<第25组><麦当劳>概要设计

版本号：第25组-Design-麦当劳

编制时间：2024年6月18日

编制人员：张云昕

图形化成员分工：

张云昕：倍速 ui界面搭建 鼠标点餐 订单状态显示 翻页

陆泽华：时钟显示 退出输出文件

庄天婧：图形化界面的优化美化 食物存储状态显示

1. 用户界面设计
   1. 命令行版本

输入:

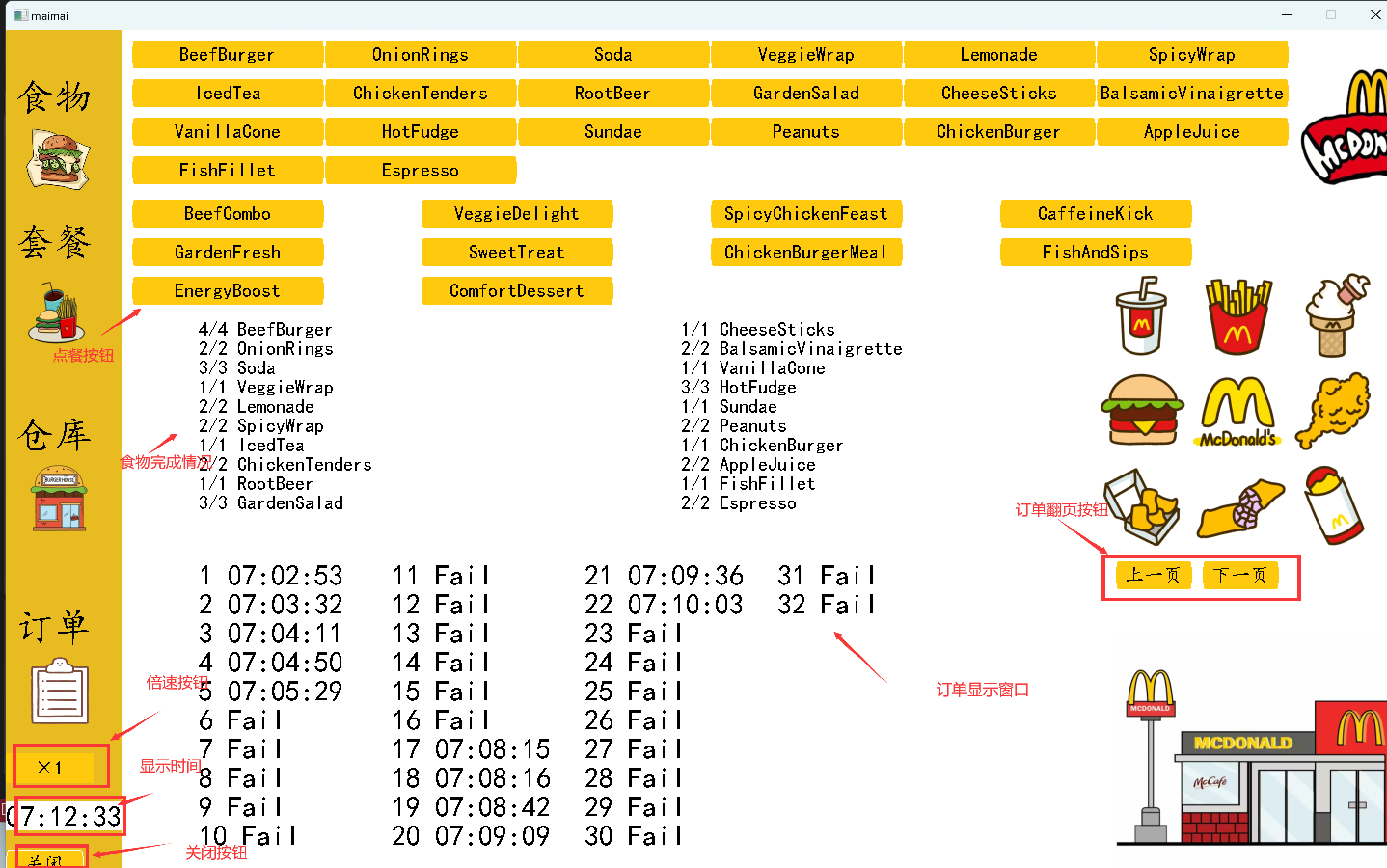
第一行包含一个整数()表示订单个数。

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。

输出:

输出包括行，按照订单顺序输出订单完成时间。对于第行，如果第个订单不成功，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()一致。

1.2图形化版本



1. 有限状态自动机状态转换图

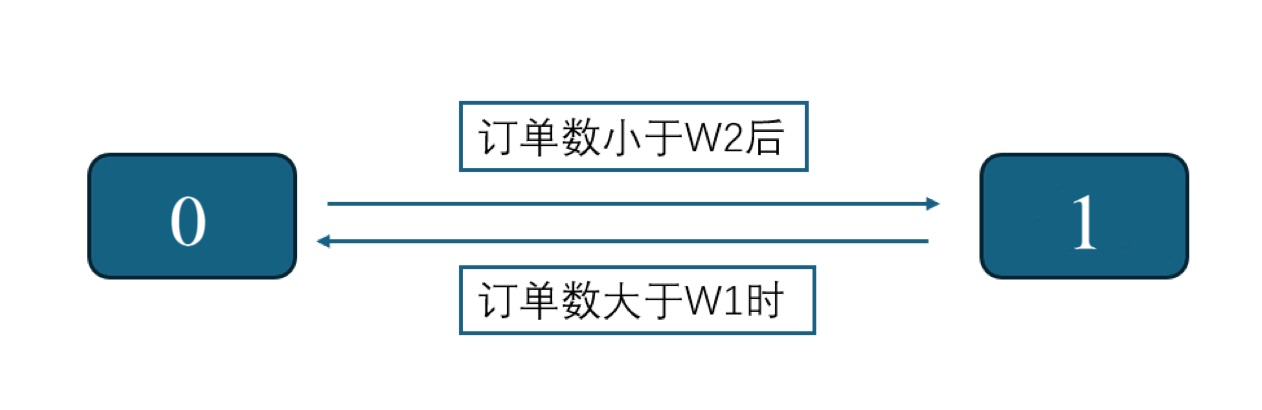
2.1 [一级状态图]

State 0:不执行接受订单操作

State 1:接受订单

state 1转化为state 0：这一秒中食物存储量超过W1且这一秒处于state1状态,则下一秒的状态为state 0;

state 0转化为state 1：这一秒中食物存储量低于W2且这一秒处于state0状态,则下一秒的状态为state 1;



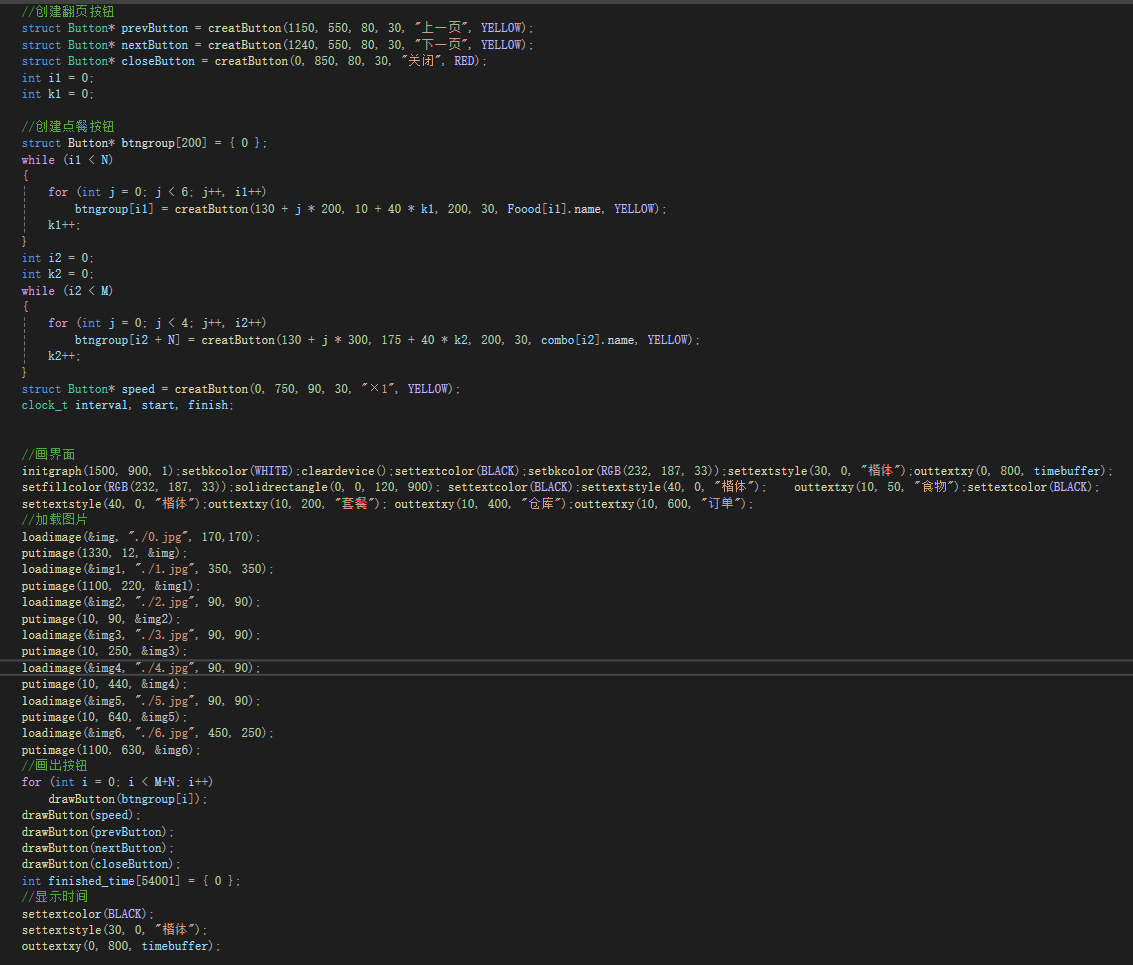
3.系统模块划分

3.1 系统模块划分

本系统程序仅有一个源文件代码maimai.c但分为四个版块.

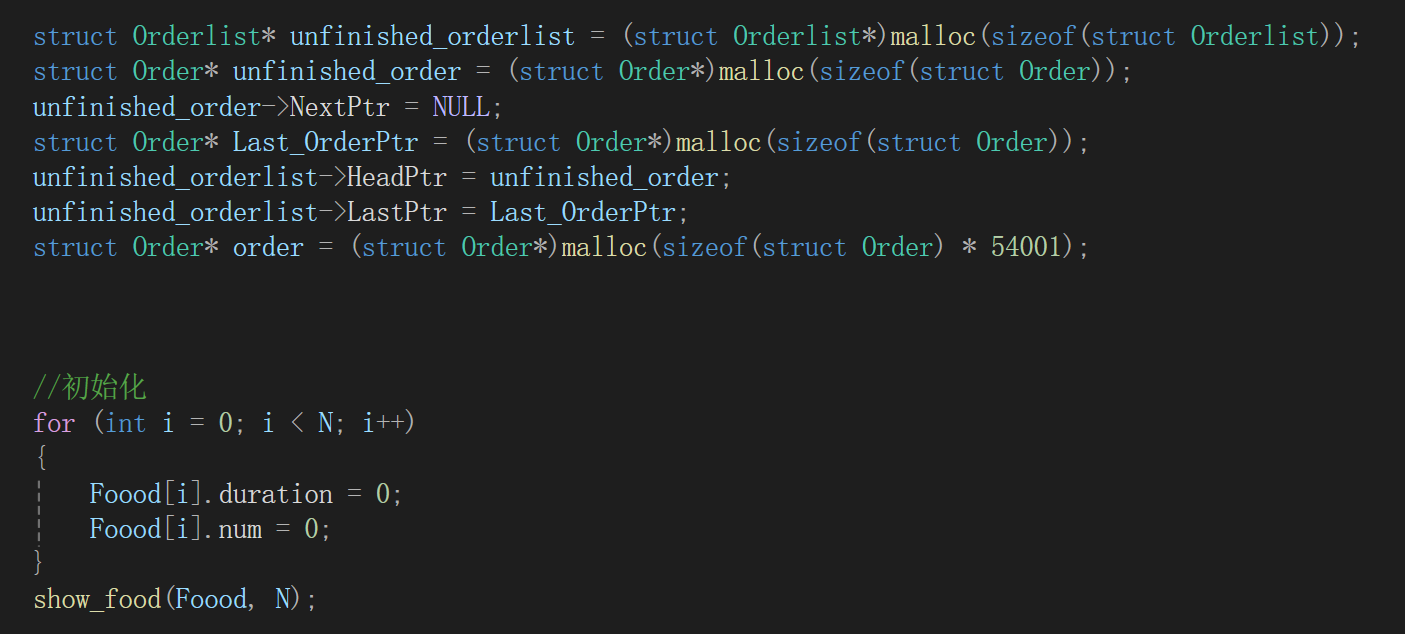
3.1.1 界面搭建

设计ui 搭建界面



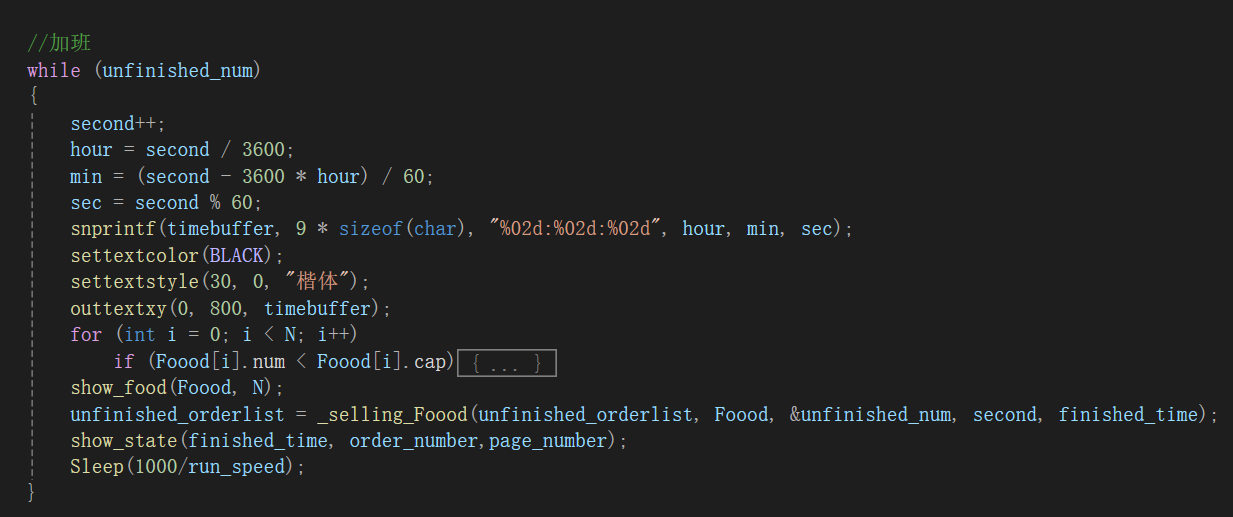
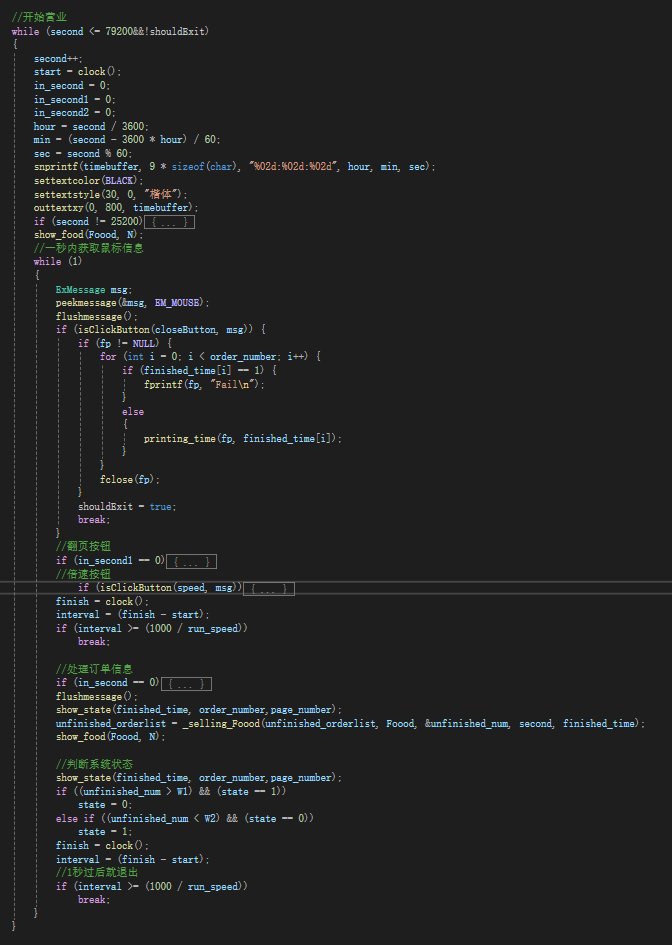
3.1.2 接受数据

创建对象并进行初始化 接受菜单信息并存储

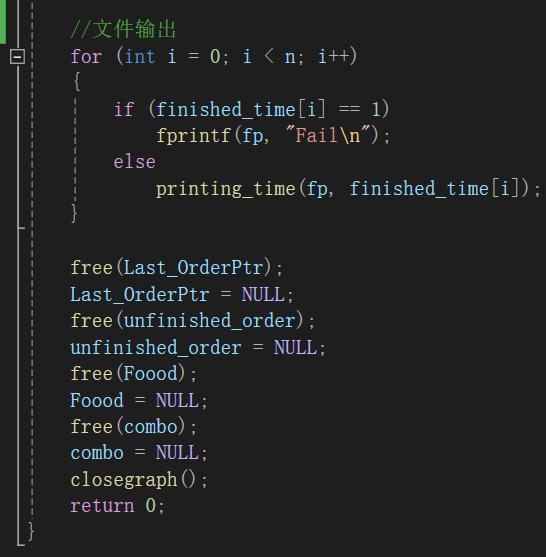


3.1.3 模拟运行

制作食物、出售食物、接受订单、处理交互信息



3.1.4 文件输出与空间释放



3.2 数据结构

此代码将食物、套餐均用结构体存储，未完成订单用链表存储，减少遍历次数；若订单为套餐，会新建一个链表存储该套餐未完成食物，减少遍历次数

食物库:

struct Food

{

char name[55] ;

int num ;

int t ;

int cap;

struct Food\* NextPtr;

int food\_number;

};

前四项均与题目输入一致

struct Food\* NextPtr 是一个自引用结构，用于为点套餐的订单制作未完成的食物链表，使得遍历时只用遍历套餐对应未完成的食物，减少遍历次数

food\_number是食物的序号，用于快速定位订单所需要的食物，不需要遍历食物列表来找到对应的食物

订单:

struct Order

{

int time;

char need\_food[55] ;

struct Order\* NextPtr;

struct Food\* NeedFood；

int order\_number;

int is\_combo;

struct Food\* food;

}

其中time用于存储订单进入时间，

need\_food用于存储订单的点的食物（或套餐）的名字，

struct Order\* NextPtr是一个自引用，用于创建未完成订单链表，使得遍历时只用遍历未完成的订单，减少遍历次数。

order\_number是按进入时间顺序排的订单序号

is\_combo用于判断该订单是否为套餐的数字，类似于布尔类型的数据，当其为0时该订单为食物，当其为1时该订单为套餐。

struct Food\* food存入的是（当订单点的是食物时）对应食物的指针，这样避免遍历食物链表比对，减少遍历次数

套餐库:

struct Combo

{

char name[55];

char food[5][55] ;

};

其中name是套餐名字

food 是包含的食物的名字

未完成订单用链表存储：

struct Orderlist

{

struct Order\* HeadPtr;

struct Order\* LastPtr;

}；

用于将未完成食物链表打包成一个整体，便于增减传输

3.3 函数说明

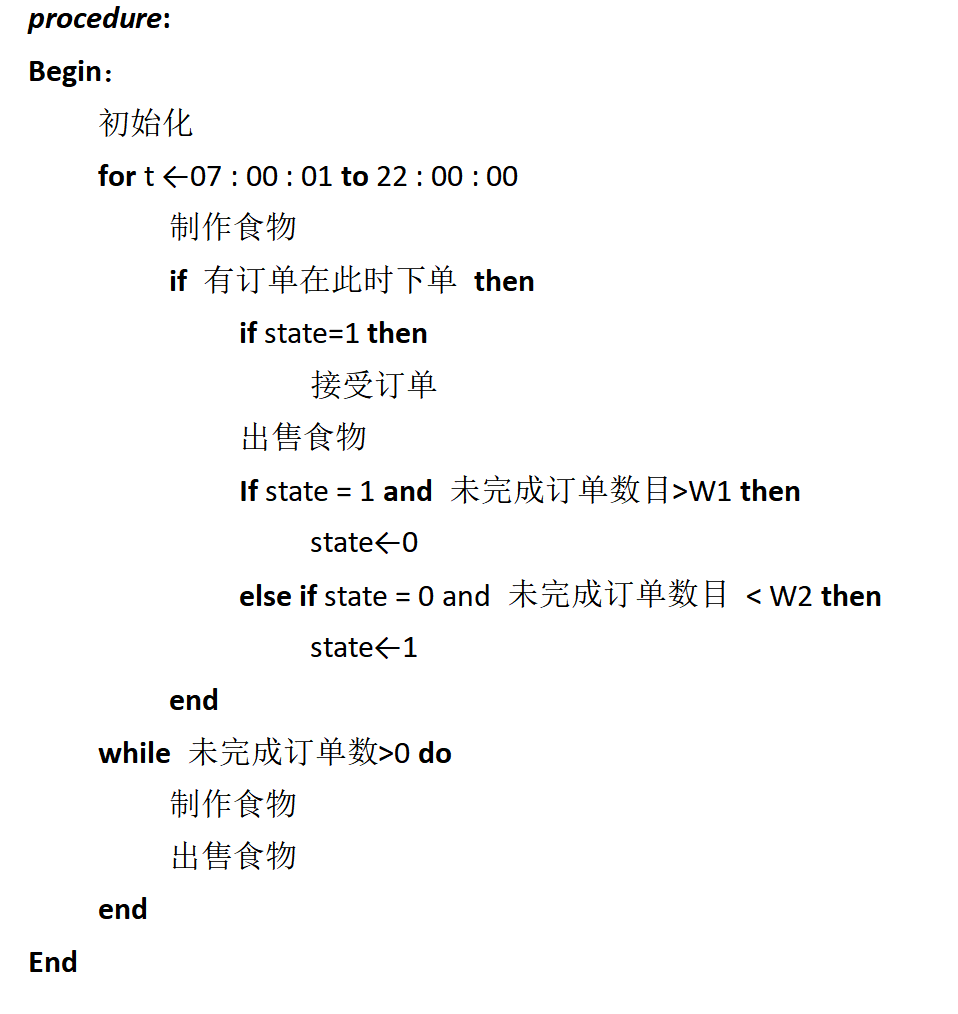
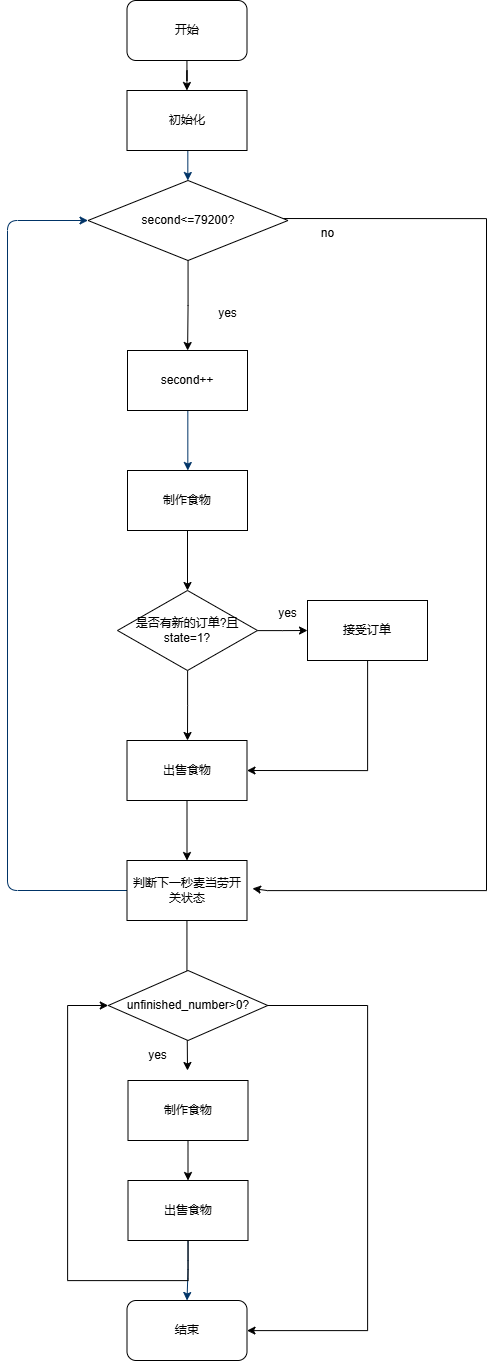
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | struct Food\* making\_foodlist | 制作食物库 | int N, FILE\* fp | struct Food\* |
| 2 | struct Combo\* making\_combo | 制作套餐库 | int M, FILE\* fp | struct Food\* |
| 3 | struct Food\* making\_food\_need | 为点套餐的订单制作食物链表 | struct struct Combo combo，struct Food\* foodlist, int N | struct Food\* |
| 4 | struct Order making\_orderlist | 制作订单库 | struct Combo\* combo, struct Food\* food, int M, int N, int number | struct Order |
| 5 | struct Orderlist\*making\_unfinished\_order | 制作未完成订单 | struct Order order, struct Order\* HeadPtr, struct Order\* LastPtr | struct Order\* |
| 6 | void printing\_time | 报时 | int second | void |
| 7 | struct Orderlist\* \_selling\_food | 出售食物 | struct Orderlist\* order, struct Food\* food, int\* unfinished\_num,int second,int \*finished\_time | struct Order\* |
| 8 | struct Button\* creatButton | 创建按钮 | int x, int y, int w, int h, const char\* str, COLORREF Color | struct Button\* |
| 9 | void drawButton | 显示按钮 | struct Button\* b | void |
| 10 | bool isClickButton | 判断按钮是否点击 | struct Button\* b, ExMessage m | bool |
| 11 | show\_state | 显示订单完成状态 | int\* time, int order\_number,int page | void |
| 12 | show\_food | 显示食物制作状态 | struct Foood\* foodlist, int N | void |

1. 核心算法设计

主函数

输入输出与题目要求一致

伪代码:

****

出售食物

流程图:

算法：出售食物

输入：未完成订单的链表，食物表，未完成订单数目，当下时间，订单完成时间数组

输出：更新后的未完成订单链表（将已经完成的订单从链表中删去）

function \_selling\_food

**while** 遍历未完成订单链表 **do**

**if** 订单点的是食物 **then**

**if** 需要的食物数目＞0 **then**

将该订单从未完成订单链表中删去

**else**

**while** 遍历订单对应套餐中未完成的食物链表 **do**

**if** 该订单点的套餐中的食物数目>0 **then**

将该食物从未完成食物链表中移(除

**if** 套餐对应的未完成食物链表为空 **then**

将该订单从未完成订单链表中删去

**return** 更新后的未完成订单链表头指针

**end** function;