C课程设计需求

项目名 WYZ-BAR

主要逻辑结构包括如下部分：

1. 点单系统
2. 下单系统
3. 进货系统
4. 存货系统
5. 人力资源系统
6. 财务系统
7. 投诉系统（视情况可选）

模块

1.信息模块

需求：打印各种相关信息，包括

1. 酒吧信息，如名称，营业时间等
2. 对于顾客的欢迎信息，酒吧调酒师等的信息
3. 询问点单的信息

接口原型如下：

```
void print_bar_info();
void print_welcome_info();
void print_hr_info();
void order_query_info();
```

内部实现函数不限定，但是接口要按这个原型给出。

2.菜单

需求：单纯的酒吧menu，包括

1. 单品名称
2. 单品价格（严格保证两位小数）
3. 单品配料
4. 可选个性化（冷热 糖度 冰度 额外加料 杯型：中/大/超大）

直接写入磁盘文件再用文件读写（也可直接用printf打印）

```
printf("contains\n"
       "contains2\n"
       "...\n");
/*此种方式实现一个printf多行打印*/
```

如果菜单单品过多或者内容过长，可以设计翻页等。

要设置好打印时**美观的格式。**可以加入多种颜色利用printf()输出。

打印菜单**接口原型**如下：

```
void print_menu();
```

内部实现函数不限定接口形式。

下面进入主题逻辑循环结构，暂时采用REPL（交互式解释器，跟python的交互命令行）。因此每次获取命令输入的时候都需要有提示符

`wyz_bar > :`

输入都在该命令符后，以回车结束。

3.点单模块

需求：获取用户的各种可能**输入**（input）进行**解析**（parse）并作出相应**反应**（response），具体包括：

1. 设计订单类并创建订单，点单部分不用全部初始化改结构体，订单的完全创建由点单部分查询是否能做以后完成。此处设计该结构体即可。

   ```
   typedef struct{
       /*...*/
   }Order;
   ```

   至少需要包括：

   a.所点单品名（最好使用enum等，不必要用完整的名称字符串，可以建立映射关系，需要打印名字时，再去查）

   b.个性化要求（如果该饮品支持）包括：冷热 糖度 冰度 额外加料 杯型：中/大/超大

   c.订单创建时间，时间格式按照`%Y-%m-%d %H:%M:%S`,比如`20XX-03-15 14:55:23`

   d.订单价格(严格保证两位小数)

   e.订单号(全局量或者别的形式，保证每次不一样)

   d.哪位调酒师等（订单处理人员）

设计一系列的宏（或者内联函数），获取相关的信息供后续使用（由于耦合度问题，后续代码如果需要获取订单相关信息，先使用直接）。例如：

```
char* get_order_name(Order* order);
char* get_time(Order* order);
float get_price(Order* order);
/*...*/
```

内部具体实现无要求，函数原型或者是宏，必须是`get_xxxx(Order *order)`或者`#define get_order(order)`

2. 设计用户的输入类

```
typedef struct {
    char *buffer;
    size_t buffer_len;
    size_t input_len;
}InputBuffer;
```

如果有更多的想法，可以完善。默认缓冲区的大小一般为`4K`

获取整行的输入可以用`getline()`等。

3. 设计输入的解析函数

   可以设计解析函数，设计结果的枚举变量

```
Parse_result input_parse(InputBuffer* input);
```

   设计结构体

```
typedef enum{
    /**/
}Parse_result;
```

   核心是对于所有用户的输入，不管是一行输入还是多行输入，又或是各种特殊字符（方向键）都要做到能响应，且程序不会崩溃。

   因为中途用户可以多次点单等，因此要做出人性化的响应过程。

   输入后要有**合适的询问**等，例如

   a.确定一次订单

   b.要什么个性化

   c. 能不能做这个个性化等

   暂时的用户可输入功能有，

   a.点单 I want to order xxxx

   b.结算 I want to have the check

   c.投诉 I want to complain

   等，至于用户输入的方式暂定为用户自由输入，即如同对话一般。

4.下单模块

需求：通过传入的order信息，查询**存货仓库**看能否做这个订单。

函数原型规定为：

```
Query_result query_order(Order *order, List* list)
```

这之后再内部查询这个单品的菜单，一个个查仓库，看是否有存货。返回结果也是一系列的enum值。

查询的接口由**存货系统**（`store_system()`）提供

如果查询成功，则直接进入订单制作（存货仓库中更新），给出客人订单结果

```
void order_done(Order *order)
```

注意人性化设置。

5.存货模块

这个仓库应该是全局的，每次改动需要同步更新 存货系统，财务系统。

利用链表维护这个仓库的所有信息。

一条**存货信息**（ListNode中的value信息）包括：**< 该货物名|货量|进货信息|保质期 >**等

其中进货信息包括 这次进货的 **<进货日期|进货商|处理人|开销>**

链表的具体实现方式，无要求，但是命名上

提供接口:

```
/*查询订单*/
Query_result query_order(Order *order,List* list);
/*打印所有的存货信息*/
void select_all(List *list);
/*插入一条存货信息*/
void insert_recd(store_recd* recd);
/*持久化*/
void
/*更多接口根据需求自定义实现*/
```

order中可能有各种各样的原材料，分别进行查询。

6. 进货模块

所有的进货商信息在一个文件

这个仓库应该是全局的，每次改动需要同步更新 存货系统，财务系统。

需求：传入详细的需求进货 或者 不传（按照默认值进货）

一条进货商包括： **<供应商|货物名|单价 （元/kg，3位小数）|保质期>**

链表实现无要求

但是，进货后所形成的store_recd 结构体除了进货商信息外，还要包括 **<进货日期|处理人|开销>**

提供接口

```
typedef struct {
    /*...*/
}purchase_info;
```

```
void purchase(purchase_info *recd);
```

purchase 内部对于供货商的选择等的具体实现不做要求，但是purchase内部要做到各种信息的更新。可以利用存货模块的接口。

7.财务模块

这个链表（可以不止一个链表）应该是全局的

所有的顾客订单交易收入，进货支出，人员工资支出

可以统计某段时间内的总支出，总收入，毛利润

可以打印财务表

可以作图（折线图，饼状图等）

可以查询某一笔具体的交易，通过订单号

可以查询某个员工手下的交易，通过处理人

8.投诉模块

由顾客提出要投诉以后进入投诉模块，要包括

投诉人

投诉人联系方式

投诉对象

投诉原因

投诉时间

是否解决

后续反馈

维护一个投诉链表（全局），后续该用户再到店里反馈也要能作出反应。一个投诉解决以后，将其从投诉链表中删除，投诉链表可以按照某些性质排序。

9. 结束模块

提供小票

是否投诉

付钱功能（考虑找零等，更新财务模块）

10. 人力资源部分

老板

经理

若干调酒师

仓库管理员

保洁

等需要维护：

工号 工资 入职年份 被投诉记录