麦当劳点餐系统概要设计书

2023211311-2023211198-杜昊阳

**OJ版部分**

# 一．麦当劳点餐系统问题描述

## 1.题面：

2023年5月，麦当劳在北邮开业。大量的学生去那里订餐。正因为如此，麦当劳的在线点餐系统经常关闭以避免拥挤，尤其是在午餐和晚餐时间。该系统的关闭时间不确定。北邮的学生认为这非常麻烦。

然而，北邮学生无所畏惧。北京邮电大学最优秀的学生之一(也是北邮ICPC团队的一员)Zhai Xie (ThomasX)在飞书上开发了一个实时监控机器人，它告诉我们麦当劳在线点餐系统的实时状态。有了这个机器人，北邮学生可以更方便地点餐。

在这个问题中，需要你像他一样完成这个任务，开发一个系统来模拟麦当劳的在线点餐系统。

北邮的麦当劳和它的点餐系统在07:00:00开始工作，在22:00:01关闭。麦当劳一共有N种食物和M种套餐类型，每种套餐中包含多种食物，具体配置信息将在菜单文件 (dict.dic) 中提供。对于制作和存储每种食物，规定第i种食物在t\_i秒内完成，其最大存储容量为cap\_i ，表示该种食物最多可以存储cap\_i个。麦当劳系统每天开放前，所有食物存储容量都为0，在任何时间点如果某种食物的存储量小于cap\_i，则会立即制作该食物，直到达到cap\_i。其中，不同种类食物可以同时制作，同种类食物只能依次制作。

从07:00:00到22:00:00(含)，学生可以在系统中点餐(如果系统未关闭)。每一天按照顺序有n个订单，第i个订单发生在时间a\_i:b\_i:c\_i，其要求一份type\_i类型(type\_i∈M\_combo∪N\_food，其中M\_combo和N\_food分别表示全体的套餐和食物的集合)的套餐或食物。如果点餐时系统关闭，会导致点餐失败。22:00以后如果还有之前的订单未完成，则麦当劳会继续加班，且保证23:59:59(含)前一定能完成所有订单。

对于订单处理存在如下规则：

在每一秒的开始，如果有新的食物完成，则首先存储食物，然后接受订单(如果存在)。

订单按照“先来先到，异步处理”原则进行处理。

先来先到：指的是对于有存量的食物，总会被分配给时间最早的订单(套餐或单点)。

异步处理：指的是当一个订单(套餐或单点)因为请求的食物没有被全部满足时，不必等待该订单完成，可以直接处理下一个订单。

食物一旦被分配给订单，就不能撤销。食物被分配给订单后，即便该订单尚未完成，该食物也不再占用对应类型的容量。

当订单(套餐或单点)中要求的所有食物，均已被分配给该订单，则该订单会立刻完成。

如果在某个时刻t\_0，有人下了一个订单，并且该订单无法立刻完成，导致未完成订单的数量大于W\_1，则系统立即自动关闭(不再接受订单)，但该订单仍然算作成功下单。

如果在某个时刻t\_1，未完成订单的数量小于W\_2，则系统将在1秒后重新打开。即系统可以接受t\_1+1时刻的订单，而不能接受t\_1时刻的订单。

你的系统需要输出：每一个订单是否下单成功，以及完成的时间。

## 2.菜单文件

本题为大家提供麦当劳的菜单文件(dict.dic)，按如下格式给出：

第一行给出和，其中表示食物的种类数()，表示套餐的种类数()。

第二行包含个字符串，每个字符串 表示第种食物的名称(不超过50个字符)。

第三行包含个整数()，其中表示第种食物的制作时长。

第四行包含个整数()，其中表示第种食物的最大存储容量。

第五行包含两个整数()。

接下来行，其中的第行包含多个字符串，第一个字符串表示第个套餐的名称(不超过50个字符)，后续的第个字符串表示第个套餐中包含的第种食物的名称。一个套餐包含的食物种类不超过5，每种食物只有1个。

注：系统每次运行时所读取的菜单文件内容可能不一样。

## 3.输入

第一行包含一个整数()表示订单个数。

对于接下来的行，用格式类似于的方式，给出第个订单的时间。然后输入一个字符串，表示套餐或食物的名称(参见dict.dic)。所有订单时间一定在内，同一个时间点不可能出现多个订单，第个订单一定早于第个()，且保证23:59:59(含)前一定能完成所有订单。

具体参见input.txt

## 4.输出

输出包括n行，按照订单顺序输出订单完成时间。对于第i行，如果第i个订单不成功，则输出Fail；否则，输出这个订单完成的时间，时间格式与输入格式(11:11:11)一致。

具体参见output.txt

# 二．需求分析

## 1.总述

开发的第一步是进行需求分析。需求分析需要从系统的数据、功能 和行为三方面进行分析。

数据方面，应当建立餐品数据，套餐数据，订单数据，任务处理数据。

功能方面，包含菜单读取，当日任务读取，订单完成情况汇报。

行为方面，包括菜单数据处理，当日任务数据处理，执行当日任务，订单完成情况输出。

## 2.数据配置

### （1）餐品数据：

struct foodkind{//食物

char name[51];//名称

int number;//当前数量

int max;//最大数量

int currenttime;//当前已制作时间

int needtime;//制作所需时间

};

### （2）套餐数据：

struct combination{//套餐

char name[51];//名称

int kindnumber;//餐品数量

struct foodkind \*link[5];//对应餐品种类

};

### （3）订单数据：

struct order{//订单

int begintime;//开始时间

int finishtime;//结束时间

int foodnumber;//餐品数量

int state;//0未开始，1成功，2失败

struct foodkind \*link[5];//对应餐品种类

};

### （4）任务处理数据：

//建立任务处理数据

int hour1;//hh:mm:ss时间-第1位

int hour2;//hh:mm:ss时间-第2位

int minute1;//hh:mm:ss时间-第3位

int minute2;//hh:mm:ss时间-第4位

int second1;//hh:mm:ss时间-第5位

int second2;//hh:mm:ss时间-第6位

int foodnumber=0;//餐品种类数量

int combnumber=0;//套餐种类数量

int ordenumber=0;//订单数量

int allowmax=0;//系统关闭订单量w1

int allowmin=0;//系统恢复订单量w2

int remainorder=0;//滞留订单量

int systemstate=1;//系统开启情况，0关闭，1开启。

int checkmin;//滞留订单最早结束时间

//建立临时变量

int i;

int j;

int k;

int temp;

char ch[51]={0};

char c;

## 3.功能配置：

### （1）菜单读取功能

菜单文件读取：

第一行读取foodnumber和combnumber，其中foodnumber表示餐品的种类数，combnumber表示套餐的种类数。

第二行包含foodnumber个字符串，每个字符串表示第i种食物的名称。

第三行包含foodnumber个整数，其中第i个数字表示第i种食物的制作时长。

第四行包含foodnumber个整数，其中第i个数字表示第i种食物的最大存储容量。

接下来combnumber行，其中的第i行包含多个字符串，第一个字符串表示第i个套餐的名称，后续的第j个字符串表示第i个套餐中包含的第j种食物的名称。

### （2）当日任务读取

输入读取：

第一行包含一个整数n(1≤n≤54001)表示订单个数。

对于接下来的n行，用格式类似于11:11:11的方式，给出第i个订单的时间。然后输入一个字符串type\_i，表示套餐或食物的名称(参见dict.dic)。所有订单时间一定在[07:00:00,22:00:00]内，同一个时间点不可能出现多个订单，第i-1个订单一定早于第i个(2≤i≤n)，且保证23:59:59(含)前一定能完成所有订单。

### （3）订单完成情况汇报

输出包括n行，按照订单顺序输出订单完成时间。对于第i行，如果第i个订单不成功，则输出Fail；否则，输出这个订单完成的时间，时间格式与输入格式(11:11:11)一致。

## 4.行为配置

### （1）数据处理

创建餐品，写入餐品名称，最大数量，制作所需时间，同时将当前数量和当前制作时间初始化为0。

创建套餐，写入套餐名称，餐品数量，将对应餐品所需数量设置为1，同时将各餐品情况的指针指向餐品。

创建订单，写入对应套餐数据，记录订单剩余未完成餐品数，订单开始时间，订单状态，订单内具体餐品完成情况。

### （2）执行当日任务

即开始营业，从第一个订单到最后一个订单，进行食物制作和订单处理。

记录每个订单完成状态（未开始，成功，失败），若成功，记录完成时间。

### （3）订单完成情况输出

每日任务完成后，按订单顺序输出完成情况/完成时间。

# 三．概要设计

## 1.数据联系图

## 2.主程序设计

主程序{

读取菜单

读取输入

当日营业

订单情况汇报

}

## 3.子程序设计

### （1）读取菜单

FILE \*fp;

fp=fopen("dict.dic","r");

fscanf(fp,"%d%d",&foodnumber,&combnumber);//读取餐品数和套餐数

combnumber+=foodnumber;//单点商品也算作套餐

struct foodkind food[foodnumber];//创建餐品数组

struct combination comb[combnumber];//创建套餐数组

for(i=0;i<foodnumber;i++) {//读取和创建餐品数据

fscanf(fp,"%s",ch);//读取餐品名称

strcpy(food[i].name,ch);

food[i].number=0;

food[i].max=0;

food[i].currenttime=0;

food[i].needtime=0;

strcpy(comb[i].name,ch);//将餐品也定义为套餐

comb[i].kindnumber=1;

comb[i].link[0]=&food[i];

}

for(i=0;i<foodnumber;i++) {//读取各餐品制作所需时间

fscanf(fp,"%d",&food[i].needtime);

}

for(i=0;i<foodnumber;i++) {//读取各餐品最大容量

fscanf(fp,"%d",&food[i].max);

}

fscanf(fp,"%d%d",&allowmax,&allowmin);//读取系统关闭订单量w1，系统恢复订单量w2

for(;i<combnumber;i++) {//读取多餐品套餐

fscanf(fp, "%s", ch);//读取套餐名称

strcpy(comb[i].name, ch);

comb[i].kindnumber = 0;

j = 0;

for (c =fgetc(fp); c==' '; c =fgetc(fp)) {//读取套餐中对应餐品名称并记录数据

fscanf(fp, "%s", ch);

comb[i].link[j] = &food[correctfood(ch, food, foodnumber)];

j++;

comb[i].kindnumber++;

}

}

fclose(fp);

**（2）读取输入**

scanf("%d",&ordenumber);//读取订单数

struct order orde[ordenumber];//创建订单数组

for(i=0;i<ordenumber;i++) {//读取订单数据

orde[i].begintime=timeread();//读取各订单开始时间并转化为时间戳

orde[i].finishtime=0;

orde[i].state=0;

scanf("%s",ch);

j=correctcomb(ch,comb,combnumber);//找出订单对应套餐编号

orde[i].foodnumber=comb[j].kindnumber;

for(k=0;k<comb[j].kindnumber;k++) {

orde[i].link[k]=comb[j].link[k];

}

}

**（3）当日营业**

①整体：

while(currentorder<ordenumber) {//按订单循环

if(进入下一时间点){

食物制作;

系统状态判断;

}

处理订单；

寻找下一时间点;

}

②食物制作

for (i = 0; i < foodnumber; i++) {

if (food[i].number < food[i].max) {

food[i].number += (minustime + food[i].currenttime) / food[i].needtime;

food[i].currenttime = (minustime + food[i].currenttime) % food[i].needtime;

}

if (food[i].number >= food[i].max) {

food[i].number = food[i].max;

food[i].currenttime = 0;

}

}

③系统状态判断

if(remainorder>allowmax)

systemstate=0;

if(remainorder<allowmin)

systemstate=1;

④处理订单

if (state == 1) {//有新增订单

if (systemstate == 1) {

orde[currentorder].state = 1;

orde[currentorder].finishtime = orde[currentorder].begintime;

for (j = 0; j < orde[currentorder].foodnumber; j++) {

orde[currentorder].link[j]->number--;

if (orde[currentorder].link[j]->number < 0)

temp = -orde[currentorder].link[j]->number \* orde[currentorder].link[j]->needtime - orde[currentorder].link[j]->currenttime+orde[currentorder].begintime;

orde[currentorder].finishtime = (orde[currentorder].finishtime > temp ? orde[currentorder].finishtime : temp);

}//处理并计算结束时间

if (orde[currentorder].finishtime > orde[currentorder].begintime) {

remainorder++;

for (temp = 0; temp <= allowmax; temp++) {

if (check[temp] == 90000) {

check[temp] = orde[currentorder].finishtime;

break;

}

}

}

}

else {

orde[currentorder].state = 2;

}

currentorder++;

}

if (state == 0) {//有结束订单

remainorder--;

for (temp = 0; temp <= allowmax; temp++) {

if (check[temp] == nowtime) {

check[temp] = 90000;

break;

}

}

}

⑤判断下一时间节点

for(i=0;i<=allowmax;i++){

checkmin=90000;

for (i = 0; i <= allowmax; i++){

checkmin=(checkmin<check[i]?checkmin:check[i]);

}

if(checkmin>=orde[currentorder].begintime){

state=1;

nowtime = orde[currentorder].begintime;

}

else{

state=0;

nowtime=checkmin;

}

}

### （4）订单完成情况汇报

for(i=0;i<ordenumber;i++) {

if(orde[i].state==3)//orde[i]失败

printf("Fail\n");

else {//orde[i]成功

//将时间戳转化为hh:mm:ss时间

second1=orde[i].finishtime%60;

second2=second1%10;

second1=second1/10;

minute1=(orde[i].finishtime%3600)/60;

minute2=minute1%10;

minute1=minute1/10;

hour1=orde[i].finishtime/3600;

hour2=hour1%10;

hour1=hour1/10;

printf("%d%d:%d%d:%d%d\n",hour1,hour2,minute1,minute2,second1,second2);

}

}

return 0;

}

## 4.辅助程序设计

int correctfood(char ch[],struct foodkind food[],int foodnumber) {//返回餐品名称ch对应序号

int n;

for(n=0;n<foodnumber;n++) {

if(strcmp(ch,food[n].name)==0)

return n;

}

}

int correctcomb(char ch[],struct combination comb[],int combnumber) {//返回套餐名称ch对应序号

int n;

for(n=0;n<combnumber;n++) {

if(strcmp(ch,comb[n].name)==0)

return n;

}

}

int timeread() {//将hh:mm:ss时间转化为时间戳

int hour=0,minute=0,second=0,time=0;

scanf("%d:%d:%d",&hour,&minute,&second);

time=second+60\*minute+3600\*hour;

return time;

}

**图形化部分**

# 工作概述

## 1.菜单文件

本题为大家提供麦当劳的菜单文件(dict.dic)，按如下格式给出：

第一行给出N和M，其中N表示食物的种类数(1≤N≤100)，M表示套餐的种类数(1≤M≤100)。

第二行包含N个字符串，每个字符串 name\_i^food表示第i种食物的名称(不超过50个字符)。

第三行包含N个整数t\_1,t\_2,...,t\_N(1≤t\_i≤70)，其中t\_i表示第i种食物的制作时长。

第四行包含N个整数cap\_1,cap\_2,...,cap\_N(1≤cap\_i≤54001)，其中cap\_i表示第i种食物的最大存储容量。

第五行包含两个整数W\_1,W\_2(2≤W\_2≤W\_1≤100)。

接下来M行，其中的第i行包含多个字符串，第一个字符串name\_i^combo表示第i个套餐的名称(不超过50个字符)，后续的第j个字符串name\_(i,j)^food表示第i个套餐中包含的第j种食物的名称。一个套餐包含的食物种类不超过5，每种食物只有1个。

## 2. 输入发生改变

总订单量由已知变为未知。

在运行到t秒时在图形界面选择食物/套餐，相当于原来的订单输入

e.g. 第07：00：05秒在图形界面选择Croutons

相当于输入：07:00:05 Croutons

## 3. 输出发生改变

运行时：

在图形界面显示系统时间（模拟）

在图形界面显示当前未完成订单信息

运行结束：

在图形界面显示所有订单信息

所选食物 开始时间 是否成功 结束时间（如果有）

生成账单信息文件

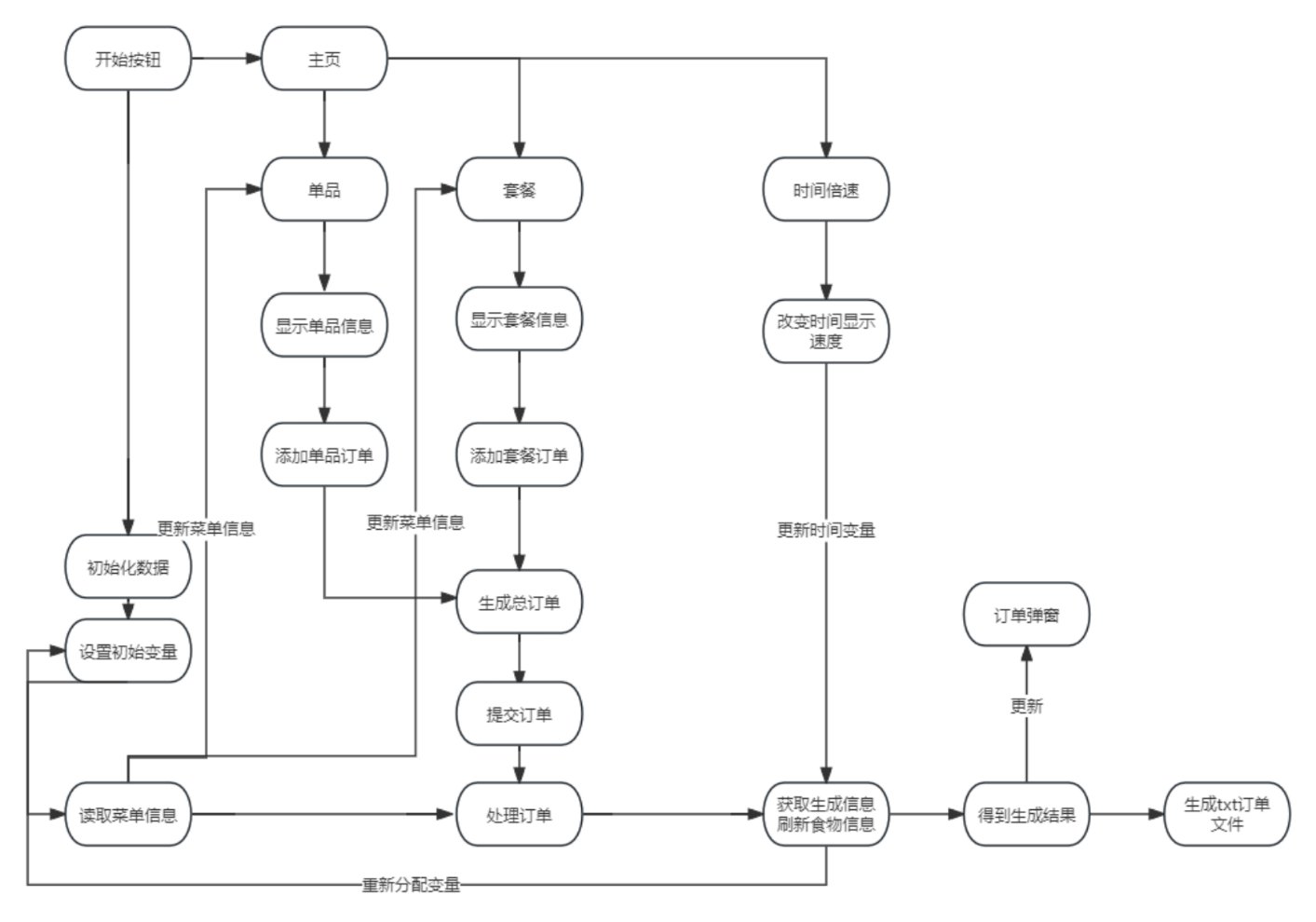
# 开发过程

## 开发软件

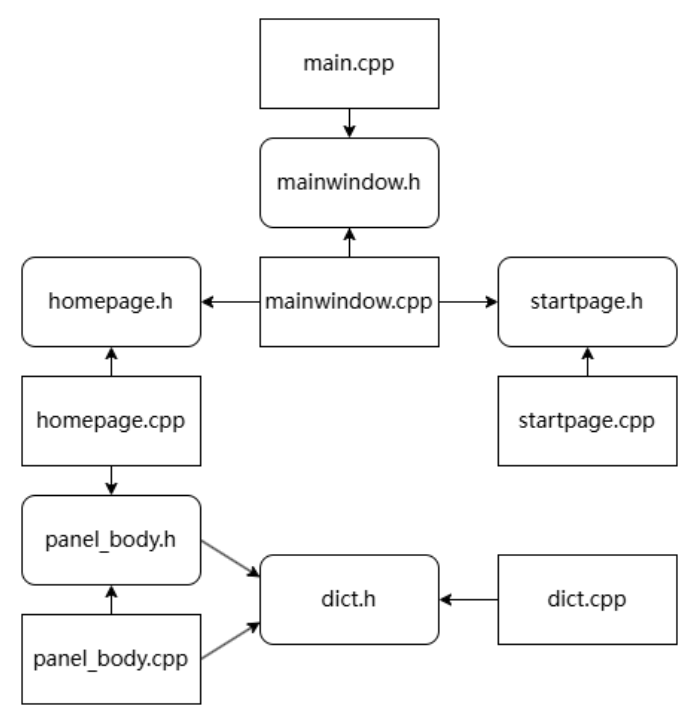
1. IDE:Clion/Visual Studio Code
2. 图形库：Qt5
3. 在线版本控制：Gitee+Sourcetree

## 2.解决方案

1. 设计思路



1. 模块划分



·dict.h : OJ版的核心函数，包含了OJ版的所有功能。

·homepage.h : 程序的主页函数，实现了单品、套餐的切换以及查看订单功能。

·mainwindow.h： 构建主体窗口。

·startpage.h: 实现了程序的登录界面。

·panel\_body.h: 实现了单品界面中心滚动菜单的功能，能根据读取菜单调整单品 图片、名称、阁窗大小。

## 实验总结

调试是开发过程中必不可少的一部分。CLion提供了强大的调试工具，能够设置断点、查看变量值、调用栈等。对于Qt项目，调试同样顺畅，特别是在处理GUI事件和槽函数时，CLion的调试器能帮助快速定位问题。需要注意的是，有时候调试Qt的插件可能会带来一些兼容性问题，这时候需要确保调试配置与Qt版本的匹配。

总的来说，使用CLion进行Qt5开发是一种高效的选择。虽然初次配置可能略显复杂，但一旦配置完成，CLion提供的强大功能和舒适的开发体验会显著提升开发效率。特别是对于跨平台开发者来说，CLion的多平台支持和Qt的跨平台特性相得益彰，使得开发、调试和部署变得更加便捷。