CPT109 Assessment 3

Group number: 49

Name	ID
Wenhao Lin (Leader)	2035578
Ruohan Wang	2039257
Muyang Li	2033850
Zhenyu Lu	2038591
ChunZe Liu	2036143

Contents

1 Problem statement

This program need to achieve the following goals:

1. This program need to allow customers to order coffee from the menu such as latte, mochaccino. Every coffee can be customized by: coffee bean type, milk type, hot/ice. Meanwhile, customer can choose extra shot of coffee and number of sugars.

2. Calculate the cost of order.

3. A account system should be provided. The program should support register and edit customer accounts, which can be identified by a unique account number. The account should store name, telephone, and order history, available balance.

4. The program should have some hints to make it easy for anyone to use the program.

5. The program is robust and any random input should not cause the program to crash or read illegal input

6. The program should provide different function for different user. The functions for shop manager are as follow:

   (a) Edit the drinks and options menus.

   (b) Set/alter charges for each drink/option.

   (c) Add/remove/edit customer information.

   (d) Provide statistics related to coffee sales.

   Meanwhile, shop managers should be able to change their initial passwords to prevent them from being cracked because the program's initial password is too simple. Also, administrator accounts can only be registered as new accounts after the administrator has logged in, so the initial administrator account needs to be written in the code

   The function for customer are as follow:

   (a) Login to account and place an order.

   (b) Review order history.

   (c) Deposit money on the account.

2 Analysis

2.1 On an input

There are many kinds of input situation. All situation should be encapsulated as functions to facilitate use and reduce code coupling. All illegal input should be detected and avoided, and the user should be given the opportunity to re-enter.

Because the count of coffee and customer may lager than nine, use number keys to choose the order or customer information is not feasible. So the program will use the up-key and down-key to choose the drink which be wanted.

When user input letter, the user can only input numeric, when other characters are detected, a prompt should be output to ask the user to re-enter. When user need input their name, the username should only have letters and have a maximum length limit. If user input other character or enter characters larger than the maximum length. The program needs to enter the appropriate prompt and ask user to input again.

2.2 On an output

The program should contain multi-level menus for customer to place orders. When user order a coffee, the coffee list included coffee name and its price should be shown. User can choose a coffee from the list. Next, the interface include option will be printed, user can choose one type of option and set the extra shot of coffee and number of sugars. And the program should provide a interface to help customers perform operation such as viewing order history and recharge their accounts. When customer review the order history, the order's name and price should be output.

The additional options for coffee should be divide into five parts: coffee type, mile type, hot/ice, extra shot of coffee and number of sugars. The different types should be output separately to facilitate selection.

The program should output the order information, including the price coffee type and other additional options, prior to checkout.

The interface for the customer is different for the shop manager. The manager can through the multi-level menus to change the user information, when the manager is edit the customer number, each of the user information will be shown on the screen. The manager can move the arrow the select which customer does he want to edit. And the program will print a hint to help manager edit the customer information. For changing the order, the program should output all the coffee like ordering drink, and the manager can select which coffee should be edited. The option change is similarity with coffee change but the option should be output separately. When the customer want to view the sale of the coffee, the

software will output all the coffee's name and the sale information of the coffee.

When the user completes some operations, some hints should be output to tell the user that the operation has been completed. For example, when the customers registers account completely, the program should output `"Register successfully"`, and when the shop manager edit the order price, the program should output `"Edit the order successfully"`.

2.3 Data structure

There are four parts of data need to stored in the .txt: **customer account information, shop manager account information, coffee order information, and option**. They are stored in four separate files. To facilitate the addition of an arbitrary number of elements, these data are stored using a linked list.

1. For the customer account structure, it include username, ID, phone number, balance and history record. And the order history for every user is not constant. To ensure that the program can save memory and store any amount of game history. So for each user, program will create a separate linked list to store game history. The header of this linked list is stored in the structure of the player account information.

2. The coffee structure include coffee name, price and sales. It can also be used to store the option data.

3. The manager account structure only need to store the username of manager and password.

The user's phone number may exceed the maximum number of integer digits, so the program uses a string to store the user's phone number.

When storing one user's information, the username and telephone number are stored first, then stored history information. Because the drink order maybe changed, so the order should be saved as string that can not be influence by the drink order.

There should also be a structure to store the order information, containing the name price of the coffee selected, as well as the name and price of the optional option. This structure can be used to calculate order prices and output order information.

2.4 Algorithm

To make it easier to read and write user account information from .txt files, the program uses `fread()` and `fwrite()` to operate the data. Another advantage of using binary storage for writing is that the data in memory can be perfectly saved and read, which prevents

distortion of data when reading and writing. It also allows perfect compatibility when saving names in different languages.

When generating unique IDs, if program use the current number of users + 1 as the new user ID, then when a user deletes their account, the next user registered will have the same account as the previous user ID (because a user was deleted). Therefore when software add users to the chain, the program will add the new user always to the end of the linked list, and then look at the previous user ID and use the previous user ID + 1 to generate the new user ID instead of simply using the number of users as the ID. To facilitate the addition of 1, the user ID is used as an integer instead of use string.

3 Design

3.1 Naming Rules

The naming of this code will follow the unix_c to make it easier to read. For code comments, the program uses Doxygen to add comments to the code to facilitate the generation of documentation and view function information.

3.2 Structure Declare

3.2.1 Customer structure

```
typedef struct{
        char name[20];
        char ID[11];
        char phone_num[20];
        float balance;
        drink_t history[5];
}user_acconut_t;
```

- `char name[20]` - store customer name.
- `char ID[11]` - store
- `char phone_num` - store customer name.
- `float balance` - store the customer's available balance.
- `drink_t history[5]` - keep a record of last 5 history.

3.2.2 Manager information

```
typedef struct{
        char name[20];
        char password[10];
}manager_account_t;
```

- `char name[20]` - store manager name.
- `char password[10]` - store password.

3.2.3 Coffee structure

The structure for coffee and option are same as follow:

```
typedef struct{
        char name[20];
        float price;
        int sales;
}drink_t;
```

- `char name[20]` - store the name of coffee or option
- `float price` - store the price of object.
- `int sales` - the sales information, option don's use this variables.

3.2.4 Order information

This structure stores information for one user's order and is used to calculate the total cost of the order and output order information.

```
typedef struct{
        char order[20];
        float order_pay;

        char coffee_bean[20];
        float coffee_bean_pay;

        char ice[20];
        float ice_pay;

        char milk[20];
```

```
        float milk_pay;

        int suger;
        int ext_coffee;
}order_info_t;
```

3.3 Linked List

3.3.1 Linked List Declare

Because there will have four types of linked list which save different structure, in order to enable the linked list storing different structures to share the linked list operation functions and reduce the coupling of the code. Therefore, the program does not use the traditional linked list form. It only stores the address of the structure in the linked list, the structure data itself is not stored in the linked list.

The one linked list element is shown as follow:

```
typedef struct{
        struct link_t *prev;
        void *pointer;
        struct link_t *next;
}link_t;
```

- `prev` - store previous linked list element.
- `pointer` - a pointer point the address which needed be store.
- `next` - store next linked list element.

As shown in this structure, this is doubly linked list which used in program. A doubly linked list makes it easier to remove and add elements to the linked list. The linked list structure include the linked list pointer points to the previous element and an another pointer points to next element. And have a pointer of undefined type to an arbitrary structure address.

The first element of linked list do not store the data, it is called the header of linked list. It only store the first element address. The advantage of it is that, the first element be deleted can not change the address of the header. It can avoid the used of secondary pointer and make program easier to read.

```
link_t *temp = account_link;
user_acconut_t *data;
```

```
temp = temp->next;
data = temp->pointer;
```

If program want to operate on the structure stored in the linked list, it can declare a pointer `data` corresponding to the type of the structure and set the pointer equal to the data stored in the linked list. After that, it can manipulate the data pointer to operate the information in structure.

3.3.2 Linked List Operation Functions

The linked list initialization function will generate a linked list header, allocates the space required for a linked list element, and assigns the values of prev and pointer to NULL, returning the value of the newly declared linked list header.

- `int *calc_link_length(link_t *p);` - calculate the number of linked list (Don't include header)
- `link_t *del_link(link_t *head, link_t *deleted);` - delete a linked list element form linked list head.
- `link_t *insert_link(link_t *positon, void *pointer);` - insert new linked list element behind the position
- `link_t *get_number_link_point(link_t *head, int num);` - get the pointer of num-th element in licked list.
- `void save_file_pointer(link_t *p, int struct_length, FILE *account_file);` - save structure data to the FILE pointer.
- `void read_file_pointer(link_t *p, int struct_length, FILE *account_file, int(*insert_link)(link_t *, void*, int));` - read structure data from the FILE pointer.
- `void read_file_name(link_t *p, int struct_length, char *file_name, int(*insert_link)(link_t *, void*, int));` - read structure data from the FILE name.
- `void save_file_name(link_t *p, int struct_length, char *file_name);` - save structure data to the file name.

The program have other linked list operate function which can be used for any linked list. This greatly facilitates the operation of linked list.

3.4 Data structure

3.4.1 Customer data

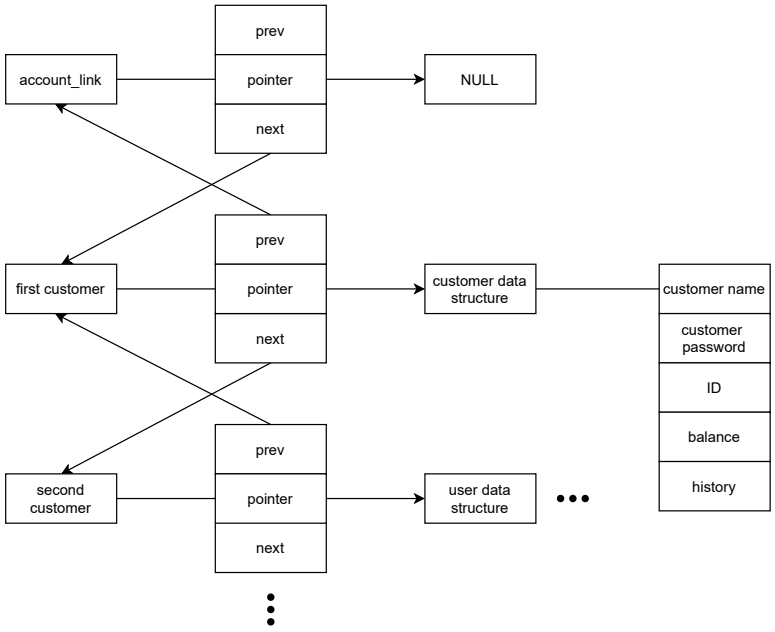

Figure 1: User information data structure diagram

The header of customers linked list is declare as:

```
link_t *account_link;
```

This linked list store all of customer name and its other information. The data structure design diagram is as follows:

First, account_link will store all of customers' information structure's address. The pointer in the account linked list point every user information structure. By manipulating the pointer, program can find the data for each customers.

Second, every user information contain username, and other information. It is type of customer structure which mentioned in Page 7.

By using this data structure, it is realized to store any number of user information using a small amount of scattered memory space, while each user can store any number of game histories. At the same time, the generic linked list structure, which can be applied to both user information and user history storage, achieves a high degree of versatility in terms of code.

The data of the customer is saved in `account.txt` file, it will be read every time the program is initialised.

3.4.2 Manager data

The header of manager linked list is declare as:

```
link_t *clerk_link;
```

The administrator's data stored in the program is same as the customer's data in a linked list, except that the storage structure is changed to manager structure. And the manager information is store in another .txt file.

The administrator data is stored in the same form as the client data, but the difference is that the initial administrator account is written to the program.

```
#define  SUPER_ADMIN_NAME1  "Admin"
#define  SUPER_ADMIN_PASSWORD1  "1234"
```

When the program finds that there is no administrator information file in the path, meaning that it is the first time the program has been run, then the initial account password in the program is used as the administrator account and the program creates a txt file to store the administrator password. Once logged in, the administrator can change the initial password, which will be saved in the txt file for reading on the next run.

3.4.3 Coffee data

The header of coffee linked list is declare as:

```
link_t *drink_link;
```

The linked list for coffee is also similar to the user data mentioned earlier, except that the structure becomes a coffee structure. It will be stored separately in drink.txt.

3.4.4 Additional optional data

The option data is stored in three linked tables:

```
link_t * coffee_type_link;
link_t * milk_type_link;
link_t * ice_type_link;
```

The structure corresponding to each of these three link table pointers is `drink_t`.

To reduce the number of txt files, the data of all three licked list are stored in the same txt file in order.

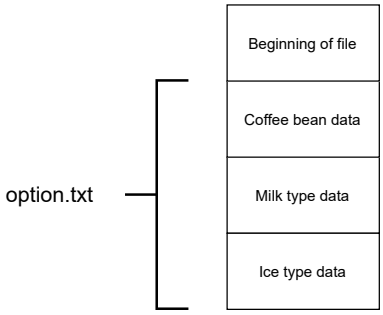

Figure 2: Additional optional data structure diagram

3.5 Input

To make the input more robust and user-friendly, the program have four input function to make input easier.

```
void input_password(char *address, int num);
void get_letter(char *p, int num);
void get_char(char * p,int num);
int input_again(char input[20], char hint[20],
                char num, void(*input_fuc)(char *ch, int num));
void input_inter_num(int *num, int count);
void input_float_num(int *num, int count);
```

3.5.1 Get Character/Letter

In get_char function, a buffer character array of length 9999 element will first be declared to received input. It can prevent can avoid memory receive overflow. Variables will be released from the stack after the function is finished, so this function will not take up much memory. The function sacnf() will stop receive when input '\t' or space. But stopping when meet a space does not help the user enter the correct username they want. This does not match the way the actual software interacts. Meanwhile, the program cannot detect whether the user is entering a space or a Enter. So this function use getc() to get char from input.

Flowchart of the function is shown as Figure 3. After getting the new input, program first determine whether the buffer length is greater than num, and then determine whether it contains spaces. If not, output the corresponding prompt and ask to re-enter.

The function `void get_letter(char *p, int num)` is same as `void get_char(char *p, int num)`. The only different is that this function should determent whether it is a letter instead of whether it is contains spaces.

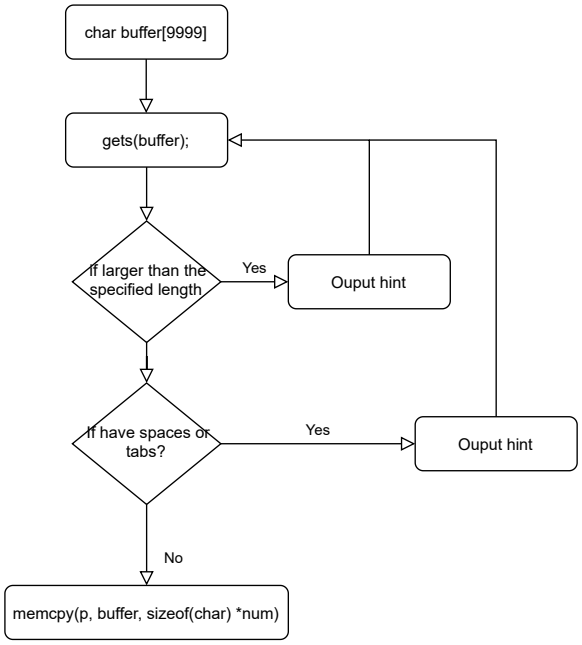

Figure 3: Flowchart of the `void get_char(char * p,int num);`

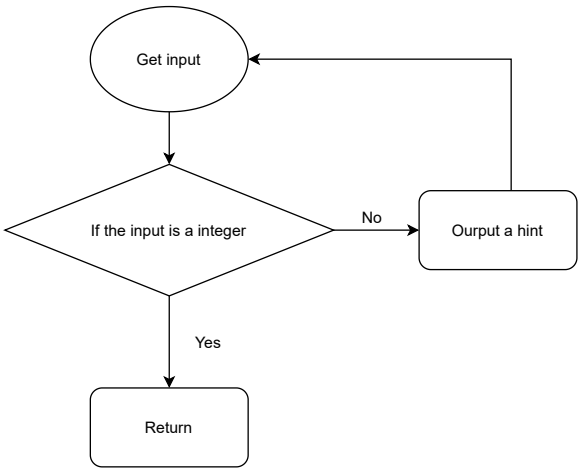

Figure 4: Flowchart of input a integer

3.5.2 Input integer/float

When the program needs to input float or integer, if the user outputs other characters, the program should be asked to re-enter them, and at the same time, in order to avoid the wrong input of the last input not read, the function `fflush(stdin);` should be used to clear the buffer to input the data again. The second parameter of this function is useless and is only intended to correspond to the function parameter interface of `input_again` below, so that this function can be used as an argument to the `input_again`function.

3.5.3 Input Again

Even if the user meets the basic input requirements (length, type), but the user still needed to re-enter. For example, the user is registering a new user name, and when the user name is entered and found to already exist. The program should asked if they want to re-enter, and the user has to enter y or n to choose.

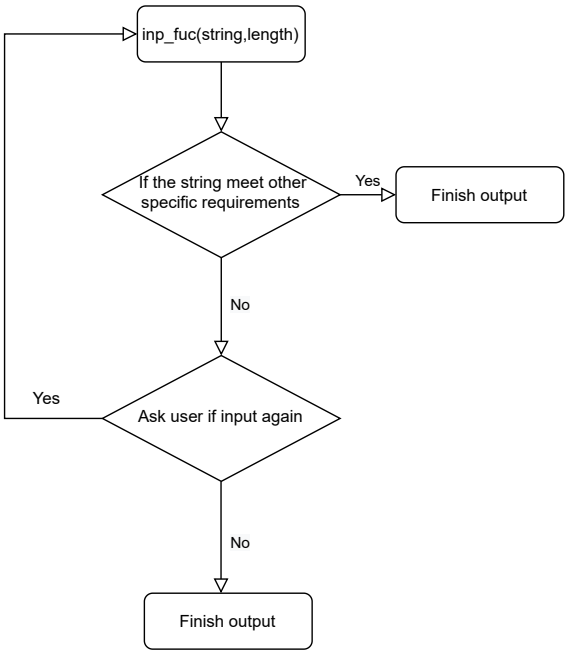

Figure 5: Flowchart of the process of user confirmation and re-entry

Since the y or n judgment is implemented in this way, reducing the number of occurrences of switch cases in the code, can significantly reduce the code length and achieve a formatted way of writing, making the code more beautiful.

```
while(1){
        switch (ch = getch()){
                case 'Y':
                case 13:
                case 'y':
                /*Corresponding operation functions*/
                break;
                case 'n':
                case 'N':
                /*Corresponding operation functions*/
                break;
                default:
                break;
```

```
        }
    }
```

The `input_again` show as Page 13, can be used to implement flowcharts that ask whether to re-input, and use input_fuc input new data. The outer call only needs to know whether the user has chosen to re-enter or exit by the return value. The first parameter of this

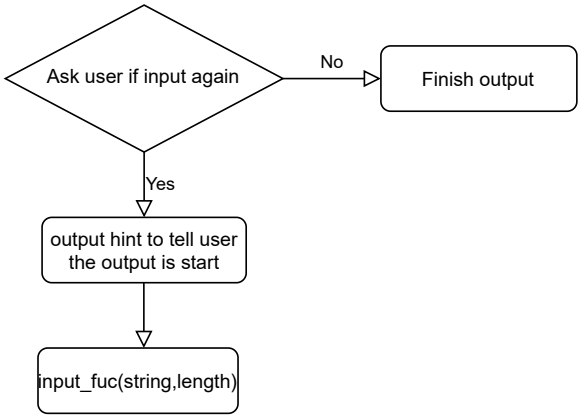

Figure 6: Flowchart for function `int input_again`

function is a variable that stores data, the second data is a prompt to help the user input, and third the fourth parameter is the structure of the receiving function, which can be any of the wrapped input functions mentioned above such as `get_password`.

For example, the judgment of whether a user name exists will be written as this.

```
    while (1){
        temp = search_name(user_account.name);
        if (temp!=NULL){
            break;
            /*Find this name from linked list, break while(1)*/
        }
        /*Do not find the name*/
        printf("User name not found\n");
        if (!input_again(user_account.name,
        "input your user name again:",20, get_char)){
            return;
        }
    }
```

The result is shown as follow.

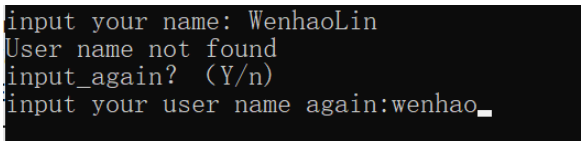

Figure 7: The effect of input again

3.6 Initialization

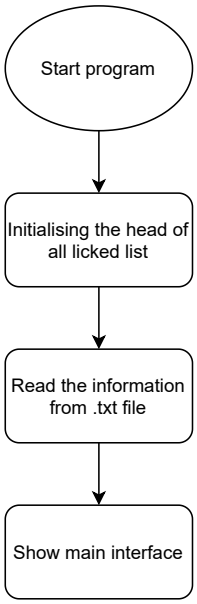

Figure 9: Flowchart for program initialization

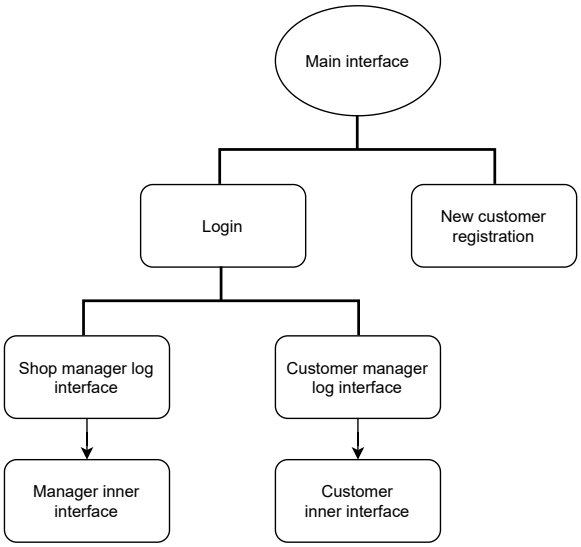

Figure 10: Flowchart for main interface

The program initialization process is shown in the diagram. The program will use the `init_link` function to initialism the linked list header and then read the stored data from the .txt file individually.

3.7 Main menu

The main will have two choices, once the program has been initialized, the main menu will be displayed. The main menu has two options, Login or Register, and after entering the login screen user can select either User Login or Administrator Login.

3.8 Account

The login and registration process for administrators and users is the same, the difference is that the information to determine the presence of the user and input is different, customers need to enter their mobile phone number, but administrators need to enter their password.

3.8.1 Resistor

The flowchart of resistor a new user is shown as Figure 12. The previously mentioned input functions will be used for user name input, so there is no need to type long words or user names with spaces. And the input again will be implemented by the function `input_again`

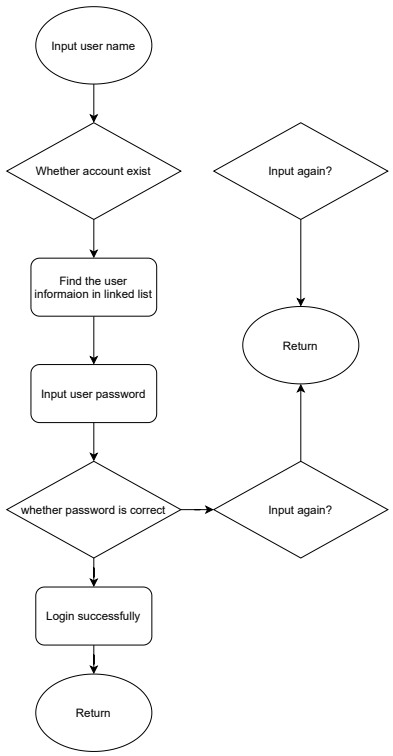

Figure 11: Flowchart for login

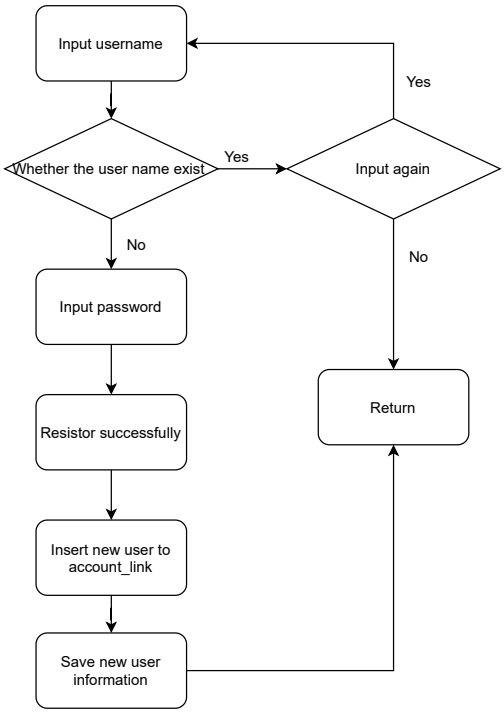

Figure 12: Flowchart for resistor a new user

3.8.2 Login

The flowchart for login is shown as Figure 11. After customer login successfully, the `link_t * purchaser_info` will be update in order to view customer account balance and update their history later

3.9 Inner manager interface

After manager login, users can choose to access these functions as follows.

3.10 Change customer information

In the change customer information interface, the user can add new customer, it is exactly the same process as the previous user registration process.

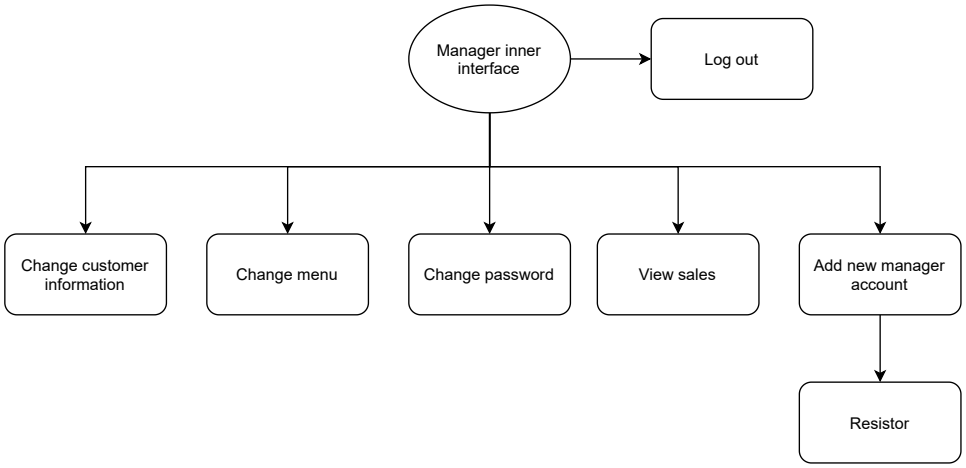

Figure 12: Flowchart for manager login

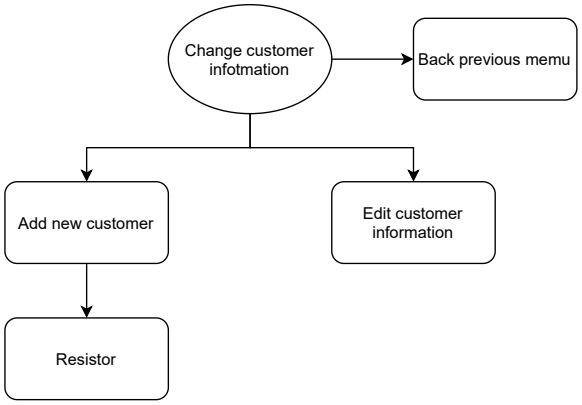

Figure 13: Flowchart for changing customer information

3.10.1 Edit customer information

When entering the edit user interface, all user information will be displayed, including user name, phone number, ID. User can use up-key and down-key the move the cursor to select the customer which want to edit. Press the 'Enter' to choose the number. In every part of this module it can return to the start interface.

This program will provide a search function, in order to avoid too many names making it difficult to find. User can input the user name, and the program will compare it with the data in the list. If find the name in the list, output it and user can continue edit, if not find the name, it will return back to the all user information interface.

3.11 Change menu information

The flowchart for changing menu interface is shown as Figure 15. In this menu, user can choose 4 function: add new coffee drink, edit/delete coffee drink, add new option, edit/delete option. And user can also back to previous interface.

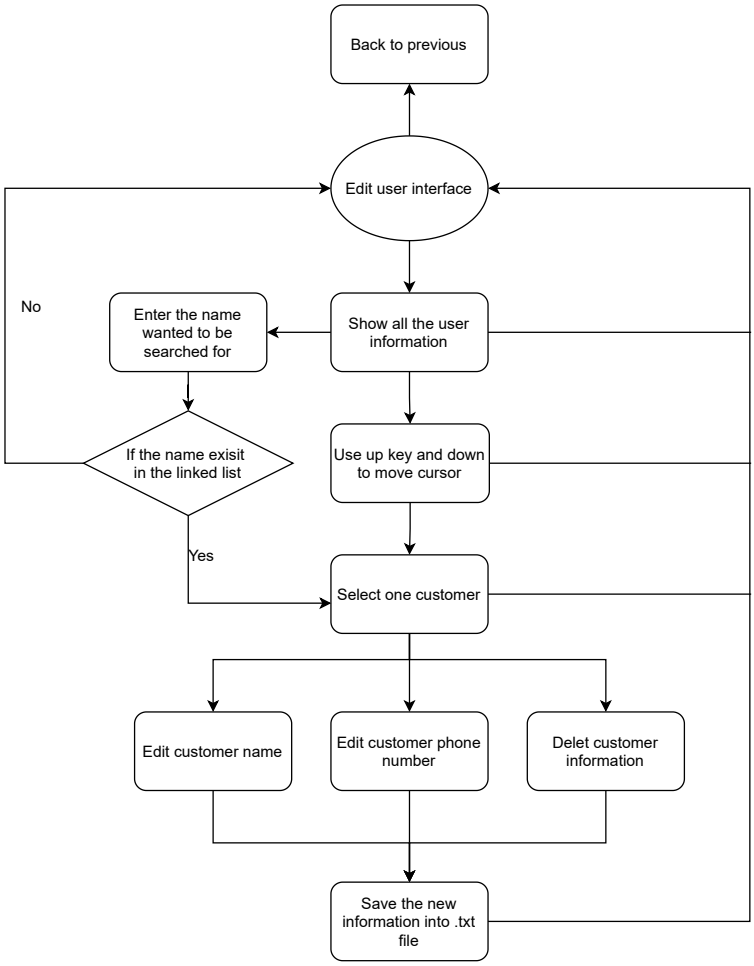

Figure 14: Flowchart for change customer information

The operation between add/edit drink and add/edit option are same. So this section only contain drink operation.

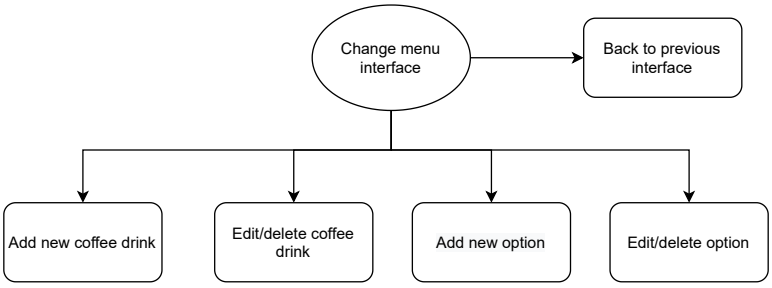

Figure 15: Flowchart for change menu

3.11.1 Add new coffee drink

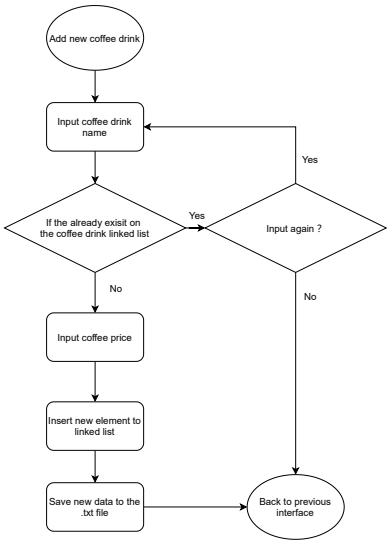

Figure 17: Flowchart for add new drink

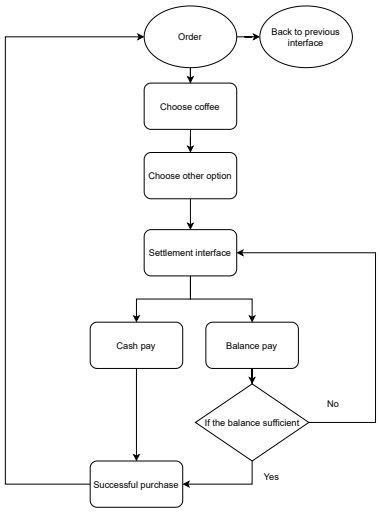

Figure 18: Flowchart for order

The flowchart for add a new coffee drink is shown as Figure17. The program will first ask user input new coffee name, then compare new name with other coffee drink in the coffee drink linked list, if have already has a same name, ask user whether input again, if user choose no, the program will go the the last interface. If have no same name in the linked list, the program will ask user input its price. Nest, initialize the other data in the `drink_t` structure, insert it to the linked list and save new data to the .txt file.

The operation for add new coffee drink and add new option are same, but add new option, the program will find if the option list already have same name.

3.11.2 Edit coffee drink

When edit coffee, there may be too many drinks for the manager to select, so use the arrow keys plus the cursor to select the drink that needs to be changed. After select the drink, user can choose which part of the drink needed to be edited, and input new data.

3.12 Inner customer interface

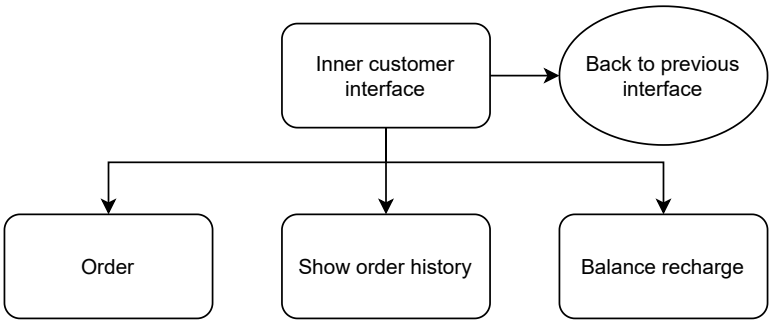

Figure 18: Flowchart for inner customer interface

The inner customer interface flow chart is shown as Figure 3.2.1. After login successfully, the program will display the inner customer interface. Users can order, top up, and view history in this interface.

3.12.1 Order

The flow chart for a user to purchase a coffee is shown as Figure 17(a). The user first makes a coffee selection, again using the arrow key method. Then select the custom item. And if the custom item is not selected, it is the default. Each type of custom item can only be selected once. Once selected, the user enters the checkout interface, which will display the name of the coffee purchased, the custom type selected and the total cost, the user can choose to pay by balance or cash, if the balance is insufficient, the user will return to the checkout interface and select cash to complete the order.

From each part of the module the user can return to the previous menu at any time and change the selected option. The history will be update after successful purchase.

3.12.2 Recharge

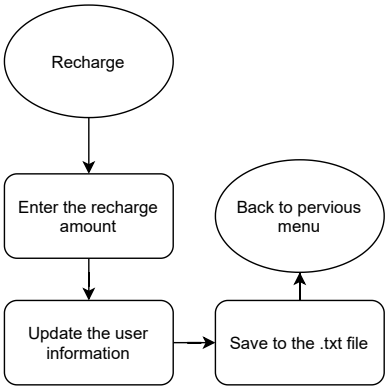

Figure 20: Flowchart for order

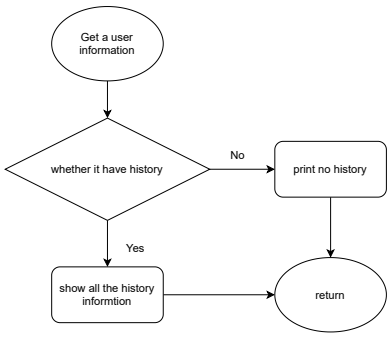

Figure 21: Flowchart for view history

The flow chart for recharge is shown as Figure 20. As the program is not networked and has no control over some of the external settings, it is not possible to top up using some of the online payment methods, but as the program is required to implement the ability for the user to pay cash for coffee, or top up the balance. Therefore, these operations are assumed to be carried out on the basis that a shop assistant is present and the user simply enters the amount of the top-up and pays the cash to the clerk to complete the recharge.

3.12.3 View the history

When the user selects View History, all 5 most recent purchases will be output, or if there are no purchases, then, print `No history at this time`

4 Implementation

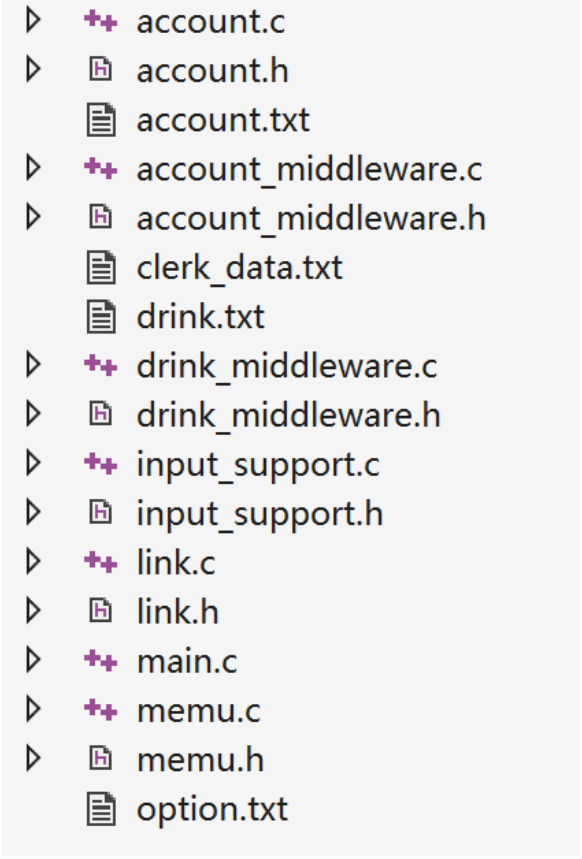

Figure 21: File structure

In order to facilitate the management of functions, different function are written in different files according to their effect.

- **`account.c`** - User account management.

- **account.h** - Define the user structure and declare corresponding function.
- **account.txt** - Store all the user information. (Created automatically after game launch)
- **account_middleware.c** - account information operation function.
- **account_middleware.h** - Define the user structure and declare corresponding function.
- **input_support.c** Four input function mentioned before.
- **input_support.h** - Declare input function.
- **drink_middleware.c** - drink information operation function
- **drink_middleware.h** - Define the drink structure and declare corresponding function.
- **menu.c** - show the order interface.
- **menu.h** - declare the declare corresponding function.
- **link.c** - Linked list operate function.
- **link.h** - Define linked list structure and declare linked list operation.
- **main.h** - Show main interface.

5 Test

5.1 Initial manager account

User name	Password
Admin	1234

Table 1: Initial manger account

5.2 Input Robustness Testing

This figure show that using the input function above instead of using `getchar` can avoid input errors. When the user enters an overly long string or a space, the user will be prompted. Meanwhile, it will output hint to make player use easier. So it is hardly to make this program crash.

Also `input_again()` function plays an important role in this input scenario, when the user enters correctly but does not find the required user name, the program out put a prompt

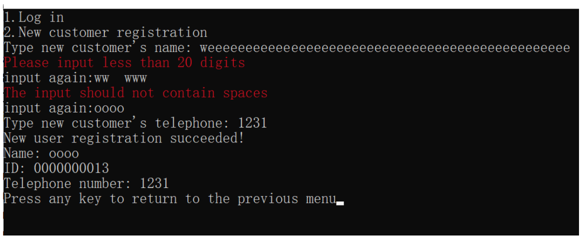

Figure 22: Input user name

and asks the user if they want to re-enter, this makes it easier for the user and ensures that the user does not get stuck in the same interface. When a number needs to be entered, if

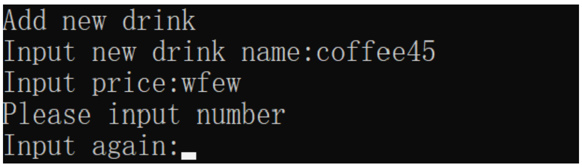

Figure 23: Input of illegal non-numeric inputs

the user enters a non-character number, a prompt will pop up asking the user to re-enter it.

5.2.1 Input float

In order to prevent the user input from being too large, the program does an overflow detection for floating point numbers, which is first received using a string and then converted to float. As shown in the diagram, the user can enter up to 5 digits, if it is greater than

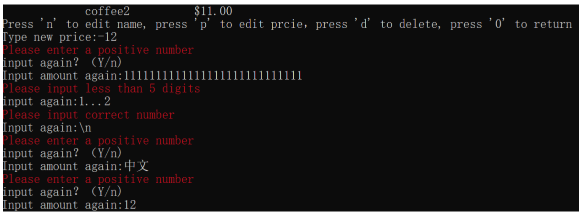

Figure 24: Input Float Test

5 digits then a hint will be entered. Since the previous `get_char` is used to get the input, there is no concern about string overflow. After converting the string to floating point, if it is less than 0, the user is asked to re-enter it.

The program can prompt and report errors for excessively long numbers, negative numbers, non digital input and illegal floating-point input, and require the user to re-enter, but at the same time, the legal input with decimal point symbol can be received normally such as `"+1.1"` is a legal input and `"+1.1.""..1""1..1" is illegal input`.

5.3 Other character username test

Because the user name is directly stored in the file by using the `fread`and`fwrite` function, the program can be perfectly compatible with the user name of other languages and log in with this user name.

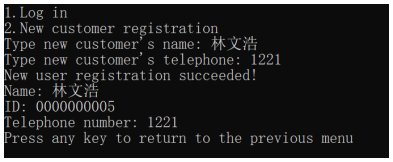

Figure 26: Use Chinese name in registration

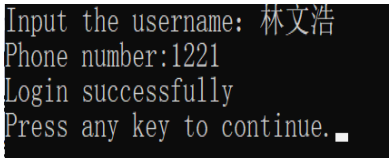

Figure 27: Login with username successfully

5.4 View and change menu

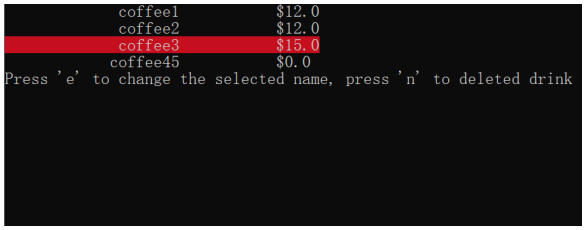

Figure 27: Change menu interface

In this interface, all drinks will be output and a red cursor will help the user to select a drink. The user can use the up-key and down-key to move up and down to select a drink, once you have found the corresponding drink, press d to delete the drink and press e to go to the edit screen to change the price or name of the drink. In this edit drink interface, user

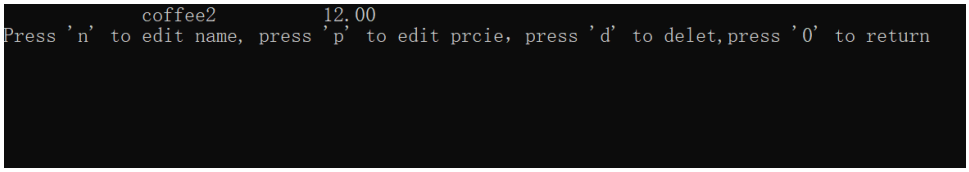

Figure 28: Edit drink interface

can press 'n' to edit name, press 'p' to edit, press 'd' to delete this drink from menu. The same input and output uses a custom function to avoid incorrect input. The input also uses a custom function to avoid incorrect input.

5.5 View and change customer data

View and modify the user information interface and the logic is the same as the menu. The interface is shown is follow:

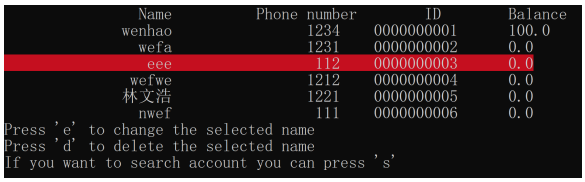

Figure 29: Change user interface

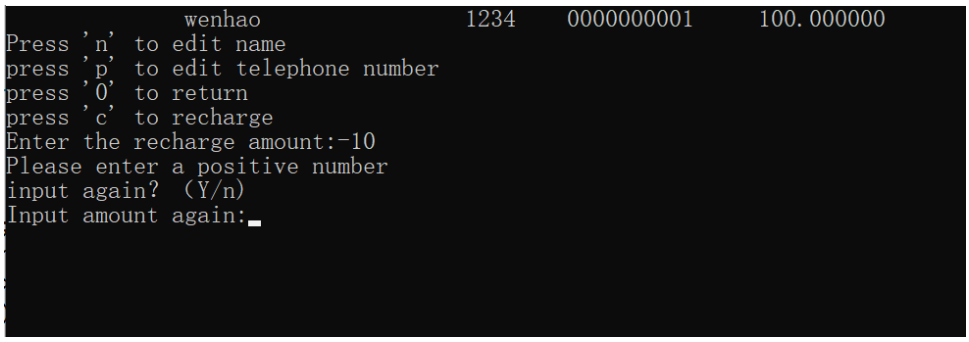

Figure 30: User recharge interface

To avoid the user outputting a negative value, the amount is checked after input and the user is asked to output it again if it is negative. In other cases of this program where a positive number is required, similar processing is still used and will not be demonstrated here.

5.6 View sales

In this interface, users can view the sales of all drinks

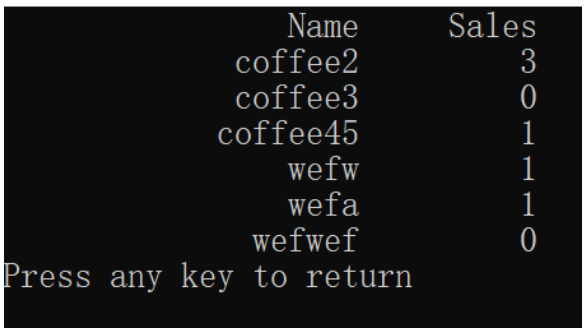

Figure 31: View sales interface

5.6.1 Order

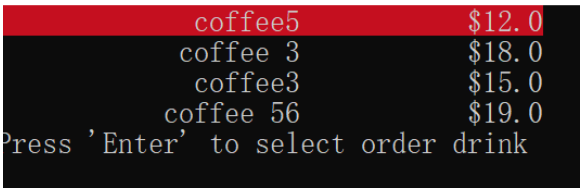

Figure 33: Order drink interface

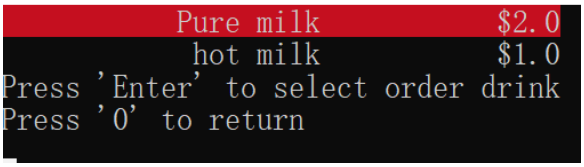

Figure 34: Choose milk interface

In the order ordering interface, the program will output all selectable drinks and allow the user to move the cursor to select.

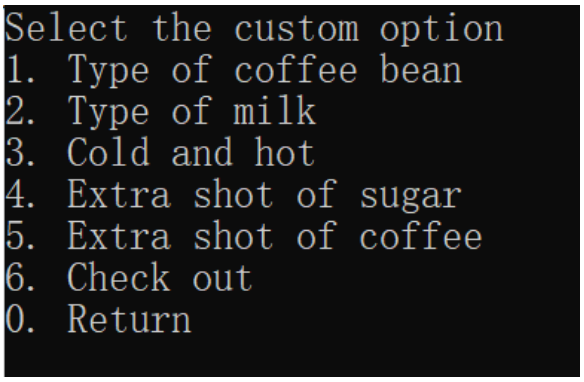

Figure 35: Choose option interface

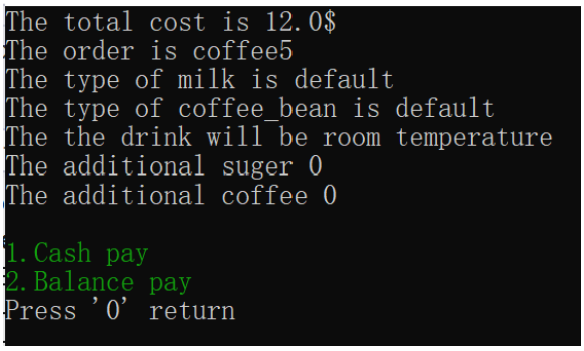

Figure 36: Checkout interface

In the checkout interface, the program will output all option types. If no option is selected, the default will be output. Finally, the total price will be output, and the options of balance payment or cash payment will pop up.

5.7 Account Storage Test

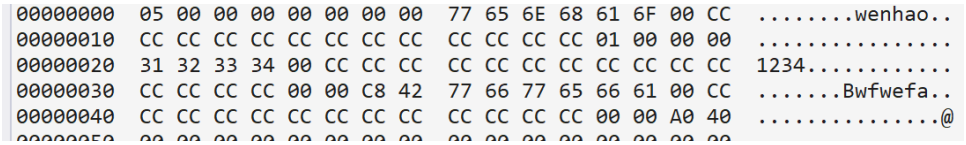

Figure 36: Open `account.txt` with binary

All of the account data will be storage in `account.txt`. In Visual Studio, the uninitialized stack space is filled with 0xCC, and the uninitialized heap space is filled with 0xCD. The 0xCCCC and 0xCDCD correspond to "tang" and "tun" characters respectively in Chinese GBK encoding. If the operating system is not Chinese, other content will be displayed.

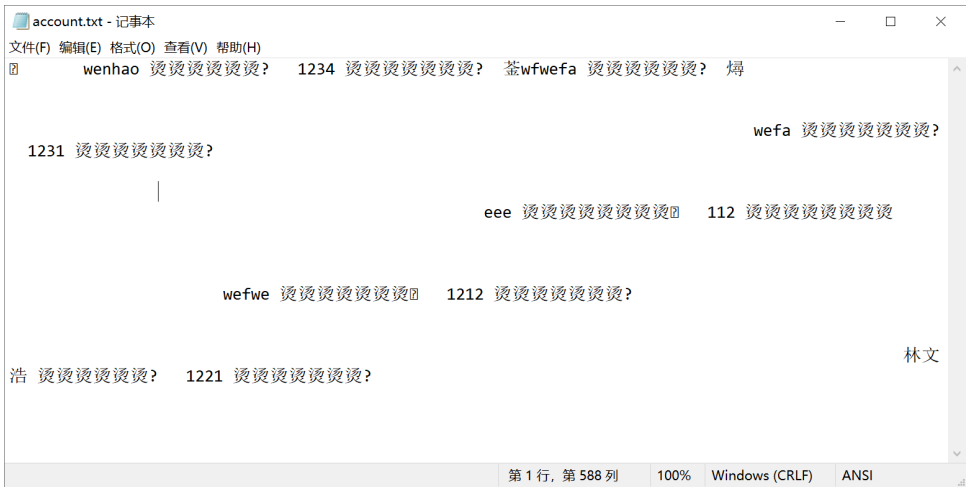

Figure 37: Open `account.txt` with Notepad

Another problem is that the user's account and password are stored explicitly in the txt, which means that viewing the program directory makes it easy to learn other people's accounts and telephone number.

In order to simplify the program, the user's purchase history is not stored using a chain table, but a fixed number of structured arrays are declared, which can lead to many users without 5 histories taking up a lot of unnecessary memory space.

Although this data format can store any number of users, the data update method is full volume, which means that as soon as one user's data changes, then all the information needs to be written to one side, which can take a lot of time if there are too many users.

Number of History	Memory/User (Byte)	User Count	Total Memory (Byte)	Cost Time(s)
5	197	1.00E+07	1.970E+10	212.5

Table 2: Write time and user number relationship

At the 400MB/s write speed of an SSD, when the count of users is too large (greater than 10,000,000), the time of re-write file will cost 212.5 second. At that time the program should consider fixing the maximum number of history records per user and then using incremental updates.