CPT106 C++ Programming and Software Engineering II

# Group Project

**Team leader**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student ID** | **1928064** | **Student Name** | **Yunze.Song** |

**Contribution table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student ID** | **Student Name**  **(Signed manually)** | **Tasks** | **Contribution (%)** |
| 1927996 |  | Mainly responsible for the analysis of consumer class, the realization of functions and the call of related files. In the report, mainly responsible for the writing and modification of the related code. | **25** |
| 1929098 |  | Mainly responsible for the compilation of manager class, the realization of functions and the call of related files. In the report, responsible for the satisfaction and debugging of related codes. | **25** |
| 1927888 |  | Mainly responsible for the implementation of chef class and the call of related functions and files. In the report part, mainly responsible for the implementation of functions related to codes. | **25** |
| 1928064 |  | Mainly responsible for the construction of vegetable,raw material class and main function construction. In the report part, mainly responsible for the analysis and test matching with the code. | **25** |
| Total | | | **100%** |

***Catalogue***

[1. Specifications 3](#_Toc6960)

[1.1 Overall Specifications 3](#_Toc2996)

[1.2 Customer Specifications 5](#_Toc18270)

[1.3 Manager Specifications 6](#_Toc18927)

[1.4 Chef Specifications 7](#_Toc18941)

[1.5 System Specifications 8](#_Toc31038)

[2. Analysis 9](#_Toc18734)

[2.1 On the inputs 11](#_Toc15481)

[2.1.1 Input on the main interface 11](#_Toc1205)

[2.1.2 Input of the chef interface 12](#_Toc9054)

[2.1.3 Input of the customer interface 12](#_Toc2107)

[2.1.4 Manager's input 13](#_Toc25161)

[2.1.4 Input error 14](#_Toc6303)

[2.2 On the outputs 14](#_Toc19657)

[2.2.1 Output on the main interface 15](#_Toc31929)

[2.2.2 Output on the chef interface 15](#_Toc20947)

[2.2.3 Output on the customer interface 16](#_Toc1334)

[2.2.4 Output on the manager interface 16](#_Toc28325)

[2.3 Data Structure 17](#_Toc5253)

[3. Design 17](#_Toc30094)

[3.1 Function in 18](#_Toc12005)

[3.2 Function in 18](#_Toc12349)

[3.3 Function in : 22](#_Toc7180)

[Fig.2. The flow chart for function void deleteDishes() 27](#_Toc24904)

[3.4 Function in 27](#_Toc12047)

[3.5 Main Function 32](#_Toc1786)

[3.6 Other Classes 33](#_Toc11881)

[4.Test 36](#_Toc4540)

[4.1 Test in the main interface 36](#_Toc25938)

[4.2 Test of the chef interface 37](#_Toc4751)

[4.3 Test of the customer interface 38](#_Toc28877)

[4.4 Test of the manager interface 39](#_Toc19046)

[4.5 Test of the input error 40](#_Toc11789)

# Specifications

## Overall Specifications

In this task, we need to do a restaurant management system. The operating system of the restaurant management system has a total of three objects , respectively:

**Chef:** The Chef is responsible for searching, browsing, adding and removing the entire menu.

**Customer:** The Customer can view the entire restaurant menu and order, delete items, etc.

**Manager:** With the highest permissions, the manager can log in to the system as the chef and the consumer respectively. In addition, only the manager can add, view and delete accounts.

Chefs, consumers, managers can use our restaurant management system, which can be more efficient management of the restaurant.

**For Chef:**

The kitchen management of the restaurant is a key management area of the restaurant. First of all, the kitchen has a lot of raw materials to manage, the chef needs to find and replenish in time when the raw materials are not enough. In addition, for a restaurant, irregular change of dishes is very necessary. If consumers often go to the same restaurants and eat the same dishes, consumers may be tired of these dishes, resulting in fewer customers. Therefore, the kitchen needs to change the dishes and modify the pricing in time.

**For customer:**

In reality, when we go to a restaurant, the waiter may hand us the ordering equipment and let us order by ourselves. The restaurant management system which we have designed can do this. The workload of the staff will be reduced by ordering their own dishes, and users can also change the dishes at anytime. If the customer want to delete the dishes which are just ordered, you can directly delete them, which is very convenient to the customers. In addition, consumers can check out in our program, which also saves labor costs.

**For manager:**

Only the manager can create chef and consumer accounts, and the manager can view and delete each account that has been saved in the system.

For the manager, the manager can log in to the chef and customer interface to change the whole system. When the manager views the chef interface, the manager can view a series of functions such as dish pricing and raw material information. In this way, the manager can avoid modifying the menu without authorization by the chef , and also supervise the chef to manage the raw materials. Once the manager finds that the raw materials do not meet the expectations, the manager can inform the chef to replenish the goods in time.

Similarly, managers can log in to the consumer's interface. When we go to a restaurant to have a meal, consumers can choose to delete the dishes if the food is not ready because of insufficient raw materials. However, the process is very troublesome. If the manager can log in the customer's interface, he can delete the dishes instead of the customer, which not only saves the customer's time but also improves the reputation of the restaurant.

## Customer Specifications

**In the user interface, we designed 6 functions, which are respectively:**

1. View the menu: consumers can view all the dishes in the restaurant, including vegetarian dishes and meat dishes. In addition, the user can also find the price of food in this interface.

2. Ordering: Users need to order according to the menu they have just browsed.

3. Cancel ordered food: When the customer does not want to eat the food just ordered,he can choose this function to cancel the food. This function is very necessary in real life. For example, we have noticed that many consumers would like to change the dishes which are ordered before because of the better choice.

4. View the ordered dishes: users can generally browse the dishes they ordered after ordering their own dishes, so as to avoid the occurrence of wrong ordering.

5. Checkout: Users can choose the check-out function after eating the meal and check out directly in the system.

1. Log out: When the customers would like to leaves, they can choose to log out.

**In our design of the consumer system, we need to focus on:**

1. According to the requirements, dishes cannot be added repeatedly. If the consumer chooses the dish just ordered, the system needs to tell the consumer that you have just chosen this dish, so there is no need to add it repeatedly. Therefore, avoiding the occurrence of more dishes will give consumers the best dining experience.

2. In addition, in the system we designed, consumers can clearly see the price of each dish when ordering, which makes it more convenient for them to order.

3.We also design the check out function. If the user chooses this function, the entire consumption bill will appear in the user's interface, and there is no need to ask the waiter to check out.

## Manager Specifications

Managers have the highest privileges which means that they can choose to log in to the system as chefs and consumers. At the same time,they can add or delete accounts and other operations. The specific functions are as follows:

1. Add accounts: Managers can add accounts at anytime. The accounts added are mainly divided into two categories: chefs and consumers.

2. View accounts: As with adding accounts, managers can view two accounts with different identities.

3. Delete the account: If the chef leaves, the manager needs to delete the chef's account, which is more convenient for the management of the whole restaurant.

4. Log in as a chef: The manager can log in as a chef and view, modify the information of the entire chef interface.

1. Log in as a consumer: The manager can log in as a consumer, and can view and modify the entire interface information. For example, if it is found that the food ordered by the customer cannot be produced normally, the manager can directly log in the interface of the consumer and delete the dishes.
2. Delete the account: The manager have the ability to delete the unnecessary account according to the real conditions.

## Chef Specifications

**In the design interface of the chef, we designed the following functions:**

1. Search for raw materials: Chefs can search for the amount of raw materials left in the kitchen

2. Browse all raw material information: the chef can browse the raw materials left in the kitchen. For the insufficient raw materials, the chef needs to replenish them in time.

3. Add dishes: Chefs can add different dishes to the menu. If the chef plans to design new dishes, the chef can add dishes to the menu.

4. Delete dishes: the chef can modify the menu in time according to the number of raw materials left in the kitchen, so as to avoid the situation that the dishes can not be done because of the lack of raw materials.

5. Browse all dishes: The chef needs to browse all dishes in the interface, so that the dishes will be modified in time according to the situation.

6. Change the inventory of raw materials: When the raw materials are replenished, the chef needs to change the inventory of raw materials in time which means that we need to change NO to YES

When the chef logs in, he can see the total number of dishes on the restaurant menu and the raw materials, which makes it easy for the chef to know the whole kitchen.

## System Specifications

For the whole system, we designed a total of three classes, respectively: chef, consumer and manager. Among them, the manager has the highest authority to directly log in to the other two interfaces. In the program, we have also designed a total of 8 head files, which play different functions respectively. Therefore, the 8 head files are:

1.Chef.h: The Chef's head file. We need to use the contents of this file in the main function.

2.Customer.h: The head file of the consumer. When we need to write the program on the consumer, we need to use the contents of this header file in the main function,too.

3.Manager. h: In this header file, we mainly write the Manager header file

1. Dishes.h: The header file of the dish. We define the ID, Name, price of the dishes and raw material here

5. Expenserecord.h: Here, we define ExpenseRecord class

6.GlobalFile.h. In this head file, we have the macro definition

7. Identity.h: In this head file, we define the Identity class and set up virtual functions

8.RawMaterial.h: In this header file, we define Class RawMaterial

For the main program, in addition to the main function, we also define three .cpp files including:

1.Chef. cpp: Here, we have designed the interface of the entire Chef and the required functions.

2. In this program, we have designed the entire consumer interface and its required functions.

3.In this program, we designed the entire manager interface and its required functions

In the **main** program, we mainly designed the login interface which is used to let the user choose the role and function of input.

# Analysis

In the program we designed, there are 4 functions in the main interface for users to choose:

1. Logout: Log out of the system

1.Manager: If the user selects manager, the user needs to enter option 1 and enter the correct user name and password

2.Chef: If the user chooses to log in as a Chef, the user needs to select 2 and enter the correct user name and password

3. Consumer: If the user chooses to log in as a consumer, the user needs to select 3 and enter the correct user name and password

In the main interface, when the user enters his/her identity and the correct user name and password, he/she will log in to the corresponding interface to perform the next step.

**In the consumer interface, the user can select the following functions**

0. Log out

1. Check the menu

2.Order the dishes

3. Cancel the dishes

4. Check the dishes which are ordered before

5. Check out

**In the Manager interface, the user can select the following functions:**

1. Log out

1. Add an account

2. View the accounts

3. Log in as a chef

4. Log in as a customer

5.Delete the account

**In the Chef interface, the user has the following options:**

1. Log out

1. Search for raw material information

2. Browse all raw material information

3. Add the dishes

4. Delete dishes

5. Browse all dishes

6. Change raw material inventory

## On the inputs

### Input on the main interface

In the main interface, users need to choose the functions they want to choose according to the prompts. In this interface, the total of user input :

1. Identity selection: including the identity of manager, consumer and chef. The user needs to choose among them

2. The account: After creating the accounts by the manager, the user can log in the system

3. Password: For different accounts, users need to enter the correct password.

In addition, if the user chooses to log out of the system, he can select 0

### Input of the chef interface

In the chef interface, the user can choose six functions and the program will perform operations according to the user's input.

**When the user enters 0**, the program will log out and returns to the main interface for the user to make a new selection

**When the user enters 1**, the program will search for raw material information. If the user does not enter the correct raw material name, the system will remind that the input error.

**When the user enters 2**, the program will browse through all the raw material information

**When the user enters 3**, the program will conduct the operation of adding dishes. Once the chef thinks that the type of dishes does not meet the requirements, the chef can choose this function to add dishes.

**When the user enters 4**, the program will delete the dishes. If the chef judges that the raw materials of the dishes are not enough, the chef can select this function to delete the existing dishes

**When the user enters 5**, the program will browse all the dishes

### Input of the customer interface

In the input interface of the consumer, the user can choose a total of six functions, and the program will execute the operation according to the user's choice

**When the user enters 0**, log out

**When the user enters 1**, view the current restaurant menu

**When the user enters 2**, the customers choose to order the dishes

**When the user enters 3**, the consumer needs to delete the dishes

**When the user enters 4**, the consumer will finally confirm that the food has been ordered

**When the user enters 5**, the consumer will check out

### 2.1.4 Manager's input

In the manager interface, we have designed a total of 5 functions for users to input, users can choose according to their own needs:

**When the user enters 0**, the manager chooses to log out and return to the main interface

**When the user enters 1**, the manager chooses to add a new account, where the user also needs to select the identity of the added account

**When the user enters 2**, the current account will be viewed. Similarly, the user also needs to select the identity

**When the user enters 3**, the manager will see information about the chef

**When the user enters 4**, the manager will view the consumer's information

### Input error

Although we have put a prompt in front of each user's input, the user may still not type as we requested, which can lead to typing errors. Examples of typing errors include:

1.In the main interface, the user may enter numbers not between 0 and 3, or enter letters or special characters

2. In the interface of the chef, the user may enter numbers that are not between 0 and 5, or enter letters or special characters

3. In the consumer interface, the user may enter numbers that are not between 0 and 5 or letters or special characters

4. In the manager's interface, the user may have entered numbers between 0 and 5 or letters or special characters

5. When users order dishes, they may input previously entered dishes. In this case, the program will tell the user input error.

## On the outputs

Similarly, in the output part of the program, we are also divided into three interfaces to introduce one by one:

### Output on the main interface

First of all, when the user chooses the corresponding identity, the program will prompt the user to enter the user’s account. When the user has entered their account, the system will prompt the user to enter a password.

If the password is entered correctly, the program will execute and enter the interface of the manager, chef and consumer.

If the password is entered incorrectly, the program will prompt the user that he enters a mismatch between the username and password and prompt the user to reenter the correct information.

### Output on the chef interface

**In the output interface of the chef, we need to output according to the user's choice**

**If the user selects 0**, log out and return to the main interface

**If the user selects 1**, the program will print the raw material information entered by the user to the screen

**If the user selects 2**, the program will present all the raw materials in the kitchen to the chef

**If the user selects 3**, the program will add new dishes to the menu as requested by the chef

**If the user selects 4**, the program will delete the corresponding dishes according to the requirements of the chef

**If the user selects 5**, the program presents all the dishes to the chef

### Output on the customer interface

**In the consumer output interface, we need to output according to the user's choice.**

If the user selects 0, log out and return to the main interface

If the user selects 1, the program presents the menu to the user

If the user selects 2, the program will record all the dishes the customer has ordered, and if the customer orders the same item, the program will remind the user that he has just ordered the dish

If the user selects 3, the program will cancel the corresponding dishes according to the requirements of the consumer

If the user selects 4, the program will follow the customer's request, browse all the dishes that have been ordered and print the dishes to the screen

If the user selects 5, the program will add up the prices of all the dishes and print the price on the screen so that the user can pay for the meal.

### Output on the manager interface

In the output interface of the manager, we need to output according to the user's choice:

If the user selects 0, log out and return to the main interface

If the user selects 1, the program will record the manager's newly added account information into the program

If the user selects 2, the program will print all the accounts to the screen for easy browsing

If the user selects 3, the program will follow the manager's instructions and jump to the chef's interface

If the user selects 4, the program will follow the manager's instructions and jump to the consumer's interface

## Data Structure

In the program we designed, there are three classes: Chef, Manager, and Customer. Therefore, the Class diagram of the classes are as follows:

# Design

Since in general, as we mentioned above, this task should create corresponding functions for three specific groups: manager, chef and customers in this restaurant, we should create the head file and cpp files correspondingly. In this task, we totally created four classes: , , , and . We can also see from the class diagram above that the relationship between the three classes is that , are the subclasses of .

However, this is general relationship, there are lots of detailed code that needed to be designed in detail. Therefore, the following section will discuss and analyse the four class functions in detail. Flow charts will be given when it comes to the complicated functions.

## 3.1 Function in

***virtual void openMenu()***

Declared as pure virtual function. It has only declaration but no declaration. The specific object can not be created. Since this class contains the virtual function, this class is a abstract class, which can only be used to inherited, but not to construct objects. The detailed definition of these pure virtual function will be given in subclasses.

We also have two public variables:

***string name***

The variable is used to store the name of the identity account.

***string password***

The variable is used to store the password of the identity account.

## 3.2 Function in

At the beginning of this class, we need to declare that the class is the subclass of class identity so that the objects are inherited. The inherit type should be public.

***Manager();***

The default constructor used to make the declaration of a class type variable valid before give specific properties to this variable.

***Manager(string tempName, string tempPassword)***

The constructor used to initialize the item manager.

***virtual void openMenu()***

This is a function used to create the interface of manager. The virtual function is used in order to achieve polymorphic. We have to rewrite the virtual function in the and other two classes in order to achieve this, which means the return value and the passed in parameters should be exactly the same. So that the parent class can access to the interface of the subclass directly.

***void addPerson()***

This function is used to add new users to chef and customer.

***void deletePerson();***

Used to delete specific person’s account.

***void searchPerson()***

This function is used to show the information of all the chefs and customers including their name and password. If the user entered is wrong, the program return to the interface.

***void initVector()***

This function is used to initialize the two vectors used in this program. Read the content in the corresponding file that store the information and store them in the vector.

***bool checkRepeat(string tempName, int userType)***

This function is used to check whether the information of the person the manager wants to add is already in the file. In order to avoid repetition.

There are also two public type vectors used in the class:

***vector<Customer>vCustomer***

This object is used to store all the information of customers.

***vector<Chef>vChef***

This object is used to store all the information of chefs.

Since the manager has the top authority, this class should communicate with two files. Thus, the main difficulty in this task is focused on the addition of the new users and check the repetition of the input user.

Before all these operations, in this program, we have to initialize the two vectors by using ***void initVector()*** to ensure that the data stored in the file is read and stored and available in this program.

***void initVector()***

In the constructor that used to initialize the object, this function ***void initVector()*** is called. Firstly, clear the two vectors to make sure that initially the two vectors are empty. Then create a data stream used to read the data stored in the file. Then open the file that used to store the information of dishes, in the mode of ios::in to read in data from file. Then use push back to store all the information in the vector ***vector<Customer>vCustomer*** Each element of the vector is an object used to store the name and the password of the customer.

Then the function will initialize the chefs’ information by using the same method. So that the second vector ***vector<Chef>vChef*** will contain the information of chefs including name, and password.

Whenever the constructor is called to create an object of type chef with user name and password in it, this function is called so that the two vectors are initialized. Then all the other function that may use the information stored in the vectors can function properly.

Then it is easy to construct the function:

***bool checkRepeat(string tempName, int userType)***

Since the information is stored in the vector, we simply need to go through all the elements in the corresponding vector to check whether the element with same name and password exist. For example, if the manager wants to add a chef, after the manager enter the name and password of the user, we can call this function by passing the corresponding user type in this function.

Then the function will go through the corresponding vector according to the passed in user type to find the same user name. if the user is found, return false. If the user is new, return true.

***void addPerson()***

Firstly, ask the manager to choose which kind of user needed to be added. Then specify the filename according to the chosen profession in order to open the corresponding file that store the correct information. Open the file as type **ios::out | ios::app** so that the new users are added to the end of the file.

Then ask the manager to enter the user name that wants to be added. Then call the function

***bool checkRepeat(string tempName, int userType)*** to check whether the user name is already existed, if the user exists, ask the manager to reenter a new user. If the user is new, ask the manager to enter the password. Then write the new user’s information into file and then call the ***void initVector()*** to update the content in the vector in case that the manager wants to have further operation to this new vector.

The flow chart of the function is as follows:

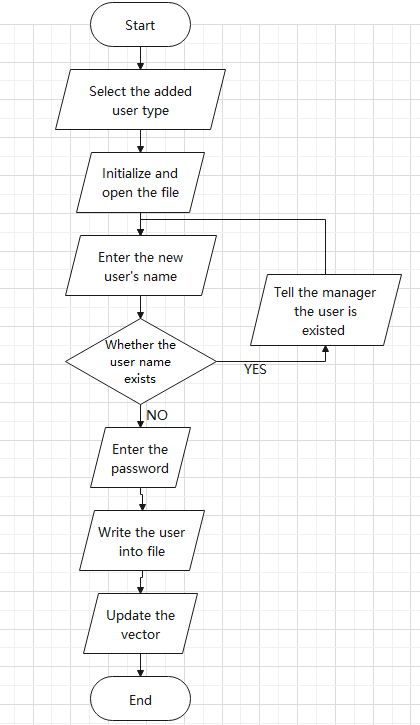


Fig.3. The flow chart of function void addPerson() in class manager

## 3.3 Function in :

***Chef()***

The default constructor used to make the declaration of a class type variable valid before give specific properties to this variable.

***Chef(string tempName, string tempPassword)***

The constructor used to initialize the item chef.

***virtual void openMenu()***

This is a function used to create the interface of chef. The virtual function is used in order to achieve polymorphic. We have to rewrite the virtual function in the and other two classes in order to achieve this, which means the return value and the passed in parameters should be exactly the same. So that the parent class can access to the interface of the subclass directly.

***void searchRawMaterial()***

This is a function used to achieve the function of searching the specific row material and judge whether the row material still has stock.

***void changeRawMaterial()***

Used to change the information of row materials including the stock. If the row material does not exist, the program will tell the user.

***void showRawMaterial()***

This is the function that used to show all the row materials in the stock.

***void addDishes()***

This is the function used to add dishes to the dish menu.

***void deleteDishes()***

This is the function used to delete dishes from the menu.

***void showDishes()***

This is the function used to show all the dishes that is already existed in the menu.

***void initVector()***

This is the function used to initialize the two vectors used in this class: the vector used to store the information for dishes and the vector used to store information for row materials.

***bool checkRepeat(int tid, string tnam)***

This is the function used when the chef wants to add dishes to the dish menu. This function can check whether the dishes the chef wants to add is already existed in the dish menu.

There are also two public type vectors used in the class: ***vector<Dishes> vDishes and vector<RawMaterial> vRawMaterial***. They are used to store the information of all dishes and row materials that is already existed in the file. They are also used when add and delete dishes.

There are numerous important parts in this class. The main difficulty concentrates on add and delete dishes. Since the operation of add and delete dishes should not only change the element stored in the vector that used to store all the dish information, but also update the new data to file.

However, before introduce the process of add and delete dishes, I will firstly introduce two functions that will be called in these two functions and have crucial functions, which are function ***void initVector()*** and function ***bool checkRepeat(int tid, string tnam).***

***void initVector()***

In the constructor that used to initialize the object, this function ***void initVector()*** is called. Firstly, clear the two vectors to make sure that initially the two vectors are empty. Then create a data stream used to read the data stored in the file. Then open the file that used to store the information of dishes, in the mode of ios::in to read in data from file. Then use push back to store all the information in the vector ***vector<Dishes> vDishes***. Each element of the vector is an object of type class Dishes, which store the dish ID, dish name dish price and row material price. After initialize the vector, close the file.

Then the function will initialize the row material information by using the same method. So that the second vector ***vector<RawMaterial> vRawMaterial***. will contain the information of row materials including name, and stock of the material.

Whenever the constructor is called to create an object of type chef with user name and password in it, this function is called so that the two vectors are initialized. Then all the other function that may use the information stored in the vectors can function properly.

***bool checkRepeat(int tid, string tnam)***

Since the two vectors are already initialized in the constructor. In this function we can directly use the information stored in them. Firstly, the ID and the name of the dish that want to be check should be passed in. Then, we used “.begin” operation to bring the built in pointer in the vector to the beginning of the vector to start to search through all the elements till the end of the vector. If the element with the same ID and name is found, then return true, if the element does not exist, return false.

Now we can explain the most complicated functions.

***Void Chef::addDishes()***

Firstly open the file in the mode of ios::out | ios::app used to store data at the end of the file. Then the program will ask the user to enter the ID and the name of the added dish. After the information is entered, the function bool checkRepeat(int tid, string tnam). will be called in order to make sure that the dish added is not exist in the menu. If the dish is already existed, program terminates and tell the user that the dish is already existed and ask the user to enter the name and ID again. If the dish does not exist, ask the user to enter the sell price of the dish and the prime cost of the dish. Then write the data into file and update the vector again to store the new element in the vector.

The flowchart of the function is as follows:

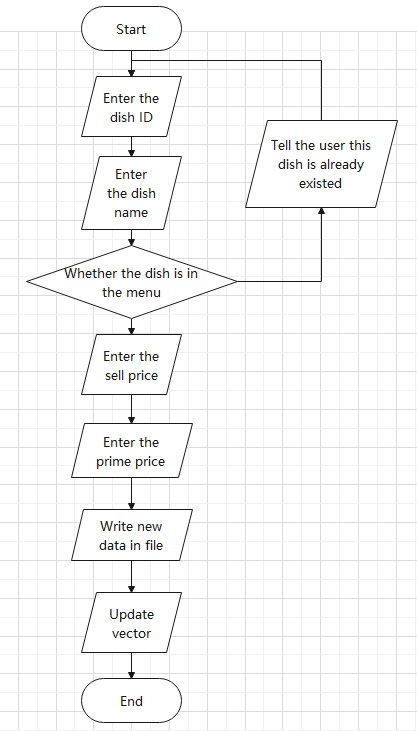


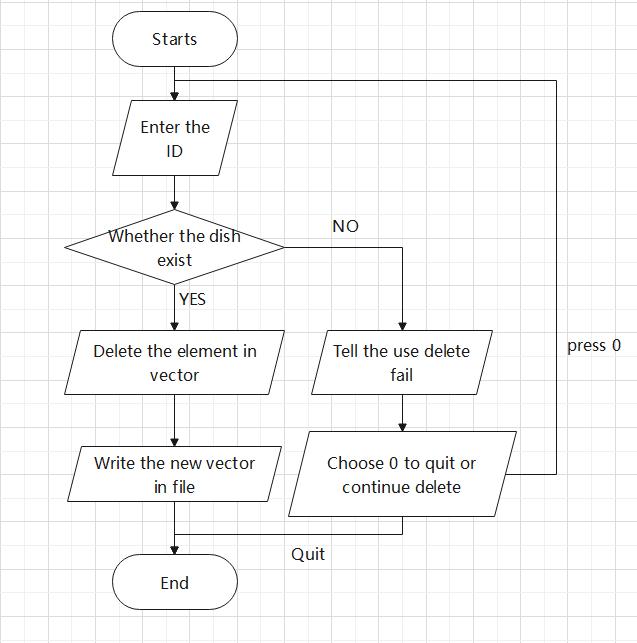
Fig.1. The flow chart of function bool checkRepeat(int tid, string tnam).

***Void Chef::deleteDishes()***

First create a bool type variable in order to judge if the delete is success and a int type variable select in order to judge whether the user want to delete more dishes or quit the program. Then the program will ask the user to enter the ID of the dish. Look through the whole vector as we mentioned above. If the dish is found, set flag variable to false. Delete the element in vector by using “.erase” operation, the current element will be delete and the element after this deleted element will automatically move one element ahead to fill the blank. This is the convenience of using vectors. If the flag is still true, that means that the dish does not exist in the menu, so the user can choose to continue delete or quit.

After the delete operation is success open the file in mode ios::out | ios::trunc so that the initial file is delete and a new file with the same name is created. Then write the information stored in the new vector in the file.

The flow chart of the function is as follows:



# Fig.2. The flow chart for function void deleteDishes()

## 3.4 Function in

At the beginning of this class, we need to declare that the class is the subclass of class identity so that the objects are inherited. The inherit type should be public.

***Customer();***

The default constructor used to make the declaration of a class type variable valid before give specific properties to this variable.

***Customer(string tempName, string tempPassword)***

The constructor used to initialize the item chef.

***virtual void openMenu()***

This is a function used to create the interface of customer. The virtual function is used in order to achieve polymorphic. We have to rewrite the virtual function in the and other two classes in order to achieve this, which means the return value and the passed in parameters should be exactly the same. So that the parent class can access to the interface of the subclass directly.

***void showAllDishes()***

This function is used to show all the dishes in the menu.

***void orderDishes()***

This function is used for the customers to order dishes.

***void cancelOrder()***

If the user wants to cancel the dishes that are already ordered, this function is called.

***void showCurrentOrder()***

This function is used to show what dish the customer has ordered.

***void checkOut()***

This is used for the user to check out and pay the money.

***bool checkRepeat(int tempID)***

This function is used for the customer to check whether the dish is already ordered, since the same dish should not be ordered twice.

***void updateOrder()***

This function is used to update the ordered dishes.

We also have two vectors used to store data.

***vector<Dishes> vDishes***

This object is used to store the dishes that is already ordered.

***vector<ExpenseRecord> vExpenseRecord***

This object is used to store the expense record of the customer.

***void initVector()***

This function is used to initialize the vector used to store the dishes.

In this class the method of initializing the vector is basically the same with the previous classes. So here we do not discuss that. For this class, the difficulty mainly concentrates on the order and cancel order, checkout. So, these two functions will be explained detailly.

Down here, we suppose the vector in the program is already initialized.

***void orderDishes()***

First, declare the int type variable in order to contain the ID of the ordered dish. Then create a bool type variable flag in order to judge whether the order is successful.

Then ask the customer to enter the ID number of the dish. If the ID is found, store the ID, name and the price of the dish into a temporary class type variable created called current order. Then check whether the dish is already ordered. If the dish is already ordered, tell the user to enter another dish. If the dish is new, push back the element in the ***vector<ExpenseRecord> vExpenseRecord*** to store the dish ordered, then call ***void updateOrder()*** to update the ordered dish information. If the order is not found by the program at the beginning, tell the user the dish does not exist and ask them to choose keep ordering or quit the program back to interface.

The flow chart of the program is as follows:

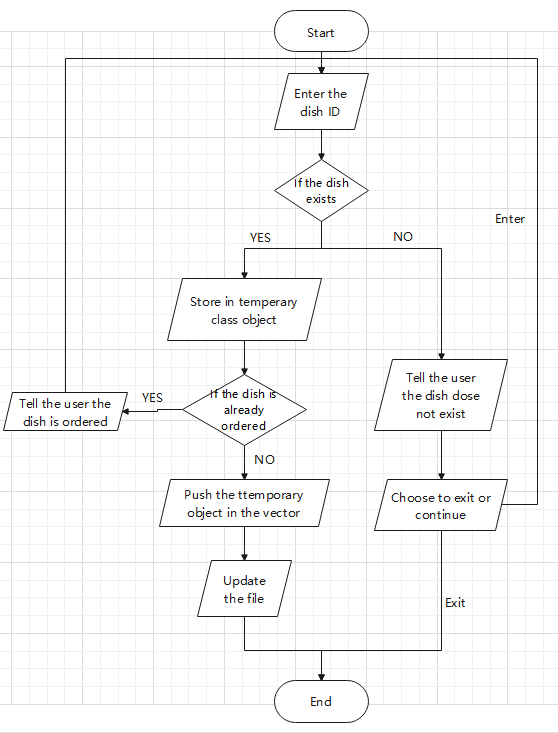


Fig.4. The flow chart for function void orderDishes()

***void cancelOrder()***

Firstly, the program ask the user to enter the ID of the dish want to be deleted. Then go through the ***vector vExpenseRecord*** to search for the dish. If the dish is found, erase the element. If the dish dose not exist, tell the user that the dish is not ordered and give them a choice to return to the interface or continue to delete the ordered dishes. After the dish is deleted, open the file used to store the ordered dishes in mode ***ios::out | ios::trunc*** so that the old file is deleted and the new file with the same name is created. Then, write all the record into the file. Since the vector is updated automatically as we mentioned, both the file and the vector is updated now.

The flow chart of the program is as follows:

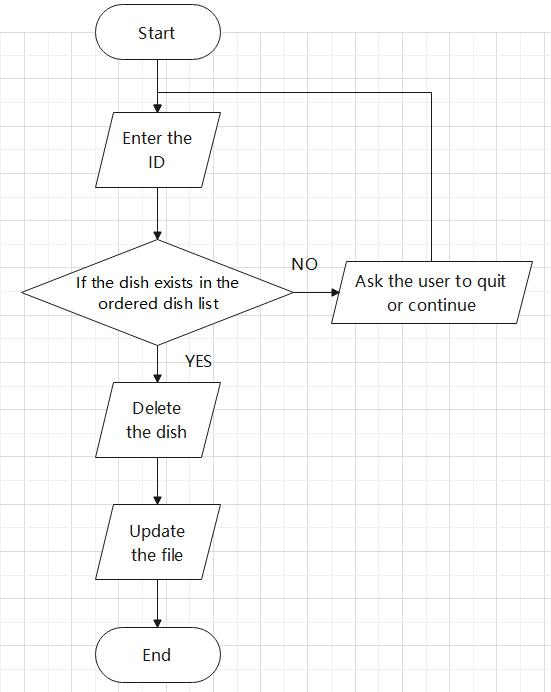


Fig.5. The flow chart for function void cancelOrder()

***void checkOut()***

When check out, open the file used to store the expense record. Read through the file to get all the dishes ordered and get the corresponding price. Add the price together to get the total cost. This is not that complicated. So the floe chart is skipped.

## 3.5 Main Function

In main function, there are four functions:

***void LoginIn(string fileName, int type)***

This function is used to switch the inter face according to the user’s choice and check if the password is right.

***void ManagerMenu(Identity\*& manager)***

This function is used when the user is a manager. Show the interface and switch between different functions.

***void ChefMenu(Identity\*& manager)***

This function is used when the user is a chef. Show the interface and switch between different functions.

***void CustomerMenu(Identity\*& manager)***

This function is used when the user is a customer. Show the interface and switch between different functions.

In function ***void LoginIn(string fileName, int type)*** , we simply get the user name and password of the user and then compare it with the password and name stored in the file. If the password and name are all correct, switch to the corresponding interface. If the login fails, ask the user to log in again.

The basic functions of the other three functions are basically the same. So, we take the ***void ManagerMenu(Identity\*& manager)*** as an example.

When the login function is called in the last step, there is a very important pointer generated:

***Identity\* User = NULL.*** This is the pointer of type class identity. This is a parent class pointer used to point to subclasses. In these three functions the first thing we do is to convert this pointer into the subclass type. For example in here: ***Manager\* tempManager = (Manager\*)manager*** so that the pointer can call all the functions and objects in the sub class.

Then, ask the user to select to different functions and call different class functions to realize them.

Whenever the user want to quit, press 0 to quit the program. The flow chart of the program is shown in the analysis part.

## 3.6 Other Classes

***class RawMaterial***

With public object

***string rawMaterialName***

The string used to store the name of the row materials.

***string isStock***

The string used to store the stock of the materials.

***class Dishes***

With public variables

***int dishID***

This object is used to store the ID of the dishes.

***string dishesName***

This object is used to store the name of the dishes.

***int dishesPrice***

This object is used to store the price of the dishes.

***int rawMaterialPrice***

This object is used to store the price of the row materials.

***class ExpenseRecord***

***int dishID;***

This object is used to store the ID of the dishes.

***string dishesName;***

This object is used to store the name of the dishes.

***int dishesPrice***

This object is used to store the price of the dishes.

***globalFile.h***

This is the head file that used to define all the file names as macro constants. So that whenever a file name is needed, we can directly use the global variable name hinted by the compiler, so that the name will not be entered wrongly.

The specific names are as follows:

***#define CUSTOMER\_FILE "Customer.txt"***

The file used to store the information about customers.

***#define CHEF\_FILE "Chef.txt"***

The file used to store the information about chefs.

***#define MANAGER\_FILE "Manager.txt"***

The file used to store the information about managers.

***#define RAWMATERIAL\_FILE "RawMeterial.txt"***

The file used to store the information about row materials.

***#define DISHESMENU\_FILE "DishesMenu.txt"***

The file used to store the information about dishes.

***#define EXPENSERECORD\_FILE "ExpenseRecord.txt"***

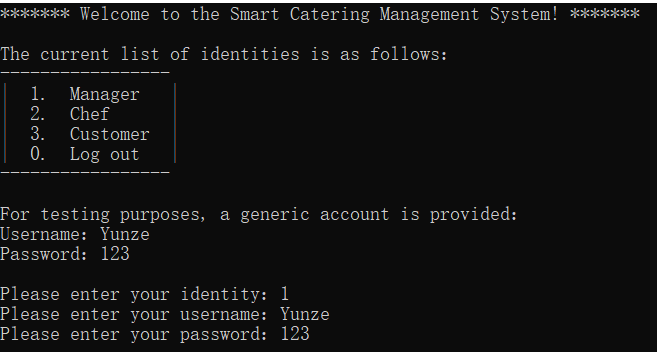
The file used to store the information about expense records

# 4.Test

## 4.1 Test in the main interface

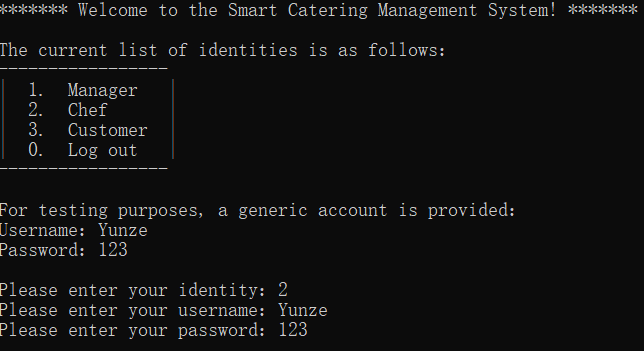
**Test 1**

**When the user selects the manager's identity, enter the correct user name and password.**



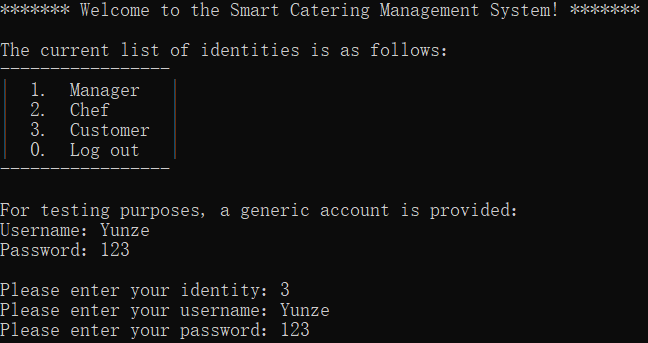
**Test 2**

**When the user selects the chef's identity, enter the correct user name and password**



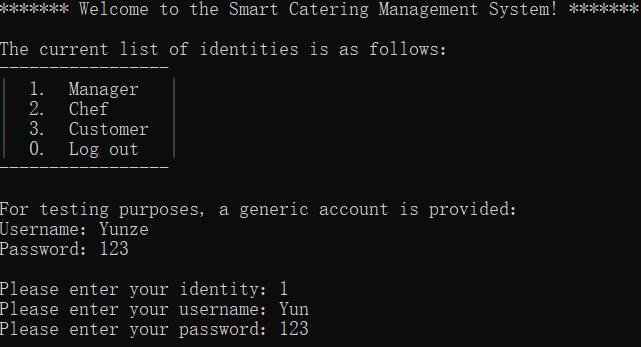
**Test 3**

**When the user selects the identity of the consumer, enter the correct user name and password**



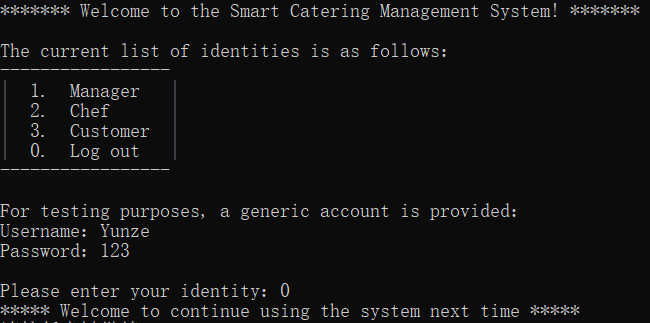
**Test 4**

**When the user selects an identity, enter a wrong account and password which are non-corresponding**



**Test 5**

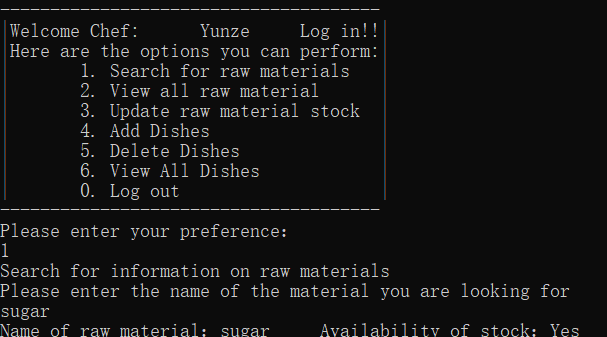
**When the user enters 0 chooses to exit**



## 4.2 Test of the chef interface

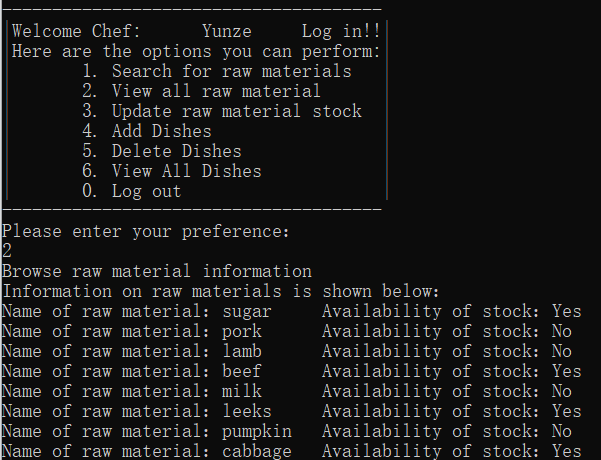
**Test 1**

**When the user chooses to search for raw material information, for example, search for "sugar"**



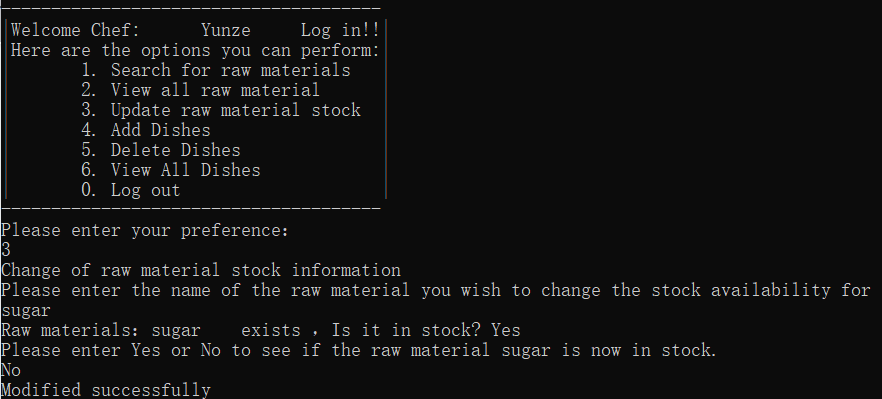
**Test 2**

**The user selects to browse all raw material information, and all raw material information will be printed on the screen**



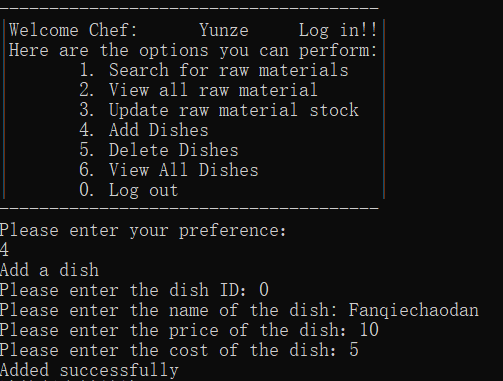
**Test 3**

**When updating the raw material stock.For example, change sugar’s stock availability**



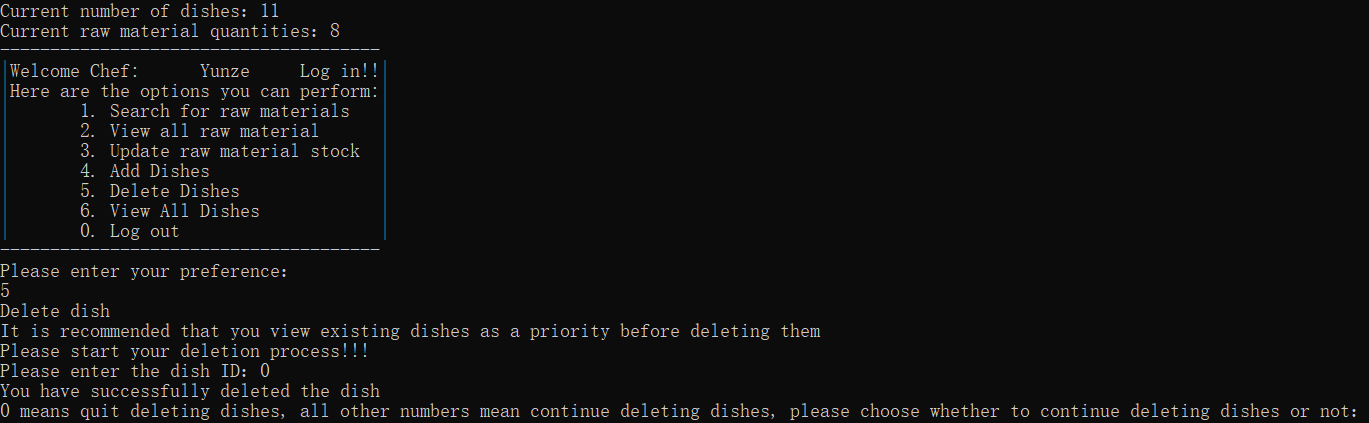
**Test 4**

**When adding dishes, the user selects to add: Fanqiechaodan, and there will be Fanqiechaodan in the menu.**



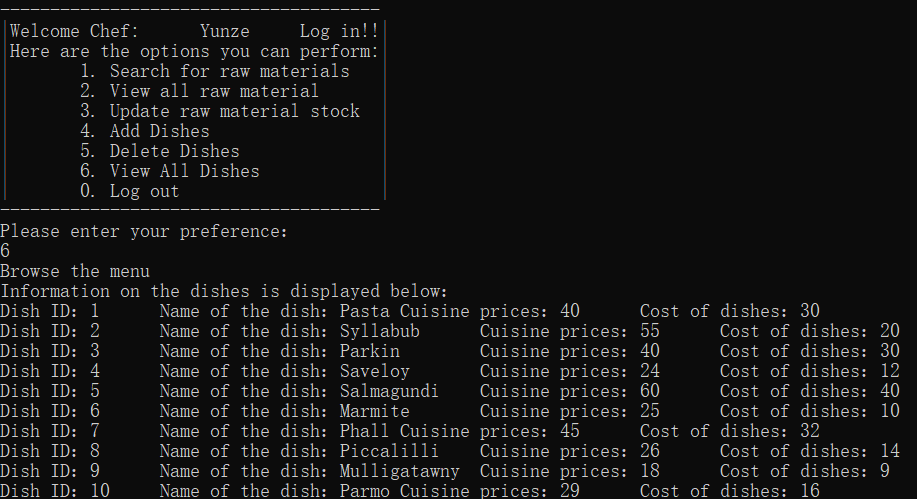
**Test 5**

**When the user chooses to delete a dish, for example, delete "Fanqiechaodan"**



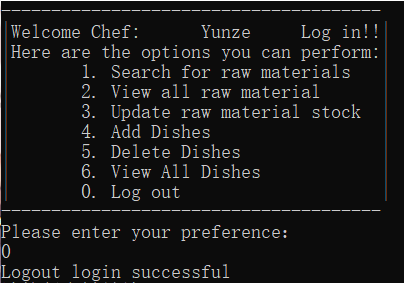
**Test 6**

**When the user chooses to browse all the information of dishes, all the information of dishes will be printed on the screen**



**Test 7**

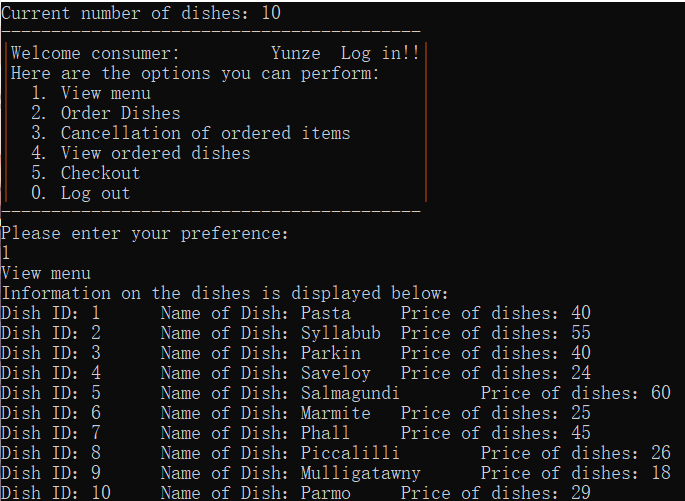
**When the user chooses to log out, the system will cut the interface to the main interface**



## 4.3 Test of the customer interface

**Test 1**

**The user selects to view the menu and all the menu information will be printed on the screen.**



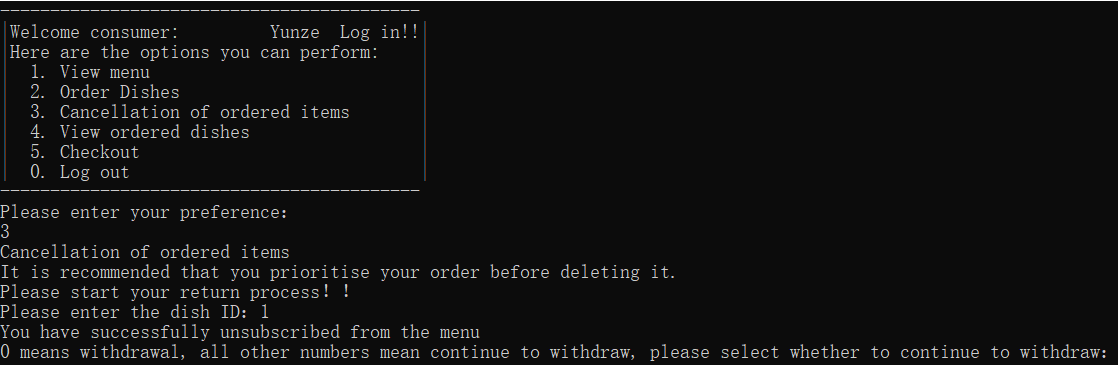
**Test 2**

**The user selects the ordering function, for example, the user enters the dish ID: 1**



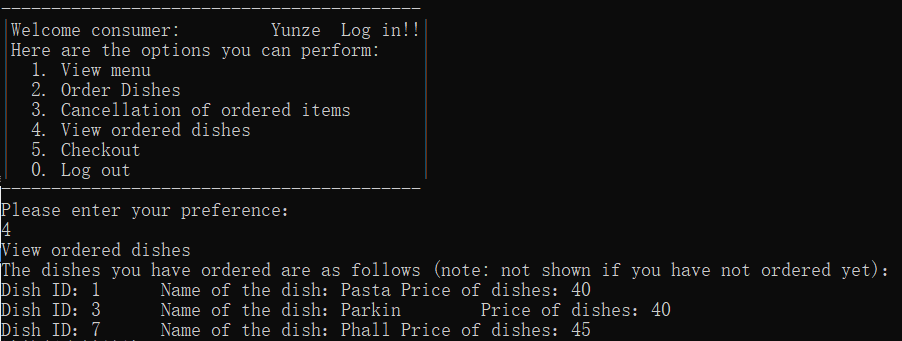
**Test 3**

**The user selects to cancel the ordered dishes. For example, the user selects to delete dishes 1, and input ID=1**



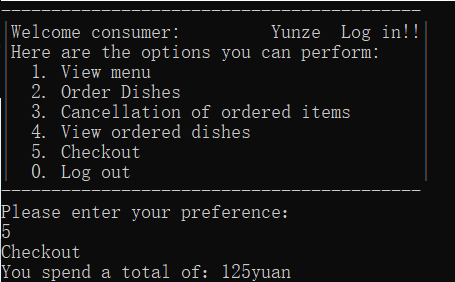
**Test 4**

**Users select to view the ordered dishes, and all the dishes selected by users will be printed on the screen**



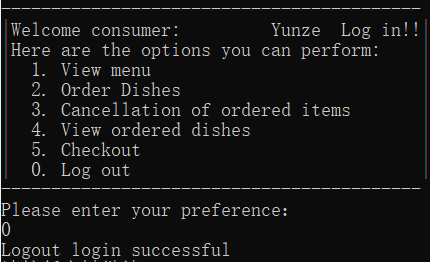
**Test 5**

**The customer selects check-out, and the total price of all the dishes will be calculated and display the results on the screen**



**Test 6**

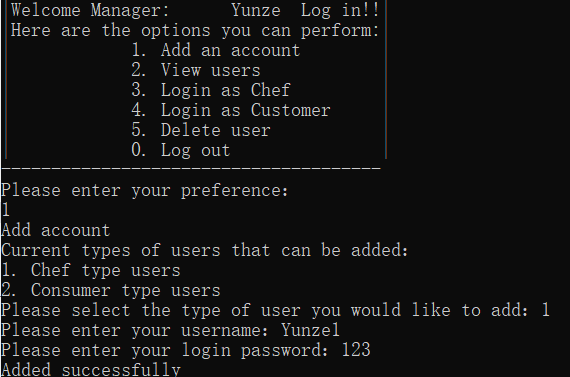
**When the user chooses to log out, the program will automatically return to the main interface**



## 4.4 Test of the manager interface

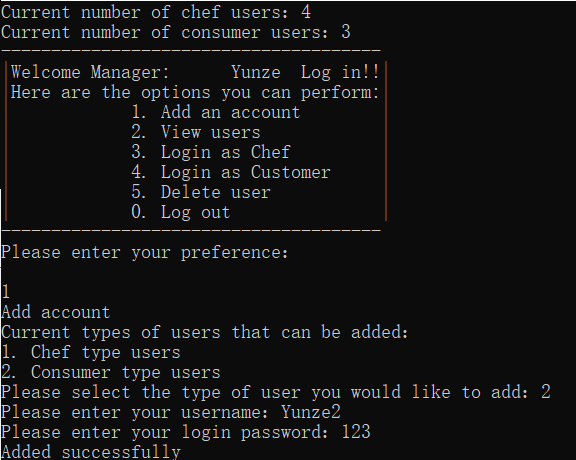
**Test 1**

**The manager chooses to add a new chef account**



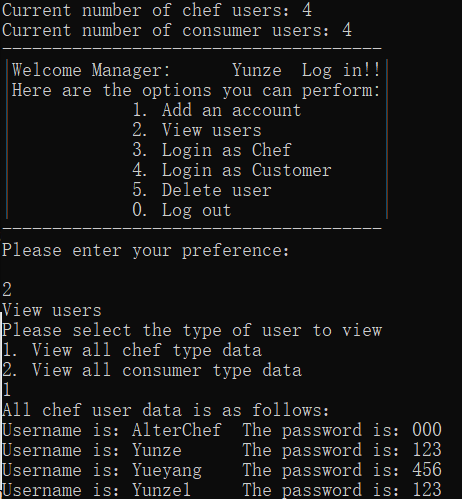
**Test 2**

**The manager chooses to create a new account for the consumer**



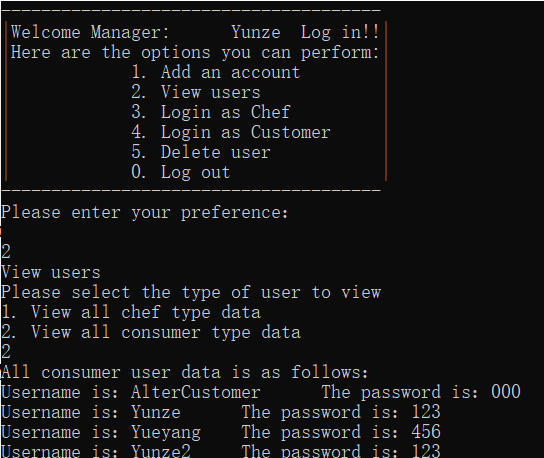
**Test 3**

**The manager chose to look at the chef's account**



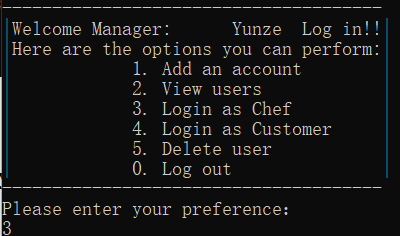
**Test 4**

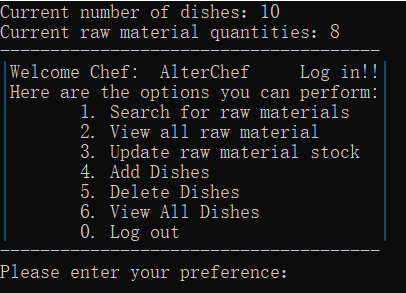
**The manager chooses to look at the consumer's account**



**Test 5**

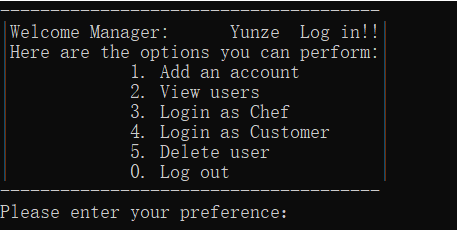
**The manager chooses to log in as the chef, and the system jumps to the interface of the chef**

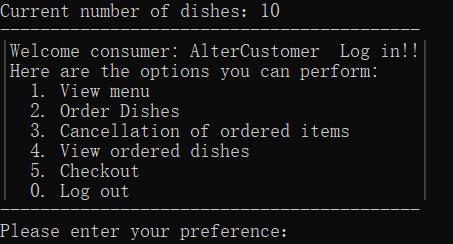




**Test 6**

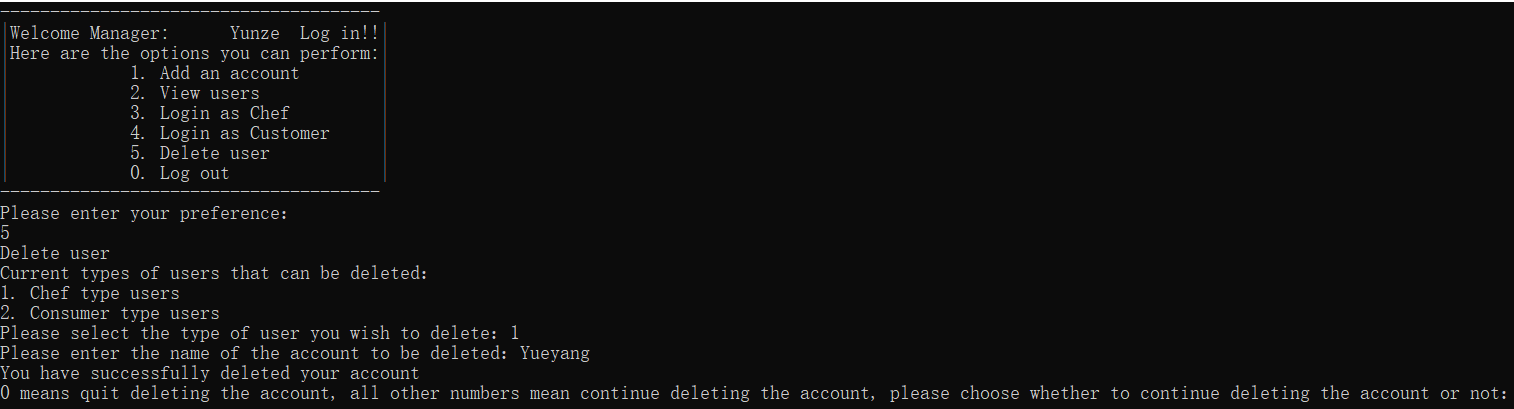
**The manager chooses to log in as a consumer, and the system jumps to the interface of the consumer**





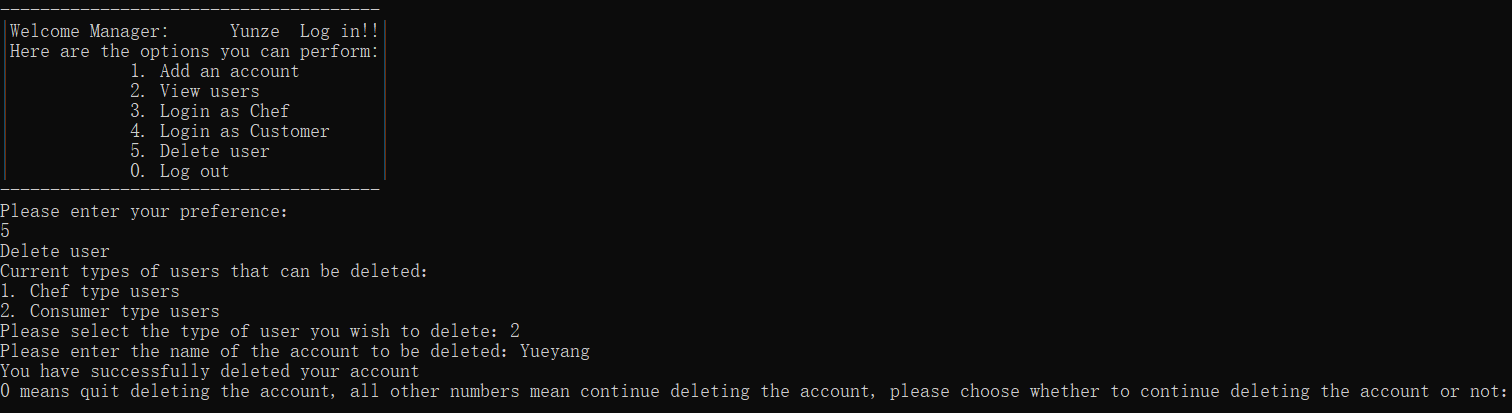
**Test 7**

**The manager chooses to delete user:Chef:Yueyang**



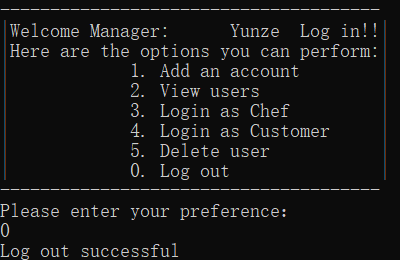
**Test 8**

**The manager chooses to delete user:Customer:Yueyang**



**Test 9**

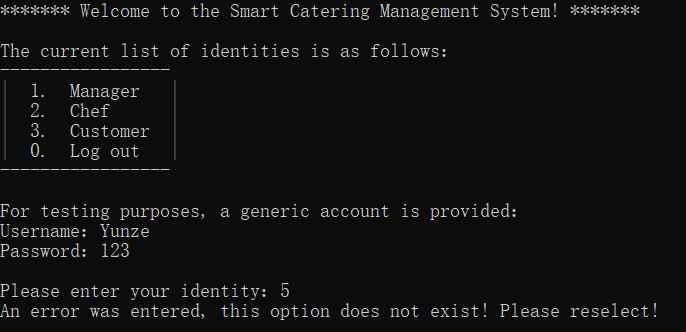
**The manager chooses to exit the account, and the system automatically jumps back to the main screen**



## 4.5 Test of the input error

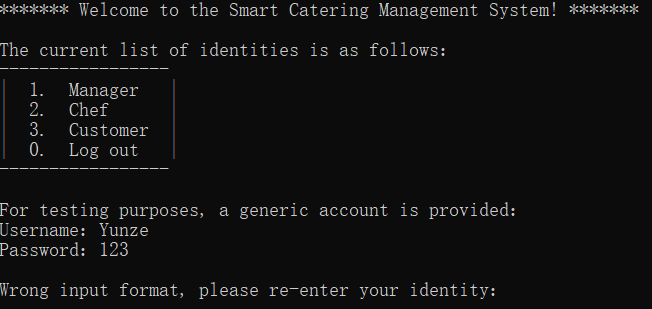
**Test 1**

**On the main screen, the user is supposed to enter a number between 0 and 3, but 5 is entered.Therefore, the system will ask the user to re-enter it.**



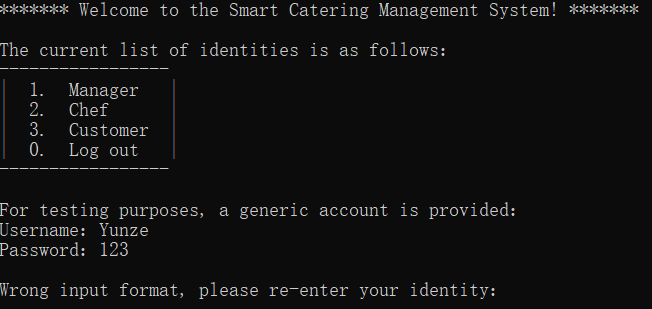
**Test 2**

**On the main screen, the user is supposed to enter a number between 0 and 3, but character p is entered. Therefore, the system will ask the user to re-enter it.**



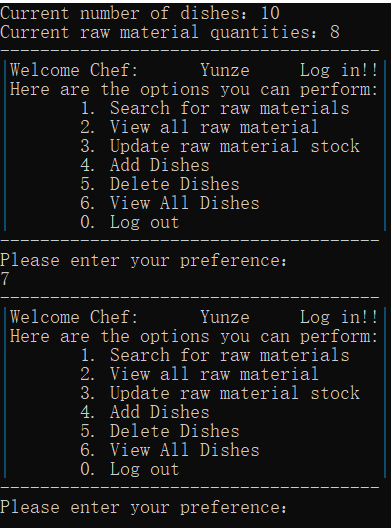
**Test 3**

**On the main screen, the user is supposed to enter a number between 0 and 3, but special character @ is entered. Therefore, the system will ask the user to re-enter it.**



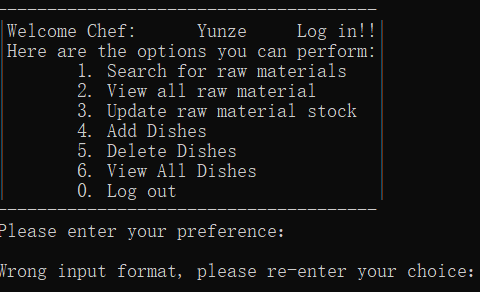
**Test 4**

**On the chef’s interface, the user is supposed to enter a number between 0 and 6, but 7 is entered.Therefore, the system will ask the user to re-enter it.**



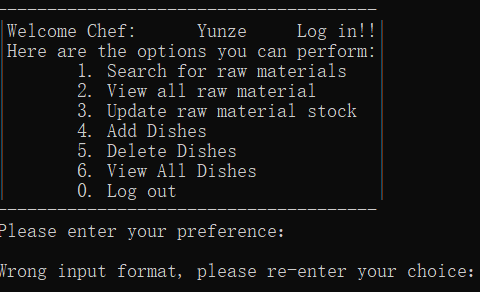
**Test 5**

**On the chef’s interface, the user is supposed to enter a number between 0 and 7, but character p is entered. Therefore, the system will ask the user to re-enter it.**



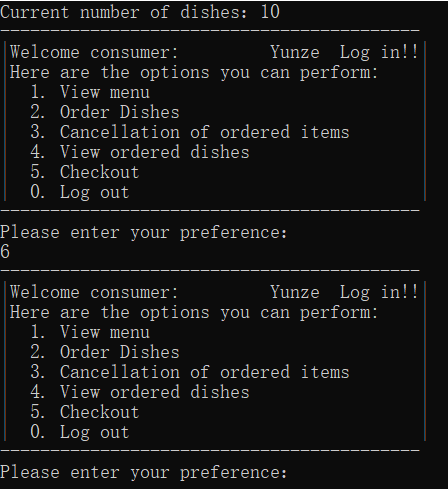
**Test 6**

**On the chef’s interface, the user is supposed to enter a number between 0 and 7, but special character @ is entered. Therefore, the system will ask the user to re-enter it.**



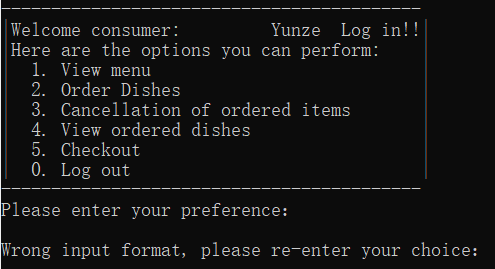
**Test 7**

**On the customer’s interface, the user is supposed to enter a number between 0 and 5, but 6 is entered.Therefore, the system will ask the user to re-enter it.**



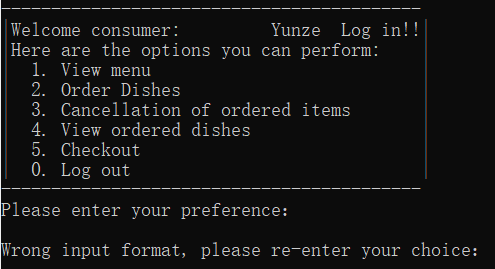
**Test 8**

**On the customer’s interface, the user is supposed to enter a number between 0 and 5, but character p is entered. Therefore, the system will ask the user to re-enter it.**



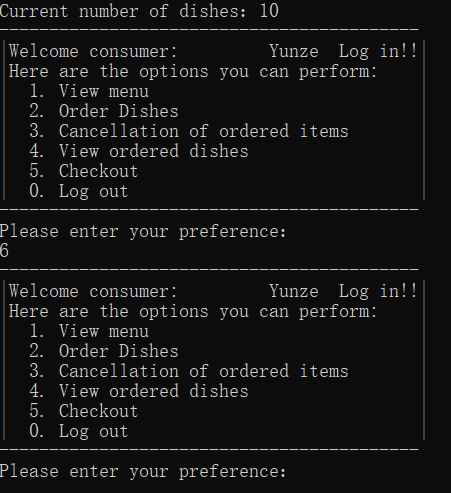
**Test 9**

**On the customer’s interface, the user is supposed to enter a number between 0 and 5, but special character @ is entered. Therefore, the system will ask the user to re-enter it.**



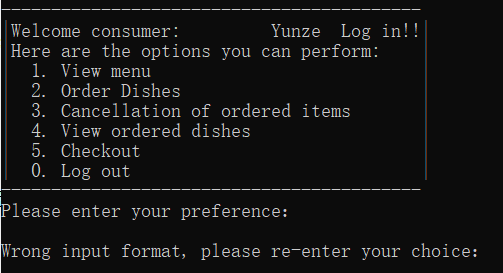
**Test 10**

**On the customer’s interface, the user is supposed to enter a number between 0 and 5, but 6 is entered.Therefore, the system will ask the user to re-enter it.**



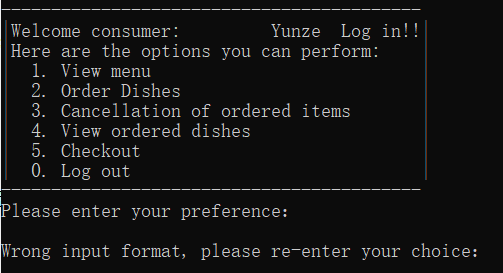
**Test 11**

**On the customer’s interface, the user is supposed to enter a number between 0 and 4, but character p is entered. Therefore, the system will ask the user to re-enter it.**



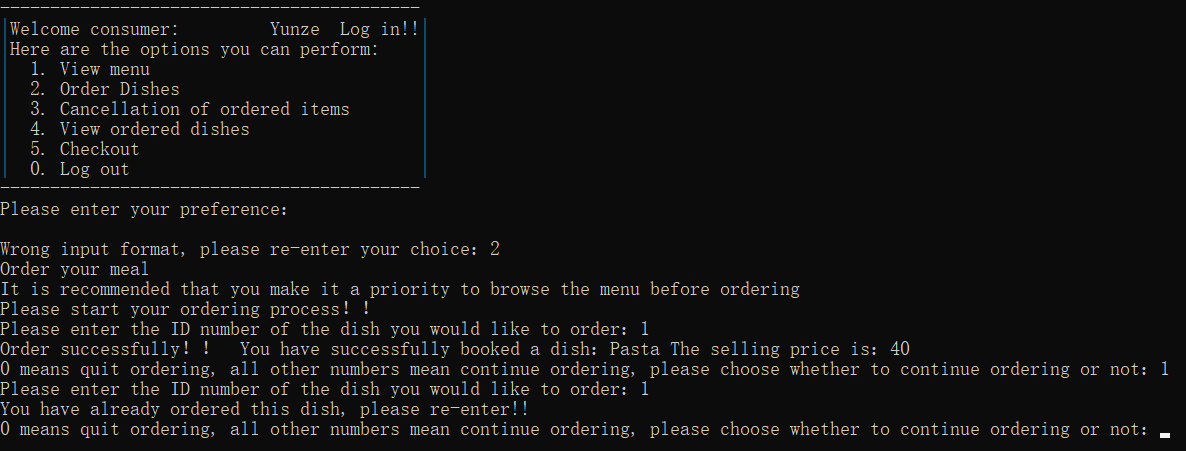
**Test 12**

**On the customer’s interface, the user is supposed to enter a number between 0 and 4, but special character @ is entered. Therefore, the system will ask the user to re-enter it.**



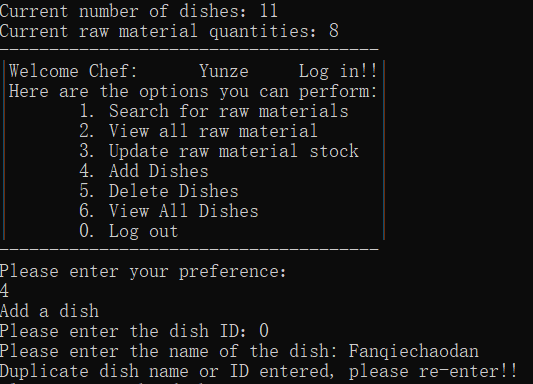
**Test 13**

**In the consumer interface, the user can not input the same dishes, when the user input the same dishes, the system will let the user re-input**



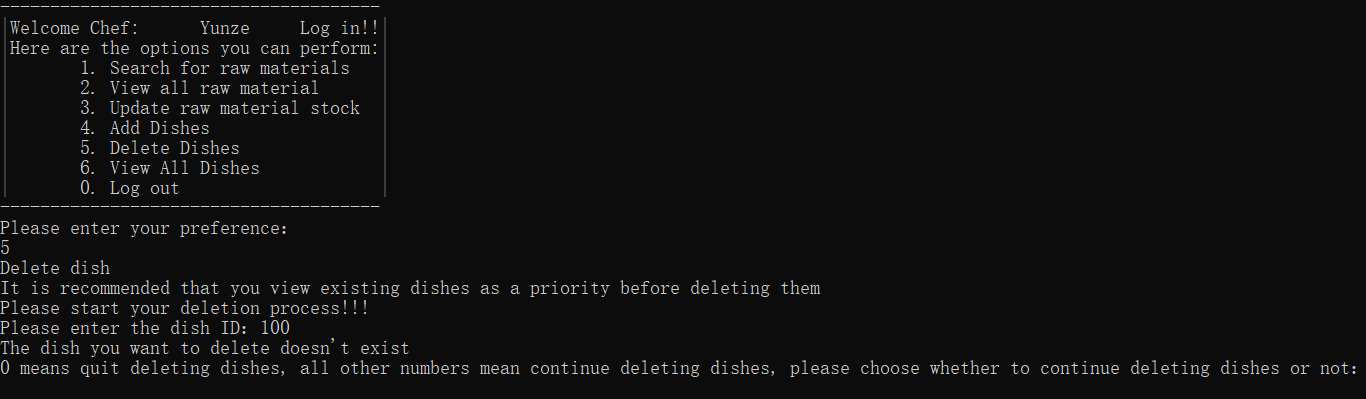
**Test 14**

**When the chef adds the same dishes, the program will tell the chef to re-input it.**



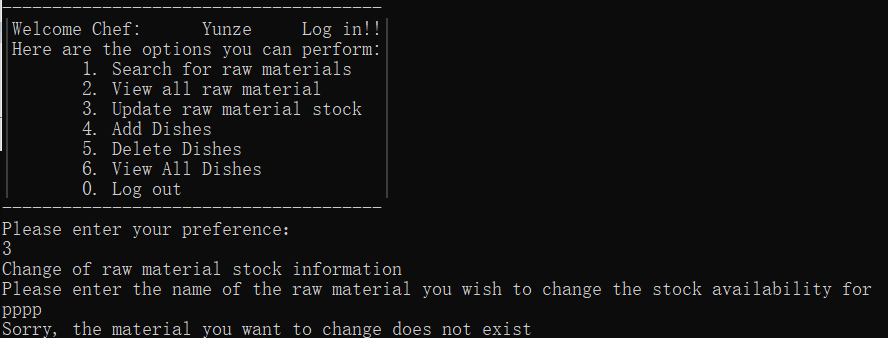
**Test 15**

**When the chef would like to delete a dish which do not exist in our menu, the program will tell him to re-input.**



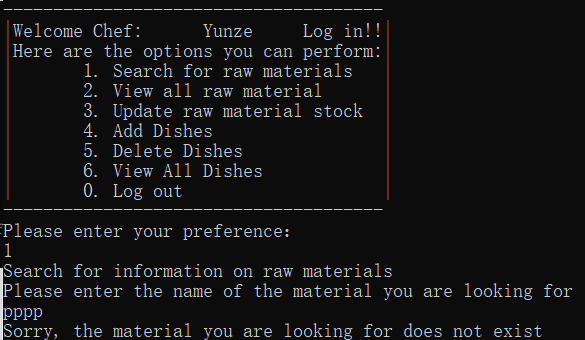
**Test 16**

**When the chef would like to update raw material stock which do not exist , the program will tell him to re-input.**



**Test 17**

**When the chef would like to search for raw materials which do not exist, the program will tell him to re-input.**



# 5. Contribution Form

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | Student ID | Design | Coding | Testing | Report | Remarks |
| Yunze.Song | 1928064 | 30 | 30 | 20 | 20 | Mainly responsible for the construction of vegetable,raw material class and main function construction. In the report part, mainly responsible for the analysis and test matching with the code |
| Yuhang.Jiang | 1927888 | 20 | 20 | 30 | 30 | Mainly responsible for the implementation of chef class and the call of related functions and files. In the report part, mainly responsible for the implementation of functions related to codes |
| Yueyang.Li | 1927996 | 30 | 30 | 20 | 20 | Mainly responsible for the analysis of consumer class, the realization of functions and the call of related files. In the report, mainly responsible for the writing and modification of the related code |
| Tiankuo.Jiao | 1929098 | 20 | 20 | 30 | 30 | Mainly responsible for the compilation of manager class, the realization of functions and the call of related files. In the report, responsible for the satisfaction and debugging of related codes |