【题目】财务报销管理系统

1. 数据格式

采用多文件存储数据，文件内容与标题对应：

1. 文件一（申请人信息）：

```
//employee.txt
工号              姓名              手机号
005               张三               13900700700
006               李四               13900800800
```

2. 文件二（处理人信息）：

```
//manager.txt
工号              姓名              手机号                  职位
001               王五               13300100100             部门副主管
002               赵一               13900200200             财务副主管
003               孙六               13800400100             部门主管
004               钱七               13900300200             财务主管
```

3. 文件三（额度限制信息）：

```
//quota.txt
工号              职位                  最大额度
001               部门副主管            50000
002               财务副主管            50000
003               部门主管              100000
004               财务主管              100000
```

4. 文件四（流程日志信息）：

```
//log.txt
报销单编号        提单人        提单金额        提单日期          状态          处理人
20220006          007           100000          20220313          创建
20220007          007           100000          20220313          审批未通过      003
20220008          007           100000          20220313          创建
```

2. 数据结构（读文件创建哈希表与队列）

将文件中的数据读入内存，采用方法为`freopen()`函数，对程序的io流进行重定向，定向至文件路径所指处读取数据；

```
//例：EmployeeMap的数据输入
freopen("employee.txt", "r", stdin);
memset(vis, 0, sizeof(vis));
string number, name, phone;
cin >> number_title >> name_title >> phone_title;
while (cin >> number >> name >> phone) {
    Employee tmp(number, name, phone);
    add(tmp);
}
freopen("CON", "r", stdin);
cin.clear();
```

之后建立以字符串工号为键值的哈希表`EmployeeMap`，`ManagerMap`，`QuotaMap`，然后建立申请人报销循环队列`LogQueue`以及`LogSave`，其内部数据成员如下：

1. Employee类与EmployeeMap类：

```
class Employee {
private:
    string number, name, phone;
    //number为工号，name为姓名，phone为手机号
};
```

```
class EmployeeMap {
private:
    bool vis[MAX_SIZE];          //记录工号是否存在
    Employee mp[MAX_SIZE];       //哈希表
    string number_title, name_title, phone_title;    //用来保存文件输出的表头
};
```

2. Manager类与ManagerMap类：

```
class Manager : public Employee {
private:
    string job;
    //继承Employee类中数据成员，job为职位
};
```

```
class ManagerMap {
private:
    bool vis[MAX_SIZE];          //记录工号是否存在
    Manager mp[MAX_SIZE];        //哈希表
    string number_title, name_title, phone_title, job_title;     //用来保存文件输出的表头
};
```

3. Quota类与QuotaMap类；

```
class Quota {
private:
    string number, job; //number为工号，job为职位
    int limits; //limits为额度限制
};
```

```
class QuotaMap {
private:
    bool vis[MAX_SIZE];     //记录工号是否存在
    Quota mp[MAX_SIZE];     //哈希表
    string number_title, job_title, limits_title;   //用来保存文件输出的表头
};
```

4. Log类、LogQueue类与LogSave类；

```
class Log {
private:
    SYSTEMTIME date;    //使用window.h头文件中的SYSTEMTIME变量，可以获取当前系统时
间
    string employee, manager, status;   //employee为申请人工号，manager为处理人
工号，status为当前处理状态
    int money;  //money为申请报销金额
};
```

```
class LogQueue {
private:
    Log* q;     //Log类指针作为队列的数组名
    int head, rear; //head为队首，rear为队尾
};
```

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

//该类用于保存审批流程至日志中，便于最后的打印展示
class LogSave {
private:
    Log* q;     //Log类指针作为队列的数组名
    int head, rear, index;      //head为队首，rear为队尾，index为
    string index_title, employee_title, money_title, date_title,
status_title, manager_title;   //保存数据表头
};
```

3. 编辑文件信息

实现文件的增删改查，设计思路与代码如下：

1. 哈希函数：使用`<string>`头文件中的`stoi`函数作为哈希函数，功能为将字符串转化为其对应的整型数据，由于工号为一一对应的分配，即独一无二的，所以哈希冲突非常低；
2. 信息增加：先将字符串哈希为整数类型，**然后在`vis`数组中查看该工号是否已经出现过（数据关联性）**，若出现过则返回`false`且不存入哈希表中，反之保存；

```
//增加信息
bool add(xxx tmp) {
    int idx = stoi(tmp.getNumber());
    if (!vis[idx])
        vis[idx] = true;
    else
        return false;
    mp[idx] = tmp;
    return true;
}
```

3. 信息删除：与信息增加相反，如果`vis`中不存在，则返回`false`，反之删除；

```
//以工号为索引删除信息
bool del(string number) {
    int idx = stoi(number);
    if (vis[idx])
        vis[idx] = false;
    else
        return false;
    return true;
}
```

4. 信息更改：先将要更改的数据从哈希表中删除，然后将更改后的数据添加进表中即可；

```
//以工号为索引更改信息
bool modify(string number, Manager tmp) {
    if (!del(number))
        return false;
    add(tmp);
    return true;
}
```

5. 信息查询：哈希后直接返回表中键值对应的数据；

```
//以工号为索引查询信息
xxx query(int idx) {
    return mp[idx];
}
```

6. 信息更新：在程序运行结束后，将内存中修改过的数据信息重新写入文件中，同样采用了`freopen()`函数，将输出流定向至文件所在位置：

```
void update() {
```

```
    freopen("employee.txt", "w", stdout);
    employee_map.display();
    cout.clear();
    freopen("manager.txt", "w", stdout);
    manager_map.display();
    cout.clear();
    freopen("quota.txt", "w", stdout);
    quota_map.display();
    cout.clear();
    freopen("log.txt", "w", stdout);
    log_save.display();
    cout.clear();
    freopen("CON", "w", stdout);
}
```

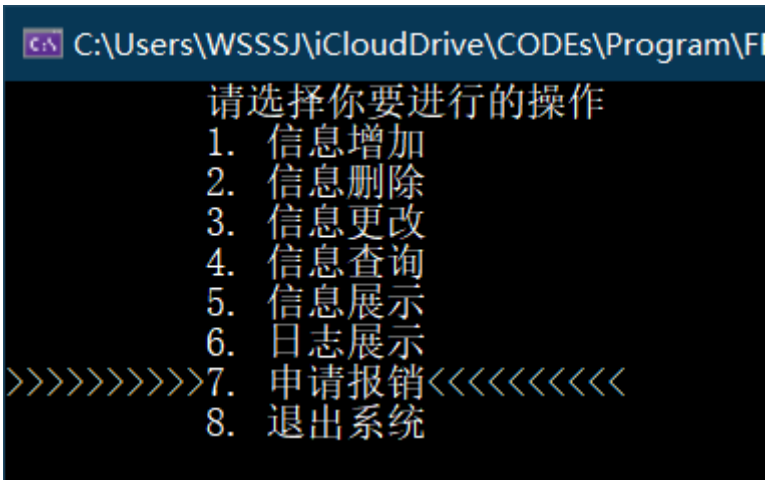
***数据关联性**：

1. 当录入的信息中出现非期望的重复时，程序将提示错误然后返回上层菜单，并不将该次输入存入内存中；
2. ManagerMap类与QuotaMap类中存在信息关联，所以不得直接单项更改或删除上述两类的数据，即ManagerMap类的删改同时要配合QuotaMap类的删改；

4. 处理报销信息

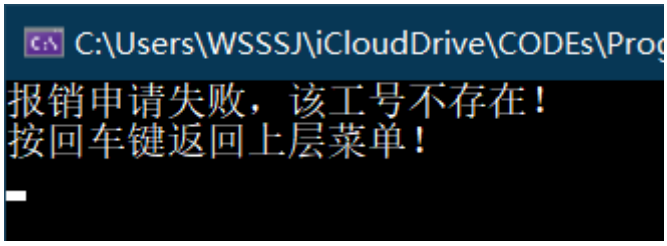
4.1 在用户交互界面中：

首先在菜单中选择`7.报销申请`，按下回车后即可进入数据录入阶段：

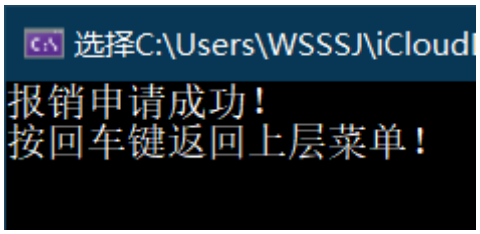


在数据录入阶段，需要输入报销申请人的工号以及期望报销的金额：

如果申请人工号并不存在，则会显示申请失败并返回上层菜单：



反之申请成功：



4.2 在程序后台运行中：

对报销申请处理的代码：

```
string number;
int money;
cout << "请输入申请人的工号："; cin >> number;
int idx = stoi(number);
cout << "请输入报销金额："; cin >> money;
Log tmp(number, money);
log_queue.push(tmp);
process();
tips();
if (employee_map.check(idx))
    cout << "报销申请成功！" << endl;
else
    cout << "报销申请失败，该工号不存在！" << endl;
if (fin())
    return;
```

当程序收到了来自前端的数据输入之后，会将其压入队列`log_queue`中，并调用`process`函数进行处理：

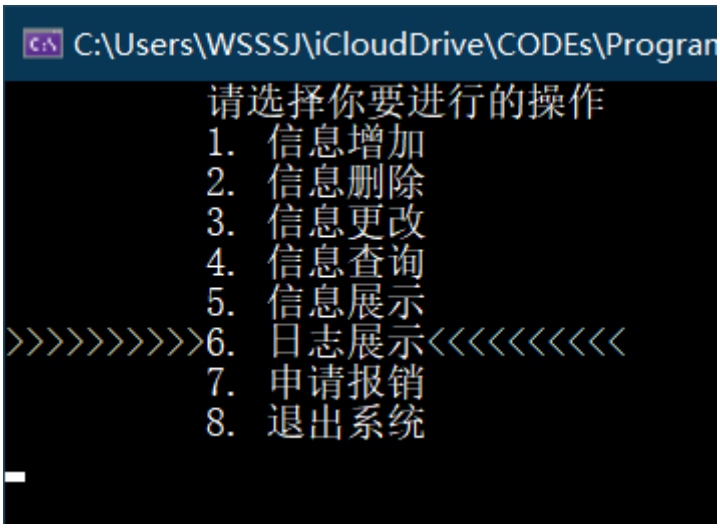
```
//小trick，获取一个随机状态，状态根据生成数字的奇偶而定
bool getRandom() {
    srand((int)time(NULL));
    int i = rand() % 100 + 1;
    return i % 2 ? true : false;
}

void process() {
    //当队列不为空时
    while (!log_queue.empty()) {
        //取出队首将其压入log_save中作为日志内容，并进行下一步操作
        Log tmp = log_queue.front();
        log_queue.pop();
        log_save.push(tmp);
        //如果当前状态是“创建”，则将移交给“部门主/副主管”进行审批
        if (tmp.getStatus() == "创建") {
```

```
            //在哈希表中线性查找，复杂度O(n)
            for (int i = 0; i < MAX_SIZE; ++i) {
                if (quota_map.check(i)) {
                    Quota quota = quota_map.query(i);
                    //根据申报数额决定分配给主管或者副主管
                    if (quota.getLimits() >= tmp.getMoney() && (quota.getJob()
== "部门主管" || quota.getJob() == "部门副主管")) {
                        //如果审批通过
                        if (getRandom()) {
                            //将其当前状态更改
                            tmp.changeStatus("部门审批");
                        }
                        else {
                            tmp.changeStatus("审批未通过");
                        }
                        //将其处理人的信息更新为刚才审批人的信息
                        tmp.changeManager(quota.getNumber());
                        //再次压入队中
                        log_queue.push(tmp);
                        break;
                    }
                }
            }
        }
        //以下步骤与上相同
        else if (tmp.getStatus() == "部门审批") {
            for (int i = 0; i < MAX_SIZE; ++i) {
                if (quota_map.check(i)) {
                    Quota quota = quota_map.query(i);
                    if (quota.getLimits() >= tmp.getMoney() && (quota.getJob()
== "财务主管" || quota.getJob() == "财务副主管")) {
                        if (getRandom()) {
                            tmp.changeStatus("财务审批");
                        }
                        else {
                            tmp.changeStatus("审批未通过");
                        }
                        tmp.changeManager(quota.getNumber());
                        log_queue.push(tmp);
                        break;
                    }
                }
            }
        }
        else if (tmp.getStatus() == "财务审批") {
            tmp.changeStatus("付款");
            log_queue.push(tmp);
        }
    }
}
```

5. 查看报销审批日志

5.1 在用户交互界面中：

在菜单中选择`6.日志展示`，即可打印出日志内容：



5.2 在程序后台运行中：

直接调用`log_save`中的成员函数`display()`打印出日志：

```
system("cls");
tips();
log_save.display();
if (fin())
    return;
```

6. 设计总结

1. 在这次程序设计中，花费时间最久的部分为用户交互界面，即菜单的设计，我采用了`<conio.h>`头文件中的`_getch()`函数达到监控键盘的效果，在程序运行时，实时读取方向键上下的状态，并且同步刷新控制台界面，实现了动态的菜单效果，同时，程序在面对异常操作（乱按键盘）时，不会给予反应；

2. 其次，在菜单的多级设计方面，我采用了函数递归回溯的方式，解决了从二级菜单返回一级菜单的问题；

```
void read(int line) {
    char c1, c2;
    while (true) {
        display(line);

        c1 = _getch();
        if (c1 == '\r') {
            if (line == 8) {
                tips();
                cout << "系统已退出！" << endl;
                return;
            }

            if (menu_status == 1) {
                operation_status = line;
                if (line > 0 && line < 6) {
                    menu_status = 2;
                    //递归回溯
                    read(1);
                    menu_status = 1;
                    continue;
                }
                else {
                    operation();
                    continue;
                }
            }
            else {
                object_status = line;
                operation();
                return;
            }
        }

        c2 = _getch();
        switch (c2) {
        case UP: line--; break;
        case DOWN: line++; break;
        default: continue;
        }

        if (menu_status == 1) {
            if (line > 8)
                line = 1;
            else if (line < 1)
                line = 8;
        }
        else if (menu_status == 2) {
            if (line > 4)
                line = 1;
            else if (line < 1)
                line = 4;
        }
    }
```

```
}

void display(int line) {
    system("cls");
    if(menu_status == 1)
        for (int i = 0; i < 9; ++i) {
            if (i != line)
                cout << spc << first[i] << spc << endl;
            else
                cout << arw_r << first[i] << arw_l << endl;
        }
    else if(menu_status == 2)
        for (int i = 0; i < 5; ++i) {
            if (i != line)
                cout << spc << second[i] << spc << endl;
            else
                cout << arw_r << second[i] << arw_l << endl;
        }
}
```

3. 在数据结构设计方面，本来打算使用STL库中的容器`map`来作为表来使用，发现其红黑树的功能有些多余，于是决定手写一个简易哈希表；
4. 有意思的两个函数：`tips()`，`fin()`：

```
//在程序中显示“处理中...”的字样，让用户感知到操作生效的反馈
void tips() {
    for (int i = 0; i < 3; ++i) {
        system("cls");
        cout << "处理中";
        for (int j = 0; j < i; ++j) {
            cout << '.';
        }
        cout << endl;
        Sleep(200);
    }
    system("cls");
}

//监视键盘，仅当读取到回车键时做出反应，可以让用户在结束一次操作之后有反应的时间，自己来决定何时返回上层菜单
bool fin() {
    cout << "按回车键返回上层菜单！" << endl;
    char c;
    while (c = _getch()) {
        if (c == '\r')
            return true;
    }
}
```

5. 程序的不足之处自然也是存在的，比如在处理人员报销信息时，采用的QuotaMap上的线性查找效率并不是很高，后期本来想采用二分查找，但是发现需要更改前面设计的数据结构，遂放弃，仍然采用线性查找算法；