麦当劳点餐系统概要设计书

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 N 和 M，其中 N 表示食物的种类数，M 表示套餐的种类数。
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。
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，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。
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 n(1≤n≤54001)表示订单个数。
第二行包含两个整数 W_1,W_2(2≤W_2≤W_1≤100)。
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≤n)，其中 cap_i 表示第 i 种食物的最大存储容量。
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。
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 {//食物种类
    char name[30];//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 combinationlink {//套餐对应链表
    struct foodkind *link;//对应餐品种类
    struct combinationlink *next;//下一餐品
};
struct combination {//套餐
    char name[30];//名称
    int kindnumber;//餐品数量
    struct combinationlink *kind;//餐品名称
};
```

（3）订单数据：

```
struct orderlink {//订单对应链表
    struct foodkind *link;//对应餐品种类
    int provided;//是否已提供餐品
    struct orderlink *next;//下一餐品
};
struct order {//订单
    int begintime;//开始时间
    int finishtime;//结束时间
    int remainfoodnumber;//未完成餐品数量
    int state;//订单状态：0 未开始，1 进行中，2 成功，3 失败
    struct orderlink *kind;//餐品详情
};
```

（4）任务处理数据：

```
//建立任务处理数据
    int time=0;//时间戳
    int hour1=0;//hh:mm:ss 时间-第 1 位
    int hour2=0;//hh:mm:ss 时间-第 2 位
    int minute1=0;//hh:mm:ss 时间-第 3 位
    int minute2=0;//hh:mm:ss 时间-第 4 位
    int second1=0;//hh:mm:ss 时间-第 5 位
    int second2=0;//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;//进行中订单量
    int systemstate=1;//系统开启情况，0 关闭，1 开启。

//建立临时变量
    int i=0;
    int j=0;
    int k=0;
    char ch[30]={0};
    char c=0;
    struct combinationlink *combinationlinknewp;
    struct combinationlink *combinationlinkcurp;
    struct orderlink *orderlinknewp;
    struct orderlink *orderlinkcurp;
```

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。
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。
输入读取：
第三行包含 foodnumber 个整数，其中 t_i 表示第 i 种食物的制作时长。
第四行包含 foodnumber 个整数，其中 cap_i 表示第 i 种食物的最大存储容量。

（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

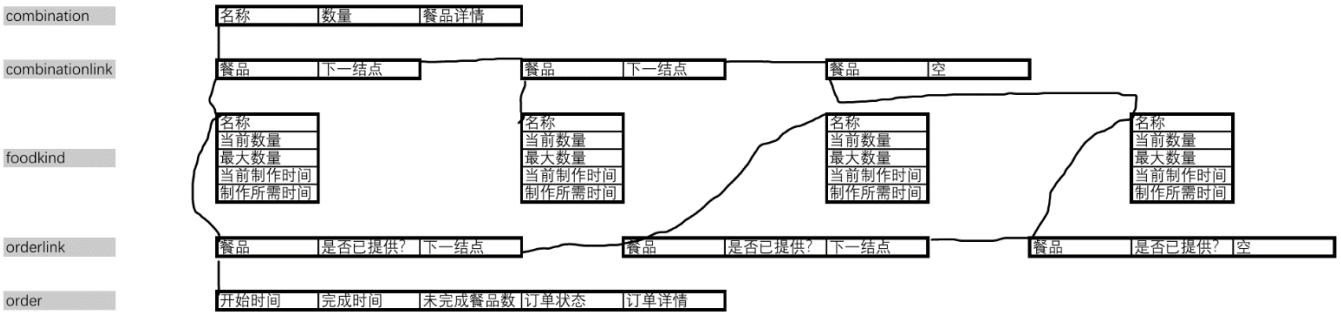
即开始营业，从 07：00：00 到 23：59：59，每秒钟进行食物制作和订单处理。
记录每个订单完成状态（未开始，进行中，成功，失败），若成功，记录完成时间。

（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];
struct combination combinations[combnumber];
for(i=0;i<foodnumber;i++) {
    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=-1;
    food[i].needtime=0;
    strcpy(combinations[i].name,ch);//将餐品也定义为套餐
    combinations[i].kindnumber=1;
```

```
    combinationlinknewp=(struct combinationlink*)malloc(sizeof(struct combinationlink));
    combinations[i].kind=combinationlinknewp;
    combinations[i].kind->link=&food[i];
    combinations[i].kind->next=NULL;
}
for(;i<combnumber;i++) {//读取多餐品套餐
    fscanf(fp,"%s",ch);
    strcpy(combinations[i].name,ch);
    combinations[i].kindnumber=0;
    combinations[i].kind=NULL;
    for(c=fgetc(fp);c!='\n';c=fgetc(fp)) {
        fscanf(fp,"%s",ch);
        combinationlinknewp=(structcombinationlink*)malloc(sizeof(struct combinationlink));
        combinationlinknewp->link=&food[correctfood(ch,food,foodnumber)];
        combinations[i].kindnumber++;
        if(combinations[i].kind==NULL) {
            combinations[i].kind=combinationlinknewp;
            combinationlinkcurp=combinationlinknewp;
        }
        else {
            combinationlinkcurp->next=combinationlinknewp;
            combinationlinkcurp=combinationlinknewp;
        }
    }
    combinationlinkcurp->next=NULL;
}
fclose(fp);
```

（2）读取输入

```
    fp=fopen("input.txt","r");
    fscanf(fp,"%d",&ordenumber);//读取订单数
    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=0;i<foodnumber;i++) {
        fscanf(fp,"%d",&food[i].needtime);
    }//读取各餐品制作所需时间
    for(i=0;i<foodnumber;i++) {
        fscanf(fp,"%d",&food[i].max);
    }//读取各餐品最大容量
    struct order orde[ordenumber];
    for(i=0;i<ordenumber;i++) {
        orde[i].begintime=timeread(fp);//读取各订单开始时间并转化为时间戳
        orde[i].finishtime=0;
        orde[i].kind=NULL;
```

```
            orde[i].state=0;
            fscanf(fp,"%s",ch);
            j=correctcomb(ch,combinations,combnumber);//找出订单对应套餐编号
            orde[i].remainfoodnumber=combinations[j].kindnumber;//将套餐信息复制给订单
            combinationlinknewp=combinations[j].kind;
            for(k=0;k<combinations[j].kindnumber;k++) {
                orderlinknewp=(struct orderlink*)malloc(sizeof(struct orderlink));
                orderlinknewp->link=combinationlinknewp->link;
                orderlinknewp->provided=0;
                combinationlinknewp=combinationlinknewp->next;
                if(orde[i].kind==NULL) {
                    orde[i].kind=orderlinknewp;
                    orderlinkcurp=orderlinknewp;
                }
                else {
                    orderlinkcurp->next=orderlinknewp;
                    orderlinkcurp=orderlinknewp;
                }
            }
            orderlinkcurp->next=NULL;
        }
        fclose(fp);
```

（3）当日营业

①整体：

```
for(time=25200;time<86400;time++) {//按秒循环
    制作食物;
    点单处理;
}
```

②制作食物：

```
    for(i=0;i<foodnumber;i++) {
        if(food[i].number<food[i].max) {
            food[i].currenttime++;
            if(food[i].currenttime==food[i].needtime) {
                food[i].number++;
                food[i].currenttime=0;
            }
        }
    }
```

③点单处理：

```
 //系统状态判断
    if(remainorder>allowmax)
        systemstate=0;
```

```
if(remainorder<allowmin)
    systemstate=1;

  //订单处理
for(i=0;i<ordenumber;i++) {
    if(orde[i].begintime>time)//未开始
        break;
    else if(orde[i].begintime==time) {//开始时
        if(systemstate==1) {//系统开放点单
            orde[i].state=1;//订单进行中
            remainorder++;
            //配餐
            for(orderlinkcurp=orde[i].kind;orderlinkcurp!=NULL;orderlinkcurp=orderlinkcurp->next) {
                if(orderlinkcurp->provided==0) {//当前餐品未配餐
                    if(orderlinkcurp->link->number!=0) {//当前餐品有存量
                        orderlinkcurp->link->number--;
                        orderlinkcurp->provided=1;
                        orde[i].remainfoodnumber--;
                    }
                }
            }

            //判断订单是否完成
            if(orde[i].remainfoodnumber==0) {
                orde[i].state=2;
                orde[i].finishtime=time;
                remainorder--;
            }
        }
        else {//系统暂停点餐
            orde[i].state=3;//点单失败
        }
    }
    else {
        if(orde[i].state==2||orde[i].state==3)//订单已结束（完成/失败）
            continue;
        else {//订单进行中
            //配餐
            for(orderlinkcurp=orde[i].kind;orderlinkcurp!=NULL;orderlinkcurp=orderlinkcurp->next) {
                if(orderlinkcurp->provided==0) {//当前餐品未配餐
                    if(orderlinkcurp->link->number!=0) {//当前餐品有存量
                        orderlinkcurp->link->number--;
                        orderlinkcurp->provided=1;
                        orde[i].remainfoodnumber--;
```

```
                    }
                }
            }

            //判断订单是否完成
            if(orde[i].remainfoodnumber==0) {
                orde[i].state=2;
                orde[i].finishtime=time;
                remainorder--;
            }
        }
    }
}
```

（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 combinations[],int combnumber) {//返回套餐名称 ch 对应序号
    int n;
    for(n=0;n<combnumber;n++) {
        if(strcmp(ch,combinations[n].name)==0)
            return n;
    }
}
int timeread(FILE *fp) {//将 hh:mm:ss 时间转化为时间戳
    int hour=0,minute=0,second=0,time=0;
    fscanf(fp,"%d:%d:%d",&hour,&minute,&second);
    time=second+60*minute+3600*hour;
    return time;
}
```