数据结构期末项目报告

完成人员

李沛珍 18307130331

需求分析

需求可以划分成两大块：

- 随机生成两个input文件
- 根据随机生成的input文件进行座位安排

对于第二块：

从输出要求进行需求分析，由于要求输出两个文件：output_data.txt和output_customer.txt，故从两个文件的输出内容进行分析。

output_data.txt

在此文件中，要求输出所有顾客的平均等候时间、平均停留时间及最后一位顾客的离开时间。据此，可以得出需要获得每组顾客：

- 等候时间
- 停留时间：等候时间 + 用餐时间(已知)
- 离开时间

output_customer.txt

在此文件中，要求输出每组顾客的：

- 编号(按时间顺序的先后排列)
- 顾客人数(已知)
- 到达时刻(已知)
- 等候时间
- 就餐时刻
- 就餐用时(已知)
- 离开时刻

初步分析可得，编号可按到达时间排序得到；就餐时刻可由到达时刻和等候时间得到；离开时刻可由就餐时刻和就餐用时得到；因此，最主要的任务是获得等候时间或者就餐时刻，二者任得其一就可获取两个输出文件中所有未知量。

概要设计

数据结构

程序抽象出了4个类，具体结构如下：

- Time：表示时刻

```
class Time {
    friend class Customers;
    friend class Table;
private:
    int hour;
    int minute;
    bool nextDay;
public:
    Time() {};
    Time(string s){     //根据string变量构造Time
        string t = "";
        int i = 0;
        while (s[i] != ':') {
            t = t + s[i];
            i++;
        }
        hour = stoi(t);
        i++;
        t = "";
        while (i < s.length()) {
            t = t + s[i];
            i++;
        }
        minute = stoi(t);
        if (hour >= 11)
            nextDay = false;
        else
            nextDay = true;
    }
    Time* afterT(int t);    //计算经过t分钟后的时刻
    int comp(Time* t1, Time* t2);   //比较两个时刻的早晚
    int length(Time* t1, Time* t2);    //计算从t1到t2经过的分钟数
    int Time2int();     //仅对就餐时刻用，将其转换成int型
    string Time2str();  //将Time转换成string便于输出
    bool getNxtDay();   //获取nextDay
};
```

- Customers：表示顾客

```
class Customers {
    friend class Table;
private:
    int ID;
    int personNum;
```

```
    int eatTime;
    int waitTime;
    Time* arriveTime;
    Time* settleDown;
    Time* leaveTime;
public:
    //通过输入初始化顾客对象，此时还没有得到所有的变量值
    Customers(int n, Time* arrT, Time* eatT){
        ID = -1;
        personNum = n;
        eatTime = eatT->Time2int();
        waitTime = -1;
        arriveTime = arrT;
        settleDown = nullptr;
        leaveTime = nullptr;
    }
    void setID(int i);  //设置ID
    void setWaitTime();     //设置waitTime
    void setSettleDown(Time* curTime);  //设置settleDown
    void setLeaveTime();    //设置leaveTime
    Time* getArriveTime();  //获得顾客到达时间
    int getPersonNum();     //获得顾客人数
    int getWaitTime();      //获得顾客等候时间
    int getStayTime();      //获得顾客停留时间
    Time* getLeaveTime();   //获得顾客离开时间
    void customerInfo(ofstream &outputFile);    //输出顾客信息到文件中
};
```

- Table：表示桌子

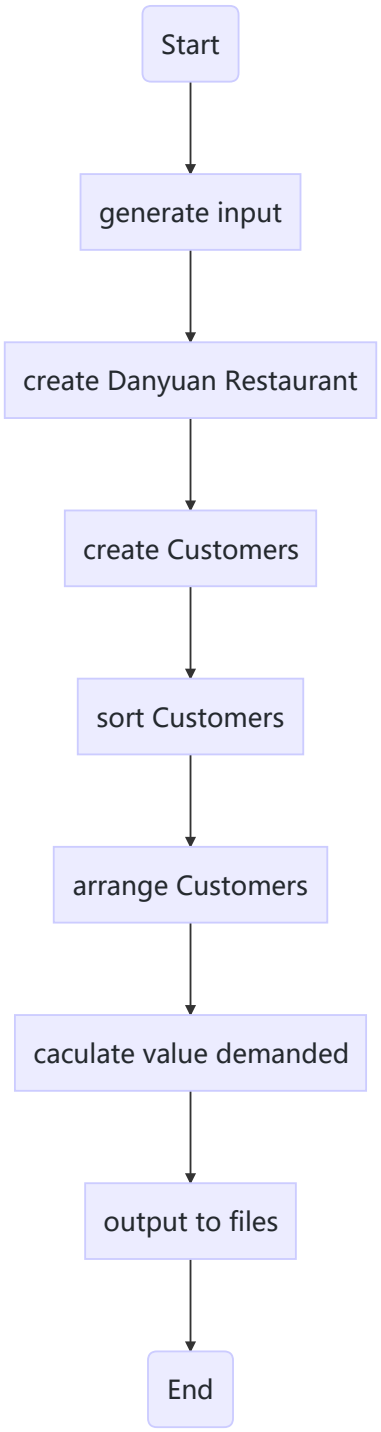
```
class Table {
    friend class Restaurant;
private:
    int capacity;   //每张桌子容量
    bool busy;      //标记是否被使用
    Time* freeTime;     //若被使用，释放时间
public:
    Table(int c) {  //初始状态桌子空闲
        capacity = c;
        busy = false;
        freeTime = nullptr;
    }
    void beUsed(Customers* c);      //桌子被customer占用
    void beFreed();     //桌子被释放
    Time* getFreeTime();    //获取桌子释放时间
    int getCapacity();      //获取桌子容量
};
```

- Restaurant：表示饭店

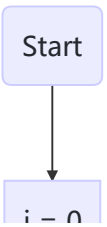
```
class Restaurant {
private:
```

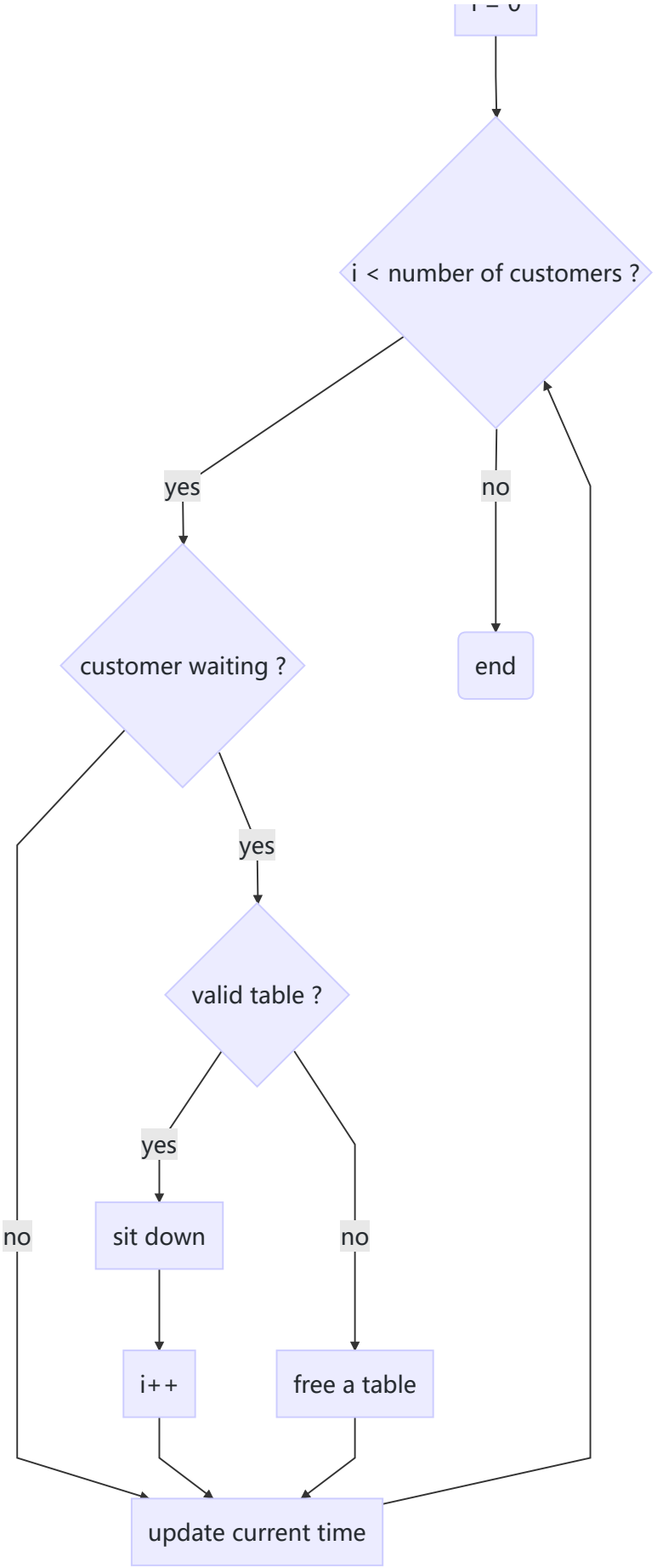
```
    vector<Table*> allTables[MaxCapacity + 1];  //该餐馆拥有的所有桌子，所有桌子按容量形成
vector数组，再形成总数组
    vector<Table*> freeTables[MaxCapacity + 1]; //该餐馆空闲的桌子，同上
    vector<Table*> busyTables[MaxCapacity + 1]; //该餐馆被占用的桌子，同上
public:
    Restaurant(vector<Table*> a) {  //初始状态所有桌子都是空闲的
        for (int i = 0; i < a.size(); i++) {
            allTables[a[i]->capacity].push_back(a[i]);
            freeTables[a[i]->capacity].push_back(a[i]);
        }
    }
    int nPersonTable(int n);    //判断n人能否就坐，若可以，返回最小桌子容量
    void useTable(int n, Customers* c);     //占用一张n人桌
    void freeTable(int n, Table* T);    //释放一张n人桌
    Table* latestTable();    //找到被占用桌子中最先被释放的桌子
};
```

算法流程



其中 arrange Customers:




i = 0
i < number of customers ?
yes
no
customer waiting ?
end
yes
valid table ?
yes
no
sit down
no
i++
free a table
update current time

详细设计

由于代码部分内容较多，因此，仅对关键部分进行详细说明，其余部分不在报告中展示。所有代码在源文件中均有详细的注释说明。

arrange Customers部分

这部分代码流程见上面的流程图，具体代码及解释如下：

```
//关键时间点：顾客到达，顾客离开(释放桌子)
//安排每组顾客的就餐，在此过程中可获得每组顾客的就坐时间，等待时间，离开时间
Time* currentTime = startTime;  //用currentTime来记录当前关键时间点
int cus_num = 0;        //cus_num记录当前被安排的顾客序号
Table* toFree;      //记录即将释放的桌子
while (cus_num < cus.size()) {
    //首先根据时间判断当前是否有已到达顾客，若有已到达顾客，对当前顾客，
    //判断是否有可容纳的空闲桌子，若有桌子，则给当前顾客安排入座；否则，
    //按时间顺序释放桌子；若没有已到达顾客，找到下一个关键时间点
    int cusWait = currentTime->comp(currentTime, cus[cus_num]->getArriveTime());
//cusWait记录当前顾客是否已经到达
    int capa = danyuanRes->nPersonTable(cus[cus_num]->getPersonNum());      //capa记录当前
顾客能入座要求的桌子容量
    if (cusWait > Early) {  //当前时刻有已到达顾客
        if (capa > 0) {     //当前有能容纳这组顾客的桌子
            cus[cus_num]->setSettleDown(currentTime);   //cus[cus_num]在当前时刻就坐
            cus[cus_num]->setLeaveTime();
            cus[cus_num]->setWaitTime();
            danyuanRes->useTable(capa, cus[cus_num]);   //cus[cus_num]占用一张capa人桌
            cus_num++;
        }
        else {      //当前没有能容纳这组顾客的桌子，按时间顺序释放桌子
            toFree = danyuanRes->latestTable();
            currentTime = toFree->getFreeTime();
            danyuanRes->freeTable(toFree->getCapacity(), toFree);
        }
    }
    else {  //当前没有已经到达的顾客，检查触发下一关键时间点的事件是顾客到达还是顾客离开，
        //若为顾客到达，更新时间点为顾客到达时间；否则，按时间顺序释放桌子即可
        Time* nextArr = cus[cus_num]->getArriveTime();      //顾客到达时间
        toFree = danyuanRes->latestTable();     //toFree == NULL说明没有可释放桌子
        if (toFree == NULL) {
            currentTime = cus[cus_num]->getArriveTime();
        }
        else {
            int k = nextArr->comp(nextArr, toFree->getFreeTime());
            if (k == Early)     //关键时间点是顾客到达
                currentTime = cus[cus_num]->getArriveTime();
            else {  //关键时间点是顾客离开
                currentTime = toFree->getFreeTime();
                danyuanRes->freeTable(toFree->getCapacity(), toFree);
            }
        }
```

```
    }
}
```

latestTable()

latestTable()为定义在Restaurant类中的函数，其作用是找到当前最先被释放的桌子。其实现方法是遍历当前所有被占用的桌子，比较得出最先被释放的桌子。

```
//找到被占用桌子中最先被释放的桌子
    Table* latestTable() {
        Table* latest = NULL;   //记录最先被释放的桌子
        for (int i = 1; i <= MaxCapacity; i++) {
            if (busyTables[i].size()) {     //只有在该类桌子有被占用的时才进行比较
                if (latest == NULL)     //遇到的第一张桌子首先被认为是最先释放的
                    latest = busyTables[i][0];
                Table* tmp = busyTables[i][0];
                int j = 0;
                while (j < busyTables[i].size()) {
                    Time* t = busyTables[i][j]->getFreeTime();
                    int res = t->comp(t, latest->getFreeTime());
                    if (res == Early)   //若当前被访问的桌子释放时间比latest早，则更新latest
为当前桌子
                        latest = busyTables[i][j];
                    j++;
                }
            }
        }
        return latest;
    }
```

sort Customers部分

对顾客排序时采用了插入排序算法，具体实现如下：

```
//对顾客数组cus按时间先后排序，采用插入排序法
for (int i = 0; i < cus.size(); i++) {
    int j = i;
    Customers* tmp = cus[i];
    Time* t_i = tmp->getArriveTime();
    while (j > 0) {
        Time* t_j;
        t_j = cus[j -1]->getArriveTime();
        int compT = cus[i]->getArriveTime()->comp(t_i, t_j);
        if (compT == Early) {
            cus[j] = cus[j - 1];
        }
        if (compT == Late) {
            cus[j] = tmp;
            break;
        }
        if (compT == Same) {    //时间相同时，按照人数由小到大排序
            if (tmp->getPersonNum() < cus[j - 1]->getPersonNum()) {
```

```
                cus[j] = cus[j - 1];
            }
            else {
                    cus[j] = tmp;
                    break;
            }
        }
        j--;
    }
    if (j == 0)     //当前元素是目前最小
        cus[j] = tmp;
}
```

generate input部分

在这一部分中，为了使随机生成的数据满足要求，用了较多的控制条件，具体情况如下：

- 生成顾客组数

```
//随机生成顾客组数CusNum
int CusNum = rand() % 200 + 101;    //保证顾客组数大于100，不超过300
```

- 生成桌子情况

```
    //随机生成桌子情况，并写入input_table.txt
    ofstream genInputTable("input_table.txt");
    int flag[MaxCapacity + 1] = { 0 };      //flag[i] = 0表示没有i人桌，flag[i] = 1表示已有i人桌
    for (int i = 0; i < MaxCapacity; i++) {
        int a = rand() % MaxTableNum + 1;   //保证每种桌子数目不多于MaxTableNum，也大于0
        int b = rand() % MaxCapacity + 1;   //保证桌子容量大于0，且不多于MaxCapacity
        if (flag[b] == 0) {
            genInputTable << a << ' ' << b << endl;
            flag[b] = 1;
        }
    }
    if (flag[MaxCapacity] == 0)
        genInputTable << 1 << ' ' << MaxCapacity << endl;   //保证所有组都能有可以容纳的桌子
    genInputTable.close();
```

```
#define MaxCapacity 10      //每张桌子最多能容纳的人数
#define MaxTableNum 5       //每种桌子最多数目
```

- 生成顾客情况

```
    //随机生成顾客情况，并写入input_customer.txt
    ofstream genInputCustomer("input_customer.txt");
```

```
    for (int i = 0; i < CusNum; i++) {
        int n = rand() % MaxCapacity + 1;       //保证就餐人数不超过最大桌子容量，且大于0
        int h = (rand() % 14 + 11) % 24;        //保证小时数在11--01(次日)之间，这里为了防止
最后一组离开时刻超过第二天2:00，故将到达时间的下限设置的比较靠前
        int m = rand() % 60;
        int eatT = rand() % 21 + 10;    //保证用餐时间在10--30min
        string s1 = to_string(h) + ":" + to_string(m);
        string s2 = "0:" + to_string(eatT);
        genInputCustomer << n << ' ' << s1 << ' ' << s2 << endl;
    }
    genInputCustomer.close();
```

output部分

- output_data.txt
  - 这里的平均停留时间和平均等待时间都被声明为double型变量，但是在输出时利用round()函数进行了四舍五入为整数。

```
//写入output_data.txt
outputData << "平均停留时间：" << round(aveStayTime) << "min" << endl;
outputData << "平均等候时间：" << round(aveWaitTime) << "min" << endl;
outputData << "最后一组顾客离开时间：" << currentTime->Time2str() << endl;
```

  - 获得最后一组离开时间：
    将所有被占用的桌子一次释放并更新currentTime，最终得到的currentTime即为最后一组离开时间。

```
//获得最后一组顾客离开时间
//若桌子全都空闲，说明所有顾客均已离开，否则，按时间顺序
//释放所有桌子，最终得到的currentTime即为最后一组离开时间
toFree = danyuanRes->latestTable();
while (toFree != NULL) {
    currentTime = toFree->getFreeTime();
    danyuanRes->freeTable(toFree->getCapacity(), toFree);
    toFree = danyuanRes->latestTable();
}
```

- output_customer.txt
  在Customers类中定义了一个customerInfo()函数，可将顾客信息输入到指定文件中，因此，遍历所有顾客并调用该函数即可。

```
    //输出顾客信息到文件
    void customerInfo(ofstream &outputFile) {
        outputFile << setw(3) << setfill('0') << ID << '\t';
        outputFile << personNum << '\t';
        outputFile << arriveTime->Time2str() << '\t';
        outputFile << waitTime / 60 << ":" << setw(2) << setfill('0') << waitTime % 60 <<
'\t';
        outputFile << settleDown->Time2str() << '\t';
        outputFile << "0:" << setw(2) << setfill('0') << eatTime << '\t';
```

```
        outputFile << leaveTime->Time2str() << endl;
    }
```

```
    //写入output_customer.txt
    outputCustomer << "编号" << '\t'
        << "顾客人数" << '\t'
        << "到来时刻" << '\t'
        << "等候用时" << '\t'
        << "就餐时刻" << '\t'
        << "就餐用时" << '\t'
        << "离开时刻" << '\t' << endl;
    for (int i = 0; i < cus.size(); i++) {
        cus[i]->customerInfo(outputCustomer);
    }
```

调试分析

测试

运行程序即可生成旦苑大饭店的桌子信息以及顾客信息，并根据生成的桌子信息和顾客信息进行座位安排，安排后的信息被存储到output_data.txt和output_customer.txt中。
运行成功后，会输出以下信息：

```
Success!
Check the following files to know the work it has done:
input_table.txt
input_customer.txt
output_data.txt
output_customer.txt
```

由于input_customer.txt和output_customer.txt中的数据行数太多，不方便截图展示，因此将上述四个文件全部放在 /test文件夹中一并提交。

P.S. 由于input_data.txt和input_customer.txt都是随机生成的，故每次运行程序其输入都不同，要想对已有的input_table.txt和input_customer.txt进行测试，需要将main.cpp中生成input_table.txt和input_customer.txt部分的代码注释掉。

时间复杂度分析

设顾客组数为n，桌子总数为m，较重要过程如下：

- 对顾客排序：采用插入排序算法，时间复杂度最差为O(n^2)
- First Come First Served过程：单层循环，最差每组顾客被访问两次(来时一次，走时一次)，时间复杂度为O(n)；在每次访问时，均要调用latestTable()来获取当前最先被释放的桌子，该函数要遍历所有被占用的桌子，时间复杂度为O(m)；总的时间复杂度为O(n*m)
- 计算平均停留时间和平均等位时间：遍历一次，时间复杂度为O(n)

综上，总的时间复杂度为O(n^2 + n*m)

调试改进

- 调试时对数据的输出样式做了改进，以便其输出格式和文档要求一致。即，时刻按照"xx:xx"格式输出，时间段按照"x:xx"格式输出
- 调试时发现快排算法实现有误，若当前值为目前最小值，则无法正确排序。调试后发现是碧娜姐条件没有控制好，导致第一个值无法正确更新

算法改进

由上面的时间复杂度分析可知，影响本程序时间复杂度的主要是排序算法和latestTable()，故采用平均性能更好的排序算法以及改进latestTable()函数的实现能让程序更好。
选用其他的排序算法，如，堆排序、归并排序等可以更快，但实现起来要更复杂。
latestTable()函数的实现是最朴素的遍历所有值，选出最小值的方法，但是目前我并不能想出更好且实现起来不复杂的算法。

课程设计总结

本次项目让我体会到了完整解决一个问题的全过程，对C++的类和对象有了直观的认识(上机课的时候没用过，感受不到)。 本次项目让我有机会亲手实现了First Come First Served调度算法，对我期末季操作系统的复习页非常有帮助。 而且这次项目进展很顺利，每当我期末季复习到暴躁时，停下来写会儿代码总能让我心情平复下来，不至于复习到崩溃。 最终能独立完成且如此顺利也让我非常有成就感。

参考资料

本项目完全独立完成，未参考资料。 本项目代码已上传GitHub个人主页：https://github.com/o-potato/Data-Structure-Final-PJ