# 职工管理系统

## 1. 管理系统需求

职工管理系统可以用来管理公司内所有员工的信息

本教程主要利用C++来实现一个基于多态的职工管理系统

公司中职工分为三类:普通员工、经理、老板，显示信息时，需要显示职工编号、职工姓名、职工岗位、以及职责普通员工职责:完成经理交给的任务

经理职责:完成老板交给的任务，并下发任务给员工

老板职责:管理公司所有事务

**管理系统中需要实现的功能如下:**

- 退出管理程序：退出当前管理系统
- 增加职工信息：实现批量添加职工功能，将信息录入到文件中，职工信息为：职工编号、姓名、部门编号
- 显示职工信息：显示公司内部所有职工信息
- 删除离职职工：按照编号删除指定的职工
- 修改职工信息：按照编号修改职工个人信息
- 查找职工信息：按照职工的编号或者职工的姓名进行查找相关的人员信息
- 按照编号排序：按照职工编号，进行排序，排序规则由用户指定
- 清空所有文档：清空文件中记录的所有职工信息（清空前需要再次确认，防止误删）

## 2. 创建项目

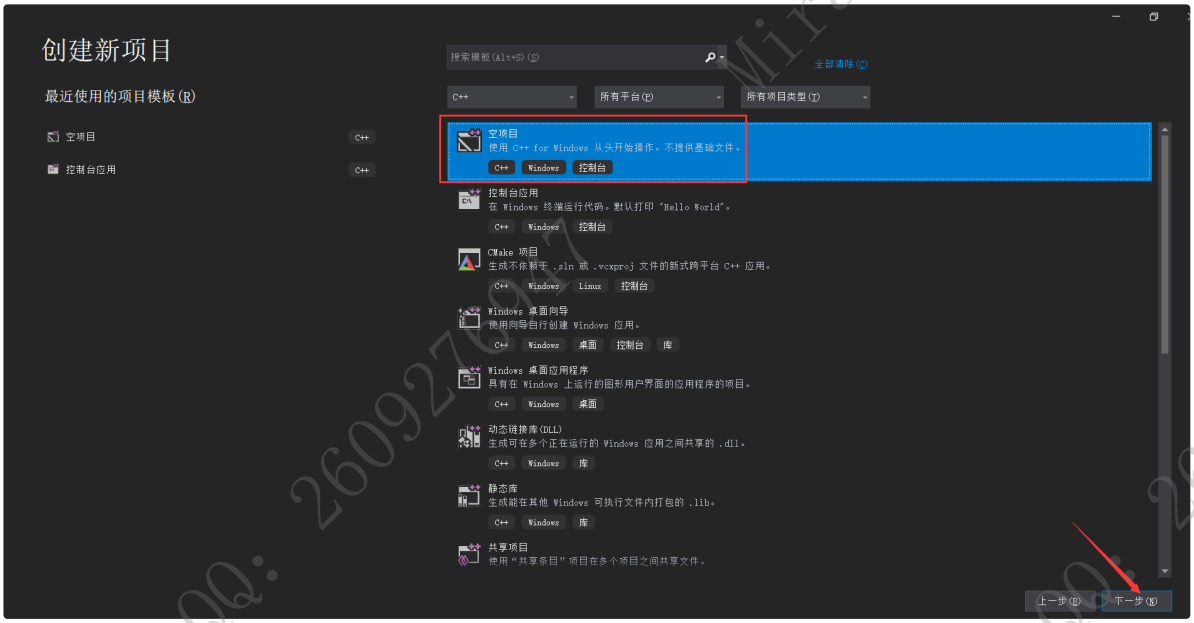
创建项目步骤如下:

- 创建新项目
- 添加文件

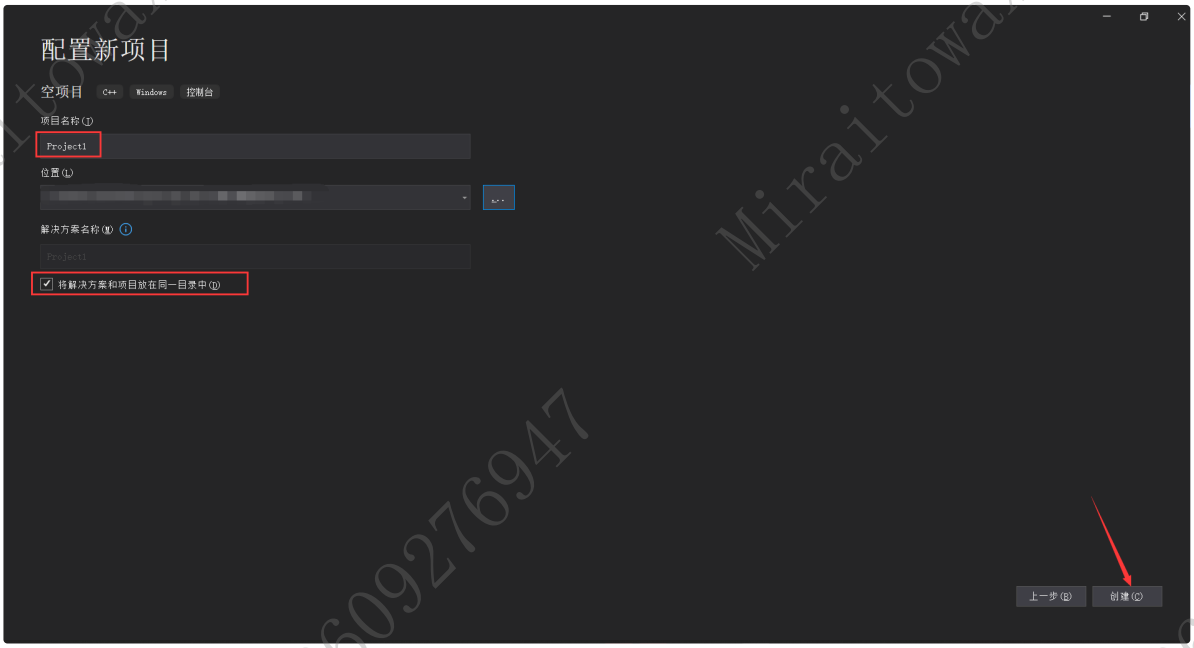
### 2.1 创建项目

打开Visual Studio 2019后，点击创建新项目，创建新的C++项目


选择空项目，点击下一步

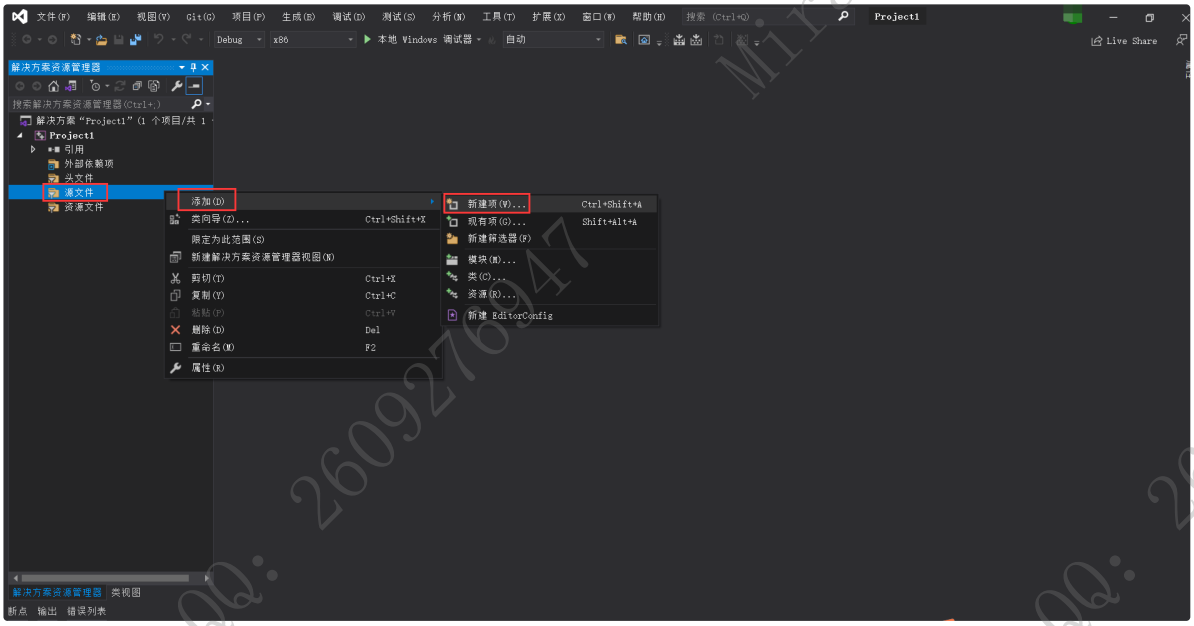



填写项目名称及项目路径，点击创建

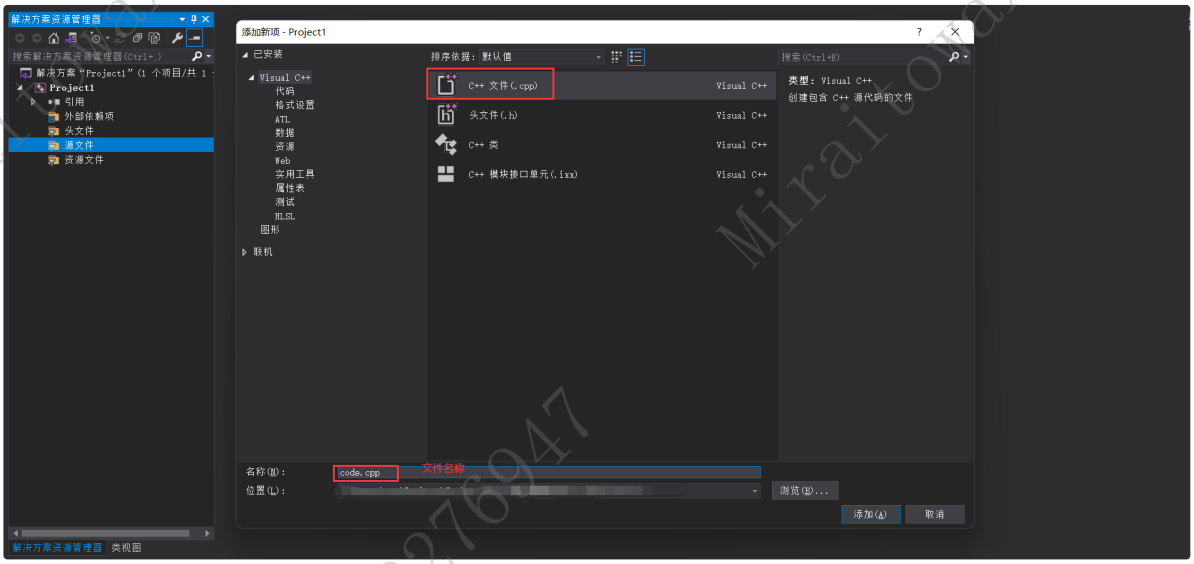


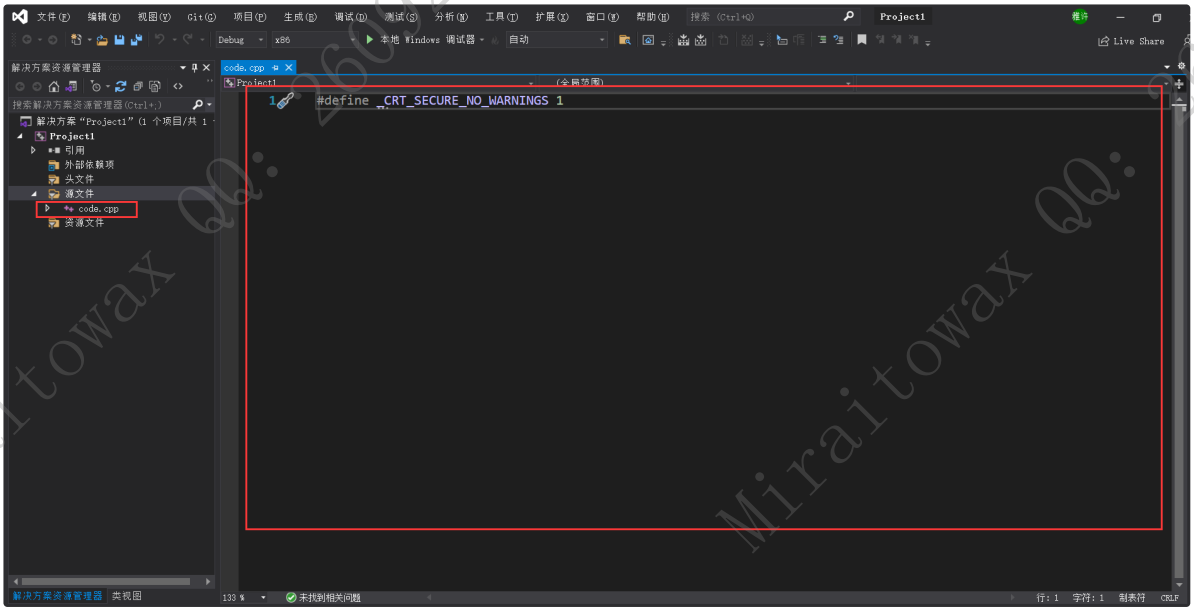

## 2.2 添加文件

右击源文件，进行添加文件操作




添加文件名称




至此，项目创建完毕

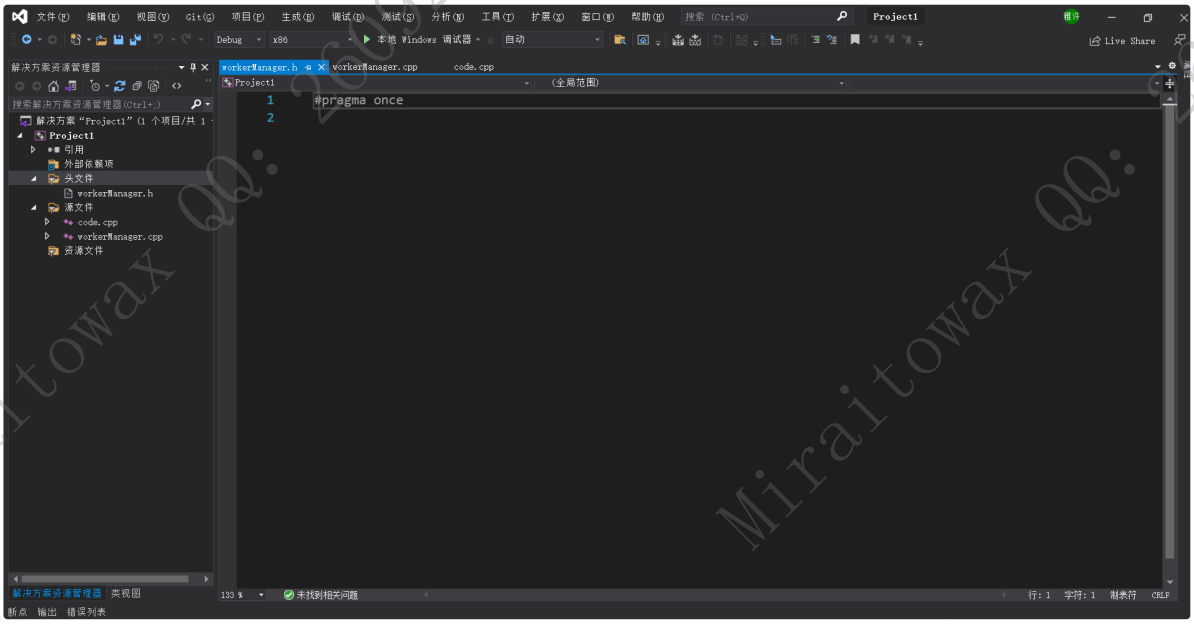
# 3. 创建管理类

管理类负责的内容如下：

- 与用户的沟通菜单
- 对职工增删改查的操作
- 与文件的读写交互

## 3.1 创建文件

在头文件和源文件的文件夹下分别创建workerManager.h和workManager.cpp文件




## 3.2 头文件的实现

在workerManager.h中设计管理类

```
#pragma once//防止头文件重复包含
#include<iostream>//包含输入输出流文件
using namespace std;//使用标准的命名空间

class WorkerManager {
public:
    WorkerManager();
    ~WorkerManager();
};
```

## 3.3 源文件实现

在workerManager.cpp中将构造和析构函数空实现补全

```cpp
#define _CRT_SECURE_NO_WARNINGS 1
#include"workerManager.h"
WorkerManager::WorkerManager() {

}
WorkerManager::~WorkerManager() {

}
```

# 4. 菜单功能

功能：与用户的沟通界面

## 4.1 添加成员函数

在管理类workManager.h中添加成员函数void Show_Menu();

```cpp
    //展示菜单
    void Show_Menu();
```

## 4.2 菜单功能实现

在管理类workerManager.cpp中实现Show_Menu()函数

```cpp
void WorkerManager::Show_Menu() {
    cout << "********************************************" << endl;
    cout << "********** 欢迎使用职工管理系统！**********" << endl;
    cout << "************* 0.退出管理程序 *************" << endl;
    cout << "************* 1.增加职工信息 *************" << endl;
    cout << "************* 2.显示职工信息 *************" << endl;
    cout << "************* 3.删除离职信息 *************" << endl;
    cout << "************* 4.修改职工信息 *************" << endl;
    cout << "************* 5.查找职工信息 *************" << endl;
    cout << "************* 6.按照编号排序 *************" << endl;
    cout << "************* 7.清空所有文档 *************" << endl;
    cout << endl;
}
```

### 4.3 测试菜单功能

在main.cpp中测试菜单功能

```cpp
#define _CRT_SECURE_NO_WARNINGS 1
#include<iostream>
#include"workerManager.h"
using namespace std;

int main() {
    //实例化管理者对象
    WorkerManager wm;
    wm.Show_Menu();
    system("pause");
    return 0;
}
```

## 5. 退出功能

### 5.1 提供功能接口

在main函数中提供分支选择，提供每个功能接口

```cpp
int main() {
    //实例化管理者对象
    WorkerManager wm;
    int choice = 0;//用来存储用户的选项
    while (true) {
        //调用展示菜单成员函数
        wm.Show_Menu();
        cout << "请输入您的选择: " << endl;
        cin >> choice;//接收用户选项
        switch (choice) {
        case 0://退出系统
            wm.ExitSystem();
            break;
        case 1://增加职工
            break;
```

```
        case 2://显示职工
            break;
        case 3://删除职工
            break;
        case 4://修改职工
            break;
        case 5://查找职工
            break;
        case 6://排序职工
            break;
        case 7://清空职工
            break;
        default:
            system("cls");//清屏
            break;
        }
    }
    system("pause");
    return 0;
}
```

## 5.2 实现退出功能

在`workerManager.h`中提供退出系统的成员函数`void ExitSystem();`

在`workerManager.cpp`中提供具体的功能实现

```
//退出系统
void WorkerManager::ExitSystem() {
    cout << "欢迎下次使用! " << endl;
    system("pause");
    exit(0);//退出程序
}
```

### 5.3 测试功能

在main函数分支0中，调用退出程序的接口

## 6. 创建职工类

### 6.1 创建职工抽象类

职工的分类：普通员工、经理、老板

将三种职工抽象到一个类（worker）中，利用多态管理不同职工种类

职工的属性为：职工编号、职工姓名、职工所在部门编号

职工的行为为：岗位职责信息描述，获取岗位名称

头文件文件夹下，创建文件`work.h`文件并且添加如下代码：

```
#pragma once
#include<iostream>
#include<string>
using namespace std;

//职工抽象类
class Worker {
public:
    //显示个人信息
    virtual void showInfo() = 0;
    //获取岗位名称
    virtual void getDeptName() = 0;

    int m_Id;//职工编号
    string m_Name;//职工姓名
    int m_DeptId;//职工所在部门名称编号
};
```

### 6.2 创建普通员工类

普通员工类继承职工抽象类，并重写父类中的纯虚函数

在头文件和源文件的文件夹下分别创建`employee.h`和`employee.cpp`文件

`employee.h`中代码如下：

```
#pragma once
#include"worker.h"

```

```cpp
#include<iostream>
using namespace std;

//员工类
class Employee :public Worker {
public:
    //构造函数
    Employee(int id, string name, int dId);
    //显示个人信息
    virtual void showInfo();
    //获取职工岗位名称
    virtual void getDeptName();
};
```

employee.cpp 中代码如下：

```cpp
#include<iostream>
using namespace std;

Employee::Employee(int id, string name, int dId) {
    this->m_Id = id;
    this->m_Name = name;
    this->m_DeptId = dId;
}
string Employee::getDeptName() {
    return string("员工");
}
void Employee::showInfo() {
    cout << "职工编号: " << this->m_Id
        << " \t职工姓名: " << this->m_Name
        << " \t岗位: " << this->getDeptName()
        << " \t岗位职责: 完成经理交给的任务" << endl;
}
```

## 6.3 测试代码

在main.cpp函数中添加以下代码进行测试：

```
//测试代码
Worker* worker = NULL;
worker = new Employee(1, "张三", 1);
worker->showInfo();
```

## 6.4 创建经理类

经理类继承职工抽象类，并重写父类中的纯虚函数，和普通员工类似

在头文件和源文件的文件夹分别创建manager.h和manager.cpp文件

manager.h中代码如下：

```
#pragma once
#include"worker.h"
#include<iostream>
using namespace std;
//经理类
class Manager : public Worker {
public:
    //构造函数
    Manager(int id, string name, int dId);
    //显示个人信息
    virtual void showInfo();
    //获取岗位名称
    virtual string getDeptName();
};
```

manager.cpp代码如下：

```
#define _CRT_SECURE_NO_WARNINGS 1
#include"manager.h"
//构造函数
Manager::Manager(int id, string name, int dId) {
    this->m_Id = id;
    this->m_Name = name;
    this->m_DeptId = dId;

```

```
}
//显示个人信息
void Manager::showInfo() {
    cout << "职工编号: " << this->m_Id
        << " \t职工姓名: " << this->m_Name
        << " \t岗位: " << this->getDeptName()
        << " \t岗位职责: 完成老板交给的任务，并且下发任务给普通员工" << endl;
}
//获取岗位名称
string Manager::getDeptName() {
    return string("经理");
}
```

## 6.5 创建老板类

老板类继承职工抽象类，并重写父类中的纯虚函数，和普通员工类似

在头文件和源文件的文件夹分别创建boss.h和boss.cpp文件

boss.h中代码如下：

```
#pragma once
#include"worker.h"
#include<iostream>
using namespace std;
//老板类
class Boss : public Worker {
public:
    //构造函数
    Boss(int id, string name, int dId);
    //显示个人信息
    virtual void showInfo();
    //获取岗位名称
    virtual string getDeptName();
};

```

boss.cpp中代码如下：

```
#define _CRT_SECURE_NO_WARNINGS 1
```

```cpp
#include"boss.h"
//构造函数
Boss::Boss(int id, string name, int dId) {
    this->m_Id = id;
    this->m_Name = name;
    this->m_DeptId = dId;
}
//显示个人信息
void Boss::showInfo() {
    cout << "职工编号: " << this->m_Id
        << " \t职工姓名: " << this->m_Name
        << " \t岗位: " << this->getDeptName()
        << " \t岗位职责: 管理公司所有事务" << endl;
}
//获取岗位名称
string Boss::getDeptName() {
    return string("老板");
}
```

## 6.6 测试多态

在职工管理系统main.cpp添加测试函数，并且运行能够产生多态

测试代码如下：

```cpp
Worker* worker2 = NULL;
worker2 = new Manager(2, "李四", 2);
worker2->showInfo();
delete worker2;

Worker* worker3 = NULL;
worker3 = new Boss(3, "王五", 3);
worker3->showInfo();
delete worker3;
```

测试成功后，代码可以保留或者注释

# 7. 添加职工

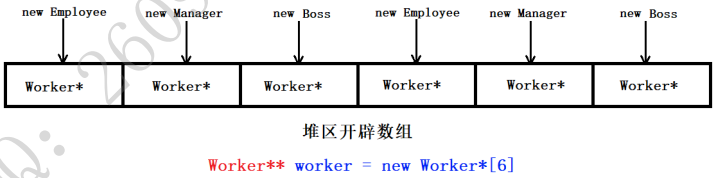
功能描述：批量添加职工，并且保存到文件中

## 7.1 功能分析

分析：用户在批量创建时，可能会创建不同种类的员工

如果想将所有不同种类的员工都放到一个数组中，可以将所有员工的指针维护到一个数组里

如果想在程序中维护这个不定长度的数组，可以将数组创建到堆区，并利用`Woker**的指针维护`




## 7.2 功能实现

在WorkManager.h头文件中添加成员属性代码：

```
//记录文件中的人数个数
int m_EmpNum;
//员工数组的指针
Worker** m_EmpArray;
```

在WorkerManager构造函数中初始化属性

```
WorkerManager::WorkerManager() {
    //初始化属性
    this->m_EmpNum = 0;
    this->m_EmpArray = NULL;
}
```

在workerManager.h中添加成员函数

```
//添加职工
void Add_Emp();
```

workerManager.cpp中实现该函数

```
//添加职工
void WorkerManager::Add_Emp() {
    cout << "请输入添加职工数量: " << endl;
    int addNum = 0;//保存用户的输入
    cin >> addNum;
```

```
    if (addNum > 0) {
        //添加
        //计算添加新的空间大小
        int newSize = this->m_EmpNum + addNum;//新空间人数大小 = 原来的人数大小
+ 新增加的人数
        //开辟新空间
        Worker** newSpace = new Worker * [newSize];
        //将原来空间下的数据拷贝到新空间下
        if (this->m_EmpArray != NULL) {
            for (int i = 0; i < this->m_EmpNum; i++) {
                newSpace[i] = this->m_EmpArray[i];
            }
        }
        //批量添加新数据
        for (int i = 0; i < newSize; i++) {
            int id;//职工编号
            string name;//职工姓名
            int dSelect;//部门选择
            cout << "请输入第" << i + 1 << "个新职工编号: " << endl;
            cin >> id;
            cout << "请输入第" << i + 1 << "个新职工姓名: " << endl;
            cin >> name;
            cout << "请选择该职工岗位: " << endl;
            cout << "1. 普通职工" << endl;
            cout << "2. 经理" << endl;
            cout << "3. 老板" << endl;
            cin >> dSelect;

            Worker* worker = NULL;
            switch (dSelect) {
            case 1:
                worker = new Employee(id, name, 1);
                break;
            case 2:
```

```
                worker = new Manager(id, name, 2);
                break;
            case 3:
                worker = new Boss(id, name, 3);
                break;
            }
            //将创建职工，保存到数组中
            newSpace[this->m_EmpNum + i] = worker;
        }
        //释放原有的空间
        delete[] this->m_EmpArray;
        //更改新空间指向
        this->m_EmpArray = newSpace;
        //更新新的职工人数
        this->m_EmpNum = newSize;
        //成功添加后，保存数据到文件中
        this->save();
        //提示添加成功
        cout << "成功添加" << addNum << "名职工" << endl;
    }
    else {
        cout << "输入数据有误,请重新输入" << endl;
    }
    //按任意键清屏，回到上级目录
    system("pause");
    system("cls");
}
```

在main.cpp中，实现析构函数

```cpp
WorkerManager::~WorkerManager() {
    if (this->m_EmpArray != NULL) {
        delete[] this->m_EmpArray;
        this->m_EmpArray = NULL;
    }
}
```

# 8. 文件交互 - 写文件

功能描述：对文件进行读写

在上一个添加功能中，我们只是将所有的数据添加到了内存中，一旦程序结束就无法保存了

因此文件管理类中需要一个与文件进行交互的功能，对于文件进行读写

## 8.1 设定文件路径

首先我们将文件路径，在`workerMangager.h`中添加宏常量，并且包含头文件`fstream`

```cpp
#include<fstream>
#define FILENAME "empFile.txt"
```

## 8.2 成员函数声明

在`workerManager.h`中类里添加成员函数`void save()`

```cpp
//保存文件
void save();
```

## 8.3 保存文件功能实现

```cpp
//保存文件
void WorkerManager::save() {
    ofstream ofs;
    ofs.open(FILENAME, ios::out);//用输出的方式打开文件
    for (int i = 0; i < this->m_EmpNum; i++) {
        ofs << this->m_EmpArray[i]->m_Id << " "
            << this->m_EmpArray[i]->m_Name << " "
            << this->m_EmpArray[i]->m_DeptId << " "
            << endl;
    }
    //关闭文件
    ofs.close();
}
```

# 9. 文件交互 - 读文件

功能描述:将文件中的内容读取到程序中

虽然我们实现了添加职工后保存到文件的操作，但是每次开始运行程序，并没有将文件中数据读取到程序中

而我们的程序功能中还有清空文件的需求

因此构造函数初始化数据的情况分为三种

1. 第一次使用，文件未创建
2. 文件存在，但是数据被用户清空
3. 文件存在，并且保存职工的所有数据

## 9.1 文件未创建

在`workerManager.h`中添加新的成员属性`m_FilesEmpty`标志文件是否为空

```cpp
//标志文件是否为空
bool m_FilesEmpty;
```

修改`WorkerManager.cpp`中构造函数代码

```cpp
WorkerManager::WorkerManager() {
    //1. 文件不存在
    ifstream ifs;
    ifs.open(FILENAME, ios::in);//读文件
    if (!ifs.is_open()) {
        cout << "文件不存在" << endl;
        //初始化属性
```

```
        //初始化记录人数
        this->m_EmpNum = 0;
        //初始化数组指针
        this->m_EmpArray = NULL;
        //初始化文件是否为空
        this->m_FileIsEmpty = true;
        ifs.close();
        return;
    }
}
```

删除文件后，测试文件不存在时初始化数据功能

## 9.2 文件存在且数据为空

在workerManager.cpp中的构造函数追加代码：

```
//2. 文件存在数据为空
char ch;
ifs >> ch;//将文件尾标志读走
if (ifs.eof()) {//ifs.eof()为真，代表文件读完
    //文件为空
    cout << "文件为空! " << endl;
    //初始化记录人数
    this->m_EmpNum = 0;
    //初始化数组指针
    this->m_EmpArray = NULL;
    //初始化文件是否为空
    this->m_FileIsEmpty = true;
    ifs.close();
    return;
}
```

在每次添加员工之后，增加一行代码，更新职工不为空的标志

```
//更新职工不为空的标志
this->m_FileIsEmpty = false;
```

添加员工，手动删除文件内容，测试代码

## 9.3 文件存在且保存职工数据

### 9.3.1 获取记录的职工人数

在`workerManager.h`中添加成员函数`int get_EmpNum();`

```
//统计人数
int get_EmpNum();
```

`workerManager.cpp`中实现

```
//统计文件中人数
int WorkerManager::getEmpNum() {
    ifstream ifs;
    ifs.open(FILENAME, ios::in);//打开文件
    int id;
    string name;
    int dId;
    int num = 0;
    while (ifs >> id && ifs >> name && ifs >> dId) {
        //统计人数变量
        num++;
    }
    return num;
}
```

`WorkerManager`构造函数中添加：

```
//3. 当文件存在，并且记录数据
int num = this->getEmpNum();
cout << "职工人数为: " << num << endl;
this->m_EmpNum = num;
```

文本文档中手动添加数据，进行测试

### 9.3.2 初始化数组

根据职工的数据以及职工数据，初始化`workerManager`中的`Worker** m_EmpArray`指针

在`WorkerManager.h`中添加成员函数`void init_Emp();`

```
//初始化员工
void init_Emp();
```

在WorkerManager.cpp实现

```cpp
//初始化数组
void WorkerManager::init_Emp() {
    ifstream ifs;
    ifs.open(FILENAME, ios::in);
    int id;
    string name;
    int dId;
    int index = 0;
    while (ifs >> id && ifs >> name && ifs >> dId) {
        Worker* worker = NULL;
        if (dId == 1) {
            worker = new Employee(id, name, dId);
        }
        else if (dId == 2) {
            worker = new Manager(id, name, dId);
        }
        else if (dId == 3) {
            worker = new Boss(id, name, dId);
        }
        this->m_EmpArray[index] = worker;
        index++;
    }
    ifs.close();
}
```

WorkerManager构造函数中添加:

```
//3. 当文件存在，并且记录数据
    int num = this->getEmpNum();
    cout << "职工人数为: " << num << endl;
    this->m_EmpNum = num;
    //开辟空间
    this->m_EmpArray = new Worker * [this->m_EmpNum];
    //将文件中的数据，存到数组中
    this->init_Emp();
    ////测试代码:
    //for (int i = 0; i < this->m_EmpNum; i++) {
    //  cout << "职工编号: " << this->m_EmpArray[i]->m_Id << " " << "姓名: " << this->m_EmpArray[i]->m_Name << " " << "部门编号" << this->m_EmpArray[i]->m_DeptId << endl;
    //}
```

# 10. 显示职工

功能描述：显示当前所有职工信息

## 10.1 显示职工函数声明

在WorkerManager.h中添加成员函数void Show_Emp();

```
//显示员工
void Show_Emp();
```

## 10.2 显示职工函数实现

在WorkerManager.cpp中实现成员函数void Show_Emp();

```
//显示职工函数
void WorkerManager::Show_Emp() {
    //判断文件是否为空或者文件是否存在
    if (this->m_FileIsEmpty) {
        cout << "文件不存在或记录为空! " << endl;
    }
    else {
        for (int i = 0; i < m_EmpNum; i++) {
            //利用多态调用程序接口
```

```
            this->m_EmpArray[i]->showInfo();
        }
    }
    //按任意键后清屏
    system("pause");
    system("cls");
}
```

# 11. 删除职工

功能描述：按照职工的编号进行删除职工操作

## 11.1 删除职工函数声明

在wokerManager.h中添加成员函数`void Del_Emp();`

```
//删除职工
void Del_Emp();
```

## 11.2 职工是否存在函数声明

很多功能都需要用到根据职工是否存在来进行操作如：删除职工、修改职工、查找职工

因此添加该公告函数，以便后续调用

在wokerManager.h中添加成员函数`int IsExit(int id);`

```
//按照职工编号判断职工是否存在，若存在则返回职工在数组中的位置，不存在返回-1
```

## 11.3 职工是否存在函数实现

在WorkerManager.cpp中实现成员函数`int IsExit(int id);`

```cpp
//判断职工是否存在，若存在则返回职工在数组中的位置，不存在返回-1
int WorkerManager::IsExit(int id) {
    int index = -1;
    for (int i = 0; i < this->m_EmpNum; i++) {
        if (this->m_EmpArray[i]->m_Id == id) {
            index = i;
            break;
        }
    }
    return index;
}
```

## 11.4 删除职工函数实现

在WorkerManager.cpp中实现成员函数void Del_Emp();

```cpp
//删除职工
void WorkerManager::Del_Emp() {
    if (this->m_FileIsEmpty) {
        cout << "文件不存在或者记录为空！" << endl;
    }
    else {
        //按照职工编号删除
        cout << "请输入想要删除的职工编号：" << endl;
        int id = 0;
        cin >> id;
        int index = this->IsExit(id);
        if (index != -1) {//说明职工存在，并且要删除index位置上的职工
            for (int i = index; i < this->m_EmpNum - 1; i++) {
                //数据前移
                this->m_EmpArray[i] = this->m_EmpArray[i + 1];
            }
            this->m_EmpNum--;
            this->save();
            cout << "删除成功" << endl;
        }
```

```
        else {
            cout << "删除失败，未找到该职工" << endl;
        }
    }
    //按任意键清屏
    system("pause");
    system("cls");
}
```

main.cpp 文件中添加以下代码：

```
wm.Del_Emp();
```

测试代码

# 12. 修改职工

功能描述：能够按照职工的编号对职工信息进行修改并保存

## 12.1 修改职工函数声明

在 workerManager.h 中添加成员函数 void Mod_Emp();

```
//修改职工
void Mod_Emp();
```

## 12.2 修改职工函数实现

在 workerManager.cpp 中实现成员函数 void Mod_Emp();

```
//修改职工
void WorkerManager::Mod_Emp() {
    if (this->m_FileIsEmpty) {
        cout << "文件不存在或记录为空！" << endl;
    }
    else {
        cout << "请输入修改职工的编号：" << endl;
        int id;
        cin >> id;
        int ret = this->IsExit(id);
        if (ret != -1) {
```

```
            //查找到编号
            delete this->m_EmpArray[ret];
            int newid = 0;
            string newName = "";
            int dSelect = 0;
            cout << "查找到编号为: " << id << "的职工, 请输入新的职工号: " <<
endl;
            cin >> newid;
            cout << "请输入新姓名: " << endl;
            cin >> newName;
            cout << "请输入岗位: " << endl;
            cout << "1. 普通员工" << endl;
            cout << "2. 经理" << endl;
            cout << "3. 老板" << endl;
            cin >> dSelect;
            Worker* worker = NULL;
            switch (dSelect) {
            case 1:
                worker = new Employee(newid, newName, dSelect);
                break;
            case 2:
                worker = new Manager(newid, newName, dSelect);
                break;
            case 3:
                worker = new Boss(newid, newName, dSelect);
                break;
            default:
                break;
            }
            //更新数据到数组中
            this->m_EmpArray[ret] = worker;
            cout << "修改成功" << endl;
            //保存到文件中
            this->save();
```

```
        }
        else {
            cout << "修改失败，查无此人，请重新输入！" << endl;
        }
    }
    //按任意键清屏
    system("pause");
    system("cls");
}
```

main.cpp 文件中添加以下代码：

```
wm.Mod_Emp();
```

测试代码

# 13. 查找职工

功能描述：提供两种查找职工方式，一种按照职工编号，一种按照职工姓名

## 13.1 查找职工函数声明

在 workerManger.h 中添加成员函数 void Find_Emp();

```
//查找职工
void Find_Emp();
```

## 13.2 查找职工函数实现

在 workerManger.cpp 中实现成员函数 void Find_Emp();

```
//查找职工
void WorkerManager::Find_Emp() {
    if (this->m_FileIsEmpty) {
        cout << "文件不存在或者记录为空！" << endl;
    }
    else {
        cout << "请输入查找的方式：" << endl;
        cout << "1. 按照职工的编号查找" << endl;
        cout << "2. 按照职工的姓名查找" << endl;
        int select = 0;
```

```
        cin >> select;
        if (select == 1) {
            //按照编号查询
            int id = 0;
            cout << "请输入查找的职工的编号: " << endl;
            cin >> id;
            int ret = this->IsExit(id);
            if (ret != -1) {
                cout << "查找成功，该员工信息如下: " << endl;
                this->m_EmpArray[ret]->showInfo();
            }
            else {
                cout << "查无此人" << endl;
            }
        }
        else if (select == 2) {
            //按照姓名查询
            string name;
            cout << "请输入查找的姓名: " << endl;
            cin >> name;
            //加入判断是否查到的标志
            bool flag = false;//默认未找到职工
            for (int i = 0; i < m_EmpNum; i++) {
                if (m_EmpArray[i]->m_Name == name) {
                    cout << "查找成功，职工编号为: "<<m_EmpArray[i]->m_Id<<"号
职工信息如下: " << endl;
                    flag = true;
                    //调用显示信息接口
                    this->m_EmpArray[i]->showInfo();
                }
            }
            if (flag == false) {
                cout << "查无此人" << endl;
            }
```

```
        }
        else {
            cout << "输入选项有误，请重新输入！" << endl;
        }
    }
    //按任意键清屏
    system("pause");
    system("cls");
}
```

main.cpp 文件中添加以下代码：

```
wm.Find_Emp();
```

测试代码

# 14. 排序

功能描述：按照职工编号进行排序，排序的顺序由用户指定

## 14.1 排序函数声明

在 workerManager.h 中添加成员函数 void Sort_Emp();

```
//排序员工
void Sort_Emp();
```

## 14.2 排序函数实现

在 workerManager.cpp 中实现成员函数 void Sort_Emp();

```
//排序员工
void WorkerManager::Sort_Emp() {
    if (this->m_FileIsEmpty) {
        cout << "文件不存在或者记录为空！" << endl;
        //按任意键清屏
        system("pause");
        system("cls");
    }
    else {
        cout << "请选择排序方式：" << endl;
```

```
        cout << "1. 按照职工号升序排列" << endl;
        cout << "2. 按照职工号降序排列" << endl;
        int select = 0;
        cin >> select;
        for (int i = 0; i < m_EmpNum; i++) {
            int minOrMax = i;//声明最小值或最大值下标
            for (int j = i + 1; j < m_EmpNum; j++) {
                if (select == 1) {//升序
                    if (this->m_EmpArray[minOrMax]->m_Id > this->m_EmpArray[j]->m_Id) {
                        minOrMax = j;
                    }
                }
                else {//降序
                    if (this->m_EmpArray[minOrMax]->m_Id < this->m_EmpArray[j]->m_Id) {
                        minOrMax = j;
                    }
                }
            }
            //判断一开始认定的最小值或最大值是不是计算的最小值或最大值
            if (i != minOrMax) {
                Worker* temp = this->m_EmpArray[i];
                this->m_EmpArray[i] = this->m_EmpArray[minOrMax];
                this->m_EmpArray[minOrMax] = temp;
            }
        }
        cout << "排序成功！排序后的结果为：" << endl;
        this->save();//排序后的结果保存到文件中
        this->Show_Emp();
    }
}
```

main.cpp 文件中添加以下代码：

```
wm.Sort_Emp();
```

测试代码

# 15. 清空文件

功能描述：将文件中记录数据全部清空

## 15.1 清空函数声明

在workerManager.h中添加成员函数void Clean_File();

```
//清空文件
void Clean_File();
```

## 15.2 清空函数实现

在workerManager.cpp中实现成员函数void Clean_File();

```
//清空文件
void WorkerManager::Clean_File() {
    cout << "确定清空? " << endl;
    cout << "1. 确定" << endl;
    cout << "2. 返回" << endl;
    int select = 0;
    cin >> select;
    if (select == 1) {
        //清空文件
        //打开模式ios::trunc 如果存在文件，删除文件后重新创建
        ofstream ofs(FILENAME, ios::trunc);
        ofs.close();
        if (this->m_EmpArray != NULL) {
            //删除堆区的每个职工对象
            for (int i = 0; i < this->m_EmpNum; i++) {

                delete this->m_EmpArray[i];
                this->m_EmpArray[i] = NULL;
            }
            //删除堆区数组指针
            delete[] this->m_EmpArray;
```

```
            this->m_EmpArray = NULL;
            this->m_EmpNum = 0;
            this->m_FileIsEmpty = true;
        }
        cout << "删除成功！" << endl;
    }
    //按任意键清屏
    system("pause");
    system("cls");
}
```

## 15.3 修改析构函数

```
WorkerManager::~WorkerManager() {
    if (this->m_EmpArray != NULL) {
        for (int i = 0; i < this->m_EmpNum; i++) {
            if (this->m_EmpArray[i] != NULL) {
                delete this->m_EmpArray[i];
                this->m_EmpArray[i] = NULL;
            }
        }
        delete[] this->m_EmpArray;
        this->m_EmpArray = NULL;
    }
}
```

main.cpp 文件中添加以下代码：

```
wm.Clean_Emp();
```

测试代码

至此本案例完成。