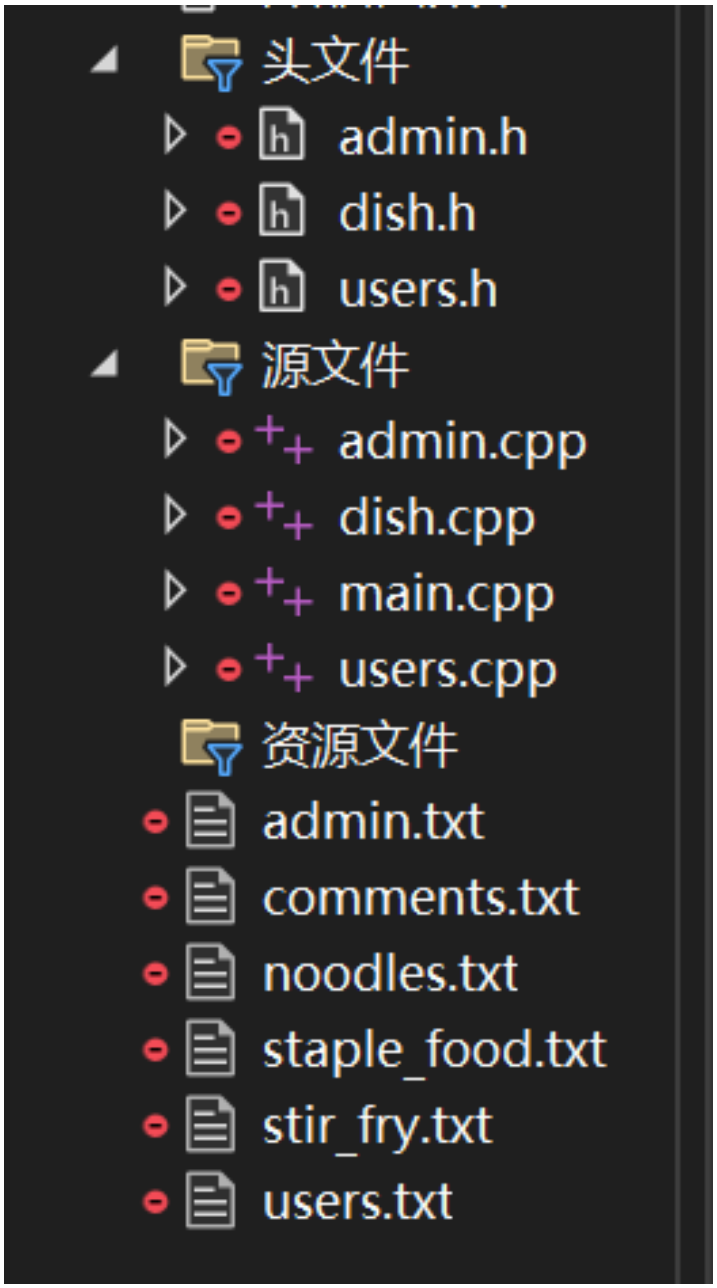
大作业报告

总体框架：



建立了 admin.h 和 user.h 和 dish.h 分别用来构建管理员功能，用户功能，菜色信息保存及显示等等的功能,并且在 main.cpp 里集中调用与实现；资源文件里，admin 和 users 分别保管管理员和用户的账号密码信息等，noodles，staple_food,stir_fry 这三个文件分别保管着不同的菜色类型对应的菜色信息（对应到 readme.md，这三个文件代表了同一个文件，也就是 dish.txt），comments 文件保存着用户对于菜色的反馈，其中包括不同加料与菜色结合的不同反馈。具体细节见下。

1.main 函数

Main 函数整体框架：

```
int main()
{
START:
    system("cls");
    cout << "*********************************************" << endl;
    cout << "饕在山青！欢迎进入山东大学青岛校区餐厅反馈系统" << endl;
    cout << "请选择以何种身份登录(请输入相应数字)？ " << endl;
    int choose;
    cout << "1.系统管理员" << endl;
    cout << "2.学生 / 教师用户" << endl;
    cout << "3.退出" << endl;
    cin >> choose;
    if (choose > 3 || choose <= 0) goto START;
    while (1)
    {
        system("cls");
        switch (choose)
        {
        case 2:
        { ... }
        case 1:
        { ... }
        case 3:
        { ... }
        default:
            break;
        }
    }
    return 0;
```

可见十分简洁，各种分支一目了然，其中应对输入数字异常也进行了相应的处理：

```
if (choose > 3 || choose <= 0) goto START;
```

Switch 函数中的不同的 case 对应 choose 的不同选择，以下会详细展开。

1）case 3 退出系统

```
case 3:
{
    cout << "感谢您的使用！给个好评哦~" << endl;
    system("pause");
    return 0;
}
default:
```

显示出相应的文字，然后直接 return 0。

2）case 1 系统管理员

```
case 2:
{ ... }
case 1:
{
CHOOSE_ADMIN:
    system("cls");
    cout << "*****************************************" << endl;
    cout << "欢迎使用管理员功能(请输入相应数字) :" << endl;
    cout << "1.登陆账号" << endl;
    cout << "2.返回" << endl;
    cout << "3.退出系统" << endl;
    int choose_admin;
    cin >> choose_admin;
    if (choose_admin > 3) goto CHOOSE_ADMIN;
    switch (choose_admin)
    {
    case 1:
    { ... }
    case 2:
    { ... }
    case 3:
    { ... }
    default:
    {
        break;
    }
    }
    break;
}
case 3:
```

看起来也是十分有规律，框架一目了然。

首先通过 system("cls")模拟了页面切换功能，显然选择管理员功能的第一步就是登陆系统，所以直接在终端输出关于登陆系统的相应内容（见橙色汉字）；

此外，与 main 函数开头相同，针对异常输入进行了处理：

```
if (choose_admin > 3) goto CHOOSE_ADMIN;
```

然后针对不同的 choose 进行相应的程序模块（这里使用了 switch 的嵌套）：

1：管理员登陆模块

```
ADMIN:
    system("cls");
    cout << "*****************************************" << endl;
    cout << "欢迎登录馨在山青-山东大学青岛校区餐厅反馈系统！ " << endl;
    cout << "请输入账号（8位数字）： " << endl;
    string name, password;
    cin >> name;
    cout << "请输入密码（9位数字）： " << endl;
    cin >> password;
    administrator b1(name, password);
    int yyyyy = b1.sign_in();
    if (yyyyy == 1)
    {
        string s = "登陆成功！ ";
        int xxxxx = 0;
        while (1) { ... }
        break;
    }
    else { ... }
```

基本思想是在常规清屏以及切换内容之后，构建一个 administrator 类型的变量（b1）(class administrator 详见 3.admin.h 及 admin.cpp)，进行其 sign_in(详见 3.admin.h 及 admin.cpp)操作：

若成功则函数返回 1，也就是 yyyyy=1，随后进行 while（1）中的语句：

每循环一次，执行一次 b1.work(s)(首次执行默认 s 是“登陆成功”)，随后根据接下来的操作返回值给 xxxxx，xxxxx 根据返回值不同更改 s 与其他变量的内容，并且再次执行 b1.work（）函数，一直到选择退出为止(返回代码为 7)：

```
登陆成功！
*****************************
请选择相应功能：
1.增加管理员
2.删除管理员
3.修改管理员密码
4.增加菜色信息
5.删除菜色信息
6.查看菜色反馈
7.退出管理员账号
```

```
    { ... }
    case 7://7.退出管理员账号
    {
        return 7;
    }
```

```
if (xxxxx == 7) break;
```

若登陆失败，则有可能是 admin.txt 丢失或者账号密码不正确导致的

```
if (yyyyy == 1) { ... }
else
{
    if (yyyyy == -1)
    {
        cout << "非常抱歉，系统文件被破坏！" << endl;
        system("pause");
        return 0;
    }
    char pd_user;
    cin >> pd_user;
    if (pd_user == 'y')
    {
        goto ADMIN;
    }
    else
    {
        goto START;
    }
    break;
}
```

其中 yyyyy=-1 是文件被破坏，否则会提示是否重新输入，并执行相关的操作。

2：返回模块

非常简单，返回上一级的页面，实现如下：

```
case 2:
{
    goto START;
    break;
}
```

3：退出系统模块

也是非常简单，choose=3 即可。（因为在一个大的 while 循环之中）

3）case2 用户

```
case 2:
{
CHOOSE_USER:
    system("cls");
    cout << "*******************************************" << endl;
    cout << "欢迎使用用户功能(请输入相应数字): " << endl;
    cout << "1.登陆账号" << endl;
    cout << "2.返回" << endl;
    cout << "3.退出系统" << endl;
    int choose_user;
    cin >> choose_user;
    if (choose_user > 3) goto CHOOSE_USER;
    switch (choose_user)
    {
    case 1:
    { ... }
    case 2:
    { ... }
    case 3:
    { ... }
    default:
    { ... }
    }
    break;
}
case 1:
```

大致框架与管理员部分相同。

```
case 1:
{
USER:
    system("cls");
    cout << "*******************************************" << endl;
    cout << "欢迎登陆馨在山青-山东大学青岛校区餐厅反馈系统! " << endl;
    string name, account, password;
    cout << "请输入姓名拼音 (例: 张三一, 请输入SanyiZhang) :" << endl;
    cin >> name;
    cout << "请输入账号 (学号或职工号) : " << endl;
    cin >> account;
    cout << "请输入密码 (初始密码为身份证号后六位) : " << endl;
    cin >> password;
    user a1(name, account, password);
    int yyyyy = a1.sign_in();
    if (yyyyy == 1)
    {
        string s = "登陆成功! ";
        int xxxxx;
        while (1){ ... }
    }
    else
    {
        if (yyyyy == -1)
        {
            cout << "非常抱歉, 系统文件被破坏! " << endl;
            system("pause");
            return 0;
        }
        char pd_user;
        cin >> pd_user;
        if (pd_user == 'y')
        {
            goto USER;
        }
        else
```

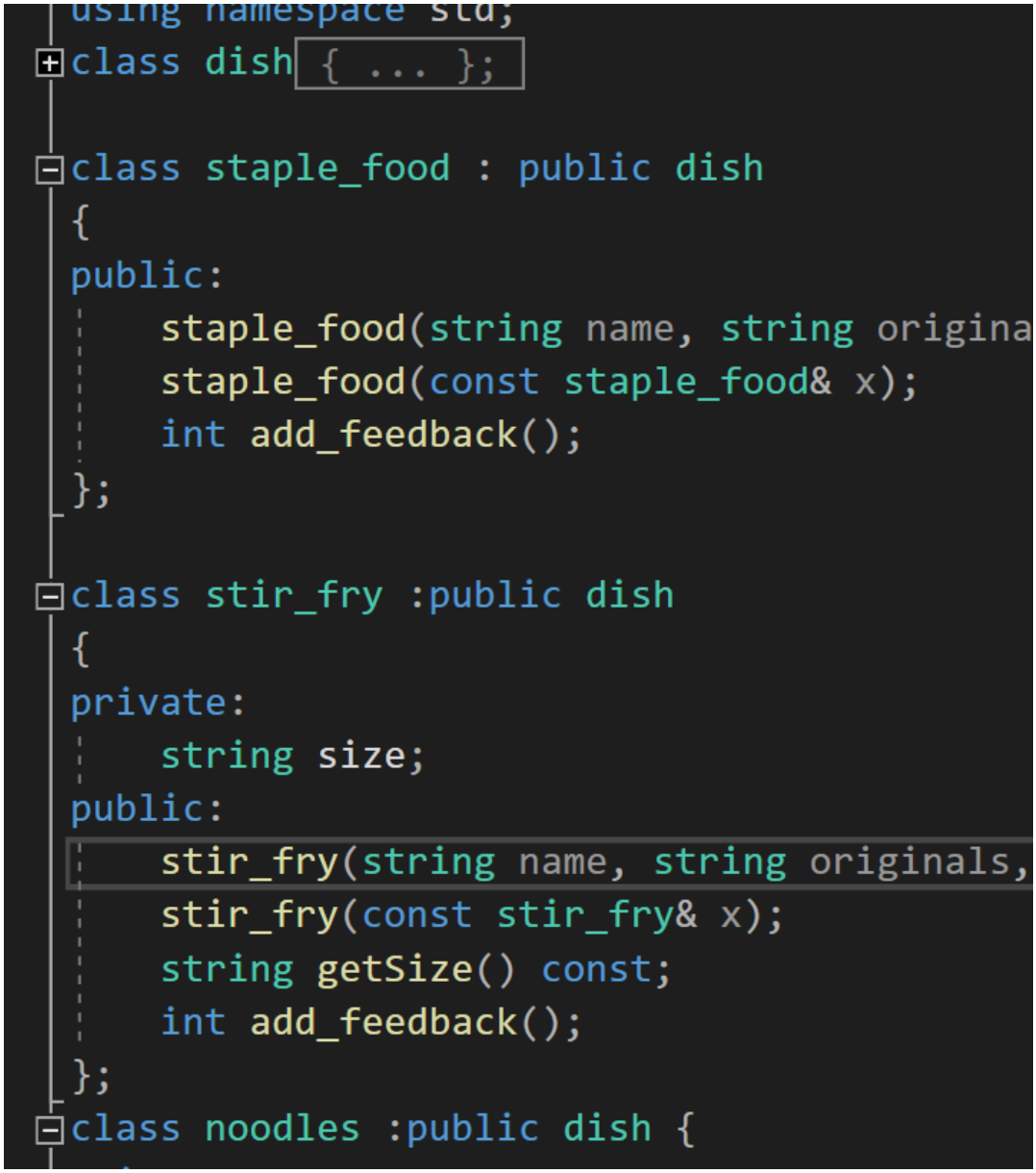
登陆部分与管理员模块相同。

```
case 2:
{
    goto START;
    break;
}
case 3:
{
    choose = 3;
    break;
}
```

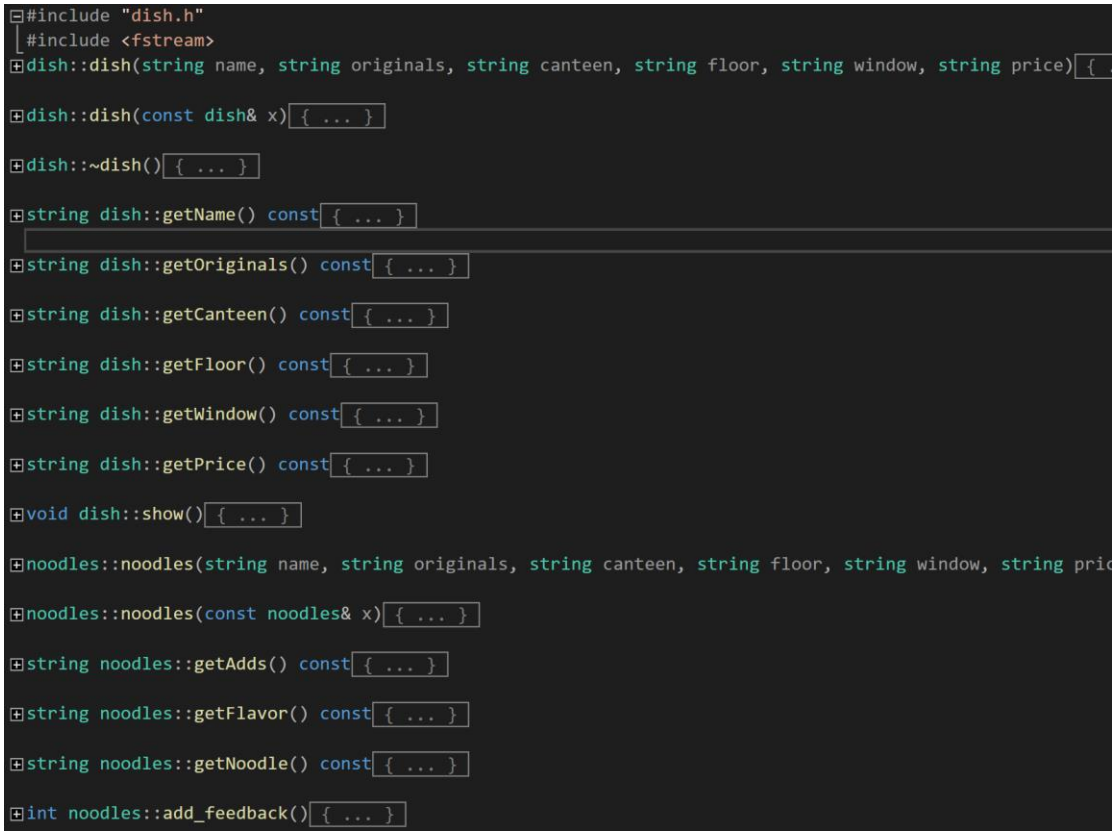
返回、退出模块与管理员模块也是相同的。

2.dish.h 及 dish.cpp

dish.h 里用类的公有继承实现了在菜色基础之上派生出炒菜类，粉面类和主食类三个菜色种类：



Dish.h 中的所有的函数具体实现均呈现在 dish.cpp 中：



其中比较需要注意的是 add_feedback 和 show 两个函数的实现。(因为其他函数的实现非常简单，只是 return 某个特定值；构造函数也是传递相应的变量值)

1）show

```
void dish::show()
{
    cout << "菜色名称: " << this->name << "    ";
    cout << "菜色原料: " << this->originals << endl;
    cout << "所在餐厅: " << this->canteen << "    ";
    cout << "所在楼层: " << this->floor << endl;
    cout << "所在窗口: " << this->window << "    ";
    cout << "菜色价格: " << this->price << endl;
    cout << endl;
}
```

Show 函数作为 dish 的接口，可同时用在其三个派生类上面，主要运用在 user 的浏览菜色和查询菜色上。

2) add_feedback

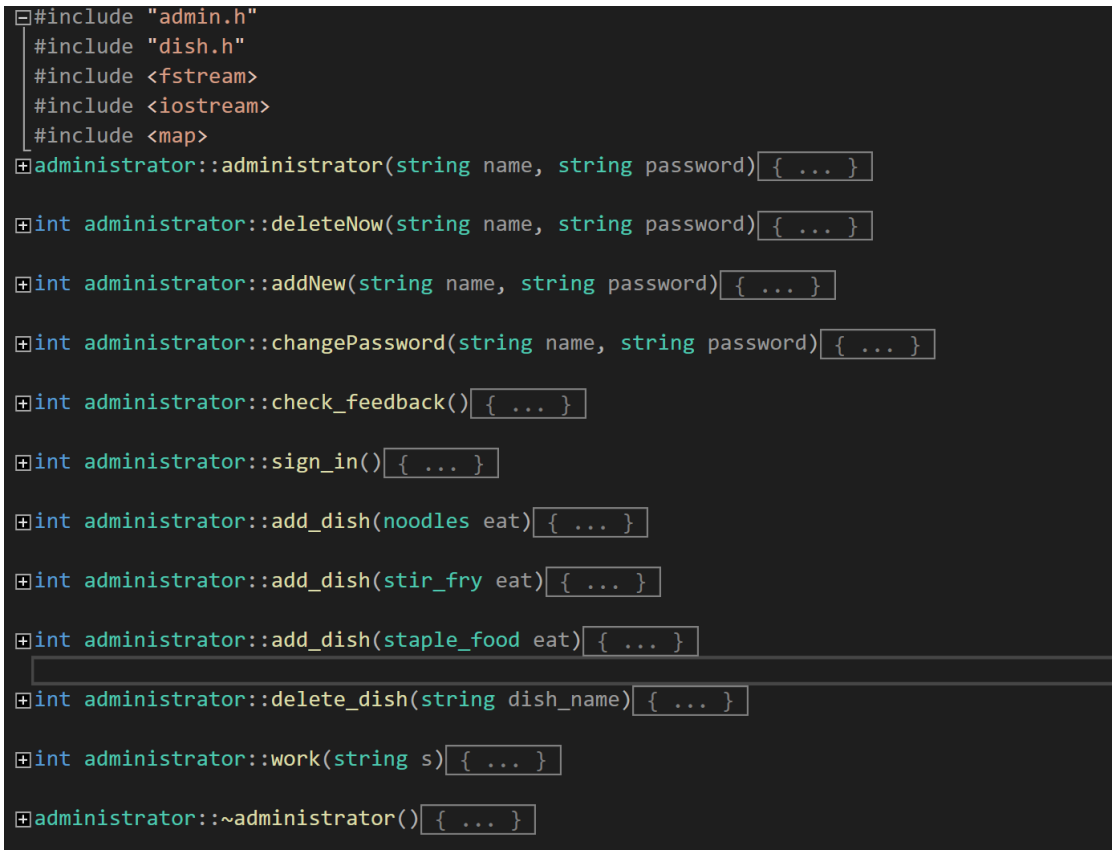
因为不同的菜色有不同的私有属性，所以在 add_feedback 函数的实现上在有些地方会稍有

不同，但是大致框架是一样的。以私有属性最多的粉面类为例：

```
int noodles::add_feedback()
{
    cout << "请输入您的反馈：" << endl;
    string feedback;
    cin >> feedback;
    ofstream fout;
    fout.open("comments.txt", ios::out | ios::app);
    if (!fout.is_open())
    {
        return -1;
    }
    else
    {
        fout << "\n" << 1 << " " << this->getName() << " " << this->adds << "
        fout.close(), fout.clear();
        return 3;
    }
    return 0;
}
```

在这里使用了 ofstream，照例做了文件打不开的异常处理，随后输出的“1”是对于粉面类做的标记，以在 admin.cpp 中读取文件时分清楚接下来的数据属于哪个类，随后做出相应的处理。

3.admin.h 及 admin.cpp



这里是本次大作业的重点之一，在这里主要有 sign_in 函数，以及 work 函数的 7 个分支，代表 7 个管理员的不同功能。其他的函数（如 add_dish(做出了重载的处理),delete_dish,changePassword 等）都是附着于 work 函数的，将会在 work 函数不同分支里详细展开。

Sign_in

源码:

```
int administrator::sign_in()
{
    int now = 0, lineCnt = 0;
    string tName, tPassword;
    ifstream fin;
    fin.open("admin.txt", ios::in);
    if (!fin.is_open())
    {
        cout << "admin.txt丢失!" << endl;
        return -1;
    }
    char c;
    while (fin.get(c))
```

```
    {
        if (c == '\n')
            lineCnt++;
    };
    lineCnt++;
    fin.close(), fin.clear();
    fin.open("admin.txt", ios::in);
    for (int now = 0; now <= lineCnt; now++)
    {
        fin >> tName >> tPassword;
        if (tName == this->name)
        {
            if (tPassword == this->password)
            {
                cout << "登陆成功！" << endl;
                fin.close(), fin.clear();
                return 1;
            }
            else
            {
                cout << "密码不正确！是否重新输入？(y/n)" << endl;
                fin.close(), fin.clear();
                return 0;
            }
        }
    }
    cout << "未找到账号！是否重新输入？(y/n)" << endl;
    fin.close(), fin.clear();
    return 0;
}
```

首先还是一个常见的判断是否能打开admin.txt的分支，否的话返回-1，对应到main函数输出相应的错误信息。随后的处理方法相对原始（因为这是一开始做的内容，思路当时还是有些混乱），简单来说就是先打开一遍文件，记录总的行数（getchar记录换行符），随后关闭文件并且clear ifstream，再次打开同一个文件，之后按照main函数中构造的对象（this指针指向的private内容）存储的信息查找文件，如果找不到账号则：

```
cout << "未找到账号！是否重新输入？(y/n)" << endl;
    fin.close(), fin.clear();
    return 0;
```

找到了相应的账号但是密码不正确就会指向下面的代码块:

```
cout << "密码不正确！是否重新输入？(y/n)" << endl;
                fin.close(), fin.clear();
                return 0;
```

而只有账号和密码都有并且一一对应时，才会执行接下来的代码块:

```
cout << "登陆成功！" << endl;
                fin.close(), fin.clear();
                return 1;
```

int 类型返回 1，登陆成功，接上了 main 函数内容：

```
int yyyyy = b1.sign_in();
if (yyyyy == 1)
{
    string s = "登陆成功！";
```

至此，sign_in 函数圆满结束。

Work_0 概述

Work 函数的整体框架非常简单，和其他的菜单界面相同，分成了显示和选择部分，并且配合 main 函数实现了不同的提示语打印：

```
628  int administrator::work(string s)
629  {
630      system("cls");
631      cout << s << endl;
632      cout << "**************************" << endl;
633      cout << "请选择相应功能: " << endl;
634      cout << "1.增加管理员" << endl;
635      cout << "2.删除管理员" << endl;
636      cout << "3.修改管理员密码" << endl;
637      cout << "4.增加菜色信息" << endl;
638      cout << "5.删除菜色信息" << endl;
639      cout << "6.查看菜色反馈" << endl;
640      cout << "7.退出管理员账号" << endl;
641      int choose_admin_work;
642      cin >> choose_admin_work;
643      switch (choose_admin_work)
644      {
645      case 1://1.增加管理员
646      { ... }
660      case 2://2.删除管理员
661      { ... }
711      case 3://3.修改管理员密码
712      { ... }
739      case 4://4.增加菜色信息
740      { ... }
870      case 5://5.删除菜色信息
871      { ... }
884      case 6://6.查看菜色反馈
885      { ... }
890      case 7://7.退出管理员账号
891      { ... }
894      }
895  }
```

Work_1 增加管理员

```
case 1://1.增加管理员
{
ADD_NEW:
    string newname, newpassword;
    cout << "请输入需要添加的管理员的账号（8位数字）：" << endl;
    cin >> newname;
    cout << "请输入需要添加的管理员的密码（9位数字）：" << endl;
    cin >> newpassword;
    int www = this->addNew(newname, newpassword);
    if (www == 666)
    {
        goto ADD_NEW;
    }
    else return www;
}
```

框架非常简洁，输入相应的信息之后，接下来的工作交给了 addNew(newname, newpassword)，并且根据返回值做出相应的调整。

addNew(newname, newpassword)具体实现如下：

```
int administrator::addNew(string name, string password)
{
    ofstream fout;
    fout.open("admin.txt", ios::out | ios::app);
    if (!fout.is_open())
    {
        cout << "admin.txt丢失!" << endl;
        return -1;
    }
    else
    {
        ifstream fin;
        fin.open("admin.txt", ios::in);
        string nowname, nowpassword;
        while (fin >> nowname >> nowpassword)
        {
            if (nowname == name)
            {
                cout << "此管理员已存在，请勿重复添加！" << endl;
                cout << "是否重新添加？(y/n)" << endl;
                char pd_cs2;
                cin >> pd_cs2;
                if (pd_cs2 == 'y')
                {
```

```
                    fout.close(), fout.clear();
                    fin.close(), fin.clear();
                    return 666;
                }
                else
                {
                    return 8;
                }
            }
        }
        fout << endl << name << " " << password;
        return 1;
    }
}
```

分析：

```
if (!fout.is_open())
    {
        cout << "admin.txt丢失!" << endl;
        return -1;
    }
```

常规判断文件丢失情况，返回特殊值 -1，使 main 函数做出相应的调整。

```
while (fin >> nowname >> nowpassword)
```

文件读取，因为存入时是有序的，所以不用担心读取错误的问题（除非人为破坏文件内容）。

```
if (nowname == name)
        {
            cout << "此管理员已存在，请勿重复添加！" << endl;
            cout << "是否重新添加？(y/n)" << endl;
            char pd_cs2;
            cin >> pd_cs2;
            if (pd_cs2 == 'y')
            {
                fout.close(), fout.clear();
                fin.close(), fin.clear();
                return 666;
            }
            else
            {
                return 8;
            }
}
```

此处判断重复，其中“是否重新添加？(y/n)”的代码块是重复使用的，在接下的不同地方还会出现

多次。

若无重复，则进行：

```
fout << endl << name << " " << password;
```

将新增的数据写入 txt 文件。至此，增加管理员操作圆满结束。

Work_2 删除管理员

```
case 2://2.删除管理员
{
DELETE_NOW:
    string newname, newpassword;
    cout << "请输入需要删除的管理员的账号（8位数字）：" << endl;
    cin >> newname;
    if (newname == this->name){……}
    cout << "请输入需要删除的管理员的密码（9位数字）：" << endl;
    cin >> newpassword;
    int kkk = this->deleteNow(newname, newpassword);
    if (kkk == 666)
    {
        goto DELETE_NOW;
    }
    else return kkk;
}
```

对比增加管理员，可见此处多出了一个判断不能删除自己的分支，具体如下：

```
if (newname == this->name)
{
    cout << "是否要删除自己？(y/n)" << endl;
    char pd_user;
    cin >> pd_user;
    if (pd_user == 'y')
    {
        system("cls");
        cout << "删除成功！请重新登录！" << endl;
        system("pause");
        ifstream fin;
        fin.open("admin.txt", ios::in);
        string nowname, nowpassword;
        string fileData = "";
        while (fin >> nowname >> nowpassword)
        {
            if (nowname == this->name)
            {
```

```
                continue;
            }
            else fileData += nowname + " " + nowpassword + "\n";
        }
        fileData.erase(fileData.end() - 1);
        fin.close(), fin.clear();
        ofstream fout;
        fout.open("admin.txt", ios::out);
        fout.flush();
        fout << fileData;
        fout.close(), fout.clear();
        return -2;
    }
    else
    {
        return 8;
    }
}
```

在实现是否要删除自己？(y/n) 的功能时，基本框架和“是否重新输入（y/n）”相似，当输入 y 之后，先实现相应的界面切换功能，（因为把自己给删除了，所以无法继续使用管理员的功能了），随后在 admin.txt 中找到 this->name 并且实现删除操作。删除操作的详细思路和操作过程在接下来给出。

下面详细介绍如何实现管理员的删除操作(在判断删除自己和 deleteNow(string name, string password)函数里都有应用:

首先定义一个 filedata 的 string 类型的变量，初始化为空，随后打开要删除内容的文件，逐行读取，若读取到要删除的内容，就 continue；否则将这一行的数据信息存储到 filedata 里，遍历完成之后将源文件清空，再将 filedata 中的内容写入源文件中。具体实现见 deleteNow(string name, string password)函数的内容:

```
int administrator::deleteNow(string name, string password)
{
    ifstream fin;
    fin.open("admin.txt", ios::in);
    string nowname, nowpassword;
    int check = 0;
    string fileData = "";
    while (fin >> nowname >> nowpassword)
    {
        if (nowname == name)
        {
            check = 1;
            if (nowpassword == password)
            {
                cout << "是否确定删除管理员：" << name << " " << "?(y/n)" << endl;
```

```
                char pd_user;
                cin >> pd_user;
                if (pd_user == 'y')
                {
                    continue;
                }
                else
                {
                    fin.close(), fin.clear();
                    return 8;
                }
            }
            else
            {
                cout << "要删除的管理员：" << name << " " << "对应的密码不正确！" <<
endl;
                cout << "是否重新输入？(y/n)" << endl;
                char pd_user;
                cin >> pd_user;
                if (pd_user == 'y')
                {
                    fin.close(), fin.clear();
                    return 666;
                }
                else
                {
                    fin.close(), fin.clear();
                    return 8;
                }
            }
        }
        else
        {
            fileData += nowname + " " + nowpassword + "\n";
        }
    }
    if (!check)
    {
        cout << "待删除的管理员不存在！" << endl;
        cout << "是否重新输入？(y/n)" << endl;
        char pd_user;
        cin >> pd_user;
        if (pd_user == 'y')
        {
```

```
            fin.close(), fin.clear();
            return 666;
        }
        else
        {
            fin.close(), fin.clear();
            return 8;
        }
    }
    else
    {
        fileData.erase(fileData.end() - 1);
        fin.close(), fin.clear();
        ofstream fout;
        fout.open("admin.txt", ios::out);
        fout.flush();
        fout << fileData;
        fout.close(), fout.clear();
        return 2;
    }
}
```

在 deleteNow(string name, string password)函数里面，引入了 check 变量观察待删除的管理员是否存在，如果不存在就进行重新选择模块（反复出现了），如果确实有则肯定不会重复（因为在添加管理员的时候就保证了账号不会重复），然后输入待删除管理员的密码（确定是你自己删除的），如果密码不正确则跳转到重新选择模块，密码正确则进行删除操作，具体实现思想已经在上面给出。
至此，删除管理员部分圆满结束。

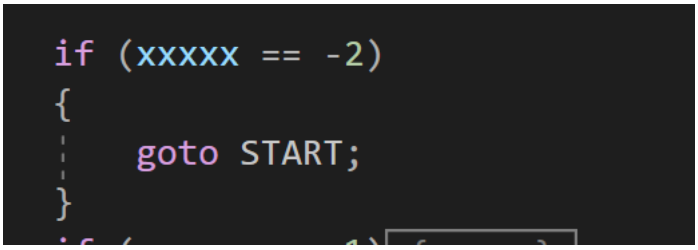
Work_3 修改管理员密码

与删除管理员类似，在修改管理员密码这里也用上了对于修改自己的判断，但是相对来说简单的是，自己的账号被保存在了 this 里，因此不用像上一个那样麻烦。具体代码如下：

```
CHANGE:
    int ck = 0;
    string changename, changepassword;
    cout << "请输入需要修改的管理员的账号（8位数字）：" << endl;
    cin >> changename;
    if (changename == this->name)
    {
        ck = 1;
    }
    cout << "请输入需要修改的管理员的原密码（9位数字）：" << endl;
    cin >> changepassword;
    int kkk = this->changePassword(changename, changepassword);
```

```
    if (kkk == 666)
    {
        goto CHANGE;
    }
    else if (kkk != 8 && ck)
    {
        system("cls");
        cout << "管理员密码修改成功!" << endl;
        cout << "您更改了自己的密码，请重新登录！" << endl;
        system("pause");
        return -2;
    }
    else return kkk;
}
```

Ck 就是判断是否修改自己的变量。在 changePassword(changename, changepassword)函数完成任务之后，根据 ck（也就是是否修改了自己）不同进行不同的代码块，对于返回特殊值-2 在 main 函数也有特殊的处理：



接下来是功能函数 changePassword(changename, changepassword)的具体实现：

```
int administrator::changePassword(string name, string password)
{
    ifstream fin;
    fin.open("admin.txt", ios::in);
    string nowname, nowpassword;
    string fileData = "";
    int check = 0;
    while (fin >> nowname >> nowpassword)
    {
        if (nowname == name)
        {
            if (nowpassword == password)
            {
                check = 1;
                string newPassword;
                cout << "请输入新的密码：" << endl;
                cin >> newPassword;
                fileData += nowname + " " + newPassword + "\n";
            }
            else
            {
```

```
                cout << "要更改密码的管理员：" << name << " " << "对应的密码不正确！"
<< endl;
                cout << "是否重新输入？(y/n)" << endl;
                char pd_user;
                cin >> pd_user;
                if (pd_user == 'y')
                {
                    fin.close(), fin.clear();
                    return 666;
                }
                else
                {
                    fin.close(), fin.clear();
                    return 8;
                }
            }
        }
        else
        {
            fileData += nowname + " " + nowpassword + "\n";
        }
    }
    if (!check)
    {
        cout << "待修改的管理员不存在！" << endl;
        cout << "是否重新输入？(y/n)" << endl;
        char pd_user;
        cin >> pd_user;
        if (pd_user == 'y')
        {
            fin.close(), fin.clear();
            return 666;
        }
        else
        {
            fin.close(), fin.clear();
            return 8;
        }
    }
    else
    {
        fileData.erase(fileData.end() - 1);
        fin.close(), fin.clear();
        ofstream fout;
```

```
            fout.open("admin.txt", ios::out);
            fout.flush();
            fout << fileData;
            fout.close(), fout.clear();
            return 3;
        }
}
```

仔细一看，这个函数与删除管理员密码中的功能函数实现非常相似，唯一不同点在于找到要删除/修改密码的管理员时，删除函数选择 continue，即不做处理（跳过）；而修改管理员密码选择的是：

```
fileData += nowname + " " + newPassword + "\n";
```

也就是原用户名加新的密码。至此，修改管理员密码部分圆满结束。

Work_4 增加菜色信息

Work_4 与其他三个相比有了很大的不同，因为那三个只是 admin 与 txt 的互动，而从 work_4 开始就有了加上 dish.h 以及相应的 txt 的内容。具体而言，其代码实现如下(部分):

```
case 4://4.增加菜色信息
{
ADDDISH:
    cout << "**************************" << endl;
    cout << "炒菜类菜色（输入1）" << endl;
    cout << "粉面类菜色（输入2）" << endl;
    cout << "主食类菜色（输入3）" << endl;
    cout << "请选择添加的菜色类别：" << endl;
    int kind;
    cin >> kind;
    if (kind > 3 || kind <= 0)
    {
        cout << "输入错误！是否重新输入？(y/n)" << endl;
        char pd_user;
        cin >> pd_user;
        if (pd_user == 'y')
        {
            goto ADDDISH;
        }
        else
        {
            return 8;
        }
    }
    else
    {
        string name = "", original = "", canteen = "";
        string floor = "", window = "";
```

```
        string price = "";
        string adds = "", flavor = "", noodle = "";
        string size = "";
        if (kind == 1)
        {
            ……
        }
        else if (kind == 2)
        {
            ……
        }
        else if (kind == 3)
        {
            ……
        }
    }
```

以上代码部分展示的是大致的菜单框架。从代码分析可知，这个界面与选择管理员功能等等的界面十分相似，都是通过操作者下一步的选择来运行不同的代码块。接下来，以附属属性最多的粉面类，也就是 kind=2 时的情况开始分析。其代码如下：

```
else if (kind == 2)
{
    cout << "请输入粉面名字：" << endl;
    cin >> name;
    cout << "请输入粉面包含的食材（多个食材请以逗号分隔）：" << endl;
    cin >> original;
    cout << "请输入粉面所在餐厅：" << endl;
    cin >> canteen;
    cout << "请输入粉面所在楼层（数字）：" << endl;
    cin >> floor;
    cout << "请输入粉面所在窗口（数字）：" << endl;
    cin >> window;
    cout << "请输入粉面的价格：" << endl;
    cin >> price;
ADDS:
    cout << "请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿）："
<< endl;
    cin >> adds;
    if (adds > "4" || adds < "0" || adds.size() > 1)
    {
        cout << "请在给定值中选择！" << endl;
        goto ADDS;
    }
FLAVOR:
    cout << "请输入口味名字（请选择数字，0：原味，1：番茄味，2：麻辣，3：酸辣，4：香
```

```
辣）" << endl;
    cin >> flavor;
    if (flavor > "4" || flavor < "0" || flavor.size() > 1)
    {
        cout << "请在给定值中选择！" << endl;
        goto FLAVOR;
    }
NOODLE:
    cout << "请输入粉面的形式（请选择数字，0：干拌面，1：汤面，2：炒面）" << endl;
    cin >> noodle;
    if (noodle < "0" || noodle > "2" || noodle.size() > 1)
    {
        cout << "请在给定值中选择！" << endl;
        goto NOODLE;
    }
    noodles eat(name, original, canteen, floor, window, price, adds, flavor, noodle);
    int kkk = this->add_dish(eat);
    if (kkk == 666)
    {
        goto ADDDISH;
    }
    else return kkk;
}
```

其中，这些是常规输入：

```
cout << "请输入粉面名字：" << endl;
    cin >> name;
    cout << "请输入粉面包含的食材（多个食材请以逗号分隔）：" << endl;
    cin >> original;
    cout << "请输入粉面所在餐厅：" << endl;
    cin >> canteen;
    cout << "请输入粉面所在楼层（数字）：" << endl;
    cin >> floor;
    cout << "请输入粉面所在窗口（数字）：" << endl;
    cin >> window;
    cout << "请输入粉面的价格：" << endl;
    cin >> price;
```

而接下来的三个部分很相似，因为让输入附属属性（加料，面条类型等，用数字），我就考虑了输入异常的情况，用 goto 进行了处理（以加料为例）：

```
ADDS:
    cout << "请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿）："
<< endl;
    cin >> adds;
    if (adds > "4" || adds < "0" || adds.size() > 1)
    {
```

```
        cout << "请在给定值中选择！" << endl;
        goto ADDS;
    }
```

实测效果如下：

```
请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿）：
5
请在给定值中选择！
请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿）：
-1
请在给定值中选择！
请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿）：
aaa
请在给定值中选择！
请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿）：
asfasfadfa
请在给定值中选择！
请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿）：
3
请输入口味名字（请选择数字，0：原味，1：番茄味，2：麻辣，3：酸辣，4：香辣）
```

，可见非常完美。剩下两个属性如法炮制，而剩下的两个菜系也是如法炮制：

```
if (kind == 1)
{
    cout << "请输入菜色名字：" << endl;
    cin >> name;
    cout << "请输入菜色包含的食材（多个食材请以逗号分隔）：" << endl;
    cin >> original;
    cout << "请输入菜色所在餐厅：" << endl;
    cin >> canteen;
    cout << "请输入菜色所在楼层（数字）：" << endl;
    cin >> floor;
    cout << "请输入菜色所在窗口（数字）：" << endl;
    cin >> window;
    cout << "请输入菜色的价格：" << endl;
    cin >> price;
SIZE:
    cout << "请输入菜色份量大小（请选择数字，0：小份，1：大份）" << endl;
    cin >> size;
    if (size > "1" || size.size() > 1 || size<"0")
    {
        cout << "请在给定值中选择！" << endl;
        goto SIZE;
    }
    stir_fry eat(name, original, canteen, floor, window, price, size);
    int kkk = this->add_dish(eat);
    if (kkk == 666)
    {
        goto ADDDISH;
```

```
    }
    else return kkk;
}
```

这是炒菜类的代码块，可见相似度是很高的。至此，添加菜色信息圆满结束。

Work_5 删除菜色信息

实现如下：

```
case 5://5.删除菜色信息
{
DELETE_DISH:
    cout << "**************************" << endl;
    cout << "请输入需要删除的菜色的名字：" << endl;
    string dish_name;
    cin >> dish_name;
    int kkk=this->delete_dish(dish_name);
    if (kkk == 666)
    {
        goto DELETE_DISH;
    }
    else return kkk;
}
```

相比较于增加菜色信息，这一代码块显得有些朴实无华，但是其功能确实是实打实完备的。框架依旧非常简单，核心是this->delete_dish(dish_name)函数，接下来看它的实现：

```
int administrator::delete_dish(string dish_name)
{
    ifstream f1, f2, f3;
    bool exist = 0;
    f1.open("noodles.txt", ios::in);
    f3.open("staple_food.txt", ios::in);
    f2.open("stir_fry.txt", ios::in);
    if (!f1.is_open() || !f2.is_open() || !f3.is_open())
    {
        return -1;
    }
    string name = "", original = "", canteen = "";
    string floor = "", window = "";
    string price = "";
    string adds = "", flavor = "", noodle = "", size = "";
    while (f1 >> name >> original >> canteen >> floor >> window >> price >> adds >>
flavor >> noodle)
    {
        if (dish_name == name)
        {
```

```
            exist = 1;
            f1.clear(), f1.close();
            break;
        }
    }
    if (f1.is_open()) f1.clear(), f1.close();
    if (exist)//in noodles
    {
        cout << "请确认是否删除菜色： " << dish_name << " ? （y/n）" << endl;
        char pd_user;
        cin >> pd_user;
        if (pd_user == 'y')
        {
            ofstream fout;
            ifstream fin;
            string fileData = "";
            fin.open("noodles.txt", ios::in);
            while (fin >> name >> original >> canteen >> floor >> window >> price >>
adds >> flavor >> noodle)
            {
                if (name == dish_name)
                {
                    continue;
                }
                else fileData += name + " " + original + " " + canteen + " " + floor
+ " " + window + " " + price + " " + adds + " " + flavor + " " + noodle + "\n";
            }
            fin.clear(), fin.close();
            fileData.erase(fileData.end() - 1);
            fout.open("noodles.txt", ios::out);
            fout.flush();
            fout << fileData;
            fout.clear(), fout.close();
            return 5;
        }
        else
        {
            f2.close(), f2.clear();
            f3.close(), f3.clear();
            return 8;
        }
    }
    else
    {
```

```
        while (f2 >> name >> original >> canteen >> floor >> window >> price >> size)
        {
            if (dish_name == name)
            {
                exist = 1;
                f2.close(), f2.clear();
                break;
            }
        }
    }
    if(f2.is_open()) f2.close(), f2.clear();
    if (exist)//in stir_fry
    {
        cout << "请确认是否删除菜色： " << dish_name << " ? （y/n）" << endl;
        char pd_user;
        cin >> pd_user;
        if (pd_user == 'y')
        {
            ofstream fout;
            ifstream fin;
            string fileData = "";
            fin.open("stir_fry.txt", ios::in);
            while (fin >> name >> original >> canteen >> floor >> window >> price >>
size)
            {
                if (name == dish_name)
                {
                    continue;
                }
                else fileData += name + " " + original + " " + canteen + " " + floor
+ " " + window + " " + price + " " + size + "\n";
            }
            fin.clear(), fin.close();
            fileData.erase(fileData.end() - 1);
            fout.open("stir_fry.txt", ios::out);
            fout.flush();
            fout << fileData;
            fout.clear(), fout.close();
            return 5;
        }
        else
        {
            f3.close(), f3.clear();
            return 8;
```

```
        }
    }
    else
    {
        while (f3 >> name >> original >> canteen >> floor >> window >> price)
        {
            if (dish_name == name)
            {
                exist = 1;
                f3.close(), f3.clear();
                break;
            }
        }
    }
    if (f3.is_open()) f3.close(), f3.clear();
    if (exist)//in staple_food
    {
        cout << "请确认是否删除菜色： " << dish_name << " ? （y/n）" << endl;
        char pd_user;
        cin >> pd_user;
        if (pd_user == 'y')
        {
            ofstream fout;
            ifstream fin;
            string fileData = "";
            fin.open("staple_food.txt", ios::in);
            while (fin >> name >> original >> canteen >> floor >> window >> price >>
size)
            {
                if (name == dish_name)
                {
                    continue;
                }
                else fileData += name + " " + original + " " + canteen + " " + floor
+ " " + window + " " + price + "\n";
            }
            fin.clear(), fin.close();
            fileData.erase(fileData.end() - 1);
            fout.open("staple_food.txt", ios::out);
            fout.flush();
            fout << fileData;
            fout.clear(), fout.close();
            return 5;
        }
```

```
            else
            {
                return 8;
            }
        }
        else
        {
            cout << "未找到要删除的菜色！是否重新输入？（y/n）" << endl;
            char pd_user;
            cin >> pd_user;
            if (pd_user == 'y')
            {
                return 666;
            }
            else
            {
                return 8;
            }
        }
}
```

看起来稍微有点多，总结起来的思想却比较简单，就是搜索三个文件，判断是否存在：若存在，就再次确认是否删除，是就像删除管理员那样对文件进行相应的操作；如果遍历完三个文件都没找到相应的菜名，就执行最后的代码块：

```
cout << "未找到要删除的菜色！是否重新输入？（y/n）" << endl;
            char pd_user;
            cin >> pd_user;
            if (pd_user == 'y')
            {
                return 666;
            }
            else
            {
                return 8;
            }
```

也就是cout加上重新选择的拼接。至此，删除菜色部分圆满结束。

Work_6 查看菜色反馈

相比 work_5，它的 case 内容甚至更简单：

```
case 6://6.查看菜色反馈
{
    cout << "*************************" << endl;
    cout << "用户反馈意见如下:" << endl;
```

```
    return this->check_feedback();
}
```

然后查看其功能函数的实现：

```
int administrator::check_feedback()
{
    ifstream fin;
    fin.open("comments.txt", ios::in);
    if (!fin.is_open())
    {
        return -1;
    }
    else
    {
        system("cls");
        int kind;
        string name = "", adds = "", flavor = "", noodle = "", size = "", feedback =
"";
        map<string, string> jl, kw, mt, sz;
        //0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿
        jl["0"] = "无", jl["1"] = "鸡蛋", jl["2"] = "牛肉", jl["3"] = "鸡肉", jl["4"]
= "火腿";
        //0：原味，1：番茄味，2：麻辣，3：酸辣，4：香辣
        kw["0"] = "原", kw["1"] = "番茄", kw["2"] = "麻辣", kw["3"] = "酸辣", kw["4"]
= "香辣";
        //0：干拌面，1：汤面，2：炒面
        mt["0"] = "干拌面", mt["1"] = "汤面", mt["2"] = "炒面";
        //0：小份，1：大份
        sz["0"] = "小份", sz["1"] = "大份";
        while (fin >> kind)
        {
            switch (kind)
            {
            case 1://noodles
            {
                fin >> name >> adds >> flavor >> noodle >> feedback;
                cout << "关于加料为 " << jl[adds] << " 的 " << kw[flavor] << "味 " <<
name << " (" << mt[noodle] << ") 的反馈如下：" << endl;
                cout << feedback << endl << endl;
                break;
            }
            case 2://staple_food
            {
                fin >> name >> feedback;
                cout << "关于" << name << " 的反馈如下：" << endl;
```

```
                cout << feedback << endl << endl;
                break;
            }
            case 3://stir_fry
            {
                fin >> name >> size >> feedback;
                cout << "关于 " << sz[size] << " " << name << " 的反馈如下：" <<
endl;
                cout << feedback << endl << endl;
                break;
            }
            default:
            {
                break;
            }
            }
        }
        cout << "------------------------------------------------" << endl;
        fin.close(), fin.clear();
        system("pause");
        return 6;
    }
}
```

其基本思想个人认为是存储菜系的升级版，也就是把这些反馈都存到一个文件里，最前边加上一个标识数字，根据标识数字的不同去执行不同的代码块：

```
while (fin >> kind)
{
    switch (kind)
    {
    case 1://noodles
    { ... }
    case 2://staple_food
    { ... }
    case 3://stir_fry
    { ... }
    default:
    {
        break;
    }
    }
}
```

除此之外，还有一个非常重要的点，就是把附加属性的数字字符串转化成汉字字符串，在这里我使用了 map 作为映射：

```
//0: 无, 1: 鸡蛋, 2: 牛肉, 3: 鸡肉, 4: 火腿
jl["0"] = "无", jl["1"] = "鸡蛋", jl["2"] = "牛肉", jl["3"] = "鸡肉", jl["4"] = "火腿";
//0: 原味, 1: 番茄味, 2: 麻辣, 3: 酸辣, 4: 香辣
kw["0"] = "原", kw["1"] = "番茄", kw["2"] = "麻辣", kw["3"] = "酸辣", kw["4"] = "香辣";
//0: 干拌面, 1: 汤面, 2: 炒面
mt["0"] = "干拌面", mt["1"] = "汤面", mt["2"] = "炒面";
//0: 小份, 1: 大份
sz["0"] = "小份", sz["1"] = "大份";
```

随后就正常的读取，显示，效果如下（可能菜名或者反馈有点不正常）：

```
关于加料为 鸡蛋 的 番茄味 铁牛牛肉面（汤面）的反馈如下：
我不吃牛肉

关于馒头 的反馈如下：
呕吼耶！太香的很~

关于 大份 羊头 的反馈如下：
简简单单，吃个羊头

关于花卷 的反馈如下：
不好吃

--------------------------------------------------
请按任意键继续. . .
```

至此，查看菜色功能圆满完成。

Work_7 退出管理员账号

```
case 7://7.退出管理员账号
{
    return 7;
}
```

```
if (xxxxx == 7) break;
```

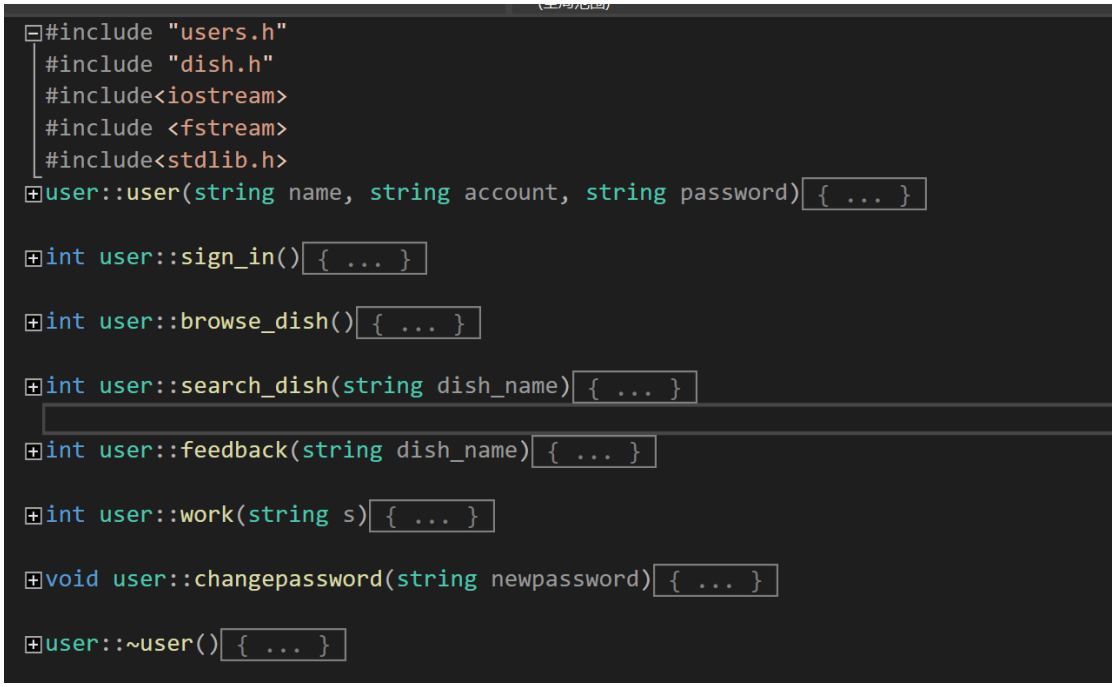
，正常结束 while 循环。至此，admin 部分功能及实现介绍完毕。

4.users.h 及 users.cpp

比起来 admin.h 和 admin.cpp，user 显得简单得多，因为它只设置了 4 个功能：

```
cout << "1.浏览所有菜色信息" << endl;
cout << "2.查询菜色" << endl;
cout << "3.提交菜色反馈意见" << endl;
cout << "4.退出登录" << endl;
```

相对来说，实现起来比较简单。与 admin.h 相似，本部分内容还是围绕 sign_in 和 work 函数展开。
其略缩图如下：



Sign_in

```
int user::sign_in()
{
    int now = 0, lineCnt = 0;
    string tName, tAccount, tPassword;
    ifstream fin;
    fin.open("users.txt", ios::in);
    if (!fin.is_open())
    {
        cout << "users.txt丢失!" << endl;
        return -1;
    }
    char c;
    while (fin.get(c))
    {
        if (c == '\n')
            lineCnt++;
    };
    lineCnt++;
    fin.close(), fin.clear();
    fin.open("users.txt", ios::in);
```

```
    for (int now = 0; now <= lineCnt; now++)
    {
        fin >> tName >> tAccount >> tPassword;
        if (tName == this->name && tAccount == this->account)
        {
            if (tPassword == this->password)
            {
                cout << "登陆成功！" << endl;
                return 1;
            }
            else
            {
                cout << "密码不正确！是否重新输入？(y/n)" << endl;
                return 0;
            }
        }
    }
    cout << "未找到用户！是否重新输入？(y/n)" << endl;
    return 0;
}
```

可见，其实现与 admin 中的 sign_in 函数的实现十分相似，也就是改了个 txt 名称和 cout 的内容而已。因此并无太多介绍。

Work_0 概述

```
int user::work(string s)
{
    system("cls");
    cout << s << endl;
    cout << "请选择相应功能(请输入相应数字): " << endl;
    cout << "1.浏览所有菜色信息" << endl;
    cout << "2.查询菜色" << endl;
    cout << "3.提交菜色反馈意见" << endl;
    cout << "4.修改密码" << endl;
    cout << "5.退出登录" << endl;
    int choose_user_work;
    cin >> choose_user_work;
    switch (choose_user_work)
    {
    case 1://1.查看所有菜色信息
    { ... }
    case 2://2.查询菜色
    { ... }
    case 3://3.提交菜色反馈意见
    { ... }
    case 4://4.修改密码
    { ... }
    case 5://5.退出登录
    { ... }
    default:
        break;
    }
}
```

总结：admin 中 work 函数的缩减版。

Work_1 查看所有菜色信息

代码非常简单：

```
case 1://1.查看所有菜色信息
{
    int kkk = this->browse_dish();
    return kkk;
}
```

现在分析其核心函数 brouse_dish：

```
int user::browse_dish()
{
    system("cls");
    ifstream f1, f2, f3;
    cout << "----------------- 餐厅现有菜色：-----------------" << endl;
    cout << "炒菜类" << endl;
    f1.open("stir_fry.txt", ios::in);
    f2.open("noodles.txt", ios::in);
    f3.open("staple_food.txt", ios::in);
    if (!f1.is_open() || !f2.is_open() || !f3.is_open())
    {
        return -1;
    }
    else
    {
        string name = "", original = "", canteen = "";
        string floor = "", window = "";
        string price = "";
        string adds = "", flavor = "", noodle = "", size = "";
        while (f1 >> name >> original >> canteen >> floor >> window >> price >> size)
        {
            stir_fry tmp(name, original, canteen, floor, window, price, size);
            tmp.show();
        }
        f1.close(), f1.clear();
        cout << "粉面类" << endl;
        while (f2 >> name >> original >> canteen >> floor >> window >> price >>
adds >> flavor >> noodle)
        {
            noodles tmp(name, original, canteen, floor, window, price, adds, flavor,
noodle);
            tmp.show();
        }
        f2.close(), f2.clear();
        cout << "主食类" << endl;
        while (f3 >> name >> original >> canteen >> floor >> window >> price)
        {
            staple_food tmp(name, original, canteen, floor, window, price);
            tmp.show();
        }
        f3.close(), f3.clear();
        cout << "-------------------------------------------" << endl;
        system("pause");
        return 1;
```

```
    }
}
```

与 comments 的读取并展示不同，dish 的读取与展示因为有三个文件所以在读取的时候要开三个 fstream，并且要及时 close 和 clear，这些在 browse_dish()函数中都得到了展现。

Work_2 查找菜色

其 case 中的内容（不带核心函数）相对来说比较简洁：

```
case 2://2.查询菜色
{
SEARCH_DISH:
    system("cls");
    cout << "**************************" << endl;
    cout << "请输入需要查询的菜色的名字：" << endl;
    string dish_name;
    cin >> dish_name;
    int kkk = this->search_dish(dish_name);
    if (kkk == 666)
    {
        goto SEARCH_DISH;
    }
    else return kkk;
    break;
}
```

可见是常规的形式。但是对于其核心函数 search_dish(dish_name)：

```
int user::search_dish(string dish_name)
{
    ifstream f1, f2, f3;
    bool exist = 0;
    f1.open("noodles.txt", ios::in);
    f3.open("staple_food.txt", ios::in);
    f2.open("stir_fry.txt", ios::in);
    if (!f1.is_open() || !f2.is_open() || !f3.is_open())
    {
        return -1;
    }
    string name = "", original = "", canteen = "";
    string floor = "", window = "";
    string price = "";
    string adds = "", flavor = "", noodle = "", size = "";
    while (f1 >> name >> original >> canteen >> floor >> window >> price >> adds >>
flavor >> noodle)
    {
```

```
        if (dish_name == name)
        {
            exist = 1;
            f1.clear(), f1.close();
            break;
        }
    }
    if (f1.is_open()) f1.clear(), f1.close();
    if (exist)//in noodles
    {
        ifstream fin;
        fin.open("noodles.txt", ios::in);
        while (fin >> name >> original >> canteen >> floor >> window >> price >>
adds >> flavor >> noodle)
        {
            if (name == dish_name)
            {
                noodles tmp(name, original, canteen, floor, window, price, adds,
flavor, noodle);
                cout << "您查询的菜色信息如下：" << endl;
                tmp.show();
                break;
            }
        }
        f2.close(), f2.clear();
        f3.close(), f3.clear();
        fin.close(), fin.clear();
        cout << "-----------------------------------------------" << endl;
        system("pause");
        return 2;
    }
    else
    {
        while (f2 >> name >> original >> canteen >> floor >> window >> price >> size)
        {
            if (dish_name == name)
            {
                exist = 1;
                f2.close(), f2.clear();
                break;
            }
        }
    }
    if (f2.is_open()) f2.close(), f2.clear();
```

```
if (exist)//in stir_fry
{
    ifstream fin;
    fin.open("stir_fry.txt", ios::in);
    while (fin >> name >> original >> canteen >> floor >> window >> price >> size)
    {
        if (name == dish_name)
        {
            stir_fry tmp(name, original, canteen, floor, window, price, size);
            cout << "您查询的菜色信息如下：" << endl;
            tmp.show();
            break;
        }
    }
    f3.close(), f3.clear();
    fin.close(), fin.clear();
    cout << "------------------------------------------------" << endl;
    system("pause");
    return 2;
}
else
{
    while (f3 >> name >> original >> canteen >> floor >> window >> price)
    {
        if (dish_name == name)
        {
            exist = 1;
            f3.close(), f3.clear();
            break;
        }
    }
}
if (f3.is_open()) f3.close(), f3.clear();
if (exist)//in staple_food
{
    ifstream fin;
    fin.open("staple_food.txt", ios::in);
    while (fin >> name >> original >> canteen >> floor >> window >> price)
    {
        if (name == dish_name)
        {
            staple_food tmp(name, original, canteen, floor, window, price);
            cout << "您查询的菜色信息如下：" << endl;
            tmp.show();
```

```
                break;
            }
        }
        fin.close(), fin.clear();
        cout << "-----------------------------------------------" << endl;
        system("pause");
        return 2;
    }
    else
    {
        cout << "未找到要查询的菜色！是否重新输入？（y/n）" << endl;
        char pd_user;
        cin >> pd_user;
        if (pd_user == 'y')
        {
            return 666;
        }
        else
        {
            return 8;
        }
    }
}
```

这段代码绝对是所有代码块中写的最复杂与混乱的代码块，没有之一。其原因在于这段功能是我一开始仅仅有了一个大致的思路就开始敲代码，导致思路简单但是实现方法及其复杂，最后写成了这么大一堆。其中心思想不难理解：

按照 3 个 txt 文件依次遍历，如果能找到相应的菜名就做出标记，返回；然后按照做出的标记对应到相应的 txt 文件再次查询菜名，并输出相应内容（调用 dish 抽象类的 show 文件）。

如果三个都没有标记，就按照失败返回结果，case 再按照返回内容做出不同的反应。

可惜，在一开始思路还比较混乱的时候就开始写代码了，导致代码最终开起来十分冗杂，这点应当在接下来的学习中注意并改正。

Work_3 提交菜色反馈意见

此处的 work 函数里的基础框架如下：

```
case 3://3.提交菜色反馈意见
{
FEEDBACK:
    system("cls");
    cout << "**************************" << endl;
    cout << "请输入您要评价的菜色名：" << endl;
    string dish_name;
    cin >> dish_name;
    int kkk = this->feedback(dish_name);
```

```
    if (kkk == 666)
    {
        goto FEEDBACK;
    }
    else return kkk;
    break;
}
```

可见，在输入相应的菜名之后，函数内部执行核心函数：feedback（dish_name），并且根据核心函数返回的不同值去执行不同的代码。

下面是核心函数 feedback（dish_name）的内容：

```
int user::feedback(string dish_name)
{
    ifstream f1, f2, f3;
    bool ex1 = 0, ex2 = 0, ex3 = 0;
    f1.open("noodles.txt", ios::in);
    f2.open("staple_food.txt", ios::in);
    f3.open("stir_fry.txt", ios::in);
    if (!f1.is_open() || !f2.is_open() || !f3.is_open())
    {
        return -1;
    }
    else
    {
        string name = "", original = "", canteen = "";
        string floor = "", window = "";
        string price = "";
        string adds = "", flavor = "", noodle = "", size = "";
        while (f1 >> name >> original >> canteen >> floor >> window >> price >>
adds >> flavor >> noodle)
        {
            if (dish_name == name)
            {
                ex1 = 1;
            ADDS:
                cout << "请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡
肉，4：火腿）：" << endl;
                cin >> adds;
                if (adds > "4" || adds < "0" || adds.size() > 1)
                {-                    cout << "请在给定值中选择！" << endl;
                    goto ADDS;
                }
            FLAVOR:
                cout << "请输入口味名字（请选择数字，0：原味，1：番茄味，2：麻辣，3：
酸辣，4：香辣）" << endl;
```

```
                cin >> flavor;
                if (flavor > "4" || flavor < "0" || flavor.size() > 1)
                {
                    cout << "请在给定值中选择！" << endl;
                    goto FLAVOR;
                }
            NOODLE:
                cout << "请输入粉面的形式（请选择数字，0：干拌面，1：汤面，2：炒面）"
<< endl;
                cin >> noodle;
                if (noodle < "0" || noodle > "2" || noodle.size() > 1)
                {
                    cout << "请在给定值中选择！" << endl;
                    goto NOODLE;
                }
                noodles tmp(name, original, canteen, floor, window, price, adds,
flavor, noodle);
                return tmp.add_feedback();
            }
        }
        while (f2 >> name >> original >> canteen >> floor >> window >> price)
        {
            if (dish_name == name)
            {
                ex2 = 1;
                staple_food tmp(name, original, canteen, floor, window, price);
                return tmp.add_feedback();
            }
        }
        while (f3 >> name >> original >> canteen >> floor >> window >> price >> size)
        {
            if (name == dish_name)
            {
                ex3 = 1;
            SIZE:
                cout << "请输入菜色份量大小（请选择数字，0：小份，1：大份）" << endl;
                cin >> size;
                if (size > "1" || size.size() > 1 || size < "0")
                {
                    cout << "请在给定值中选择！" << endl;
                    goto SIZE;
                }
                stir_fry tmp(name, original, canteen, floor, window, price, size);
                return tmp.add_feedback();
```

```
            }
        }
        if ((ex1 | ex2 | ex3) == 0)
        {
            cout << "未找到要反馈的菜色！是否重新输入？（y/n）" << endl;
            char pd_user;
            cin >> pd_user;
            if (pd_user == 'y')
            {
                return 666;
            }
            else
            {
                return 8;
            }
        }
    }
}
```

这一部分的框架设计也是不够简洁，在这里我按照三个文件依次查找设计的这样的函数框架，但是实际上只需要将三个文件合并成一个文件，并且在每一行数据的开头加上一个判断标识符就可以（这个框架在查看反馈菜色的时候得到实现）。回到这个框架本身，它针对不同的文件中不同的内容（因为不同菜品有加料的不同）做出了不同的处理。就粉面类而言，它在找到相应的菜名之后会提示输入加料类型，以针对不同的加料组合做出最准确的反馈：

```
ADDS:
                cout << "请输入加料名字（请选择数字，0：无，1：鸡蛋，2：牛肉，3：鸡肉，4：火腿）：" << endl;
                cin >> adds;
                if (adds > "4" || adds < "0" || adds.size() > 1)
                {-                  cout << "请在给定值中选择！" << endl;
                    goto ADDS;
                }
            FLAVOR:
                cout << "请输入口味名字（请选择数字，0：原味，1：番茄味，2：麻辣，3：酸辣，4：香辣）" << endl;
                cin >> flavor;
                if (flavor > "4" || flavor < "0" || flavor.size() > 1)
                {
                    cout << "请在给定值中选择！" << endl;
                    goto FLAVOR;
                }
            NOODLE:
                cout << "请输入粉面的形式（请选择数字，0：干拌面，1：汤面，2：炒面）" << endl;
                cin >> noodle;
```

```
if (noodle < "0" || noodle > "2" || noodle.size() > 1)
{
    cout << "请在给定值中选择！" << endl;
    goto NOODLE;
}
```

这里的代码块复制了在管理员添加菜品时的代码，其中异常处理非常明确且有效。再之后，函数里构造的相应的 dish 类的对象，并且执行 dish 类派生的对象的 public 函数：add_feedback()函数，详情见 2) add_feedback.至此，work_3：提交菜色反馈意见部分圆满结束。

Work_4 退出登录

```
case 4://4.退出登录
{
    return 4;
    break;
}
```

其思想与 admin 的 work 函数中退出相同。至此，user.h,user.cpp 的实现全部完成。

5.添加功能/修复 BUG

显然，仅有 readme 文件里给出的那些功能是不够的，还缺少一些非常基本的功能，比如用户的修改密码功能。除此之外，还有一些很不容易被发现的漏洞，比如删除菜品的时候应该不仅仅删除 dish 文件里面保存的信息，还应该提示是否删除相应的评论（考虑到相关的菜品已经被删除，反馈到现实生活中，被删除的菜品的反馈意见已经没有了参考价值，所以相应的反馈意见也应该被删除）。除此之外，在管理员功能上新加了增加用户，删除用户两个功能。具体的实现见下。

1.为 user 添加修改密码功能：

需要原因：用户一般都有一个初始密码，但是一般来说初始密码过于简单，导致这个用户的安全性比较差，因此需要添加更改用户密码的功能来保证用户的安全性。
将输出菜单更改成：

```
cout << "请选择相应功能(请输入相应数字)： " << endl;
cout << "1.浏览所有菜色信息" << endl;
cout << "2.查询菜色" << endl;
cout << "3.提交菜色反馈意见" << endl;
cout << "4.修改密码" << endl;
cout << "5.退出登录" << endl;
```

然后再在 work 函数里添加相应的分支：

```
case 4://4.修改密码
{
CHANGE:
```

```
        cout << "请输入原来的密码：" << endl;
        string nowpassword;
        cin >> nowpassword;
        if (nowpassword != this->password)
        {
            cout << "原密码错误！" << endl;
            cout << "是否重新输入？(y/n)" << endl;
            char pd_user;
            cin >> pd_user;
            if (pd_user == 'y')
            {
                goto CHANGE;
            }
            else
            {
                return 8;
            }
        }
        else
        {
            cout << "请输入新密码：" << endl;
            string newpassword;
            cin >> newpassword;
            this->changepassword(newpassword);
            system("cls");
            cout << "修改密码成功，请重新登录！" << endl;
            system("pause");
            return 4;
        }
        break;
    }
```

这个框架和 admin 函数中管理员修改密码时的框架基本相同，都是先输入原来的密码（确保你是本人在操作），随后输入新的密码，并且调用核心功能函数 changepassword（newpassword）进行相应功能的实现。

下面是功能函数 changepassword（newpassword）的具体实现：

```
void user::changepassword(string newpassword)
{
    ifstream fin;
    fin.open("users.txt", ios::in);
    string nowname, nowaccount, nowpassword;
    string filedata = "";
    while (fin >> nowname >> nowaccount >> nowpassword)
    {
        if (nowname == this->name && nowaccount == this->account)
```

```
            {
                filedata += nowname + " " + nowaccount + " " + newpassword + "\n";
            }
            else
            {
                filedata += nowname + " " + nowaccount + " " + nowpassword + "\n";
            }
        }
        fin.clear(), fin.close();
        ofstream fout;
        fout.open("users.txt", ios::out);
        fout.flush();
        filedata.erase(filedata.end() - 1);
        fout << filedata;
        fout.clear(), fout.close();
}
```

可见这一部分与 admin 中的功能实现函数部分也是非常的相似，无论是框架还是实现思路都与其大差不差，故省略具体描述。

2.对于删除菜色部分的 bug 修复

在 admin 的 cpp 文件里的 delete_dish 函数中做出了修复。原函数见 Work_5 删除菜色信息。更改之后的代码如下（以 noodles 删除为例）：

```
if (exist)//in noodles
{
    cout << "请确认是否删除菜色： " << dish_name << " ? （y/n）" << endl;
    char pd_user;
    cin >> pd_user;
    if (pd_user == 'y')
    {
        ofstream fout;
        ifstream fin;
        string fileData = "";
        fin.open("noodles.txt", ios::in);
        while (fin >> name >> original >> canteen >> floor >> window >> price >>
adds >> flavor >> noodle)
        {
            if (name == dish_name)
            {
                continue;
            }
            else fileData += name + " " + original + " " + canteen + " " + floor + " "
+ window + " " + price + " " + adds + " " + flavor + " " + noodle + "\n";
        }
```

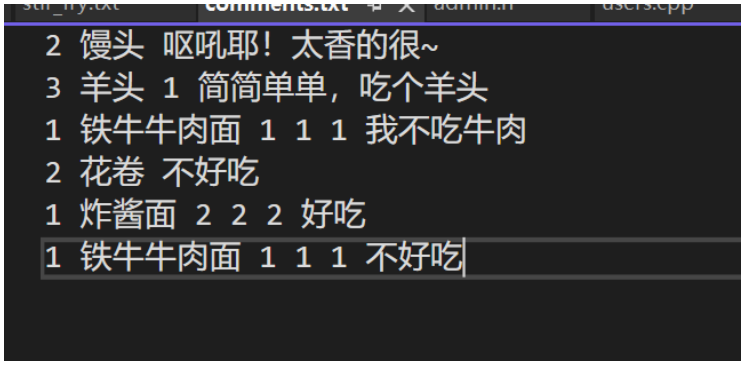
```
            fin.clear(), fin.close();
            fileData.erase(fileData.end() - 1);
            fout.open("noodles.txt", ios::out);
            fout.flush();
            fout << fileData;
            fout.clear(), fout.close();
            //下面是增加的删除评论部分
            fin.open("comments.txt");
            fileData = "";
            int kind;
            while (fin >> kind)
            {
                switch (kind)
                {
                case 1://noodles
                {
                    fin >> name >> adds >> flavor >> noodle >> feedback;
                    if (name == dish_name) continue;
                    else fileData+= "1 " + name + " " + adds + " " + flavor + " " +
noodle + " " + feedback + "\n";
                    break;
                }
                case 2://staple_food
                {
                    fin >> name >> feedback;
                    fileData += "2 " + name + " " + feedback + "\n";
                    break;
                }
                case 3://stir_fry
                {
                    fin >> name >> size >> feedback;
                    fileData += "3 " + name + " " + size + " " + feedback + "\n";
                    break;
                }
                default:
                {
                    break;
                }
                }
            }
            fin.clear(), fin.close();
            fileData.erase(fileData.end() - 1);
            fout.open("comments.txt", ios::out);
            fout.flush();
```

```
        fout << fileData;
        fout.clear(), fout.close();
        return 5;
    }
    else
    {
        f2.close(), f2.clear();
        f3.close(), f3.clear();
        return 8;
    }
}
```

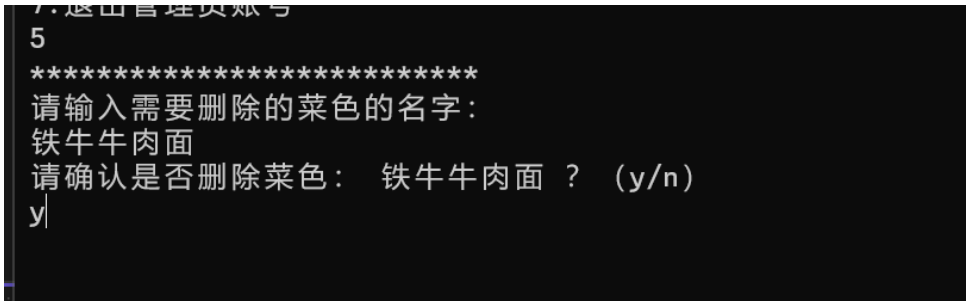
```
fout.clear(), fout.close();
//下面是增加的删除评论部分
fin.open("comments.txt");
fileData = "";
```

其中，这之下的代码部分是新增加的部分。分析其代码框架，发现这与 admin 的 cpp 文件中的 check_feedback 函数非常相似（见 Work_6 查看菜色反馈），有区别的地方在于查看菜色反馈的时候直接输出，而在这里参考了删除管理员账号的方法。测试情况如下：

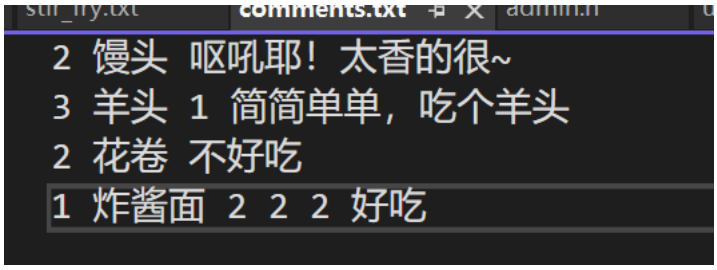
删除之前：



删除信息：



删除之后：



可见，删除相应的菜色之后，不仅相同类型的菜色反馈内容没有被误删，而且同一菜名的所有反馈都被删除了，bug 得到了修复。

3.admin——增加用户

代码：

```
case 7://7.增加用户
{
ADD_NEW_USER:
    string newname, newpassword, newnum;
    cout << "请输入需要添加的用户的拼音（例：张三一，请输入SanyiZhang）:" << endl;
    cin >> newname;
    cout << "请输入需要添加的用户的账号（学号或职工号）:" << endl;
    cin >> newnum;
    cout << "请输入需要添加的用户的密码（该用户的身份证号后六位）：" << endl;
    cin >> newpassword;
    int www = this->addNew_USER(newname, newnum,newpassword);
    if (www == 666)
    {
        goto ADD_NEW_USER;
    }
    else return www;
}
```

总体思路和增加管理员差不多，改动的地方不是很多，只有 fstream 打开的文件名字和输入到文件的变量格式的改变。，其核心代码 addNew_USER(newname, newnum,newpassword) 和增加管理员那块差不多，略。

4.删除用户

代码：

```
case 8://8.删除用户，8被占用改成88
{
DELETE_NOW_USER:
    string nowname;
    cout << "请输入需要删除的用户的拼音（例：张三一，请输入SanyiZhang）：" << endl;
    cin >> nowname;
    int kkk = this->deleteNow_USER(nowname);
```

```
    if (kkk == 666)
    {
        goto DELETE_NOW_USER;
    }
    else return kkk;
}
```

首先，case 8 按照惯例应该 return 8，但是 return 8 在这个大量使用的代码块里出现过了：

```
char pd_user;
cin >> pd_user;
if (pd_user == 'y')
{
    ……
}
else
{
    ……
    return 8;
}
```

所以改成了return 88代替。除此之外，在逻辑上来说，显然管理员删除用户是不需要指导用户的密码的，而且也不会造成删除自己的问题（不是管理员删除管理员），所以这里相比删除管理员简单一些。其核心函数：`deleteNow_USER(nowname)`与删除管理员的核心函数很相似，略。

6.总结

回顾整个程序，我主要设计了 dish,user,admin 三个类，并且联系现实对不同的类实现了不同的功能，并且通过 main 函数对其进行了统一调配。其中在函数的具体实现上，我充分运用了这个学期的所学知识，包括但不限于类的构造与成员函数、成员数据，类的继承与派生，STL 容器的使用（在 Work_6 查看菜色反馈中运用了 map），文件的读取和输出等知识。通过此次大作业，我感觉自己对于 C++的理解，掌握与运用得到了进一步的提升。总而言之，这是一次非常有意义的大作业。