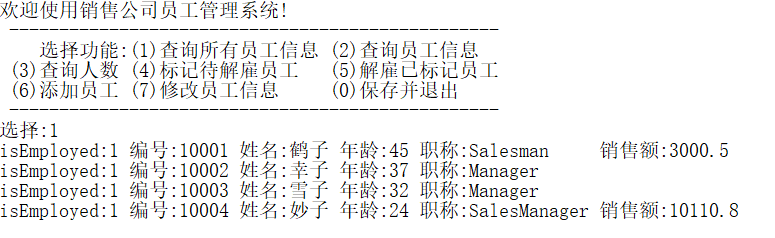
**当文件为空:**



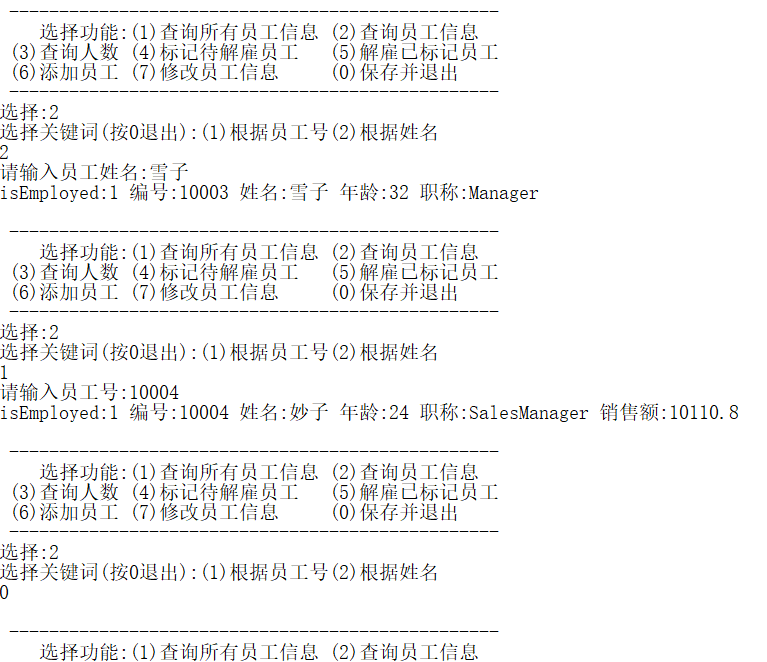
**正常进入:**



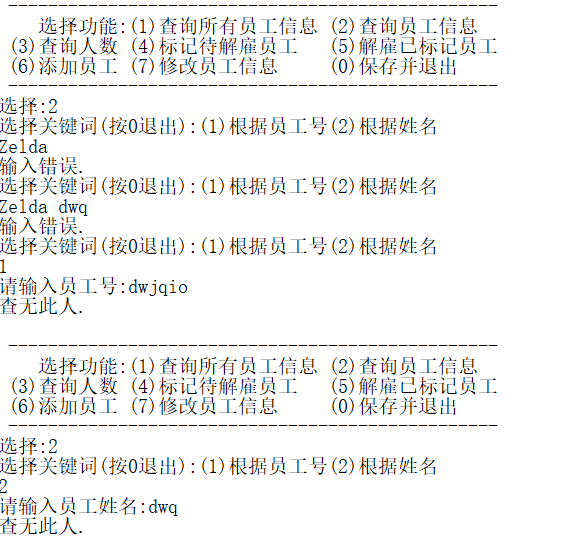
## 4.1 查询所有员工



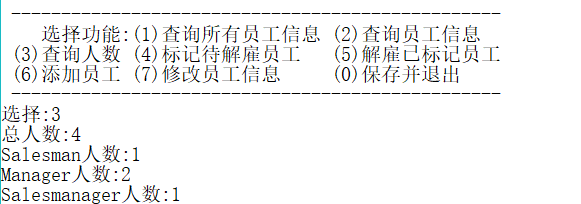
## 4.2 按员工号或姓名查询



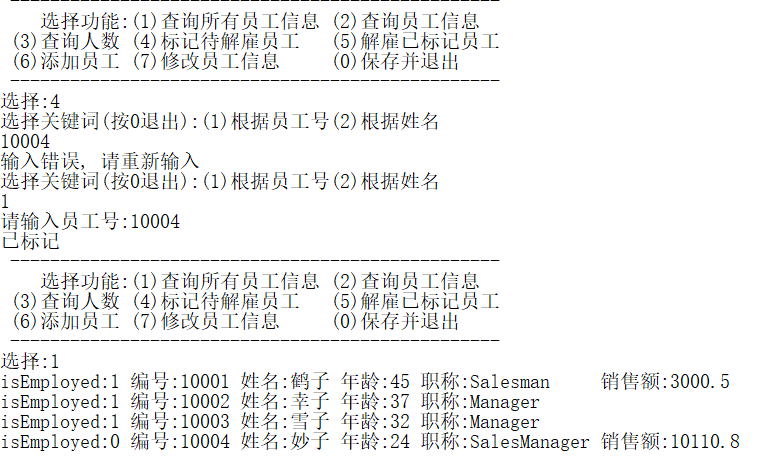
**当用户误操作:**

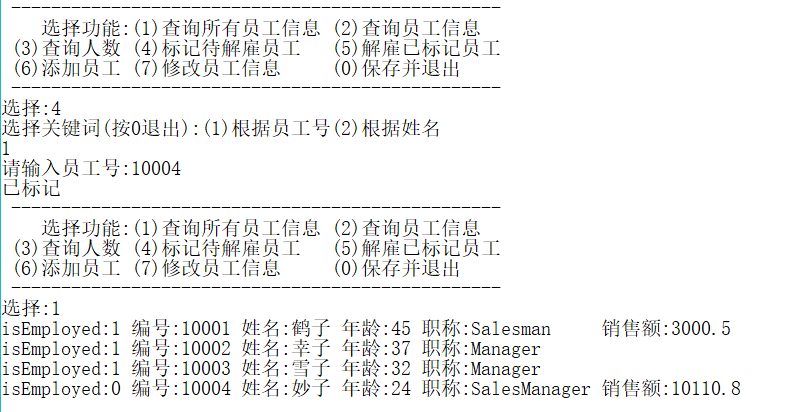


## 4.3 查询人数

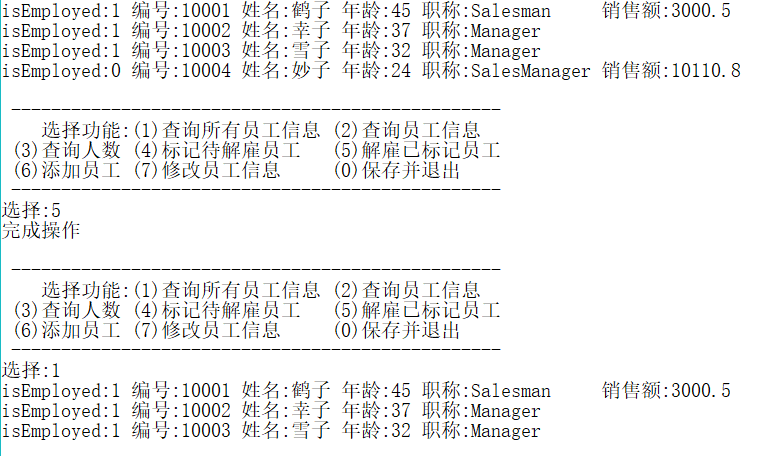


## 4.4 标记待解雇员工

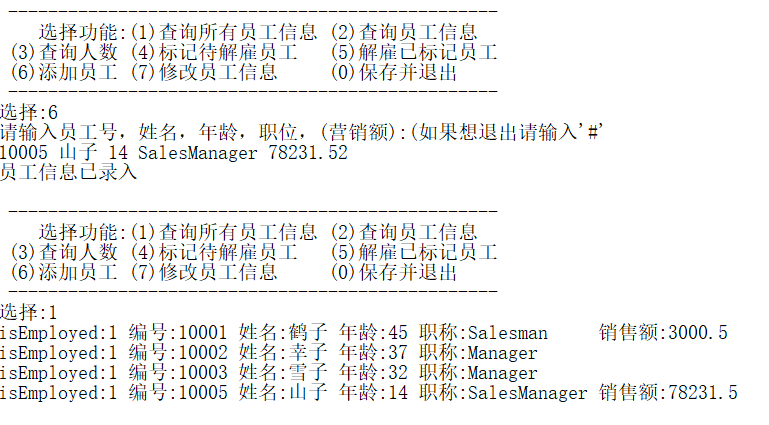




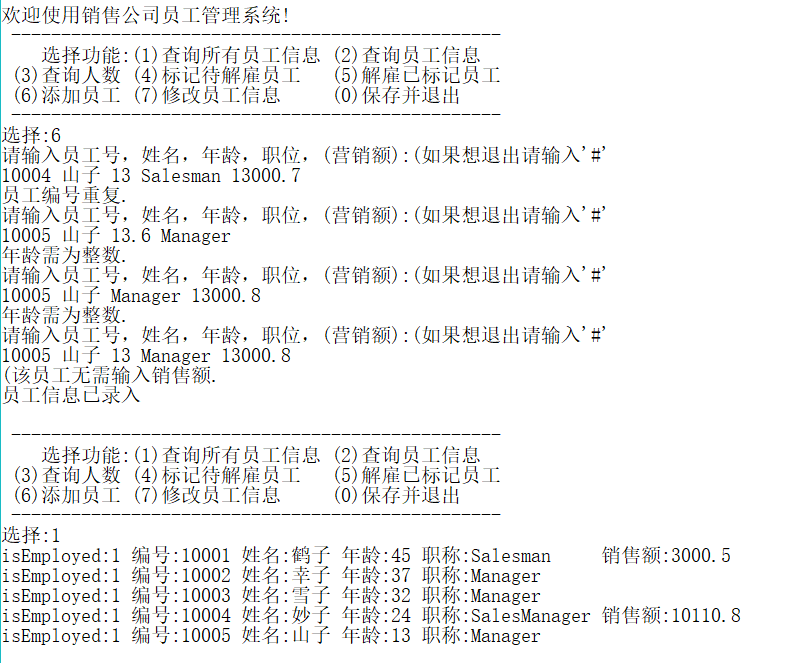
## 4.5 解雇已标记员工



## 4.6 添加员工



**当用户误操作:**

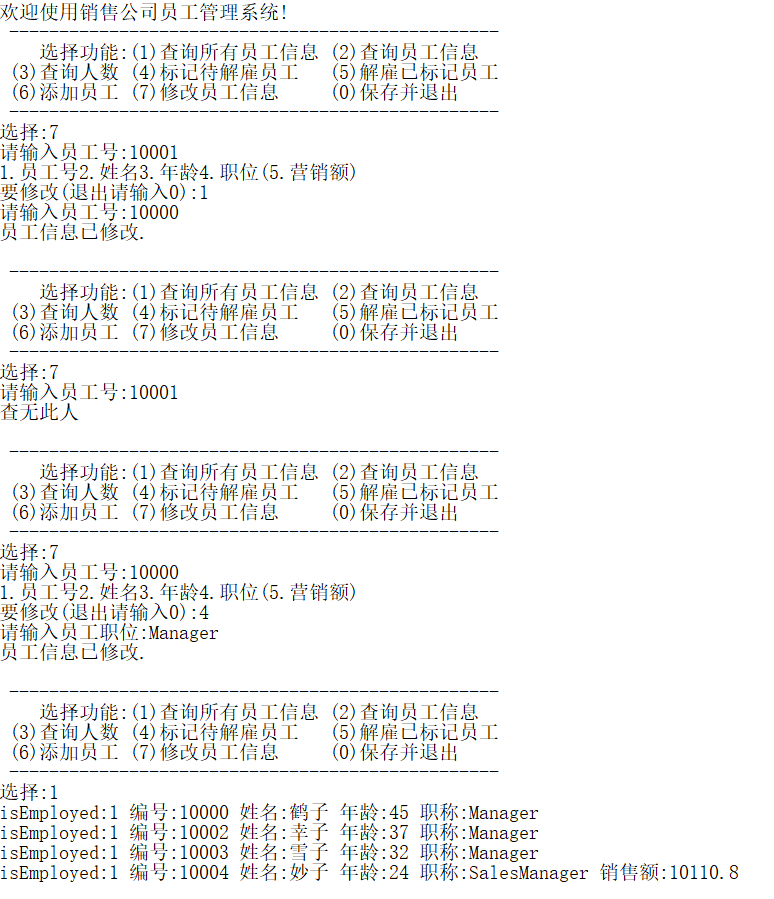


这里有几点说明:

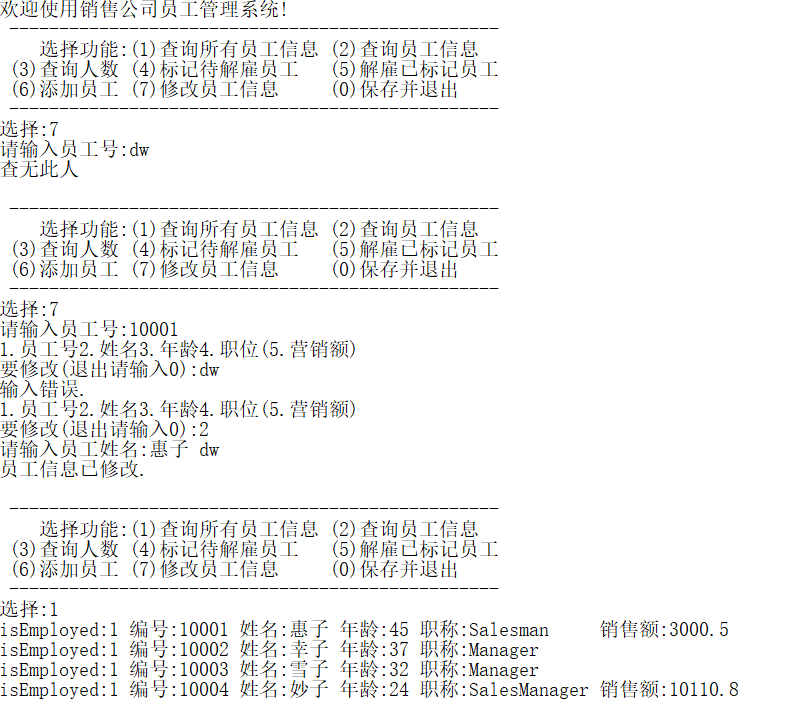
1)当输入的年龄为double型时自动转换成int型

2)当Manager或者Salesman/SalesManager的sales后面有多余输入时自动忽略, 录入之前的信息

## 4.7 修改员工信息



**当用户误操作:**



**保存并退出就不再赘述了,就是写回数据.**

# 5 程序源代码

#include <iostream>

#include <fstream>

#include <cstring>

#include <stdlib.h>

#include <vector>

using namespace std;

class Staff{

    protected://这样派生类也能继承

    string no; //编号

    string name; //姓名

    int age; //年龄

    public:

    Staff(string no, string name, int age){

        this->no = no;

        this->name = name;

        this->age = age;

    }

    virtual ~Staff(){}

    //声明为public, 对外接口

    virtual void showData() const = 0; //输出员工信息(纯虚函数-派生类必须实现)

    virtual string getTitle() const = 0;//得到员工的职称(纯虚函数)

    string getNo() const{ return no;} //得到员工的编号(常函数)

    void setNo(string no){this->no = no;};//修改员工编号

    string getName() const{return name;}//得到员工的姓名

    void setName(string name){this->name = name;};//修改员工姓名

    int getAge() const{return age;}//得到员工的年龄

    void setAge(int age){this->age = age;};//修改员工年龄

    virtual double getSales()const{}//得到员工销售额(如果有的话)(虚函数-派生类可不实现)

    virtual void setSales(double sales){}//修改员工销售额(如果有的话)

};

class Salesman: virtual public Staff{

    protected:

    double sales;//销售额

    public:

    Salesman(string no, string name, int age, double sales): Staff(no, name, age){

        this->no = no;

        this->name = name;

        this->age = age;

        this->sales = sales;

    }

    virtual ~Salesman(){}

    virtual void showData() const{ //纯虚函数重写格式需严格

        cout<<"编号:"<<no<<" ";

        cout<<"姓名:"<<name<<" ";

        cout<<"年龄:"<<age<<" ";

        cout<<"职称:"<<"Salesman"<<"     ";

        cout<<"销售额:"<<sales<<endl;

    }

    virtual string getTitle() const{//重写,返回各个类职称

        return "Salesman";

    }

    double getSales() const{return this->sales;}

    void setSales(double sales){this->sales = sales;};//修改员工Sales

};

class Manager: virtual public Staff{

    //数据成员都由基类继承来

    public:

    Manager(string no, string name, int age): Staff(no, name, age){

        this->no = no;

        this->name = name;

        this->age = age;

    }

    virtual ~Manager(){}

    virtual void showData() const{

        cout<<"编号:"<<no<<" ";

        cout<<"姓名:"<<name<<" ";

        cout<<"年龄:"<<age<<" ";

        cout<<"职称:"<<"Manager"<<endl;

    }

    virtual string getTitle() const{

        return "Manager";

    }

    //虽然继承了getSales()和setSales()但函数体为空

};

class SalesManager: public Salesman, public Manager{

    //sales由Salesman继承来

    public:

    SalesManager(string no, string name, int age, double sales): Staff(no, name, age),

    Salesman(no, name, age, sales), Manager(no, name, age){

        this->no = no;

        this->name = name;

        this->age = age;

        this->sales = sales;

    }

    virtual ~SalesManager(){}

    virtual void showData() const{

        cout<<"编号:"<<no<<" ";

        cout<<"姓名:"<<name<<" ";

        cout<<"年龄:"<<age<<" ";

        cout<<"职称:"<<"SalesManager"<<" ";

        cout<<"销售额:"<<sales<<endl;

    }

    virtual string getTitle() const{

        return "SalesManager";

    }

    double getSales() const{return this->sales;}//防止歧义, 重写

    void setSales(double sales){this->sales = sales;};

};

struct Node{ //vector中的元素

    Staff \*staff;//指向Staff的指针

    bool isEmployed;//待解雇用

};

class ManagementSystem{

    private:

    vector<Node> vec;//动态数组

    int StaffNum;//员工总人数

    //private,将功能封装, 用户只能通过SelectFunction()接口调用

    void fileIn();//文件录入

    void showStaff();//显示所有信息

    void showCount();//显示所有员工人数

    void searchStaff();//显示单个人的信息

    void addData();//添加员工

    void markStaffToBeFired();//标记待解雇员工

    void updateStaff();//修改员工信息

    void fireStaffs();//解雇isEmployed为false的员工

    void recombineFile();//保存文件并退出

    //辅助函数

    void hire(string no, string name, int age, string title, double sales = 0);//往vector中新加结点

    bool hasRepeatted(string no);//员工号为码,判断vector中是否已存在相同员工号

    bool inTitles(string title){//判断输出title是否正确

        return (!title.compare("Salesman"))||(!title.compare("Manager"))||(!title.compare("SalesManager"));

    }

    bool hasSales(string title){//判断是否拥有sales这个数据成员

        return ((!title.compare("Salesman")) || (!title.compare("SalesManager")));}

    bool isSalesman(string title){return (!title.compare("Salesman"));}//判断是否是Manager

    bool isManager(string title){return (!title.compare("Manager"));}

    bool isSalesManager(string title){return (!title.compare("SalesManager"));}

    int getStaffNum(){return StaffNum;}

    public:

    ManagementSystem();

    ~ManagementSystem(){}

    void SelectFunction();//选择功能

};

ManagementSystem::ManagementSystem(){

    this->StaffNum = 0;

    fileIn();//读接口

}

void ManagementSystem::fileIn(){

    string no, name, title;

    int age;

    double sales;

    ifstream infile("staff.txt", ios::in);

    if(!infile) cout<<"文件新创建, 请先添加员工再操作!\n"<<endl;

    infile.seekg(0, ios::end); //将文件指针指向文件末端

    streampos fp = infile.tellg(); //fp为文件指针的偏移量

    if (int(fp) == 0) cout<<"文件为空, 请先添加员工再操作!\n"<<endl;

    infile.seekg(0, ios::beg); //将文件指针指向文件末端

    while(infile >> no >> name >> age >> title){ //依次录入数据

        sales = 0;

        if (hasSales(title)) infile >> sales;

        hire(no, name, age, title, sales);

    }

}

//往链表末尾添加一个新结点, 同时StaffNum++;

void ManagementSystem::hire(string no, string name, int age, string title, double sales){

    Node \*newNode = (Node \*) malloc(sizeof(Node)); //分配一块新结点的空间, 并让 newNode指向它

    //根据不同的职位,生成不同的Node

    if (isSalesman(title)) {

        newNode->staff = new Salesman(no, name, age, sales);;

        newNode->isEmployed = true;

    } else if (isManager(title)) {

        newNode->staff = new Manager(no, name, age);

        newNode->isEmployed = true;

    } else if (isSalesManager(title)) {

        newNode->staff = new SalesManager(no, name, age, sales);

        newNode->isEmployed = true;

    }

    else{

        cout<<"输入错误, 请重新输入:"<<endl;

    }

    vec.push\_back(\*newNode);

    StaffNum++;//员工数要记得加1

}

//输出所有员工的信息

void ManagementSystem::showStaff(){

    for(Node it: vec){

            cout << "isEmployed:" << it.isEmployed <<" ";

            it.staff->showData();

    }

}

//判断员工号是否重复

bool ManagementSystem::hasRepeatted(string no){

    vector<Node>::iterator it;

        for(it = vec.begin(); it != vec.end(); it++){

            if(!no.compare(it->staff->getNo())){

                return true;

            }

        }

}

//按员工号或者姓名索引信息

void ManagementSystem::searchStaff(){

    string searchNo, searchName;

    while(1){

    int c;

    cout<<"选择关键词(按0退出):(1)根据员工号(2)根据姓名"<<endl;

    cin>>c;

    if (cin.good() == 1);//判断是否输入的是整数

    else{cout<<"输入错误."<<endl;cin.clear();cin.sync();continue;}

    if(c == 1){

        cout<<"请输入员工号:";

        cin>>searchNo;

        for(Node it: vec){

           if(!searchNo.compare(it.staff->getNo())){

                cout << "isEmployed:" << it.isEmployed <<" ";

                it.staff->showData();

                return;

            }

        }

        cin.ignore(100, '\n');//防止输入多余信息

        cout<<"查无此人."<<endl;

        break;

    }

    else if(c == 2){

        bool flag = false;

        cout<<"请输入员工姓名:";//需考虑可能重名的情况

        cin>>searchName;

        for(Node it: vec){

           if(!searchName.compare(it.staff->getName())){

                cout << "isEmployed:" << it.isEmployed <<" ";

                it.staff->showData();

                flag = true;

            }

        }

        if(flag) return;

        cin.ignore(100, '\n');

        cout<<"查无此人."<<endl;

        break;

    }

    else if(c == 0) return;

    else{

        cout<<"输入错误, 请重新输入"<<endl;

    }

    }

}

//输出人数

void ManagementSystem::showCount(){

    int SalesmanNum=0, ManagerNum=0, SalesmanagerNum=0;

    vector<Node>::iterator it;

    for(it = vec.begin(); it != vec.end(); it++){

        string ss = it->staff->getTitle();

        if(isSalesman(ss)) SalesmanNum++;

        else if(isManager(ss)) ManagerNum++;

        else if(isSalesManager(ss)) SalesmanagerNum++;

        else{}

    }

    cout<<"总人数:"<<getStaffNum()<<endl<<"Salesman人数:"<<SalesmanNum

        <<endl<<"Manager人数:"<<ManagerNum<<endl<<"Salesmanager人数:"<<SalesmanagerNum;

}

void ManagementSystem::addData(){//需考虑员工号重复情况

    int age;

    string no, name, title;

    double sales;

    while(1){

    cout<<"请输入员工号，姓名，年龄，职位，(营销额):(如果想退出请输入'#'"<<endl;

    cin>>no; //判断输入是否为#

    if(!no.compare("#")) return;

    cin>>name;

    cin>>age;//age 需为int型

    if (cin.good() == 1){cin.ignore(100, ' ');cin>>title;}//如果是double型则只取小数点前的

    else{cout<<"年龄需为整数."<<endl;cin.clear();cin.sync();continue;}

    //cin.ignore(100, ' ');//如果是double型则只取小数点前的

    if(inTitles(title));// title只能为3种

    else{cout<<"职位输入错误."<<endl;cin.ignore(100, '\n');continue;}

    if(hasRepeatted(no)){cout<<"员工编号重复."<<endl;continue;}

    if (hasSales(title)){

        while(1){

            cin >> sales;cin.ignore(100, '\n');//防止用户多输入数据

            if (cin.good() == 1);

            else{cout<<"销售额需为复数."<<endl;cin.clear();cin.sync();continue;}

            break;

        }

    }

    else{

        if(cin.eof()){//防止用户多余输入

            cout<<"(该员工无需输入销售额."<<endl;

            cin.ignore(100, '\n');

        }

    }

    hire(no, name, age, title, sales);//录入并退出函数

    break;

    }

    cout << "员工信息已录入" << endl;

}

void ManagementSystem::markStaffToBeFired(){

    string searchNo, searchName;

    while(1){

    cout<<"选择关键词(按0退出):(1)根据员工号(2)根据姓名"<<endl;

    int c;cin>>c;

    if (cin.good() == 1);

    else{cout<<"输入错误."<<endl;cin.clear();cin.sync();continue;}

    if(c == 1){

        cout<<"请输入员工号:";

        cin>>searchNo;

        for(Node &it: vec){//此处需用引用

           if(!searchNo.compare(it.staff->getNo())){

                it.isEmployed = false;

                cout<<"已标记"<<endl;

                return;

            }

        }

        cin.ignore(100, '\n');

        cout<<"查无此人"<<endl;

        break;

    }

    else if(c == 2){

        cout<<"请输入员工姓名:";//需考虑可能重名的情况

        cin>>searchName;

        for(Node &it: vec){

           if(!searchName.compare(it.staff->getName())){

                it.isEmployed = false;

                cout<<"已标记"<<endl;

                return;

            }

        }

        cin.ignore(100, '\n');

        cout<<"查无此人"<<endl;

        break;

    }

    else if(c == 0){return;}

    else{

        cout<<"输入错误, 请重新输入"<<endl;

    }

    }

}

void ManagementSystem::updateStaff(){

    int c, age;

    string searchNo, no, name, title;

    double sales;

    cout<<"请输入员工号:";

    cin>>searchNo;

    vector<Node>::iterator it;

    for(it = vec.begin(); it != vec.end();){

        if(!searchNo.compare(it->staff->getNo())){

            no = it->staff->getNo();

            name = it->staff->getName();

            age = it->staff->getAge();

            while(1){

                while(1){

                    cout<<"1.员工号2.姓名3.年龄4.职位(5.营销额)\n要修改(退出请输入0):";

                    cin>>c;

                    if (cin.good() == 1);//输入需为整数

                    else{cout<<"输入错误."<<endl;cin.clear();cin.sync();continue;}

                    break;

                }

            if(c == 0) return;//是0则退出

            switch (c)

            {

            case 1:

                cout<<"请输入员工号:";cin>>no;

                cin.ignore(100, '\n');

                it->staff->setNo(no);

                break;

            case 2:

                cout<<"请输入员工姓名:";cin>>name;

                cin.ignore(100, '\n');

                it->staff->setName(name);

                break;

            case 3:

                while(1){

                    cout<<"请输入员工年龄:";cin>>age;

                    if (cin.good() == 1) cin.ignore(100, ' ');

                    else{cout<<"年龄需为整数."<<endl;cin.clear();cin.sync();continue;}

                    break;

                }

                it->staff->setAge(age);

                break;

            case 4:

                while(1){

                cout<<"请输入员工职位:";cin>>title;

                cin.ignore(100, '\n');

                if(inTitles(title)) ;// title只能为3种

                else{cout<<"职位输入错误."<<endl;cin.ignore(100, '\n');continue;}

                break;

                }

                if(isManager(it->staff->getTitle())){

                    if(isManager(title)){cout<<"职位重复"<<endl;return;}

                    else{

                        cout<<"请再输入其销售额:";

                        cin>>sales;

                        Node \*newNode = (Node \*) malloc(sizeof(Node)); //分配一块新结点的空间, 并让newNode指向它

                        if (isSalesman(title)) {

                            newNode->staff = new Salesman(no, name, age, sales);

                            newNode->isEmployed = true;

                        }

                        else{

                            newNode->staff = new SalesManager(no, name, age, sales);

                            newNode->isEmployed = true;

                        }

                        it = vec.erase(it); //返回指向被删除元素的下一个的指针

                        it = vec.insert(it, \*newNode);//往指向元素前增加一个元素

                    }

                }

                else{

                    if(!title.compare(it->staff->getTitle())){

                        cout<<"职位重复"<<endl;

                        return;

                    }

                    else{

                        if(isManager(title)){

                            Node \*newNode = (Node \*) malloc(sizeof(Node));

                            newNode->staff = new Manager(no, name, age);

                            newNode->isEmployed = true;

                            it = vec.erase(it); //返回指向被删除元素的下一个的指针

                            it = vec.insert(it, \*newNode);//往指向元素前增加一个元素

                        }

                        else if(isSalesman(title)){

                            Node \*newNode = (Node \*) malloc(sizeof(Node));

                            newNode->staff = new Salesman(no, name, age, it->staff->getSales());

                            newNode->isEmployed = true;

                            it = vec.erase(it);

                            it = vec.insert(it, \*newNode);

                        }

                        else{

                            Node \*newNode = (Node \*) malloc(sizeof(Node));

                            newNode->staff = new SalesManager(no, name, age, it->staff->getSales());

                            newNode->isEmployed = true;

                            it = vec.erase(it);

                            it = vec.insert(it, \*newNode);

                        }

                    }

                }

                break;

            case 5:

                if(isManager(it->staff->getTitle())){

                    cout<<"经理无销售额一说."<<endl;

                    return;

                }

                else{

                    cout<<"请输入销售额:";cin>>sales;

                    cin.ignore(100, '\n');

                    it->staff->setSales(sales);

                }

                break;

            default:

                break;

            }

            cout << "员工信息已修改." << endl;

            return;

            }

        }

        else it++;

    }

    cout<<"查无此人"<<endl;

}

void ManagementSystem::fireStaffs(){

    vector<Node>::iterator it;

    for(it = vec.begin(); it != vec.end();){

        if(it->isEmployed == false){

            it = vec.erase(it); //返回指向被删除元素的下一个的指针

        }//如果删除的是最后一个元素则指向删除后的最后一个

        else it++;

    }

    cout<<"完成操作"<<endl;

}

void ManagementSystem::recombineFile(){

    fireStaffs();//保存前删除待解雇员工

    ofstream outfile("staff.txt", ios::out | ios::trunc);//重写文件

    vector<Node>::iterator it;

    for(it = vec.begin(); it != vec.end(); it++){

        outfile<<it->staff->getNo()<<" "<<it->staff->getName()

        <<" "<<it->staff->getAge()<<" "<<it->staff->getTitle()<<" ";

        if(!isManager(it->staff->getTitle()))

            outfile<<it->staff->getSales()<<endl;

        else outfile<<endl;

    }

    outfile.close();

}

void ManagementSystem::SelectFunction(){

    int c, age;

    string no, name, title;

    double sales;

    cout<<"欢迎使用销售公司员工管理系统!"<<endl;

    while(true){

        cout<<

        " -------------------------------------------------\n";

        cout<<

        "    选择功能:(1)查询所有员工信息 (2)查询员工信息\n"<<

        " (3)查询人数 (4)标记待解雇员工   (5)解雇已标记员工\n"<<

        " (6)添加员工 (7)修改员工信息     (0)保存并退出    \n"<<

        " -------------------------------------------------"<<endl;

        cout<<"选择:";

        cin>>c;

        if (cin.good() == 1);

        else{cout<<"输入错误, 请重新输入."<<endl;cin.clear();cin.sync();continue;}

        switch (c)

        {

        case 1:

            showStaff();cout<<endl;

            break;

        case 2:

            searchStaff();cout<<endl;

            break;

        case 3:

            showCount();cout<<endl;

            break;

        case 4:

            markStaffToBeFired();

            break;

        case 5:

            fireStaffs();cout<<endl;

            break;

        case 6:

            addData();cout<<endl;

            break;

        case 7:

            updateStaff();cout<<endl;

            break;

        case 0:

            recombineFile();cout<<endl;

            break;

        default:

            cout<<"输入错误, 请重新输入."<<endl;

            cin.ignore(100, '\n');//防止错误输入 清除缓冲区

            break;

        }

        if(c == 0) break;

    }

}

int main(){

    ManagementSystem ms;

    ms.SelectFunction();

    return 0;

}