EEE102 C++ Programming and Software Eng. II

Final group project

Project C: Warehouse Management System

Ruichen Zheng 1405184
Zhaozhong Wu 1404831
Ziheng Zhang 1405898
Chongyun Wang 1406074
Shifan Zhao 1404833

5-25-2016

Contribution form

Ruichen Zheng: Design the program framework, basic functions and file operation, write the report except for testing and user manual.

Zhaoshi Fan: Conclude the user specification; accomplish and test Time and Container class.

Zhaozhong Wu: Accomplish and test User, Customer, Keeper and Goods class.

Chongyun Wang: Accomplish and test Database class; enhance the file operation.

Ziheng Zhang: Design main function and user interface; draw all the tables and graphs in the report.

		Spec	Design	Coding	Testing	Doc.
Ruichen.Zheng	Rate	2	5	4	1	4
1405184	Work Load	10%	35%	30%	0%	30%
Ziheng.Zhang	Rate	2	2	2	4	4
1405898	Work Load	20%	15%	15%	25%	20%
Chongyun.Wang	Rate	2	3	3	3	2
1406074	Work Load	20%	25%	20%	25%	15%
Shifan.zhao	Rate	4	1	3	3	2
1404833	Work Load	30%	10%	15%	25%	15%
Zhaozhong.Wu	Rate	2	2	3	4	3
1404831	Work Load	20%	15%	20%	25%	20%

	How did the student appreciate the work within the group?
Ruichen.Zheng 1405184	I designed the framework and file operation of the program. My teammates are nice enough to give feedbacks and accomplish and test the whole program. I also wrote many parts of the report.
Ziheng.Zhang 1405898	I suppose the teamwork as a success, full completion of all project requirements. During the teamwork, I focused on the testing code and improved user interface, with relative low contribution in Spec and Design. My teammates were excellent and decisive in proposing ideas to design the project and meanwhile, they were good to debug the program. While I prefer to what I can handle, such as drawing UML, CRC etc.
Chongyun.Wang 1406074	I participated in every part of this project. Mainly focus on building class Database, file operations and the connection between Datebase and other specific classes.
Shifan.zhao 1404833	I made some efforts to conclude customer's specifictions. But I didn't do much about the design of this project. The Time and Container classes are my parts. I also test the Time part.
Zhaozhong.Wu 1404831	My contribution to our group in this project is, for the programming part, writing the goods class, general program test for the whole function, and specific test for the goods, container, keeper, and customer class. For the report part, I wrote the testing part of the goods, container, keeper, and customer class, and also the user manual.

Signature:

Ruichen Zheng

Ziheng Zhang

Chongyun Wang

Shifan Zhao

Zhaozhong Wu

Contents

Contribution form 1
Specification 3
Analysis and Design 4
  Program framework design 4
    Simplification 4
    Entities and relations 4
    Data read, manipulation and storage 5
    The sub pointer vector 6
    Rule for customer and goods deleting 6
    Time class and main function 7
    User function choose 7
  Class design 7
    Container class design 7
    Time class design 8
    Goods class design 9
    User, Customer and Keeper class design 10
    Database class design 13
    Friendship and user authority achievement 16
    The main function design 17
Testing 18
    Time class test (by Zhaoshi Fan) 18
    Container, Goods, Customer and keeper class test (by Zhaozhong Wu) 23
    Database class test (by Chongyun Wang) 30
    The main function test (by Ziheng Zhang) 40
Debugs report 46
  Pointer and vector 46
  0 or delete 47
  Endl 47
  Include 47
User Manual 47
  Login 47
  Keeper 47
  Customer 48
  Exit and data saving 48

Specification

This project aims to design a warehouse management system (WMS) for the management of the warehouse. The WMS should meet the most of the customer's requirements.

The specifications were discussed and sorted in group discussion. Five entities were picked up from the requirements: warehouse, container, goods, keeper and customer. The specific requirements of each entity are listed below:

Warehouse:

1. There are two types of warehouse, the large one and the small one.
2. The large warehouse has 50*10 standard containers; the small warehouse has 40*10 small containers.

Container:

1. There are two types of containers, the standard one and the small one.
2. One container may belong to one specific warehouse with its location.
3. The 2600 containers exist regardless of their states whether they are stored with goods.

Goods:

1. Goods can be stored in more than 1 container.
2. One batch of goods can only be stored in one warehouse.
3. The fee of goods depends on both its space and duration.

Keepers:

1. Keeper can edit warehouse information.
2. Keeper is not permitted to modify goods information.
3. The Keeper may have the authority to add/delete customers, but he may have no authority to change customer's password and goods information.

Customers:

1. Customer can only search their own goods.
2. Customer can check fee to pay.
3. Customer can check in/out goods.
4. Customer can change their own goods information and duration.
5. Customer may only have access to their goods. How goods are stored and where it is stored depends on the arrangement of the WMS.

After group discussion, all the functionalities of the user are eventually determined below:

Customer:

1. Check in new goods
2. Brower current goods
3. Modify current goods
4. Check out current goods

5. Modify personal information

Keeper:

1. Display all warehouses
2. Display all customers
3. Search container
4. Search goods
5. Search customers
6. Add customer
7. Delete customers
8. Modify account information

Analysis and Design

Program framework design

Simplification

In order to make this WMS programmable, several simplifications are made:

1. The id, type(size), the warehouse it belongs to and location in warehouse of each container is initialized, fixed and cannot be changed during this program. The system will automatically generate required txt files if they are not found.
2. Due to the complexity of shape and volume comparison, the sizes of goods and containers are simplified with the unit space. Moreover, a rule is set that one (batch of) goods can be stored in more than one containers; one container can only store one part of goods.
3. Only one keeper is allowed in this WMS.

Entities and relations

After group discussion, the customer, keeper, container and goods are regarded as the 4 entities in this program. (Note that the information of warehouse can be directly obtained by the set of containers.) The ER model was made below to illustrate their relationships (Figure 1).

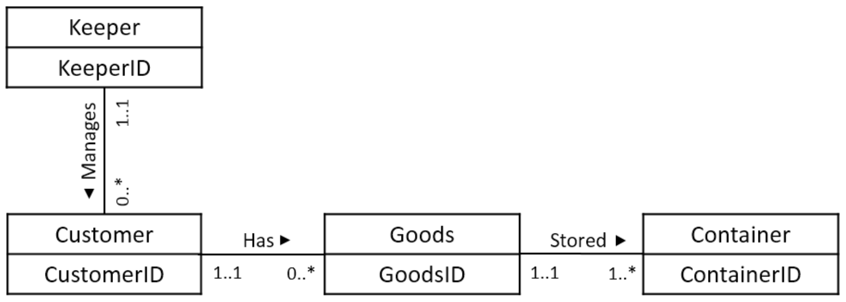


Figure 1: ER model

In words, their relationships are:

1. 1 keeper can manage more than 0 customers; 1 customer can only be managed by 1 keeper.
2. 1 customer can have 0 or more goods; 1 goods can only be haven by 1 customer.
3. 1 goods can be stored in at least 1 container; 1 container can only store 1 part of goods.

The container, goods, keeper and customer are linked by their id, which is unique and be seen as its primary key in the WMS. The advantage of this model is the authority. As displayed, the customer has only access of the goods, rather than container. Additionally, the keeper has the authority to manage customer, but he is not allowed to have access to goods to modify its information.

These 4 entities will be designed as classes in the WMS programming. Note that the class named User is determined as a base class, which the customer and keeper will inherent from.

Data read, manipulation and storage

When the classes were determined, the types of objects in this program are determined. However, the problem arose as how and where to temporarily store and manipulate these objects. More importantly, how to store this data after manipulation.

Class attribute for storing

Before considering file read and storage, it is necessary to list all the attributes of each class that needs to be stored:

1. Keeper: userId, password, name
2. Customer: userId, password, name
3. Goods: goodsId, userID, startTime, endTime, goodsInfo, goodSize
4. Container: containerID, locX, loxY, warehouseNum, goodsId, containerSize

In class temporary storage and manipulation

One specific class named database was made to read, temporarily store and manipulate these objects. Three vectors, customer vector, goods vector and container vector were created in the database class to store all those objects, because of their uncertain size (expect for the container objects, which was fixed to be 2600 by the project requirement).

TXT file

In this program, data preserve should be taken into consideration. The data was decided to be stored in txt files, which may ensure that each alteration can be made the for the next time that the program is running.

Due to the one to many relations of different entities, whether to store all the attributes in one single file or in different files is one of the most critical problems in this WMS program design. After group discussion, the plan of storing in three different files (Keeper and customer store in one file) is chosen. The reasons are as follows:

1. Storing in one single file needs 2600 long record. When there are only a few goods stored, abundant of unnecessary data will be read.
2. Storing in different file can reduce the information duplication. For example, if a goods is stored in 2 containers, there is no need to write the goods information twice.
3. Moreover, when an error is made when reading the file, it is much difficult to locate the error.

File read and write

As is shown above, one entity has the attributes of different type. To solve that, an algorithm was written to read each line of the program as one string. Afterwards, divide the long string into different substrings on the basis of number of attributes. To minimize the problem in format transformation, string stream is utilized.

To maintain the file integrity, whenever a file is missing, all the data replaced by default one and be written to the file.

After manipulation, the data is saved in the data as the same format. (Note that, the information of the only keeper is stored in the same file as the customers)

The sub pointer vector

This is the most innovative but also the most controversial part in this WMS design. This idea derives from the 1 to many relations.

For the goods class, a vector attribute is made to store the pointers that pointing to the container objects, which has the same goodsId as the goods object, stored in container vector inside the database class. The same, customer class has a vector attribute that stores the pointers that pointing to the goods objects, which has the same customerId as the customer object, stored in goods vector inside the database class.

This action aims to reduce object duplication and unnecessary searching, as well as maintaining the data integrity. It is the one of the most critically design in this program. Previous, whenever a customer object needs to modify his goods, a specific function is required to be made inside the database class for searching goods with the same customerId. But now, the customer object can directly modify the data inside its class, in which case all the changes will be also made in the data stored in the database.

However, it is still a controversial design. During group discussion, its disadvantages have been argued a lot:

1. Complexity: Store the address of vector member has potential danger. When an element is added or removed from a vector, its memory will be re-allocated, which may case error. The details will be covered in the debug part.
2. Limitation: For the customer, the addition of sub pointer vector may realize most of the functionalities. However, when the function needs the manipulation of the whole vector, it must be made in the database class. For example, the function adding goods needs adding new element in the original vector.
3. Programming difficulty: It is the most controversial of this design. When the sub pointer class is included in both goods and customer class, these three class are not independent any more. It may make it much difficult to let group members to write single class and test it.

Although the disadvantages it has, the design was eventually approved in group discussion. To contend with the programming difficulties, the group leader is required to write the basic framework of the program and label the functions of each class, as well as designing and accomplishing the basic functions of each class to make the program works. Group members are responsible for perfecting basic functions and adding advanced features in each class to complete the program, as well as the testing.

Rule for customer and goods deleting

Because the number of elements in container vector is fixed, which cannot be added or deleted. As a result, when a container is empty, its goodsid is set to 0.

To maximize usage of sub pointer vector, a decision is made when deleting a goods or customers, its id is set to zero rather than directly erase it from the original vector. When the file is saved, all the goodsId and customerId with id equals to 0 will be remove from the original vector.

Time class and main function

A specific Time class is designed to resolve the time setting and calculation. The WMS is the main class for this program.

User function choose

To avoid unnecessary error, no cin was decided to be used through the program. Alternatively, getline() is used to read user's input into string and convert into integer if necessary. By now, invalid input will be rejected rather than destroy running of the whole program.

Class design

There are totally 7 classes with header files and one class with main function in this WMS program.

Container class design

Container class is a simple and independent class. It defines the basic data of a container, as well as several functions to print out its information. Its structure and relation with other classes are illustrated in Figure 2 and 3.

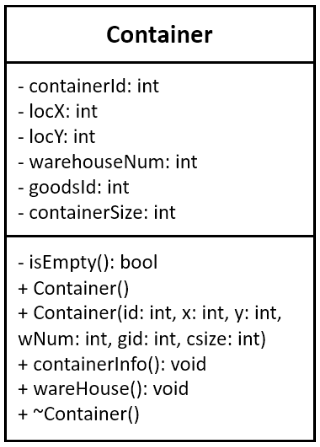


Figure 2: UML of Container class

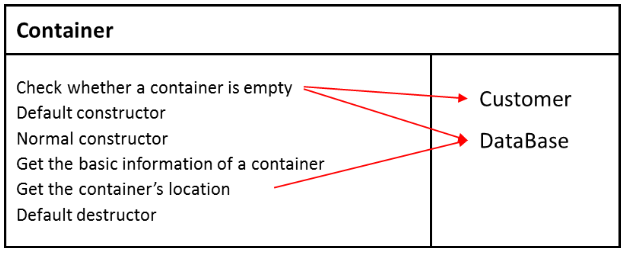

Container
Check whether a container is empty
Default constructor
Normal constructor
Get the basic information of a container
Get the container's location
Default destructor
Customer
DataBase

Figure 3: CRC of Container class

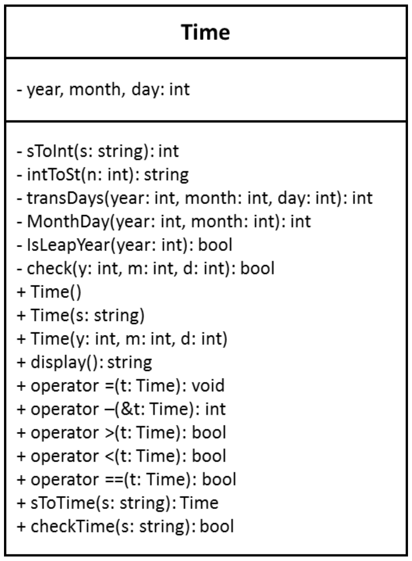
(1) isEmpty() is a private bool function that is used to check whether this container is empty by comparing its goodsId to 0. It is made private to ensure only its friend classes have the access to this function.

(2) Container() and ~Container() are the default constructor and destructor. They are kept default setting because no code will directly create a container without parameters.

(3) Container(int id, int x, int y, int wNum, int gid, int csize) is a normal constructor that create and initialize a container with passed data.
(4) containerInfo() is a void function to print out the basic information of this container.
(5) warehouse() is another void function to show the warehouse it belongs to, as well as the its location in that warehouse.

Time class design

Time class is a simple and independent class. It defines the basic data of a Time. It also encloses many function to check the whether the time is valid, as well as the operator overloading for Time calculation and comparison. Its structure and relation with other classes are illustrated in Figure 4 and 5.

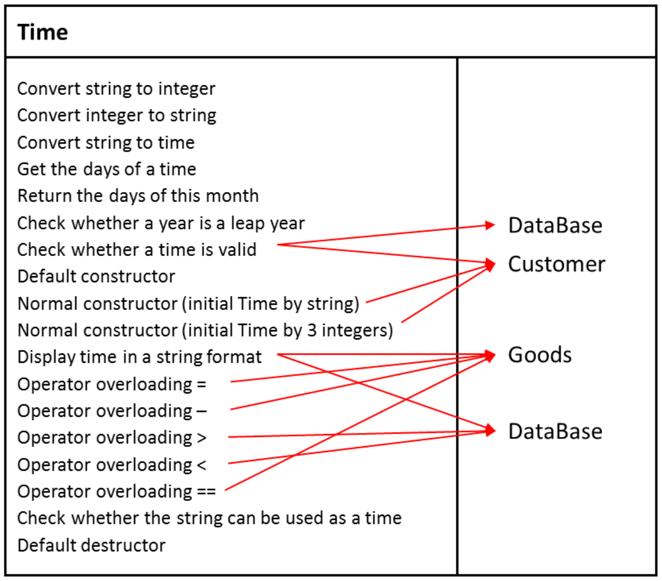


Time

- year, month, day: int

- sToInt(s: string): int
- intToSt(n: int): string
- transDays(year: int, month: int, day: int): int
- MonthDay(year: int, month: int): int
- IsLeapYear(year: int): bool
- check(y: int, m: int, d: int): bool
+ Time()
+ Time(s: string)
+ Time(y: int, m: int, d: int)
+ display(): string
+ operator =(t: Time): void
+ operator –(&t: Time): int
+ operator >(t: Time): bool
+ operator <(t: Time): bool
+ operator ==(t: Time): bool
+ sToTime(s: string): Time
+ checkTime(s: string): bool

Figure 4: UML of Time class



Time

Convert string to integer
Convert integer to string
Convert string to time
Get the days of a time
Return the days of this month
Check whether a year is a leap year
Check whether a time is valid
Default constructor
Normal constructor (initial Time by string)
Normal constructor (initial Time by 3 integers)
Display time in a string format
Operator overloading =
Operator overloading –
Operator overloading >
Operator overloading <
Operator overloading ==
Check whether the string can be used as a time
Default destructor

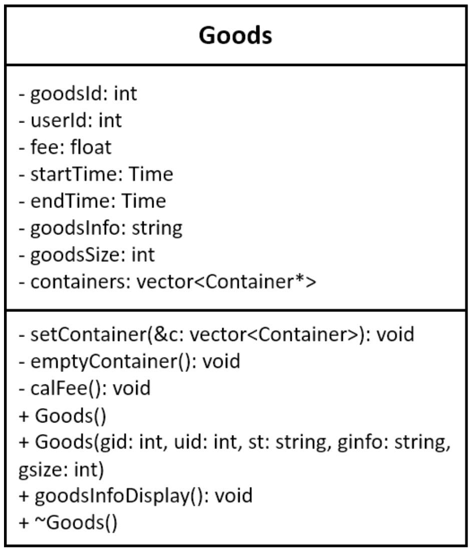
Figure 5: CRC of Time class

(1) sToInt(string s) and intToSt(int n) are private function to convert between string and integer.

(2) transDay(int year, int month, int day) is the private int function to get the total days of a given time (The year is not included for the day calculation).
(3) MonthDay(int year, int month) is a private int function to get the number of day of a given Time.
(4) IsLeapYear(int year) is a private bool function to check whether it is a leap year.
(5) check(int y,int m,int d) is private bool function to check whether time is valid by given integer parameters.
(6) Time() and ~Time() are the default constructor and destructor.
(7) Time(string s) and Time(int y, int m, int d) are default constructors that create and initialize Time with string or integer paramters.
(8) display() is a string function to return the Time in string format for output.
(9) Operator =,-,>,<,== are defined for Time object.
(10) sToTime(string s) is a function to convert a string to Time
(11) checkTime(string s) is a function to check whether a time is valid by string parameter.

Goods class design

Goods class is a simple and dependent class, which includes Container and Time class. It defines the basic data of a goods, as well as several functions to print out its information. Its structure and relation with other classes are illustrated in Figure 6 and 7.



Goods

- goodsId: int
- userId: int
- fee: float
- startTime: Time
- endTime: Time
- goodsInfo: string
- goodsSize: int
- containers: vector<Container*>

- setContainer(&c: vector<Container>): void
- emptyContainer(): void
- calFee(): void
+ Goods()
+ Goods(gid: int, uid: int, st: string, ginfo: string, gsize: int)
+ goodsInfoDisplay(): void
+ ~Goods()

Figure 6: UML of Goods class

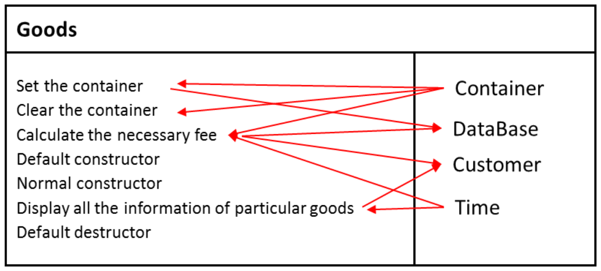


Figure 7:CRC of Goods class

(1) vector<Container*> containers is the sub pointer vector mentioned previously. Its includes the pointers pointing to the containers that has the same goodsId.
(2) setContainer(vector<Container> &c) is a void function to initialize the sub pointer vector.
(3) emptyContainer() is a private void function to clear the container by setting the goodsid to 0.
(4) calFee() is a private function to calculate the fee by the startTime, endTime and containers it stored.
(5) Goods() and ~Goods() are default constructor and destructor. They are kept default setting because no code will directly create a goods without parameters.
(6) Goods(int gid, int uid, string st, string et, string ginfo, int gsize) is a normal constructor to create and initialize goods with passed parameters.
(7) goodsInfoDisplay() is the function to print out the basic information of this goods.

User, Customer and Keeper class design

User class design

User class is a base class where the Customer and Keeper will inherent from. Its structure and relation with other classes are illustrated in Figure 8 and 9.

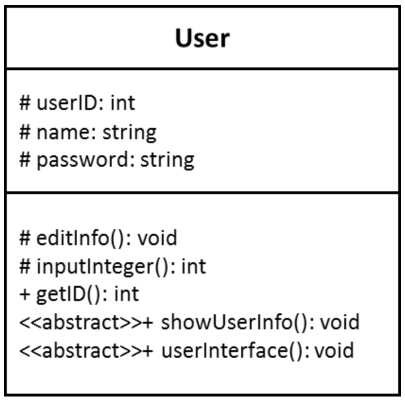


Figure 8: UML of User class

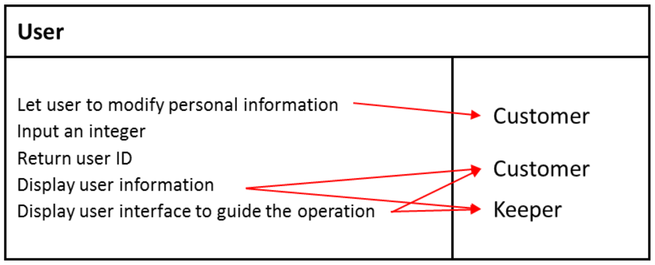

User
Let user to modify personal information
Input an integer
Return user ID
Display user information
Display user interface to guide the operation
Customer
Customer
Keeper

Figure 9: CRC of User class

(1) editInfo() is a protected function aims to let the user change his name and password.
(2) inputInteger() is a funtion ask user to input a integer by storing in string and return converted integer. It is made protected because it will be inherited and used in Customer and Keepr class.
(3) getID() is a intger function to return its userId.
(4) showUserInfo() is a virtual void function to the output of basic information. It will be used in main class by polymorphism.

(5) userInterface() is another virtual void function. It shows the function can only be done through the class, which is no need to be realized through database. It works combine the private functions in this class to protect privacy. It will be used in main class by polymorphism.

Customer class design

Customer class is a depend class inherited from User class, which also includes the Goods class. Its structure and relation with other classes are illustrated in Figure 10 and 11.

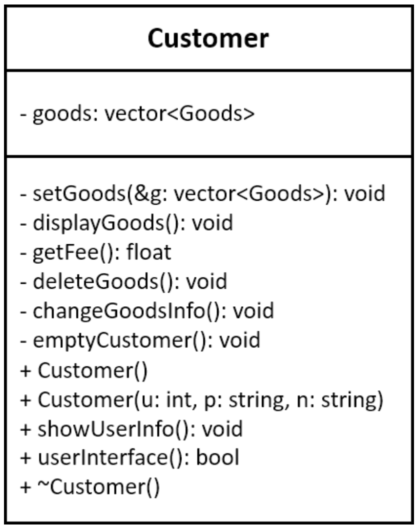


Figure 10: UML of Customer class

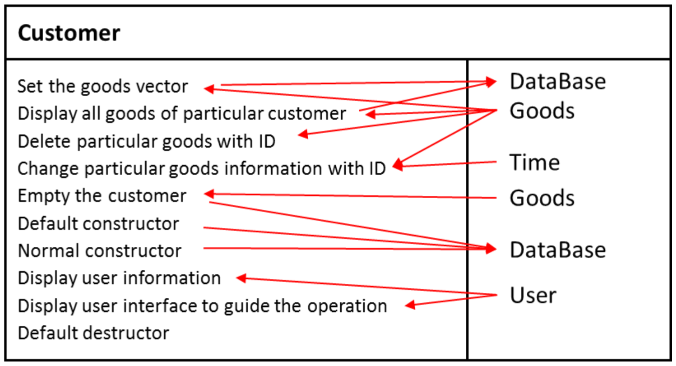

Customer
Set the goods vector
Display all goods of particular customer
Delete particular goods with ID
Change particular goods information with ID
Empty the customer
Default constructor
Normal constructor
Display user information
Display user interface to guide the operation
Default destructor
DataBase
Goods
Time
Goods
DataBase
User

Figure 11: CRC of Customer class

(1) vector<Goods*> goods is the sub pointer vector mentioned previously. Its includes the pointers pointing to the goods that has the same customerId.
(2) setGoods(vector<Goods> &g) is a void function to initialize the sub pointer vector.
(3) displayGoods() is the function to display all the goods belonging to this customer.
(4) getFee() is the function to return total fee of all the goods this customer is stored.
(5) deleteGoods() is the function to delete certain goods with id. Note that the goods are limited to the goods inside the sub pointer vector, which are the goods owned by this customer.
(6) changeGoodsInfo() is the function to change the goods information.
(7) emptyCustomer() is the function to delete this customer by changing the goodsId of all the goods owned by this customer to 0. It also changes the customerId to 0. It can only be called by keeper.

(8) Customer() and ~Customer() are default constructor and destructor. They are kept default setting because no code will directly create a Customer without parameters.
(9) Customer(int u, string p, string n) is the normal constructor to create and initialize a customer with passed parameters.
(10) showUserInfo() and userInterface() are functions inherited by User. They are the only public function in this class, which aims to protect privacy.

Keeper class design

Keeper class is a depend class inherited from User class. It cannot directly have access to Customer's class, which can put privacy protection under guarantee. Its structure and relation with other classes are illustrated in Figure 12 and 13.

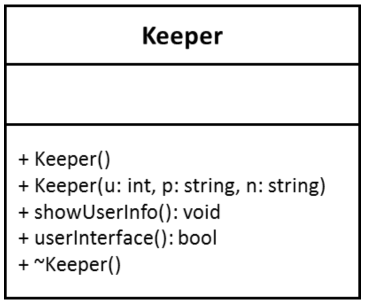


Figure 12: UML of Keeper class

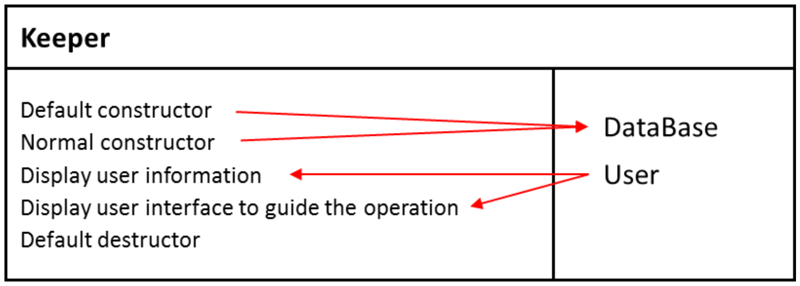


Figure 13:CRC of Keeper class

(1) Keeper()and ~Keeper() are default constructor and destructor.
(2) Keeper(int u, string p, string n)is the normal constructor to create and initialize a Keeper with passed parameters.
(3) showUserInfo() and userInterface() are functions inherited by User. They are the only public function in this class, which aims to protect privacy.

As is shown, only the basic functionality can be done inside the keeper class. The reason is that this class cannot have access the original Customer vector stored in the database class. As a result, the functionalities of the Keeper can only be realized by the arrangement of database class.

Inheritance

The inheritance is illustrated below (Figure 14).

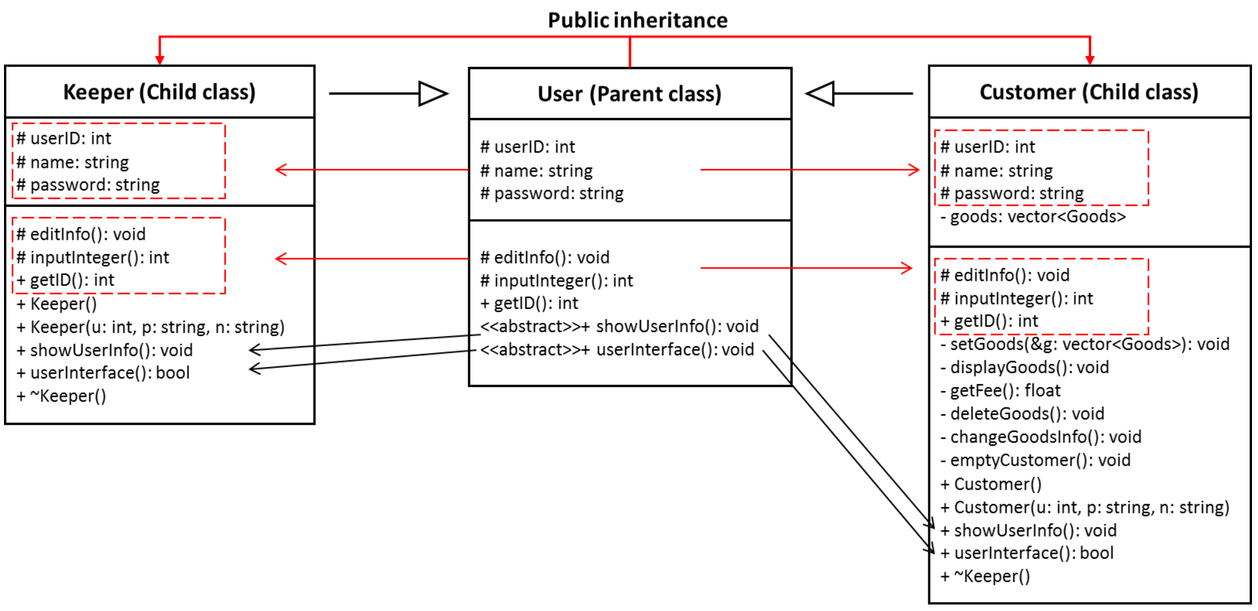
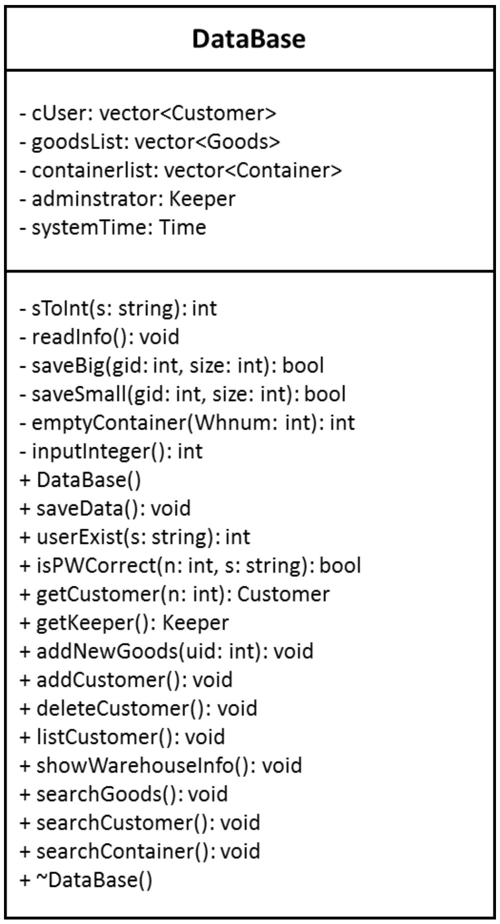


Figure 14: class inheritance

Database class design

Database class is the most complicated and critical class in the program. It not only directly or indirectly include all the other classes, but also is the friend class of all other class except for Time class. This class control the data read, manipulation and write. This class also defines some specific functions to achieve functionalities for Keeper and Customer. Its structure and relation with other classes are illustrated in Figure 15 and 16.



```
DataBase
- cUser: vector<Customer>
- goodsList: vector<Goods>
- containerlist: vector<Container>
- adminstrator: Keeper
- systemTime: Time
- sToInt(s: string): int
- readInfo(): void
- saveBig(gid: int, size: int): bool
- saveSmall(gid: int, size: int): bool
- emptyContainer(Whnum: int): int
- inputInteger(): int
+ DataBase()
+ saveData(): void
+ userExist(s: string): int
+ isPWCorrect(n: int, s: string): bool
+ getCustomer(n: int): Customer
+ getKeeper(): Keeper
+ addNewGoods(uid: int): void
+ addCustomer(): void
+ deleteCustomer(): void
+ listCustomer(): void
+ showWarehouseInfo(): void
+ searchGoods(): void
+ searchCustomer(): void
+ searchContainer(): void
+ ~DataBase()
```

Figure 15: UML of Database class

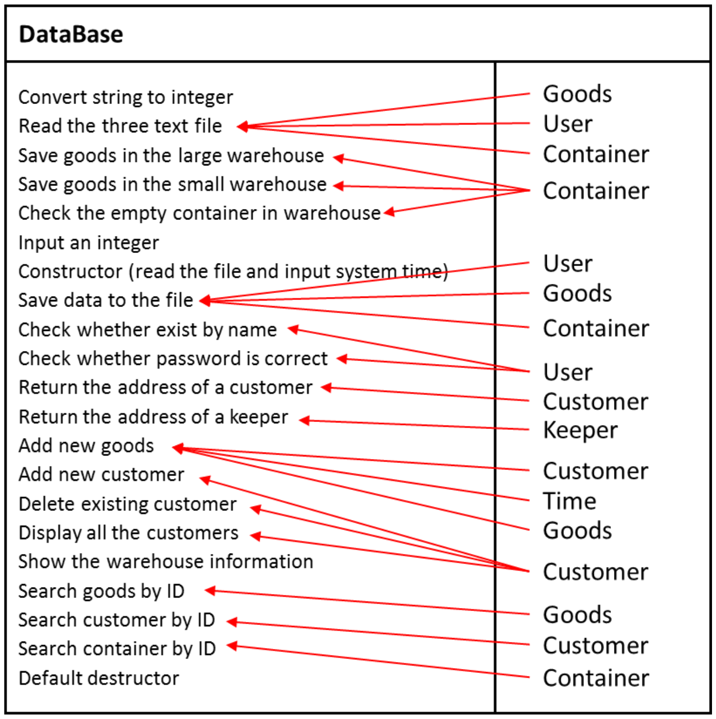

DataBase
Convert string to integer
Read the three text file
Save goods in the large warehouse
Save goods in the small warehouse
Check the empty container in warehouse
Input an integer
Constructor (read the file and input system time)
Save data to the file
Check whether exist by name
Check whether password is correct
Return the address of a customer
Return the address of a keeper
Add new goods
Add new customer
Delete existing customer
Display all the customers
Show the warehouse information
Search goods by ID
Search customer by ID
Search container by ID
Default destructor
Goods
User
Container
Container
User
Goods
Container
User
Customer
Keeper
Customer
Time
Goods
Customer
Goods
Customer
Container

Figure 16: CRC of Database class

Attributes

(1) cUser, goodList and containerList are three vectors that is used to store all the three kinds of objects that read from the file.
(2) Administrator is the only Keeper object throughout the program. It is firstly created and will be initialized when the file is read.
(3) systemTime is a Time object that let the user input the time they would like to set. It is also the earliest start time that goods can be set.

Functions

(1) sToInt(string s) is the function to convert string into integer. It is mainly used for declaring the objects when files are read.
(2) readInfo() aims to read the three txt file, which is added in constructor. The order of file for reading is container.txt, goods.txt, customer.txt. Whenever a file is read, corresponding objects will be created and pushed back to vectors. The sub pointer vector will also be initialized. Note that if one file is found missing, all the files will be written with default data.
(3) saveBig(int gid, int size) and saveSmall(int gid, int size) are bool functions to store goods to big(regular) or small container. A Boolean value will be returned depends on whether storing succeeds.
(4) emptyContainer(int WHnum) aims to return the number of empty containers by the passed warehouse number.
(5) inputInteger() is the function ask user to input a integer by storing in string and return converted integer.
(6) DataBase() is the default constructor. File will be read and the system time will be asked to set when the database object is created.

(7) saveData() is the function to save the data. It works by write all the data in the vectors to the txt file. It is not put into destructor because one chance is given if the user doesn't want to save his change.
(8) userExist(string s) is the function to check whether the user exists by the user name.
(9) isPWCorrect(int n, string s) is the function to check whether the password of username is the same as the record in database.
(10) getCustomer(int n) aims to return a customer pointer pointing to the customer object with the passed userId in the customer vector.
(11) getKeeper() aims to return a keeper pointer pointing to the only keeper object stored in database class.
(12) addNewGoods(int uid) is the function for user to add new goods. How goods are stored depend on the written algorithm. According to requirement, one batch of goods can only be stored in one warehouse. Note that because the customer cannot choose the container, the goods storing plan will firstly minimize customer's payment.
(13) addCustomer() is the function for the keeper to add new customers.
(14) deleteCustomer() is to let the keeper to delete an existing customer. When a customer is deleted, his goods will also be deleted. Moreover, the containers stored his goods will also be emptied.
(15) listCustomer() is to let the keeper to check all the customers with basic information.
(16) showWarehouseInfo() aims to let the keeper to show all the warehouse information.
(17) searchGoods(), searchCustomer() and searchContainer() are functions for the keeper to search particular item with goods id.
(18) ~DataBase() is the default destructor.

Friendship and user authority achievement

The friendship of each class is delineated below (Figure 17):

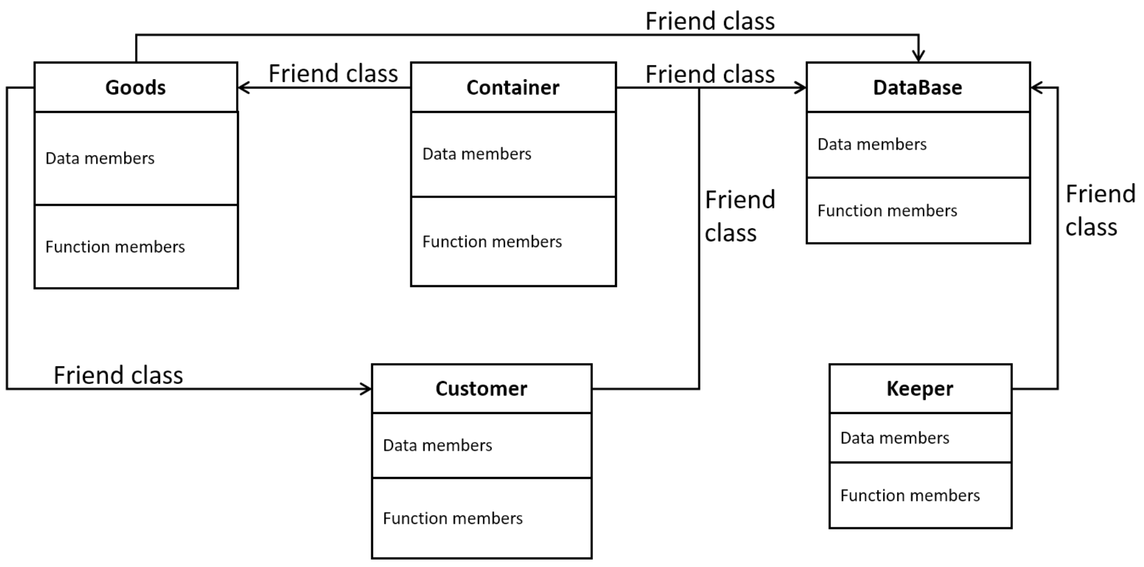


Figure 17: Friendship

Table 1 demonstrate the functionalities required by specification. Just the same as the friendships, the user, customer and keeper, can only have access to the data belonging to them. The realization of other functionalities is allocated by database. This design will achieve the authority by different user. The reasons are as lists:

1. Achieved by database: In order to make all the functionalities achieved its own class, the customer class should have access to the goods vector stored in database while the keeper class should have access to all the container, goods and customer vector. It is paradoxical to what is required by the project. In fact, the customer can only have access to goods belong to him. The keeper cannot have access to customer' password, either. As a result, let the database to arrange particular functions is a better choice.
2. Achieved by own class: Originally, all the functionalities should be allocated by database. However, by the sub pointer vector, some functionalities are achieved by the class itself. It is true that this design makes the program more complicated. But this method will reduce unnecessary searches. By only the database, a large amount of unnecessary search will be made to link customer, goods and container together. The enrollment of sub pointer vector may enhance the program efficiency when dealing a large amount of data.

Table 1: functionality and class

User	Functionality	Achieved by
Customer	Check in new goods	Database class
	Brower current goods	Customer class
	Modify current goods	
	Check out current goods	
	Modify personal information	
Keeper	Display all warehouses	Database class
	Display all customers	
	Search container	
	Search goods	
	Search customers	
	Add customer	
	Delete customers	
	Modify account information	Keeper class

In conclusion, this design will sacrifice simplicity but will simultaneously achieve different user authority and enhance the program efficiency.

The main function design

In the main function, a database object is first created. One user pointer is also created. It will point to particular object in the database depends on the which user is logged in. Note that the dynamic allocation is not used because every change made in this user will directly affect the data stored in database. The flow chart is illustrated below (Figure 18).

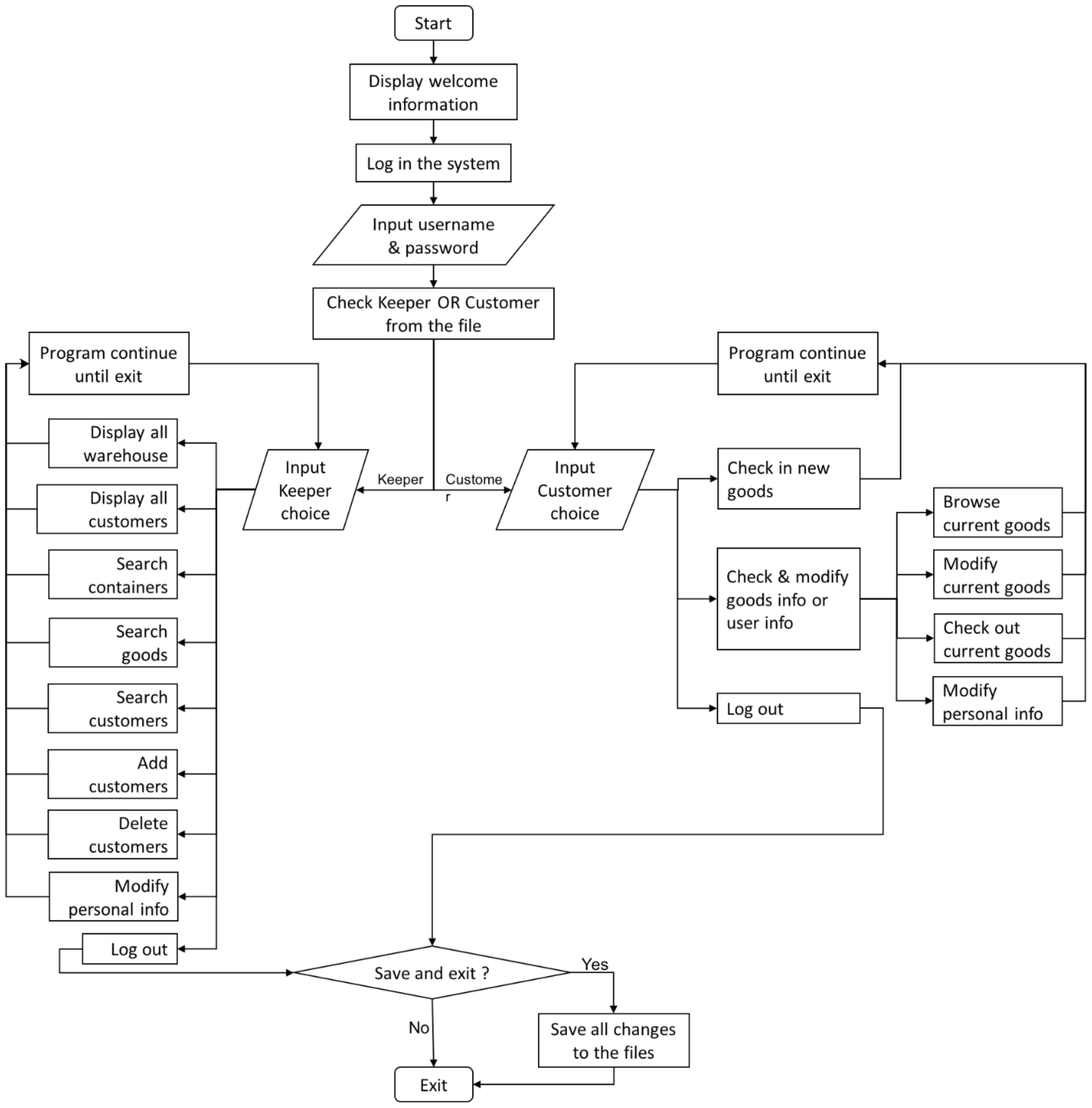


Figure 18: Flow chart

Testing

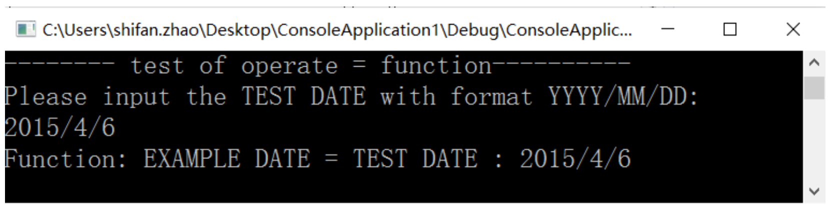
Note that the dependency of classes may the test much difficult. As a result, the test is done by groups rather than individual classes.

Time class test (by Zhaoshi Fan)

There are six main member functions in the Time class.

All of following functions are test in an independent main function

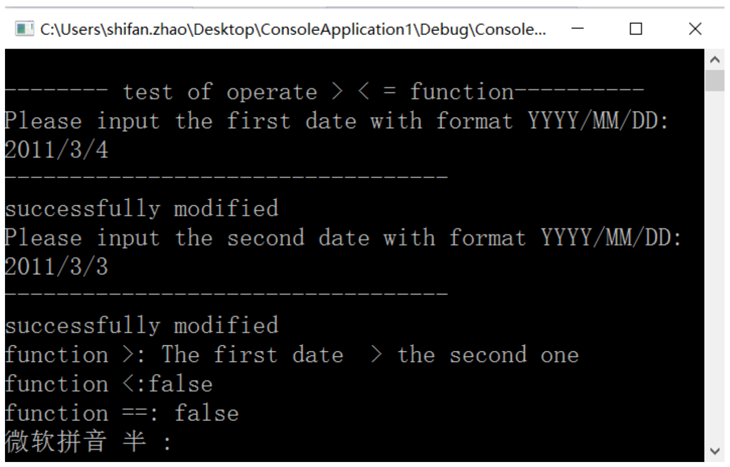
(1) operator=(Time t): It is a void public function that can make the one member be assigned to the other one.



```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\ConsoleApplic...
-------- test of operate = function----------
Please input the TEST DATE with format YYYY/MM/DD:
2015/4/6
Function: EXAMPLE DATE = TEST DATE : 2015/4/6
```

Figure 19: Time test for = operator

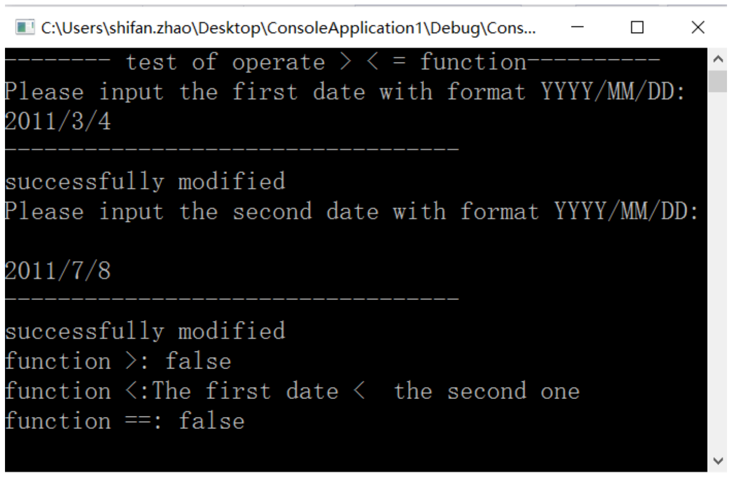
(2) operator >(Time t): It is a bool public function that only returns true when the time is bigger than the compared one.



```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\Console...

-------- test of operate > < = function----------
Please input the first date with format YYYY/MM/DD:
2011/3/4
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:
2011/3/3
----------------------------------
successfully modified
function >: The first date  > the second one
function <:false
function ==: false
微软拼音 半 :
```

Figure 20: Time test for > operator

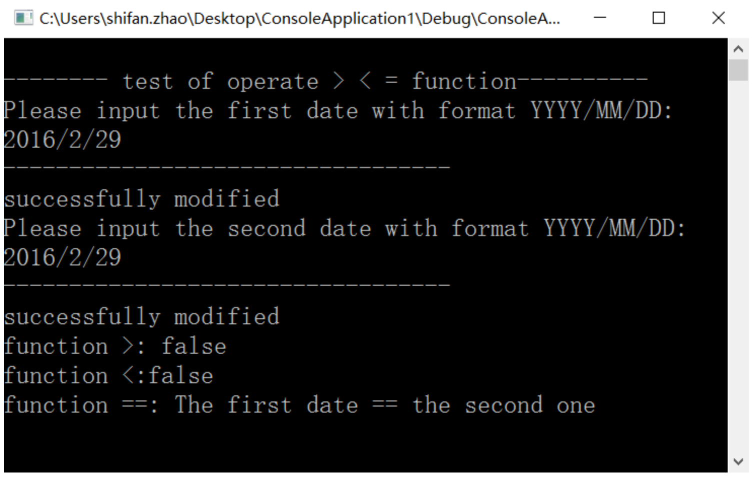
(3) operator<(Time t): It is a bool public function that only return true when the time is smaller than the compared one.



```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\Cons...
-------- test of operate > < = function----------
Please input the first date with format YYYY/MM/DD:
2011/3/4
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:

2011/7/8
----------------------------------
successfully modified
function >: false
function <:The first date <  the second one
function ==: false
```

Figure 21: Time test for < operator

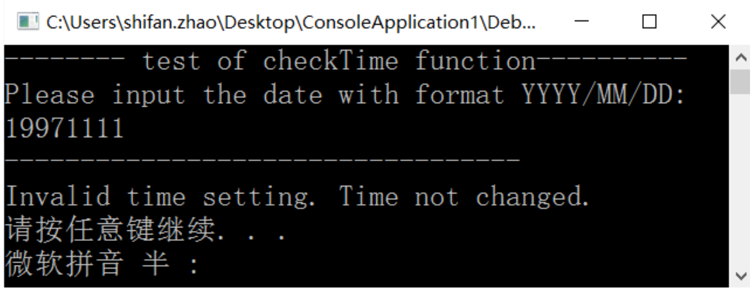
(4) operator==(Time t): It is a bool public function that only returns true when the time is the same with the compared one.



```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\ConsoleA...
-------- test of operate > < = function----------
Please input the first date with format YYYY/MM/DD:
2016/2/29
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:
2016/2/29
----------------------------------
successfully modified
function >: false
function <:false
function ==: The first date == the second one
```

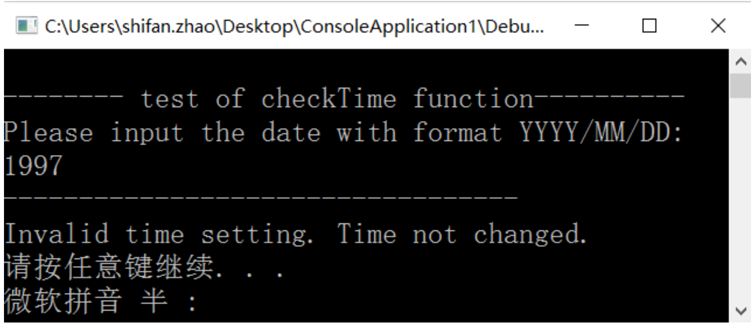
Figure 22: Time test for == operator

(5) checkTime(string s): It is a bool public function that whether the date inputted by the user is right.



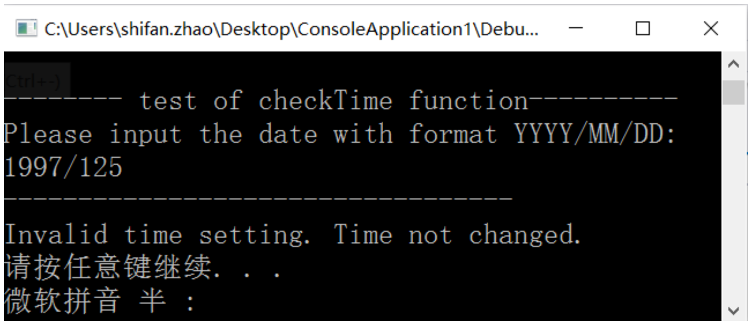
```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Deb...
-------- test of checkTime function----------
Please input the date with format YYYY/MM/DD:
19971111
----------------------------------
Invalid time setting. Time not changed.
请按任意键继续. . .
微软拼音 半 :
```

Figure 23: Time test for invalid input 1



```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debu...
-------- test of checkTime function----------
Please input the date with format YYYY/MM/DD:
1997
----------------------------------
Invalid time setting. Time not changed.
请按任意键继续. . .
微软拼音 半 :
```

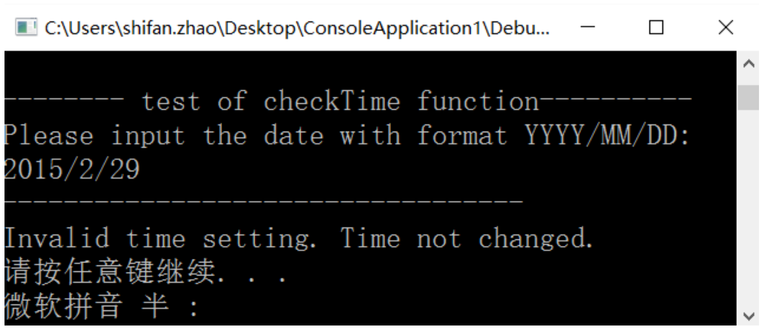
Figure 24: Time test for invalid input 2



```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debu...
-------- test of checkTime function----------
Please input the date with format YYYY/MM/DD:
1997/125
-----------------------------------

Invalid time setting. Time not changed.
请按任意键继续. . .
微软拼音 半 :
```

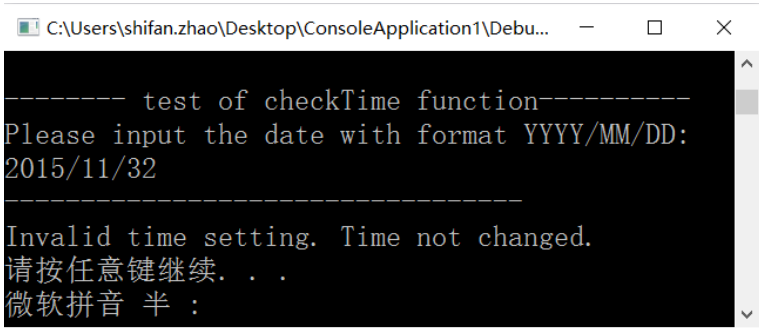
Figure 25: Time test for invalid input 3

Test above shows that this function succeeded in rejecting invalid input.



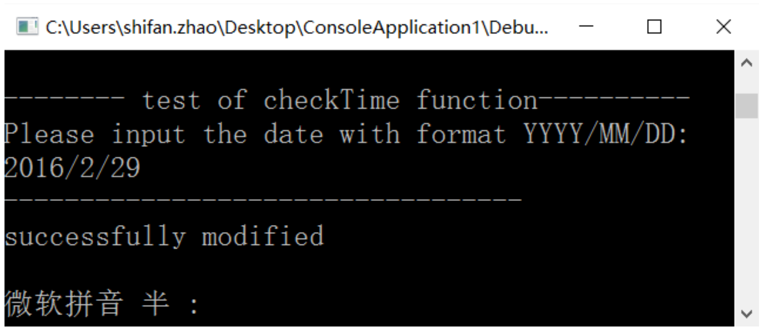
```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debu...
-------- test of checkTime function----------
Please input the date with format YYYY/MM/DD:
2015/2/29
-----------------------------------

Invalid time setting. Time not changed.
请按任意键继续. . .
微软拼音 半 :
```

Figure 26: Time test for invalid date 1



```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debu...
-------- test of checkTime function----------
Please input the date with format YYYY/MM/DD:
2015/11/32
-----------------------------------

Invalid time setting. Time not changed.
请按任意键继续. . .
微软拼音 半 :
```

Figure 27: Time test for invalid date 2

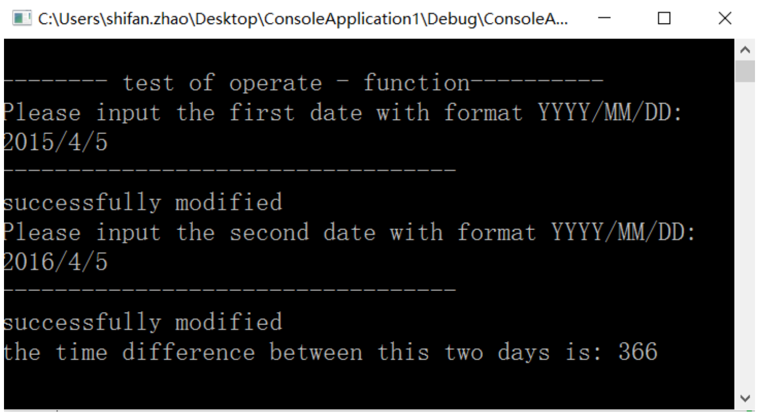


```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debu...
-------- test of checkTime function----------
Please input the date with format YYYY/MM/DD:
2016/2/29
-----------------------------------

successfully modified

微软拼音 半 :
```

Figure 28: Time test for valid date

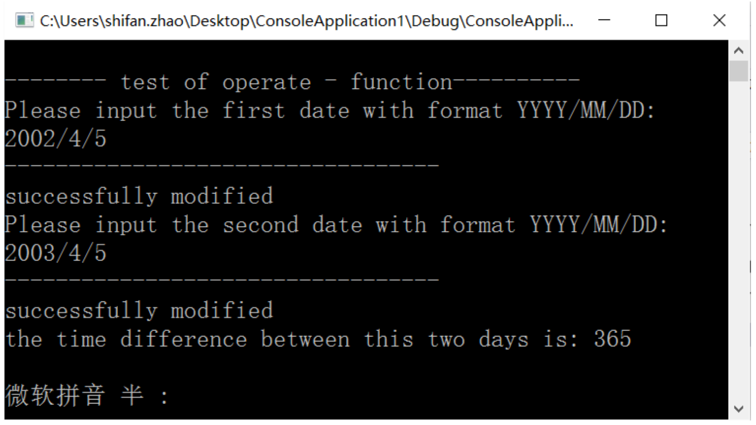
The rest three tests showed that this function succeeded in dealing with wrong date input.

(6) operator-(Timer &t): It is an int public function that can calculate the time deference between the two inputted dates.



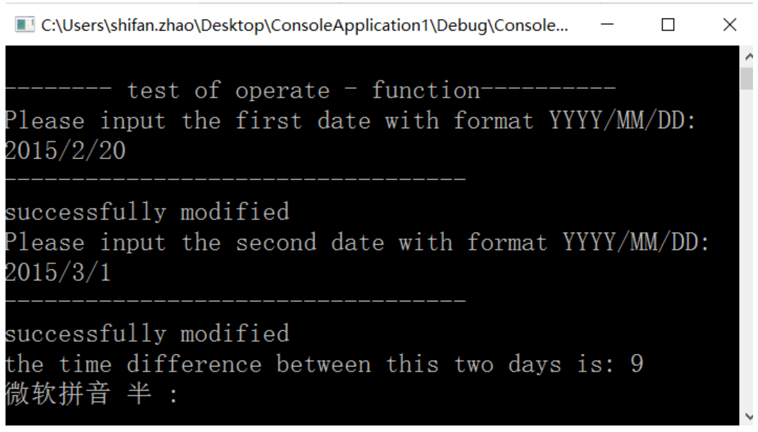
```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\ConsoleA...
-------- test of operate - function----------
Please input the first date with format YYYY/MM/DD:
2015/4/5
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:
2016/4/5
----------------------------------
successfully modified
the time difference between this two days is: 366
```

Figure 29: Time test for - operator (leap year)



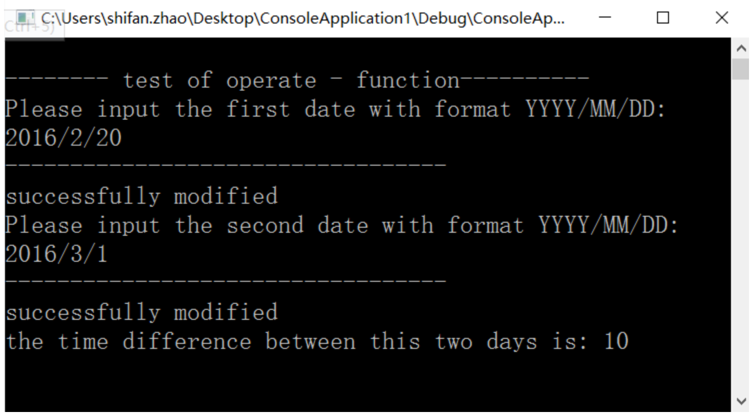
```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\ConsoleAppli...
-------- test of operate - function----------
Please input the first date with format YYYY/MM/DD:
2002/4/5
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:
2003/4/5
----------------------------------
successfully modified
the time difference between this two days is: 365

微软拼音 半 ：
```

Figure 30: Time test for - operator (normal year)



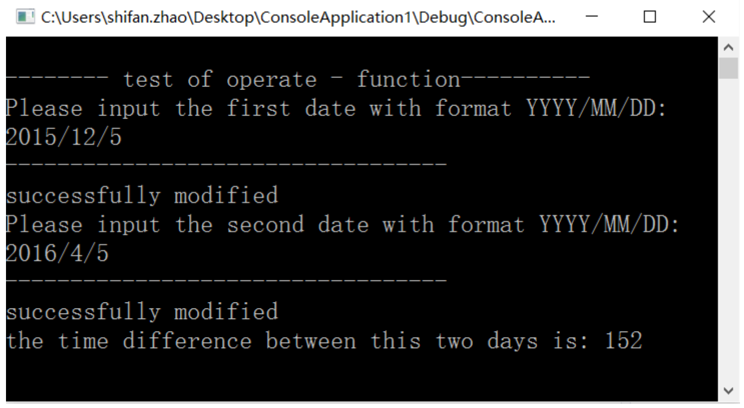
```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\Console...
-------- test of operate - function----------
Please input the first date with format YYYY/MM/DD:
2015/2/20
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:
2015/3/1
----------------------------------
successfully modified
the time difference between this two days is: 9
微软拼音 半 ：
```

Figure 31: Time test for - operator (normal year February)



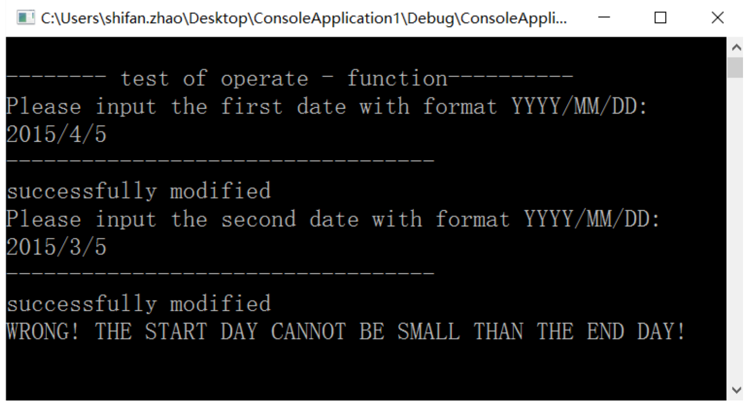
```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\ConsoleAp...
-------- test of operate - function----------
Please input the first date with format YYYY/MM/DD:
2016/2/20
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:
2016/3/1
----------------------------------
successfully modified
the time difference between this two days is: 10
```

Figure 32: Time test for - operator (leap year February) 1



```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\ConsoleA...
-------- test of operate - function----------
Please input the first date with format YYYY/MM/DD:
2015/12/5
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:
2016/4/5
----------------------------------
successfully modified
the time difference between this two days is: 152
```

Figure 33: Time test for - operator (leap year February) 2



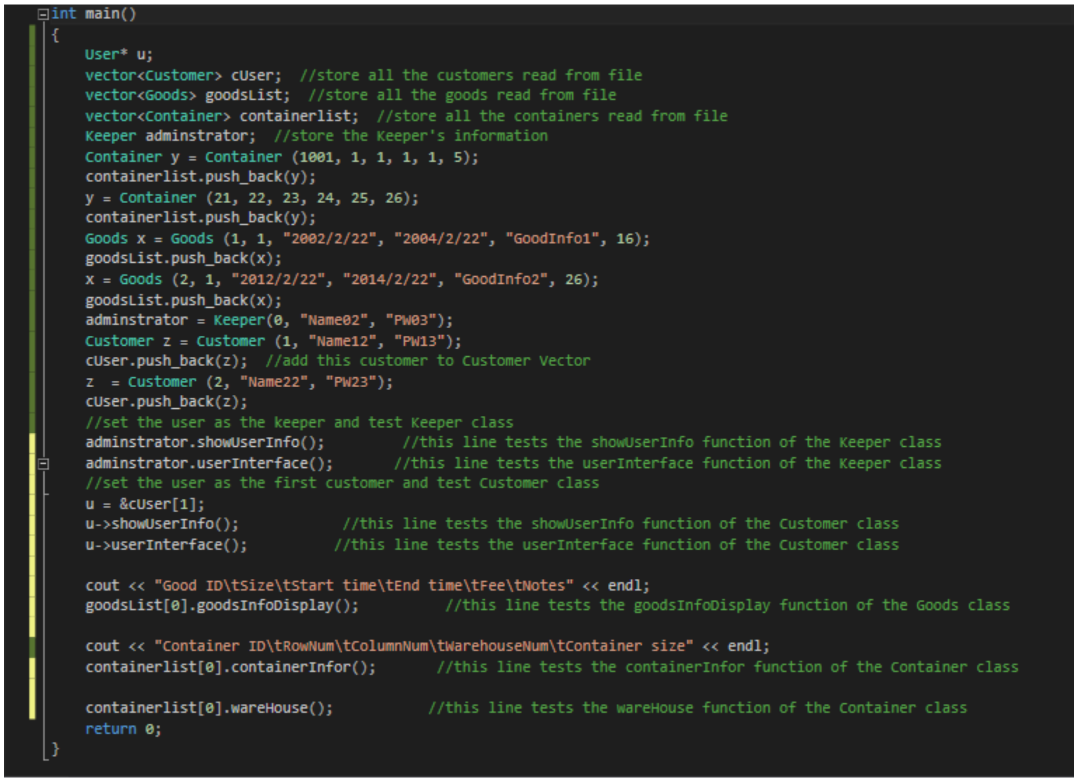
```
C:\Users\shifan.zhao\Desktop\ConsoleApplication1\Debug\ConsoleAppli...
-------- test of operate - function----------
Please input the first date with format YYYY/MM/DD:
2015/4/5
----------------------------------
successfully modified
Please input the second date with format YYYY/MM/DD:
2015/3/5
----------------------------------
successfully modified
WRONG! THE START DAY CANNOT BE SMALL THAN THE END DAY!
```

Figure 34: Time test for - operator (invalid minus)

The – operator was tested successfully. Note that the condition shown in Figure 34 will not happen because specific function is designed to reject a smaller second date.

Container, Goods, Customer and keeper class test (by Zhaozhong Wu)

The container, goods and keeper classes are tested by a test.cpp file, the code is shown below,



```
int main()
{
    User* u;
    vector<Customer> cUser;  //store all the customers read from file
    vector<Goods> goodsList;  //store all the goods read from file
    vector<Container> containerlist;  //store all the containers read from file
    Keeper adminstrator;  //store the Keeper's information
    Container y = Container (1001, 1, 1, 1, 1, 5);
    containerlist.push_back(y);
    y = Container (21, 22, 23, 24, 25, 26);
    containerlist.push_back(y);
    Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
    goodsList.push_back(x);
    x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
    goodsList.push_back(x);
    adminstrator = Keeper(0, "Name02", "PW03");
    Customer z = Customer (1, "Name12", "PW13");
    cUser.push_back(z);  //add this customer to Customer Vector
    z  = Customer (2, "Name22", "PW23");
    cUser.push_back(z);
    //set the user as the keeper and test Keeper class
    adminstrator.showUserInfo();         //this line tests the showUserInfo function of the Keeper class
    adminstrator.userInterface();       //this line tests the userInterface function of the Keeper class
    //set the user as the first customer and test Customer class
    u = &cUser[1];
    u->showUserInfo();           //this line tests the showUserInfo function of the Customer class
    u->userInterface();          //this line tests the userInterface function of the Customer class

    cout << "Good ID\tSize\tStart time\tEnd time\tFee\tNotes" << endl;
    goodsList[0].goodsInfoDisplay();          //this line tests the goodsInfoDisplay function of the Goods class

    cout << "Container ID\tRowNum\tColumnNum\tWarehouseNum\tContainer size" << endl;
    containerlist[0].containerInfor();       //this line tests the containerInfor function of the Container class

    containerlist[0].wareHouse();           //this line tests the wareHouse function of the Container class
    return 0;
}
```

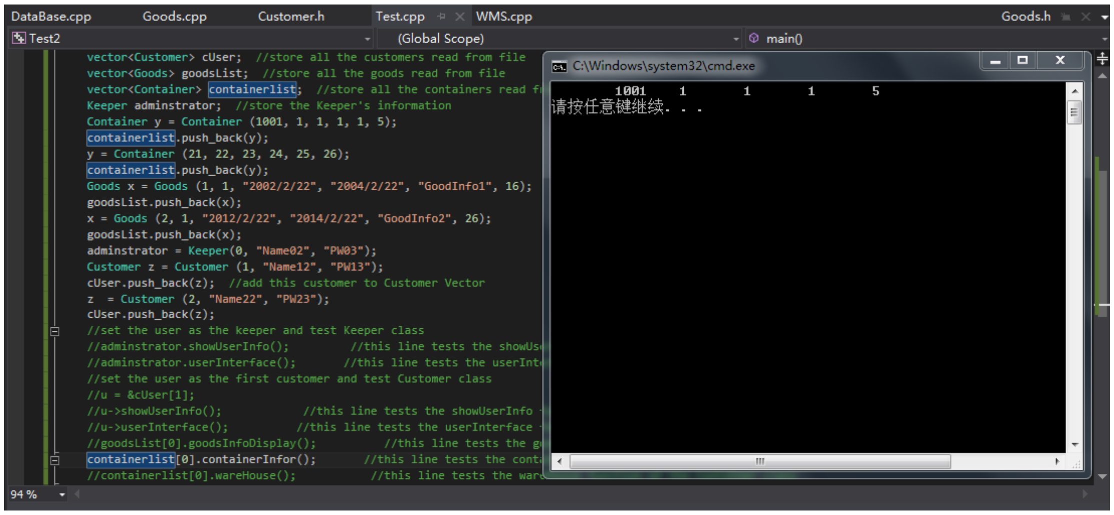
By setting up all the needed vectors in the main function, the four class is tested separated from the whole program.

Container

There are two member functions in the Container class:

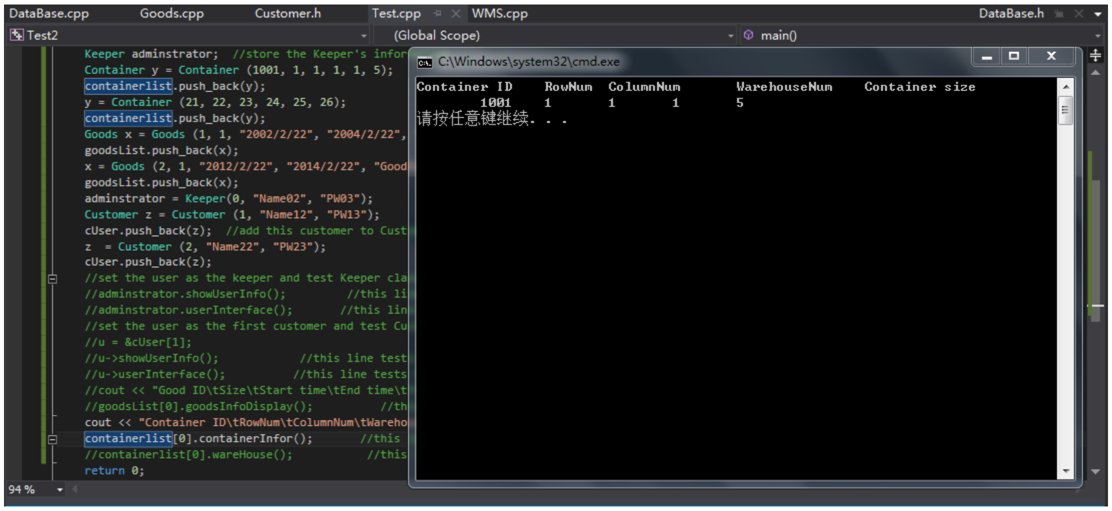
(1) containerInfor ()

It is a void public function that displays the ID, the coordinates, warehouse number, and the size of the container. It is called in the function named searchGoods in the database class. The test is successful.



```
vector<Customer> cUser;  //store all the customers read from file
vector<Goods> goodsList;  //store all the goods read from file
vector<Container> containerlist;  //store all the containers read fr
Keeper adminstrator;  //store the Keeper's information
Container y = Container (1001, 1, 1, 1, 1, 5);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Customer Vector
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper class
//adminstrator.showUserInfo();         //this line tests the showUs
//adminstrator.userInterface();       //this line tests the userInt
//set the user as the first customer and test Customer class
//u = &cUser[1];
//u->showUserInfo();           //this line tests the showUserInfo
//u->userInterface();          //this line tests the userInterface
//goodsList[0].goodsInfoDisplay();          //this line tests the g
containerlist[0].containerInfor();       //this line tests the cont
//containerlist[0].wareHouse();           //this line tests the war
```

```
        1001    1       1       1       5
请按任意键继续. . .
```

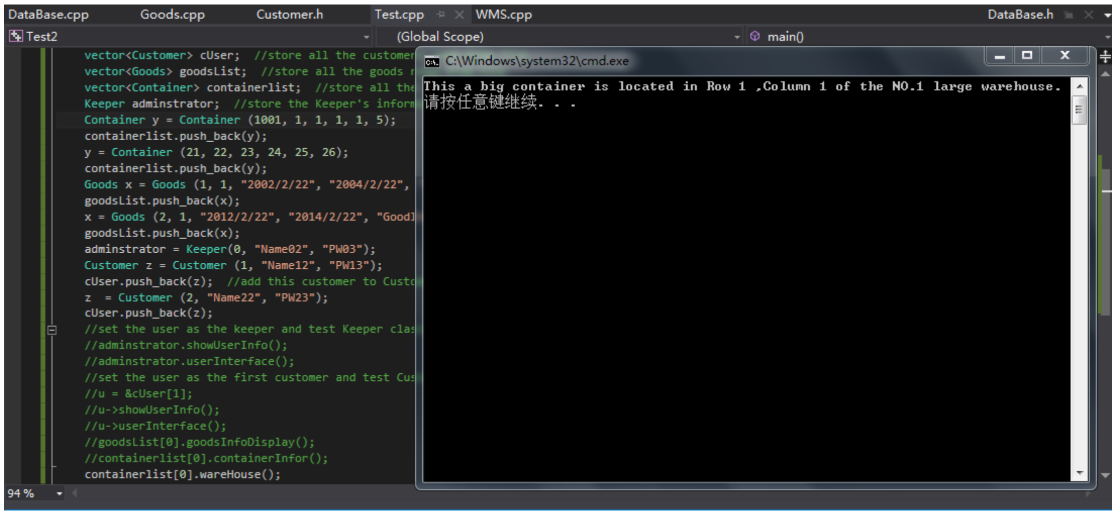


```
Keeper adminstrator;  //store the Keeper's infor
Container y = Container (1001, 1, 1, 1, 1, 5);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22",
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "Good
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Cust
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper cla
//adminstrator.showUserInfo();         //this li
//adminstrator.userInterface();       //this lin
//set the user as the first customer and test Cu
//u = &cUser[1];
//u->showUserInfo();            //this line test
//u->userInterface();          //this line tests
//cout << "Good ID\tSize\tStart time\tEnd time\t
//goodsList[0].goodsInfoDisplay();          //th
cout << "Container ID\tRowNum\tColumnNum\tWareho
containerlist[0].containerInfor();        //this
//containerlist[0].wareHouse();          //this
return 0;
```

```
Container ID    RowNum  ColumnNum       WarehouseNum    Container size
        1001    1       1       1       5
请按任意键继续. . .
```

(2) warehouse()

This is a void public function that displays the information about which warehouse the container in and whether the container is small or large. In the full program It is called in the searchContainer function in the database class. During the test the function runs well.



```
vector<Customer> cUser;  //store all the customer
vector<Goods> goodsList;  //store all the goods r
vector<Container> containerlist;  //store all the
Keeper adminstrator;  //store the Keeper's inform
Container y = Container (1001, 1, 1, 1, 1, 5);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22",
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "Good1
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Custo
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper clas
//adminstrator.showUserInfo();
//adminstrator.userInterface();
//set the user as the first customer and test Cus
//u = &cUser[1];
//u->showUserInfo();
//u->userInterface();
//goodsList[0].goodsInfoDisplay();
//containerlist[0].containerInfor();
containerlist[0].wareHouse();
```

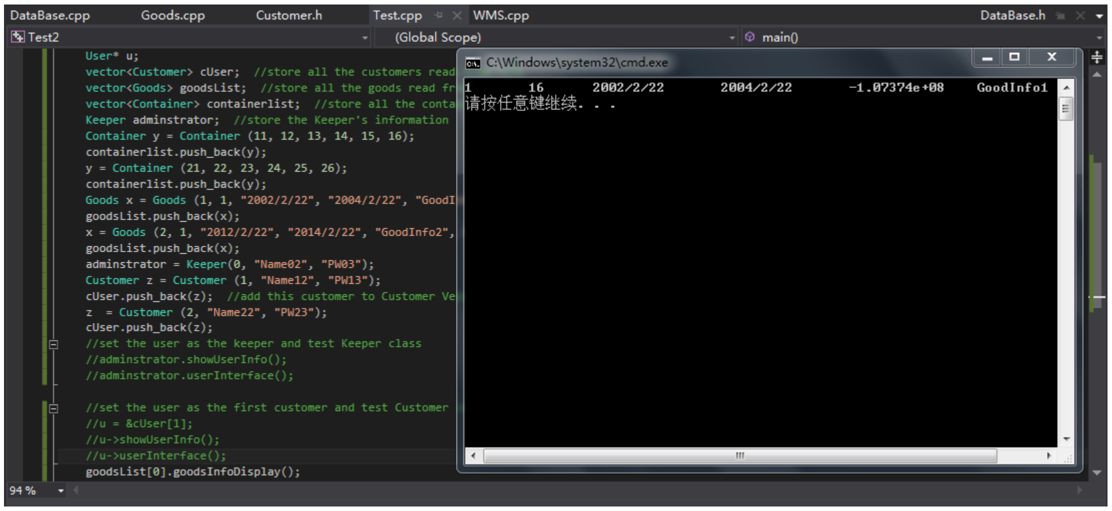
```
This a big container is located in Row 1 ,Column 1 of the NO.1 large warehouse.
请按任意键继续. . .
```

Goods

There is one member function in the Goods class:

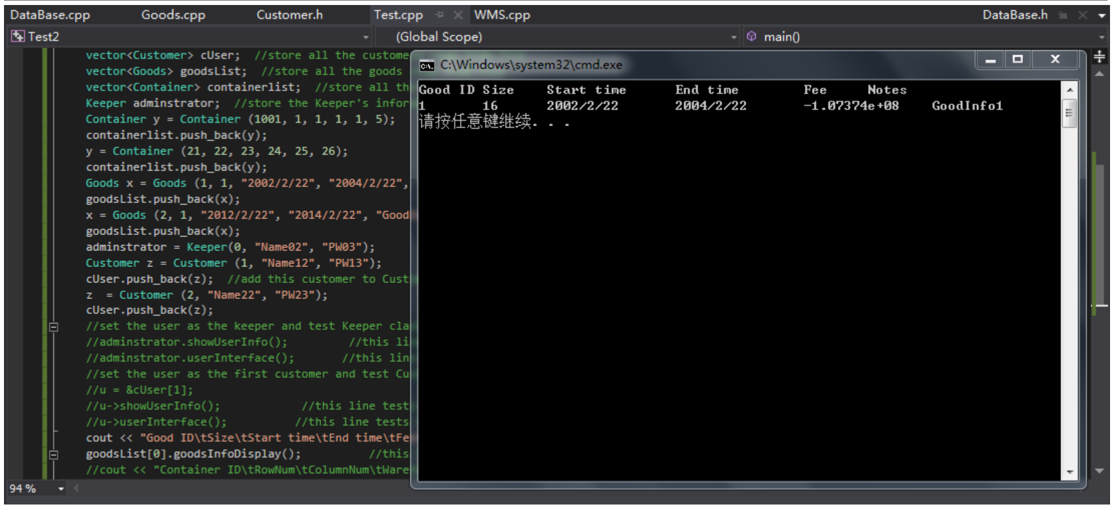
(1) goodsInfoDisplay ()

It is a void public function that displays the ID, size, start time, end time, and other information of specific goods. It is called from the Customer class. In the test this function ran well.



```
User* u;
vector<Customer> cUser;  //store all the customers read
vector<Goods> goodsList;  //store all the goods read fr
vector<Container> containerlist;  //store all the conta
Keeper adminstrator;  //store the Keeper's information
Container y = Container (11, 12, 13, 14, 15, 16);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodI
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2",
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Customer Ve
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper class
//adminstrator.showUserInfo();
//adminstrator.userInterface();

//set the user as the first customer and test Customer
//u = &cUser[1];
//u->showUserInfo();
//u->userInterface();
goodsList[0].goodsInfoDisplay();
```

```
1        16       2002/2/22        2004/2/22        -1.07374e+08     GoodInfo1
请按任意键继续. . .
```



```
vector<Customer> cUser;  //store all the custome
vector<Goods> goodsList;  //store all the goods
vector<Container> containerlist;  //store all th
Keeper adminstrator;  //store the Keeper's infor
Container y = Container (1001, 1, 1, 1, 1, 5);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22",
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "Good
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Cust
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper cla
//adminstrator.showUserInfo();         //this li
//adminstrator.userInterface();       //this lin
//set the user as the first customer and test Cu
//u = &cUser[1];
//u->showUserInfo();           //this line test
//u->userInterface();          //this line tests
cout << "Good ID\tSize\tStart time\tEnd time\tFe
goodsList[0].goodsInfoDisplay();         //this
//cout << "Container ID\tRowNum\tColumnNum\tWare
```

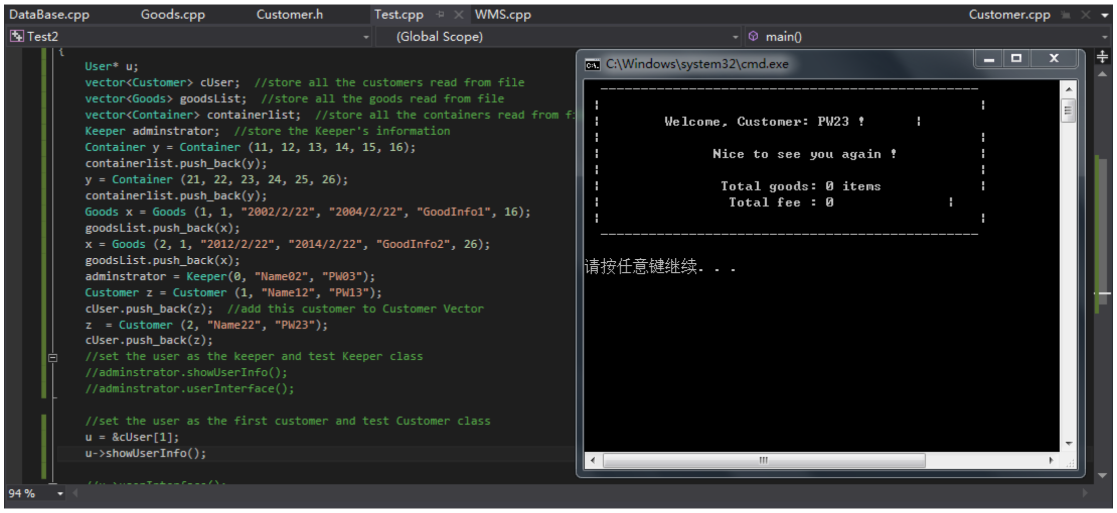
```
Good ID Size     Start time       End time         Fee      Notes
1        16       2002/2/22        2004/2/22        -1.07374e+08     GoodInfo1
请按任意键继续. . .
```

Customer

There are two member functions in the Customer class:

(1) showUserInfo()

It is a void public function that displays an interface before showing the information of the user. During the test, the program displays the interface of a customer, and the result of this test is successful.

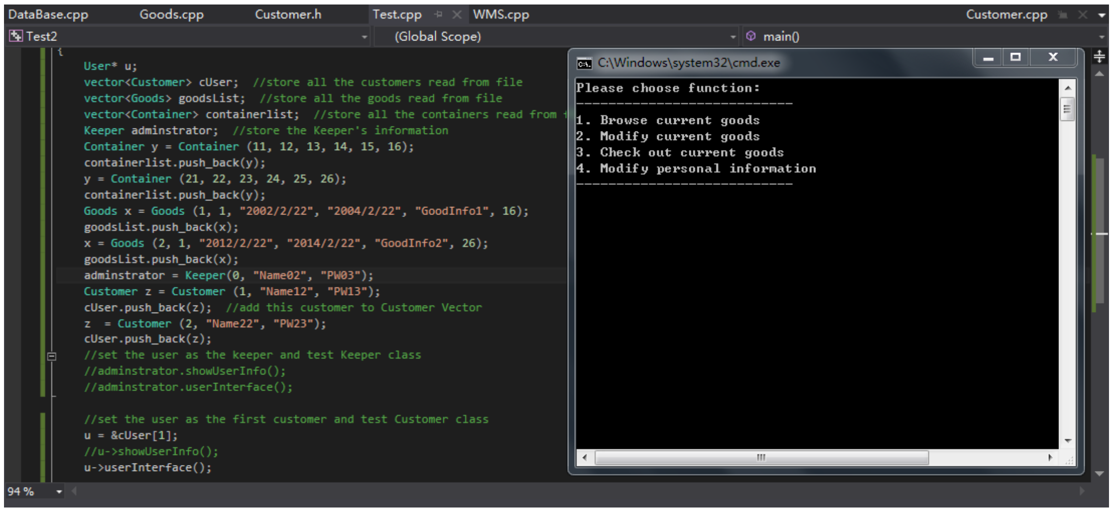


```
User* u;
vector<Customer> cUser;  //store all the customers read from file
vector<Goods> goodsList;  //store all the goods read from file
vector<Container> containerlist;  //store all the containers read from f
Keeper adminstrator;  //store the Keeper's information
Container y = Container (11, 12, 13, 14, 15, 16);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Customer Vector
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper class
//adminstrator.showUserInfo();
//adminstrator.userInterface();

//set the user as the first customer and test Customer class
u = &cUser[1];
u->showUserInfo();
```

```
 ------------------------------------------------
¦                                                ¦
¦        Welcome, Customer: PW23 !       ¦
¦                                                ¦
¦              Nice to see you again !           ¦
¦                                                ¦
¦               Total goods: 0 items             ¦
¦                Total fee : 0               ¦
¦                                                ¦
 ------------------------------------------------

请按任意键继续. . .
```

(2) UserInterface()

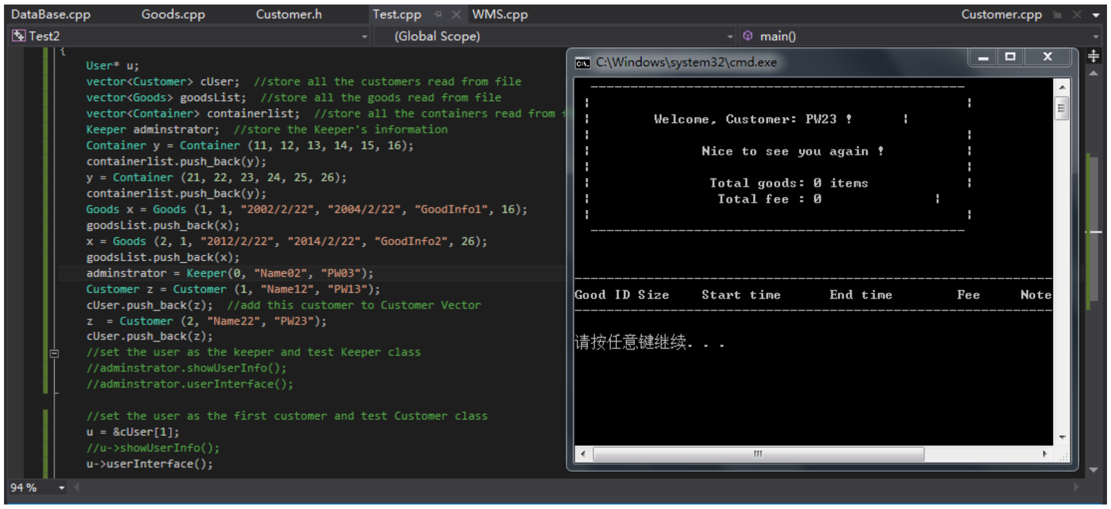
This is a function that allows the user to choose the private functions in the Customer class to display goods, modify goods, delete goods or edit the account information.



```
User* u;
vector<Customer> cUser;  //store all the customers read from file
vector<Goods> goodsList;  //store all the goods read from file
vector<Container> containerlist;  //store all the containers read from
Keeper adminstrator;  //store the Keeper's information
Container y = Container (11, 12, 13, 14, 15, 16);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Customer Vector
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper class
//adminstrator.showUserInfo();
//adminstrator.userInterface();

//set the user as the first customer and test Customer class
u = &cUser[1];
//u->showUserInfo();
u->userInterface();
```

```
Please choose function:
--------------------------------
1. Browse current goods
2. Modify current goods
3. Check out current goods
4. Modify personal information
--------------------------------
```

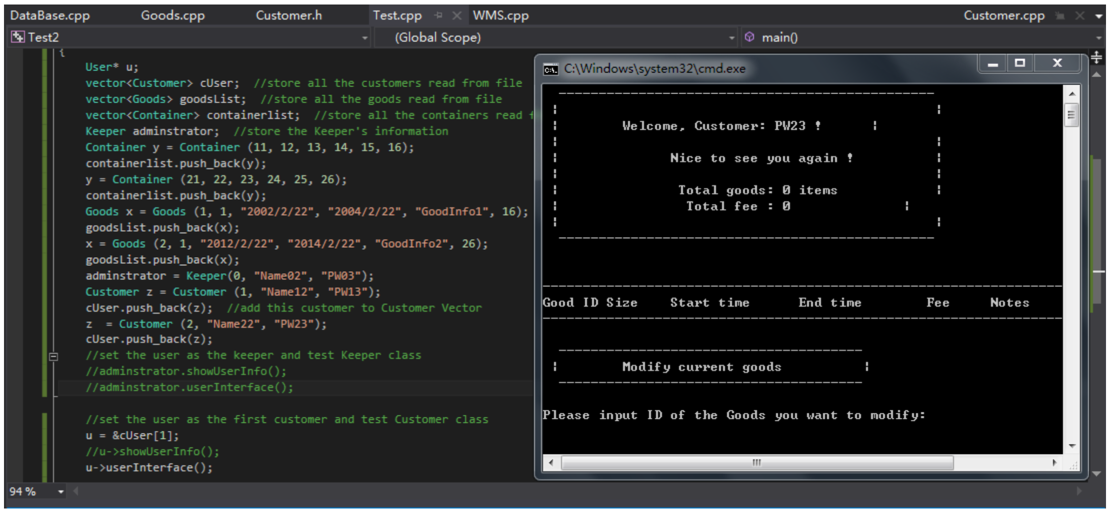
The first function of displaying goods is successfully tested as shown below,



```
User* u;
vector<Customer> cUser;  //store all the customers read from file
vector<Goods> goodsList;  //store all the goods read from file
vector<Container> containerlist;  //store all the containers read from
Keeper adminstrator;  //store the Keeper's information
Container y = Container (11, 12, 13, 14, 15, 16);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Customer Vector
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper class
//adminstrator.showUserInfo();
//adminstrator.userInterface();

//set the user as the first customer and test Customer class
u = &cUser[1];
//u->showUserInfo();
u->userInterface();
```

```
 ------------------------------------------------
¦                                                ¦
¦        Welcome, Customer: PW23 !       ¦
¦                                                ¦
¦              Nice to see you again !           ¦
¦                                                ¦
¦               Total goods: 0 items             ¦
¦                Total fee : 0               ¦
¦                                                ¦
 ------------------------------------------------


------------------------------------------------------------
Good ID Size    Start time      End time        Fee     Note
------------------------------------------------------------

请按任意键继续. . .
```

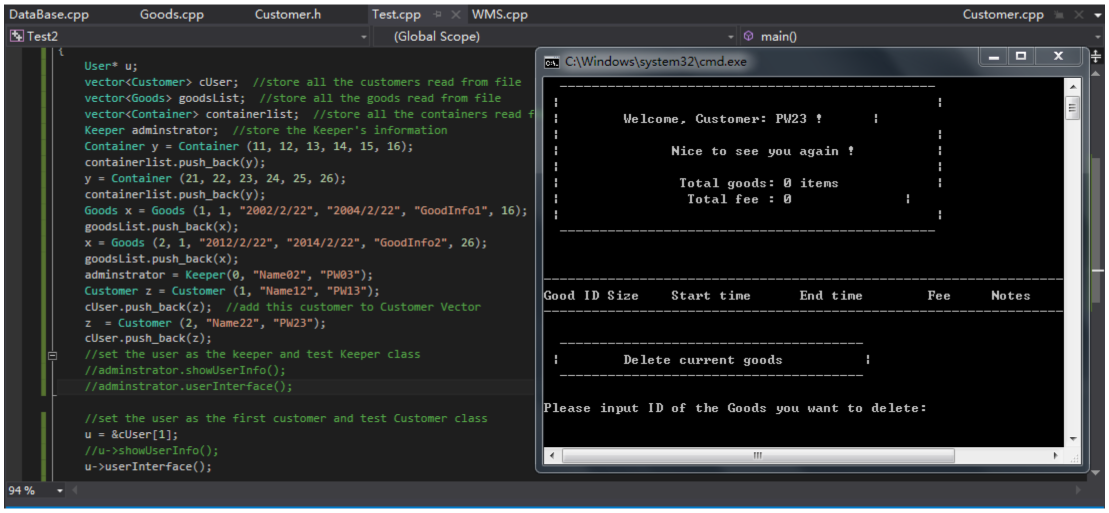
The second function of modifying goods is successfully tested as shown below,



```
{
    User* u;
    vector<Customer> cUser;  //store all the customers read from file
    vector<Goods> goodsList;  //store all the goods read from file
    vector<Container> containerlist;  //store all the containers read f
    Keeper adminstrator;  //store the Keeper's information
    Container y = Container (11, 12, 13, 14, 15, 16);
    containerlist.push_back(y);
    y = Container (21, 22, 23, 24, 25, 26);
    containerlist.push_back(y);
    Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
    goodsList.push_back(x);
    x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
    goodsList.push_back(x);
    adminstrator = Keeper(0, "Name02", "PW03");
    Customer z = Customer (1, "Name12", "PW13");
    cUser.push_back(z);  //add this customer to Customer Vector
    z  = Customer (2, "Name22", "PW23");
    cUser.push_back(z);
    //set the user as the keeper and test Keeper class
    //adminstrator.showUserInfo();
    //adminstrator.userInterface();

    //set the user as the first customer and test Customer class
    u = &cUser[1];
    //u->showUserInfo();
    u->userInterface();
```

```
 ------------------------------------------------------
|                                                      |
|          Welcome, Customer: PW23 !        |
|                                                      |
|                Nice to see you again !               |
|                                                      |
|                 Total goods: 0 items                 |
|                  Total fee : 0               |
|                                                      |
 ------------------------------------------------------


----------------------------------------------------------------
Good ID Size    Start time        End time         Fee      Notes
----------------------------------------------------------------

  ------------------------------------------
 |          Modify current goods            |
  ------------------------------------------

Please input ID of the Goods you want to modify:
```

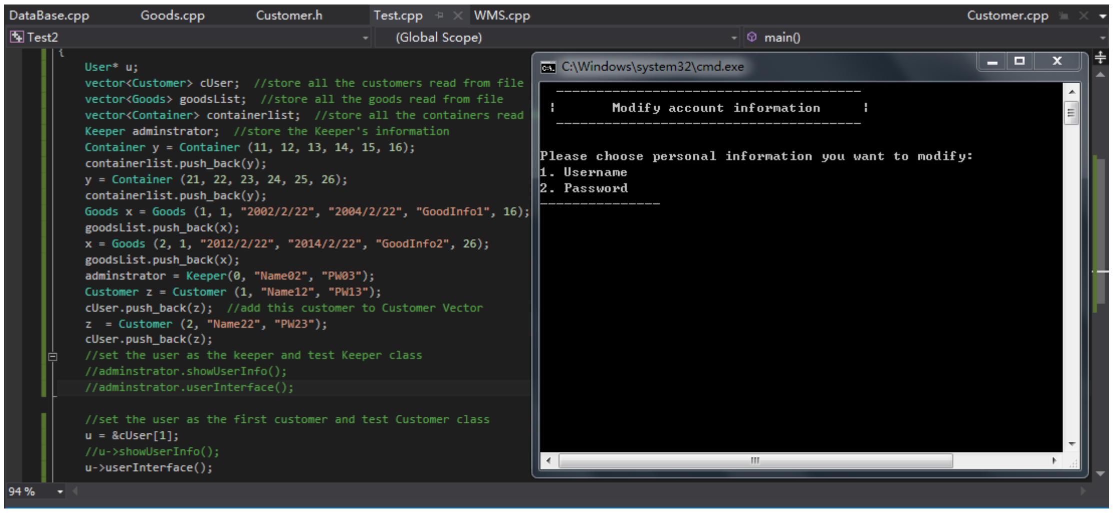
The third function of deleting goods is successfully tested as shown below,



```
{
    User* u;
    vector<Customer> cUser;  //store all the customers read from file
    vector<Goods> goodsList;  //store all the goods read from file
    vector<Container> containerlist;  //store all the containers read f
    Keeper adminstrator;  //store the Keeper's information
    Container y = Container (11, 12, 13, 14, 15, 16);
    containerlist.push_back(y);
    y = Container (21, 22, 23, 24, 25, 26);
    containerlist.push_back(y);
    Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
    goodsList.push_back(x);
    x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
    goodsList.push_back(x);
    adminstrator = Keeper(0, "Name02", "PW03");
    Customer z = Customer (1, "Name12", "PW13");
    cUser.push_back(z);  //add this customer to Customer Vector
    z  = Customer (2, "Name22", "PW23");
    cUser.push_back(z);
    //set the user as the keeper and test Keeper class
    //adminstrator.showUserInfo();
    //adminstrator.userInterface();

    //set the user as the first customer and test Customer class
    u = &cUser[1];
    //u->showUserInfo();
    u->userInterface();
```

```
 ------------------------------------------------------
|                                                      |
|          Welcome, Customer: PW23 !        |
|                                                      |
|                Nice to see you again !               |
|                                                      |
|                 Total goods: 0 items                 |
|                  Total fee : 0               |
|                                                      |
 ------------------------------------------------------


----------------------------------------------------------------
Good ID Size    Start time        End time         Fee      Notes
----------------------------------------------------------------

  ------------------------------------------
 |          Delete current goods            |
  ------------------------------------------

Please input ID of the Goods you want to delete:
```

The fourth function of editing customer infomation is successfully tested as shown below,



```
{
    User* u;
    vector<Customer> cUser;  //store all the customers read from file
    vector<Goods> goodsList;  //store all the goods read from file
    vector<Container> containerlist;  //store all the containers read
    Keeper adminstrator;  //store the Keeper's information
    Container y = Container (11, 12, 13, 14, 15, 16);
    containerlist.push_back(y);
    y = Container (21, 22, 23, 24, 25, 26);
    containerlist.push_back(y);
    Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
    goodsList.push_back(x);
    x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
    goodsList.push_back(x);
    adminstrator = Keeper(0, "Name02", "PW03");
    Customer z = Customer (1, "Name12", "PW13");
    cUser.push_back(z);  //add this customer to Customer Vector
    z  = Customer (2, "Name22", "PW23");
    cUser.push_back(z);
    //set the user as the keeper and test Keeper class
    //adminstrator.showUserInfo();
    //adminstrator.userInterface();

    //set the user as the first customer and test Customer class
    u = &cUser[1];
    //u->showUserInfo();
    u->userInterface();
```

```
  ------------------------------------------
 |        Modify account information        |
  ------------------------------------------

Please choose personal information you want to modify:
1. Username
2. Password
---------------
```

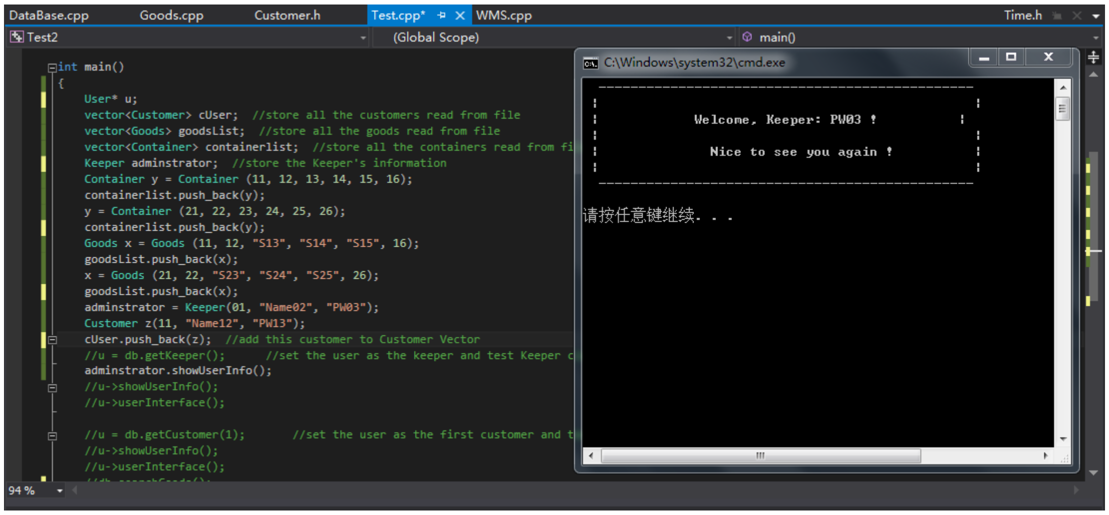
Invalid input was not accepted.

Keeper

There are two member functions in the Keeper class:

(1) showUserInfo()

It is a void public function that displays an interface before showing the information of the user, which is the keeper in this case, the test run of this function is successful.

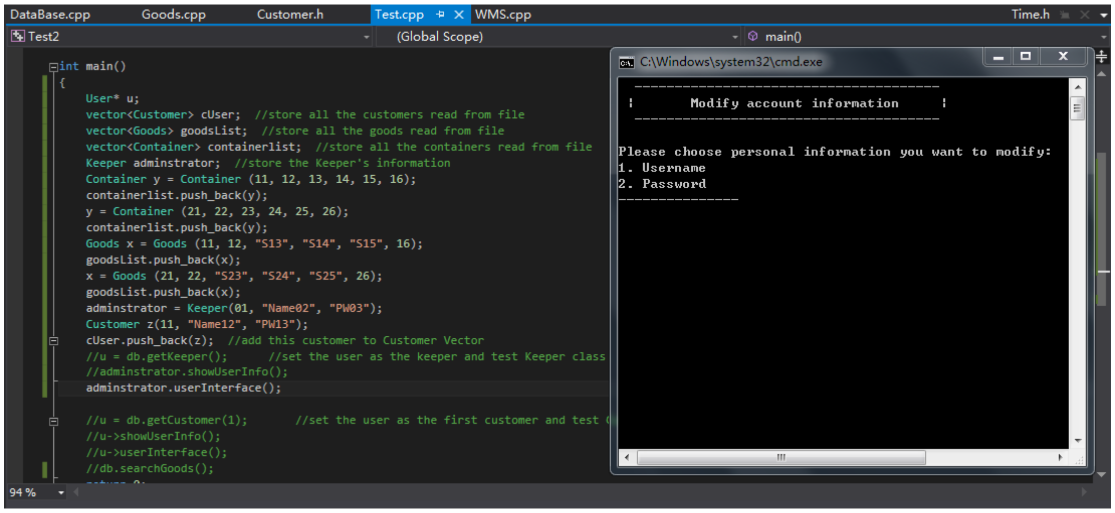


```
int main()
{
    User* u;
    vector<Customer> cUser;  //store all the customers read from file
    vector<Goods> goodsList;  //store all the goods read from file
    vector<Container> containerlist;  //store all the containers read from fi
    Keeper adminstrator;  //store the Keeper's information
    Container y = Container (11, 12, 13, 14, 15, 16);
    containerlist.push_back(y);
    y = Container (21, 22, 23, 24, 25, 26);
    containerlist.push_back(y);
    Goods x = Goods (11, 12, "S13", "S14", "S15", 16);
    goodsList.push_back(x);
    x = Goods (21, 22, "S23", "S24", "S25", 26);
    goodsList.push_back(x);
    adminstrator = Keeper(01, "Name02", "PW03");
    Customer z(11, "Name12", "PW13");
    cUser.push_back(z);  //add this customer to Customer Vector
    //u = db.getKeeper();      //set the user as the keeper and test Keeper c
    adminstrator.showUserInfo();
    //u->showUserInfo();
    //u->userInterface();

    //u = db.getCustomer(1);       //set the user as the first customer and t
    //u->showUserInfo();
    //u->userInterface();
```

```
 ---------------------------------------------
 |                                           |
 |        Welcome, Keeper: PW03 !          |
 |                                           |
 |          Nice to see you again !          |
 |                                           |
 ---------------------------------------------

请按任意键继续. . .
```

(2) UserInterface()

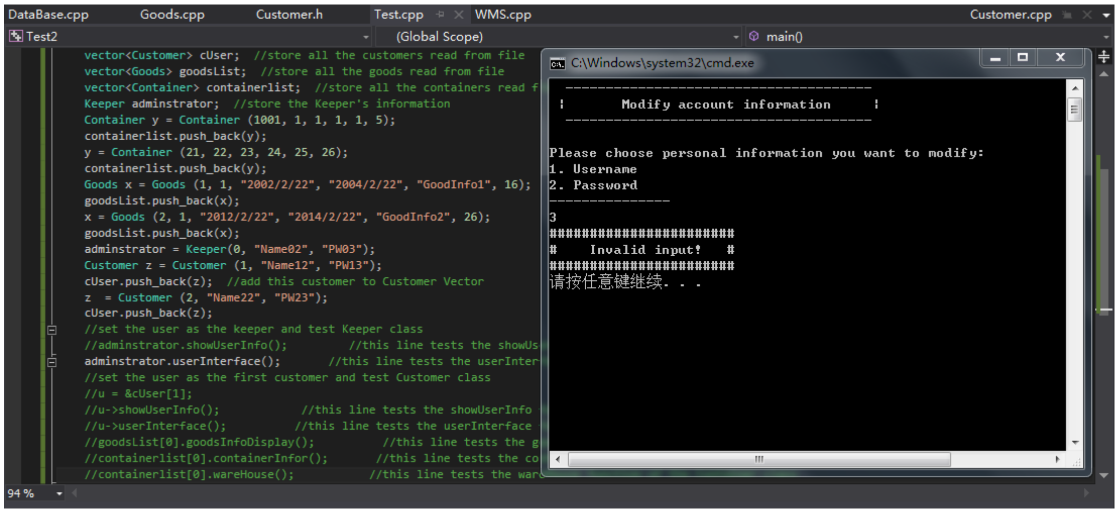
This is a function that calls the function named editInfo in the base class User to edit the information of the users within the database. The test of it called the function successfully.



```
int main()
{
    User* u;
    vector<Customer> cUser;  //store all the customers read from file
    vector<Goods> goodsList;  //store all the goods read from file
    vector<Container> containerlist;  //store all the containers read from file
    Keeper adminstrator;  //store the Keeper's information
    Container y = Container (11, 12, 13, 14, 15, 16);
    containerlist.push_back(y);
    y = Container (21, 22, 23, 24, 25, 26);
    containerlist.push_back(y);
    Goods x = Goods (11, 12, "S13", "S14", "S15", 16);
    goodsList.push_back(x);
    x = Goods (21, 22, "S23", "S24", "S25", 26);
    goodsList.push_back(x);
    adminstrator = Keeper(01, "Name02", "PW03");
    Customer z(11, "Name12", "PW13");
    cUser.push_back(z);  //add this customer to Customer Vector
    //u = db.getKeeper();      //set the user as the keeper and test Keeper class
    //adminstrator.showUserInfo();
    adminstrator.userInterface();

    //u = db.getCustomer(1);       //set the user as the first customer and test (
    //u->showUserInfo();
    //u->userInterface();
    //db.searchGoods();
```

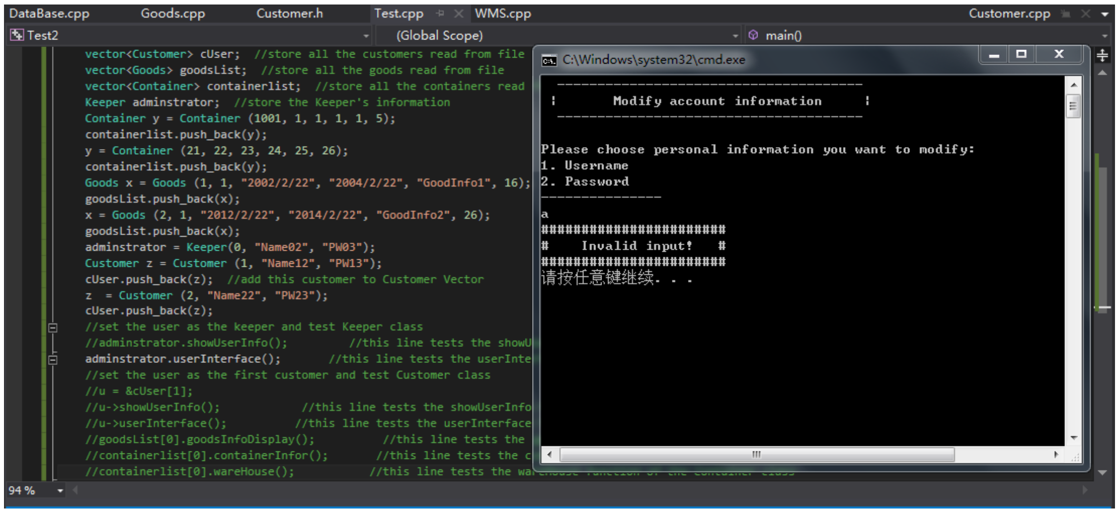
```
 ----------------------------------------
 |       Modify account information      |
 ----------------------------------------

Please choose personal information you want to modify:
1. Username
2. Password
---------------
```

Invalid input has been rejected, it would not be accepted.



```
vector<Customer> cUser;  //store all the customers read from file
vector<Goods> goodsList;  //store all the goods read from file
vector<Container> containerlist;  //store all the containers read f
Keeper adminstrator;  //store the Keeper's information
Container y = Container (1001, 1, 1, 1, 1, 5);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Customer Vector
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper class
//adminstrator.showUserInfo();         //this line tests the showUs
adminstrator.userInterface();       //this line tests the userInter
//set the user as the first customer and test Customer class
//u = &cUser[1];
//u->showUserInfo();           //this line tests the showUserInfo
//u->userInterface();          //this line tests the userInterface
//goodsList[0].goodsInfoDisplay();          //this line tests the g
//containerlist[0].containerInfor();       //this line tests the co
//containerlist[0].wareHouse();           //this line tests the war
```

```
 ------------------------------------------
 !       Modify account information      !
 ------------------------------------------

Please choose personal information you want to modify:
1. Username
2. Password
---------------

3
########################
#    Invalid input!   #
########################
请按任意键继续. . .
```



```
vector<Customer> cUser;  //store all the customers read from file
vector<Goods> goodsList;  //store all the goods read from file
vector<Container> containerlist;  //store all the containers read
Keeper adminstrator;  //store the Keeper's information
Container y = Container (1001, 1, 1, 1, 1, 5);
containerlist.push_back(y);
y = Container (21, 22, 23, 24, 25, 26);
containerlist.push_back(y);
Goods x = Goods (1, 1, "2002/2/22", "2004/2/22", "GoodInfo1", 16);
goodsList.push_back(x);
x = Goods (2, 1, "2012/2/22", "2014/2/22", "GoodInfo2", 26);
goodsList.push_back(x);
adminstrator = Keeper(0, "Name02", "PW03");
Customer z = Customer (1, "Name12", "PW13");
cUser.push_back(z);  //add this customer to Customer Vector
z  = Customer (2, "Name22", "PW23");
cUser.push_back(z);
//set the user as the keeper and test Keeper class
//adminstrator.showUserInfo();         //this line tests the showU
adminstrator.userInterface();       //this line tests the userInte
//set the user as the first customer and test Customer class
//u = &cUser[1];
//u->showUserInfo();           //this line tests the showUserInfo
//u->userInterface();          //this line tests the userInterface
//goodsList[0].goodsInfoDisplay();          //this line tests the
//containerlist[0].containerInfor();       //this line tests the c
//containerlist[0].wareHouse();           //this line tests the warehouse function of the container class
```

```
 ------------------------------------------
 !       Modify account information      !
 ------------------------------------------

Please choose personal information you want to modify:
1. Username
2. Password
---------------

a
########################
#    Invalid input!   #
########################
请按任意键继续. . .
```

By now, this one group of classes is tested successfully.

Database class test (by Chongyun Wang)

Database is the trunk of this program, which links all the other class and passed data.

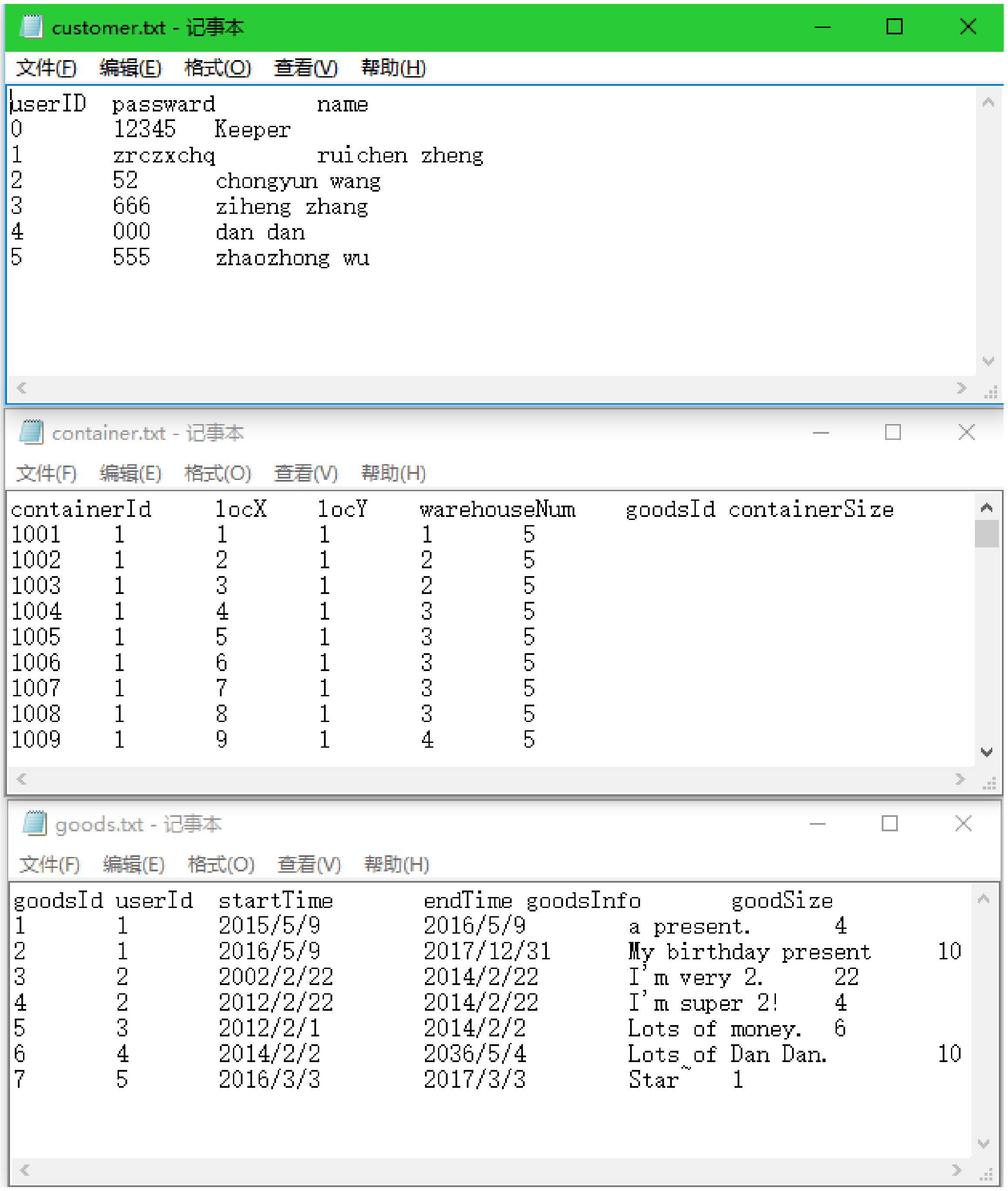
This class include several member functions which are listed below.

1. Database()

This member function operates at the beginning of program, which passes all the data from files to the program.

(1) readInfo()

Normally, when the function operates successfully, at first it will call another function readInfo(), this function reads information from three text files "Customer.txt", "Goods.txt" and "Container.txt" (Figure 35).


customer.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
container.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
goods.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
userID  passward        name
0       12345   Keeper
1       zrczxchq        ruichen zheng
2       52      chongyun wang
3       666     ziheng zhang
4       000     dan dan
5       555     zhaozhong wu
```

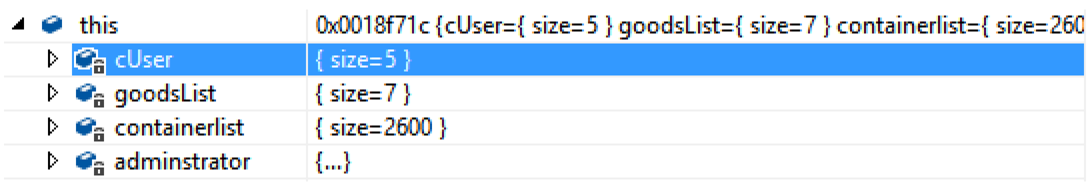
```
containerId     locX    locY    warehouseNum    goodsId containerSize
1001    1       1       1       1       5
1002    1       2       1       2       5
1003    1       3       1       2       5
1004    1       4       1       3       5
1005    1       5       1       3       5
1006    1       6       1       3       5
1007    1       7       1       3       5
1008    1       8       1       3       5
1009    1       9       1       4       5
```

```
goodsId userId  startTime       endTime goodsInfo       goodSize
1       1       2015/5/9        2016/5/9        a present.      4
2       1       2016/5/9        2017/12/31      My birthday present     10
3       2       2002/2/22       2014/2/22       I'm very 2.     22
4       2       2012/2/22       2014/2/22       I'm super 2!    4
5       3       2012/2/1        2014/2/2        Lots of money.  6
6       4       2014/2/2        2036/5/4        Lots of Dan Dan.        10
7       5       2016/3/3        2017/3/3        Star~   1
```

Figure 35: Database class readInfo() test files

All the data stored in these is listed line by line. Ones any data was missing or in wrong format, huge bugs would appear. If any of these three files are missing, this function could not judge it and would continue searching, which is another bug.

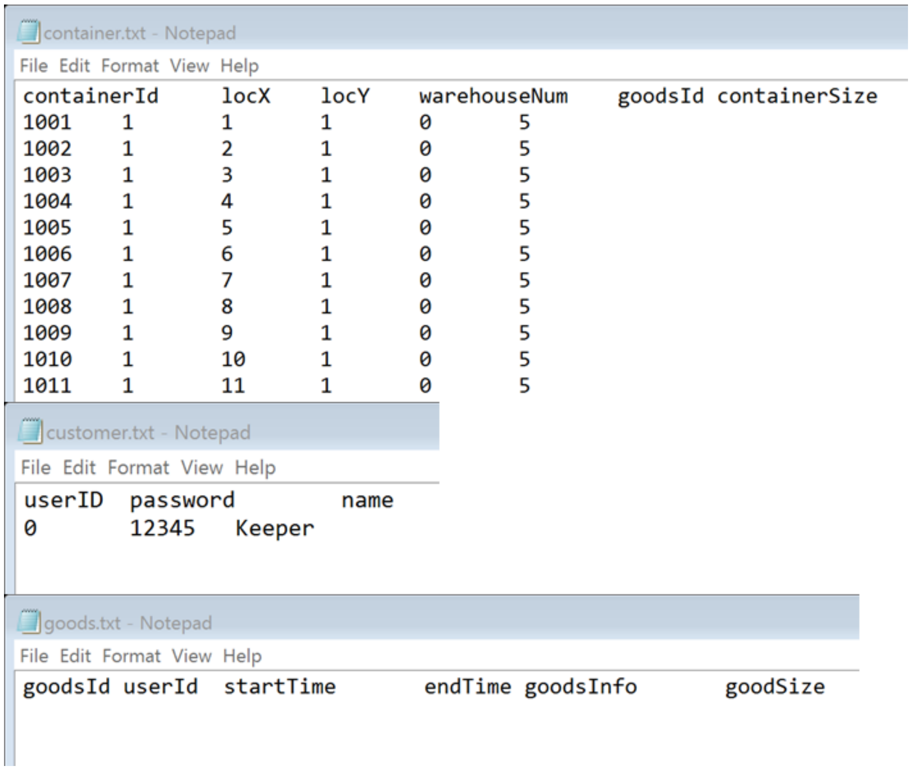
Once all the data files are correct, after operating, it will return four vectors. cUser, goodsList and containerlist stores all the information of users, goods and containers while administrator contains information of keeper (Figure 36).



```
this            0x0018f71c {cUser={ size=5 } goodsList={ size=7 } containerlist={ size=260
  cUser         { size=5 }
  goodsList     { size=7 }
  containerlist { size=2600 }
  adminstrator  {...}
```

Figure 36: Database class readInfo() results

If one file is missing, all the files will be set to default states (Figure 37).



container.txt - Notepad
File Edit Format View Help

```
containerId	locX	locY	warehouseNum	goodsId containerSize
1001	1	1	1	0	5
1002	1	2	1	0	5
1003	1	3	1	0	5
1004	1	4	1	0	5
1005	1	5	1	0	5
1006	1	6	1	0	5
1007	1	7	1	0	5
1008	1	8	1	0	5
1009	1	9	1	0	5
1010	1	10	1	0	5
1011	1	11	1	0	5
```

customer.txt - Notepad
File Edit Format View Help

```
userID	password	name
0	12345	Keeper
```

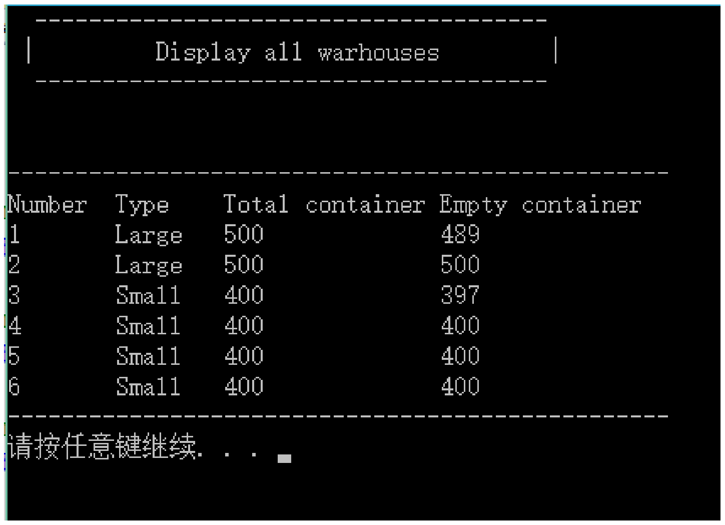
goods.txt - Notepad
File Edit Format View Help

```
goodsId userId	startTime	endTime goodsInfo	goodSize
```

Figure 37: Database class readInfo() for missing or empty files result

2. showWarehouseInfo()

This function uses function emptyContainer to get the number of empty containers in each warehouse and output as a list.

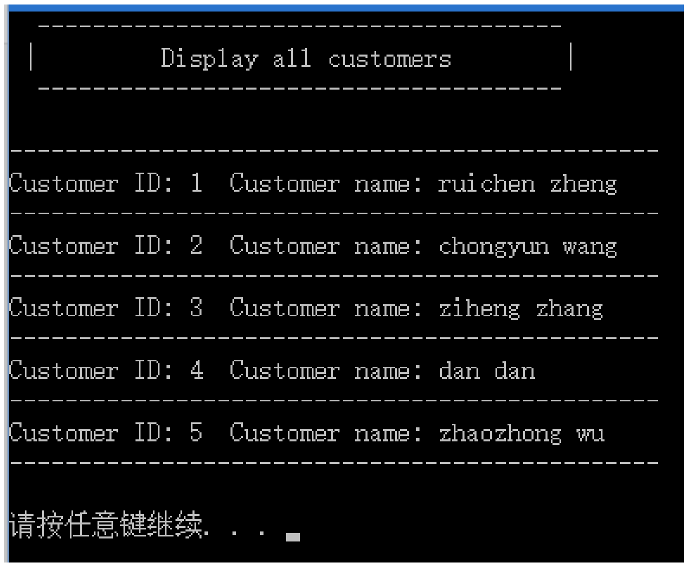


```
 --------------------------------------
|         Display all warhouses        |
 --------------------------------------


------------------------------------------------
Number  Type    Total container Empty container
1       Large   500             489
2       Large   500             500
3       Small   400             397
4       Small   400             400
5       Small   400             400
6       Small   400             400
------------------------------------------------
请按任意键继续. . .
```

Figure 38: Database class showWarehouseInfo() test

3. listCustomer()

This function can simply output all the username and ID in vector cUser, but cannot access password.

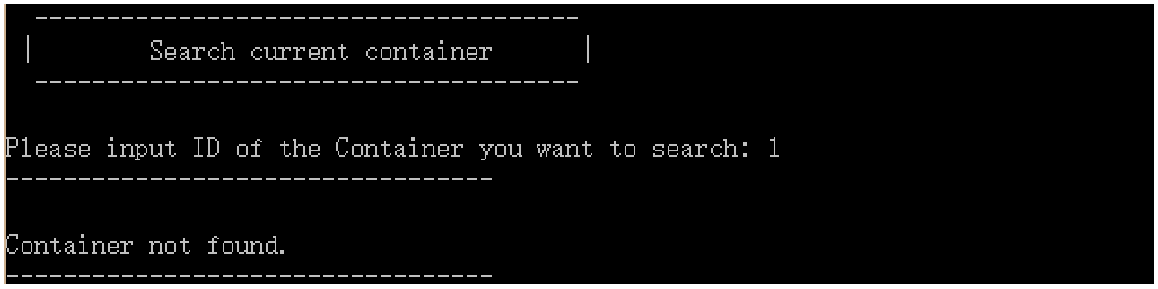


```
 --------------------------------------
|         Display all customers        |
 --------------------------------------

------------------------------------------------
Customer ID: 1  Customer name: ruichen zheng
------------------------------------------------
Customer ID: 2  Customer name: chongyun wang
------------------------------------------------
Customer ID: 3  Customer name: ziheng zhang
------------------------------------------------
Customer ID: 4  Customer name: dan dan
------------------------------------------------
Customer ID: 5  Customer name: zhaozhong wu
------------------------------------------------

请按任意键继续. . .
```

Figure 39: Database class listcustomer() test

4. searchGoods(), searchContainer() & searchCustomer()

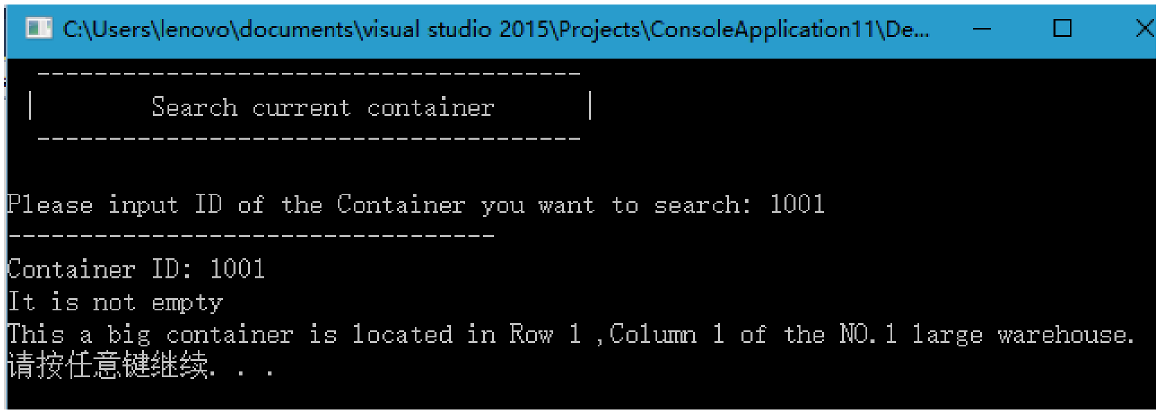
These functions scans data vector to find an item whose ID matches the input one. Invalid input and wrong ID will be warned that such item does not exist.



```
 ---------------------------------------
|        Search current container        |
 ---------------------------------------

Please input ID of the Container you want to search: 1
-----------------------------------

Container not found.
-----------------------------------
```

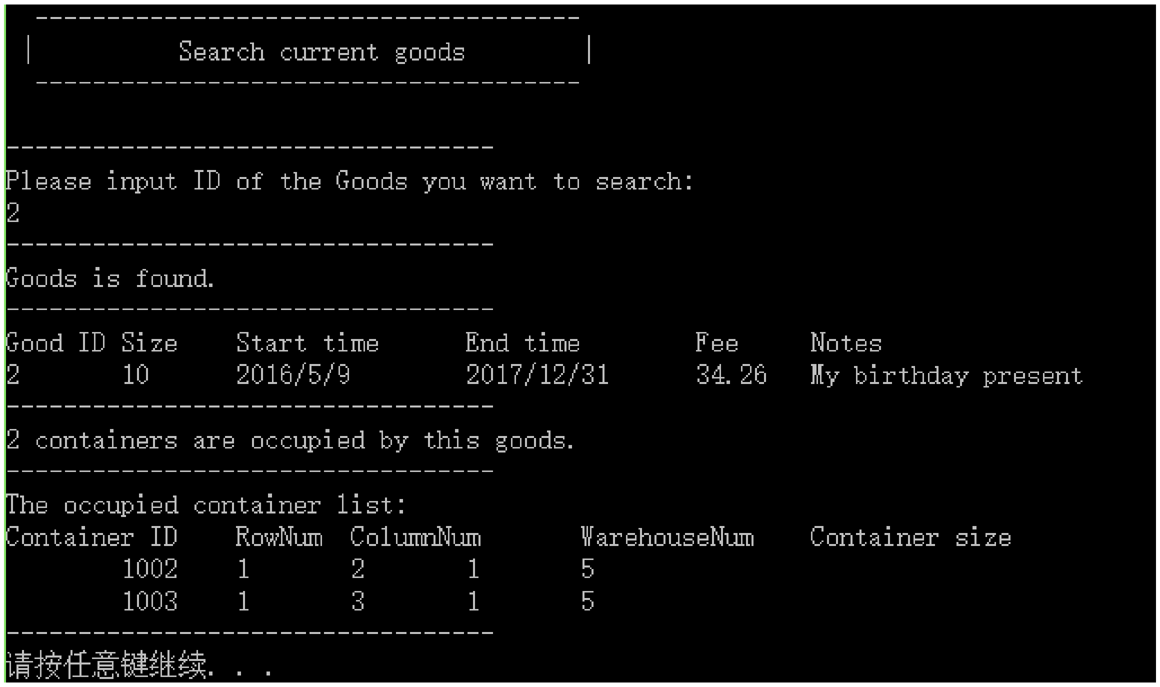
Figure 40: Database class searchContainer() test customer not found

However, once an item matches, related information will be output on screen.



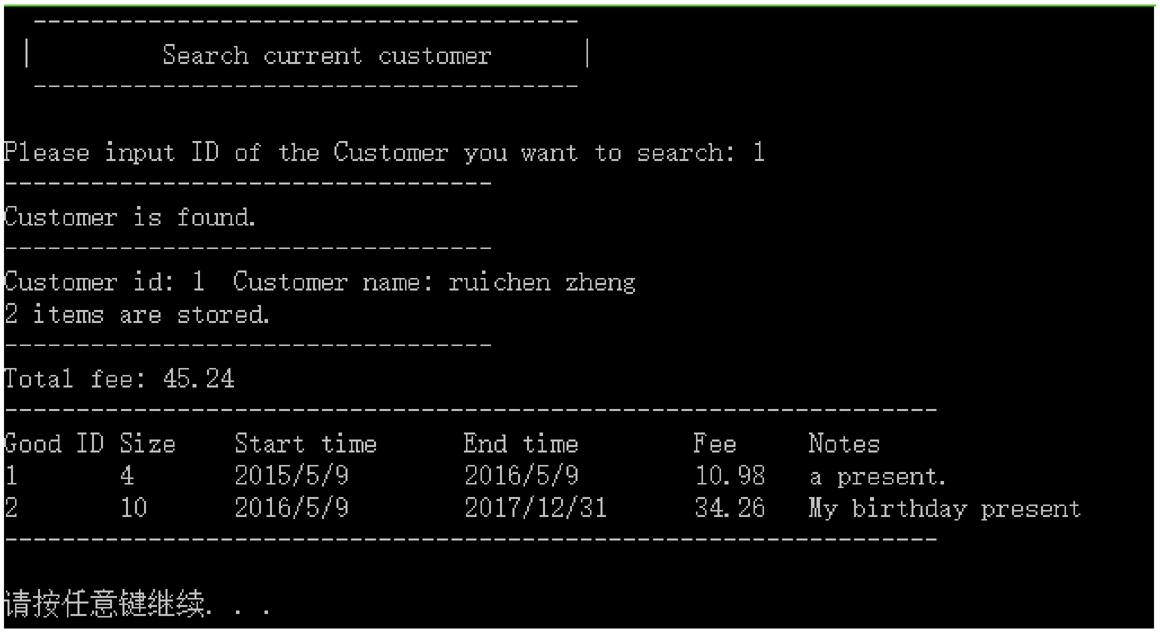
```
C:\Users\lenovo\documents\visual studio 2015\Projects\ConsoleApplication11\De...
 ---------------------------------------
|        Search current container        |
 ---------------------------------------

Please input ID of the Container you want to search: 1001
-----------------------------------
Container ID: 1001
It is not empty
This a big container is located in Row 1 ,Column 1 of the NO.1 large warehouse.
请按任意键继续. . .
```

Figure 41: Database class searchContainer() test container found



```
 ---------------------------------------
|          Search current goods          |
 ---------------------------------------


-----------------------------------
Please input ID of the Goods you want to search:
2
-----------------------------------
Goods is found.
-----------------------------------
Good ID Size    Start time      End time        Fee     Notes
2       10      2016/5/9        2017/12/31      34.26   My birthday present
-----------------------------------
2 containers are occupied by this goods.
-----------------------------------
The occupied container list:
Container ID    RowNum  ColumnNum       WarehouseNum    Container size
        1002    1       2       1       5
        1003    1       3       1       5
-----------------------------------
请按任意键继续. . .
```

Figure 42: Database class searchGoods() test goods found

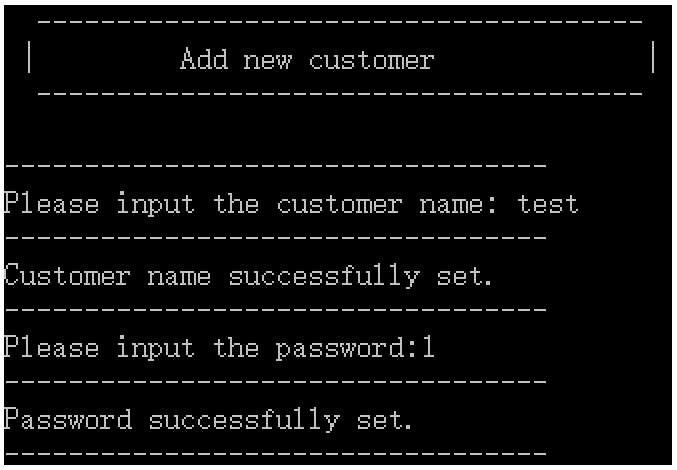


```
 --------------------------------------
|        Search current customer      |
 --------------------------------------

Please input ID of the Customer you want to search: 1
----------------------------------
Customer is found.
----------------------------------
Customer id: 1  Customer name: ruichen zheng
2 items are stored.
----------------------------------
Total fee: 45.24
-----------------------------------------------------------------
Good ID Size    Start time      End time        Fee     Notes
1       4       2015/5/9        2016/5/9        10.98   a present.
2       10      2016/5/9        2017/12/31      34.26   My birthday present
-----------------------------------------------------------------

请按任意键继续. . .
```

Figure 43: Database class searchCustomer() test customer found

5. addNewCustomer()

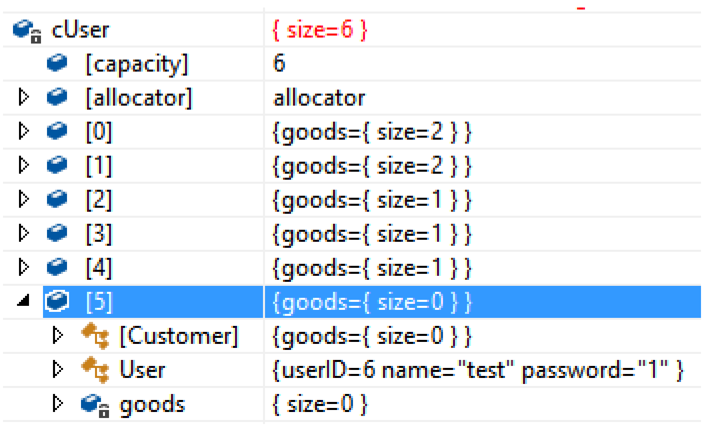
This function allow user to create a new user account.



```
 ------------------------------------------
|         Add new customer                 |
 ------------------------------------------


----------------------------------
Please input the customer name: test
----------------------------------
Customer name successfully set.
----------------------------------
Please input the password:1
----------------------------------
Password successfully set.
----------------------------------
```

Figure 44: Database class addNewCustomer() test

When the user account is created successfully, its information will be added to the end of vector cUser (Figure 45).

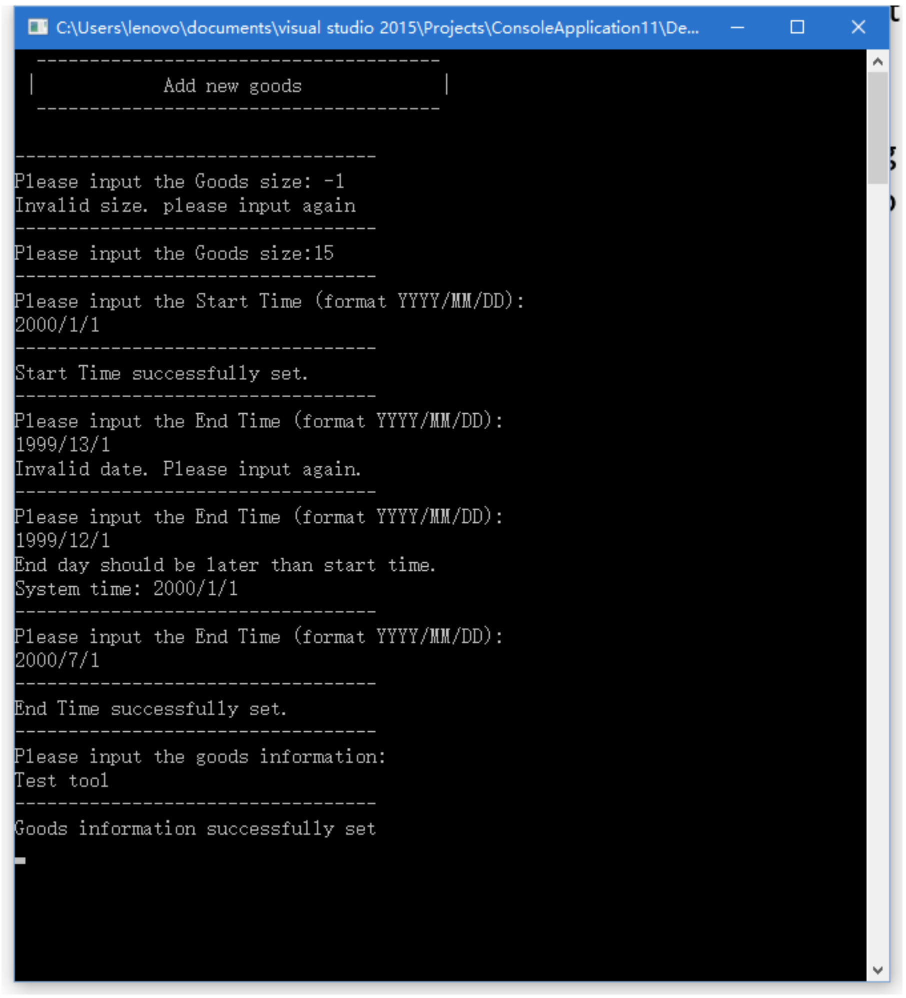


```
cUser               { size=6 }
  [capacity]        6
▷ [allocator]       allocator
▷ [0]               {goods={ size=2 } }
▷ [1]               {goods={ size=2 } }
▷ [2]               {goods={ size=1 } }
▷ [3]               {goods={ size=1 } }
▷ [4]               {goods={ size=1 } }
◢ [5]               {goods={ size=0 } }
  ▷ [Customer]      {goods={ size=0 } }
  ▷ User            {userID=6 name="test" password="1" }
  ▷ goods           { size=0 }
```

Figure 45: Database class addNewCustomer() test result

6. addNewGoods(int uid)

These function is used to add a new good to database. Invalid inputs like wrong good size, incorrect time format and end time earlier than start time will be asked to input again.



```
C:\Users\lenovo\documents\visual studio 2015\Projects\ConsoleApplication11\De...
 --------------------------------------
|              Add new goods           |
 --------------------------------------

----------------------------------
Please input the Goods size: -1
Invalid size. please input again
----------------------------------
Please input the Goods size:15
----------------------------------
Please input the Start Time (format YYYY/MM/DD):
2000/1/1
----------------------------------
Start Time successfully set.
----------------------------------
Please input the End Time (format YYYY/MM/DD):
1999/13/1
Invalid date. Please input again.
----------------------------------
Please input the End Time (format YYYY/MM/DD):
1999/12/1
End day should be later than start time.
System time: 2000/1/1
----------------------------------
Please input the End Time (format YYYY/MM/DD):
2000/7/1
----------------------------------
End Time successfully set.
----------------------------------
Please input the goods information:
Test tool
----------------------------------
Goods information successfully set
```

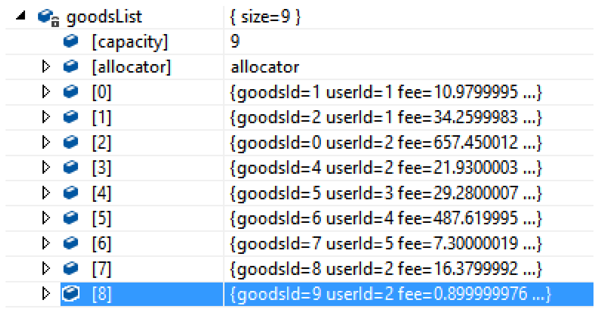
Figure 46: Database class addNewGoods(int uid) test

Even if all the information is valid, this function will use function emptyContainer() to check whether there is enough space to store. If there is enough space for storing this good, 2 functions will be called considering fees calculated (Figure 46).

(1) saveSmall(int gid, int size)
(2) saveBig(int gid, int size)

These two functions can adjust empty container to used container.

After enough empty containers adjusted, information of this new added good will be added to the end of goodsList (Figure 47).

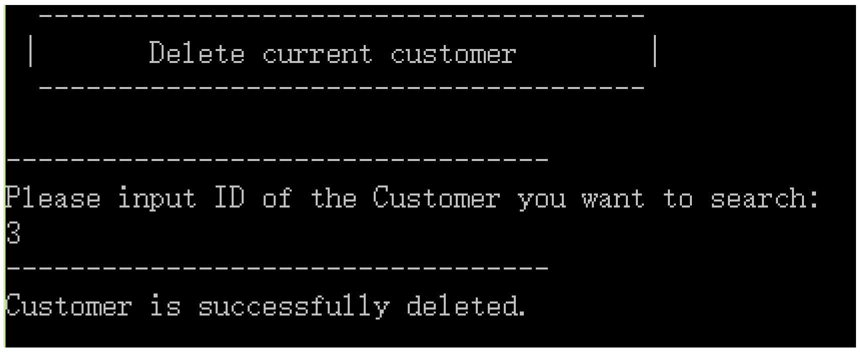


```
goodsList      { size=9 }
  [capacity]   9
  [allocator]  allocator
  [0]          {goodsId=1 userId=1 fee=10.9799995 ...}
  [1]          {goodsId=2 userId=1 fee=34.2599983 ...}
  [2]          {goodsId=0 userId=2 fee=657.450012 ...}
  [3]          {goodsId=4 userId=2 fee=21.9300003 ...}
  [4]          {goodsId=5 userId=3 fee=29.2800007 ...}
  [5]          {goodsId=6 userId=4 fee=487.619995 ...}
  [6]          {goodsId=7 userId=5 fee=7.30000019 ...}
  [7]          {goodsId=8 userId=2 fee=16.3799992 ...}
  [8]          {goodsId=9 userId=2 fee=0.899999976 ...}
```

Figure 47: Database class addNewGoods(int uid) test result

7. deleteCustomer()

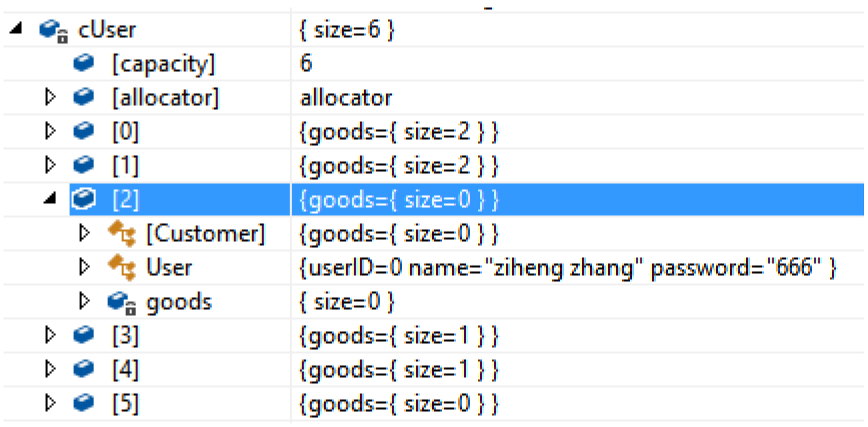
This function will search vector cUser to find an item which has the same ID as input. Once that item is found, its ID will be deleted from cUser. Deleting there is simply set id to 0.



```
  ----------------------------------------
 |        Delete current customer        |
  ----------------------------------------


-----------------------------------
Please input ID of the Customer you want to search:
3
-----------------------------------
Customer is successfully deleted.
```

Figure 48: Database class deleteCustomer() test

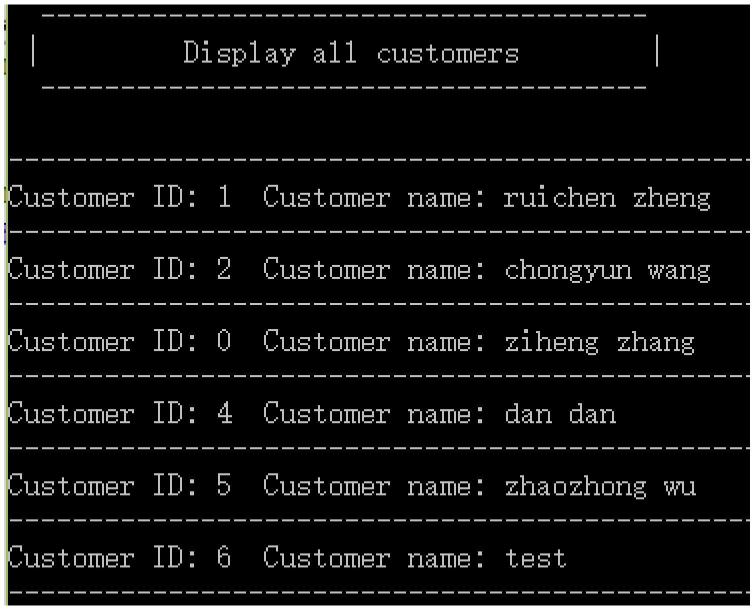
True deleting process is in saving process.



```
◢ cUser            { size=6 }
    [capacity]     6
  ▷ [allocator]    allocator
  ▷ [0]            {goods={ size=2 } }
  ▷ [1]            {goods={ size=2 } }
  ◢ [2]            {goods={ size=0 } }
    ▷ [Customer]   {goods={ size=0 } }
    ▷ User         {userID=0 name="ziheng zhang" password="666" }
    ▷ goods        { size=0 }
  ▷ [3]            {goods={ size=1 } }
  ▷ [4]            {goods={ size=1 } }
  ▷ [5]            {goods={ size=0 } }
```

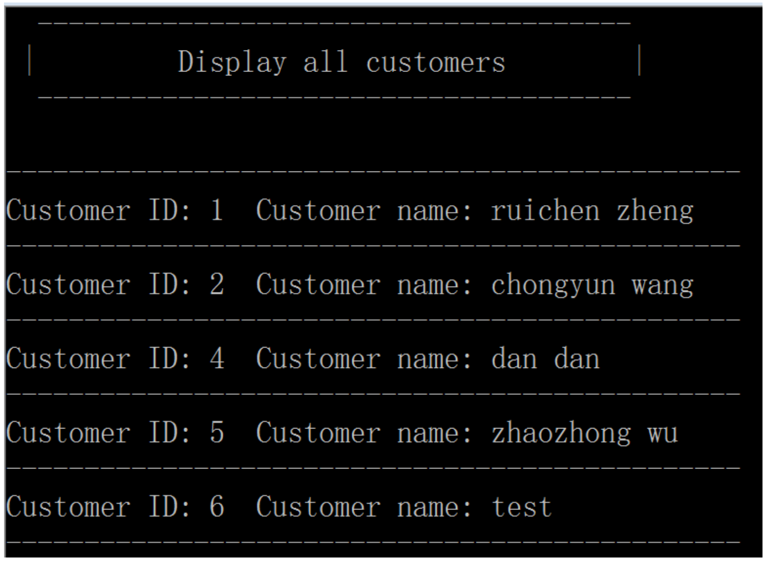
Figure 49: Database class deleteCustomer() test result

Therefore, if user asks to display all the customer, there will have a small bug. However, this problem is solved by erasing items with 0 id to avoid trouble.



```
 ----------------------------------------
|          Display all customers          |
 ----------------------------------------


-------------------------------------------------
Customer ID: 1  Customer name: ruichen zheng
-------------------------------------------------
Customer ID: 2  Customer name: chongyun wang
-------------------------------------------------
Customer ID: 0  Customer name: ziheng zhang
-------------------------------------------------
Customer ID: 4  Customer name: dan dan
-------------------------------------------------
Customer ID: 5  Customer name: zhaozhong wu
-------------------------------------------------
Customer ID: 6  Customer name: test
-------------------------------------------------
```

Figure 50: Database class deleteCustomer() test bug

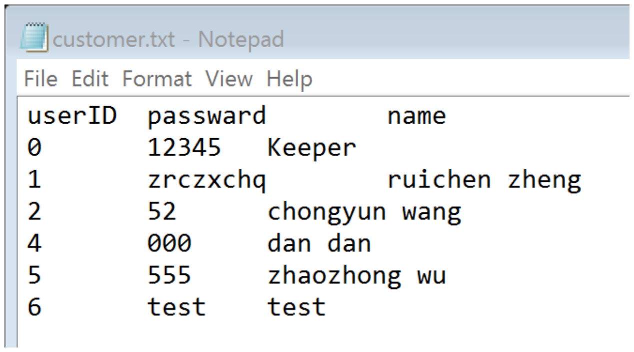


```
 ------------------------------------
|         Display all customers      |
 ------------------------------------


---------------------------------------------
Customer ID: 1  Customer name: ruichen zheng
---------------------------------------------
Customer ID: 2  Customer name: chongyun wang
---------------------------------------------
Customer ID: 4  Customer name: dan dan
---------------------------------------------
Customer ID: 5  Customer name: zhaozhong wu
---------------------------------------------
Customer ID: 6  Customer name: test
---------------------------------------------
```

Figure 51: Database class deleteCustomer() test bug solved

8. saveData()

This function can save all the data in program to files by simply overwriting them. The test is proved in

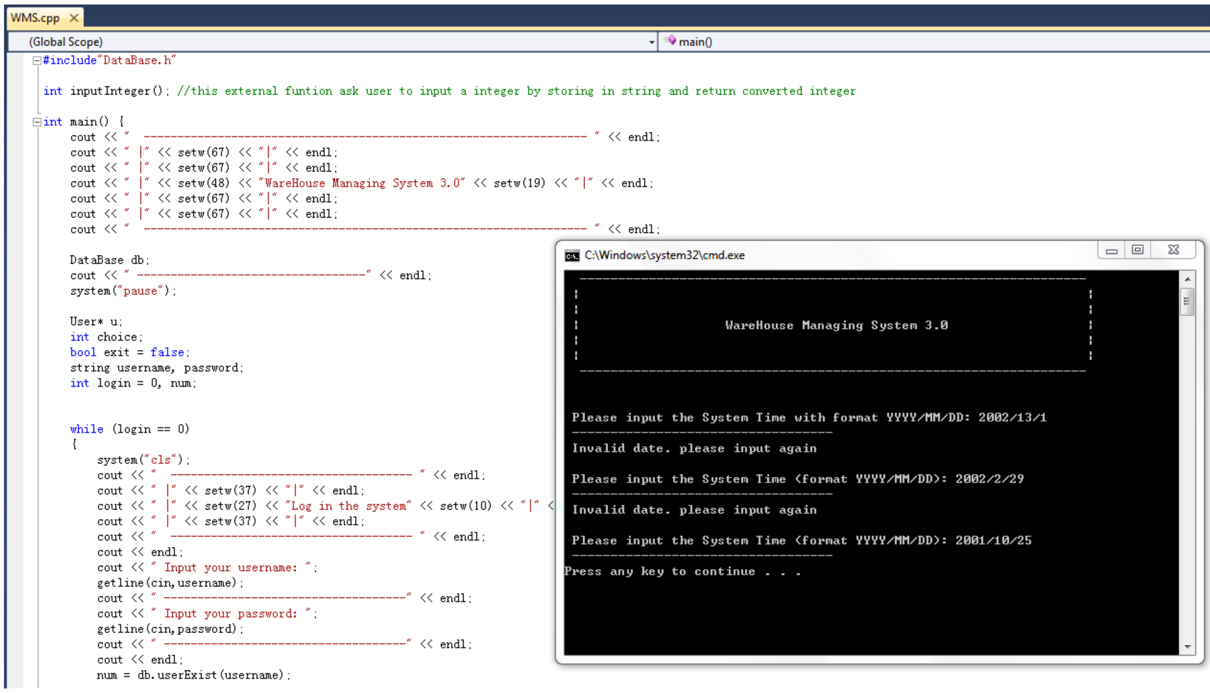


```
userID	passward	name
0	12345	Keeper
1	zrczxchq	ruichen zheng
2	52	chongyun wang
4	000	dan dan
5	555	zhaozhong wu
6	test	test
```

Figure 52: Database class saveData() test

The main function test (by Ziheng Zhang)

This part will focus on the testing of the main function “WMS.cpp” (WMS represents Warehouse Management System)



WMS.cpp

(Global Scope) main()

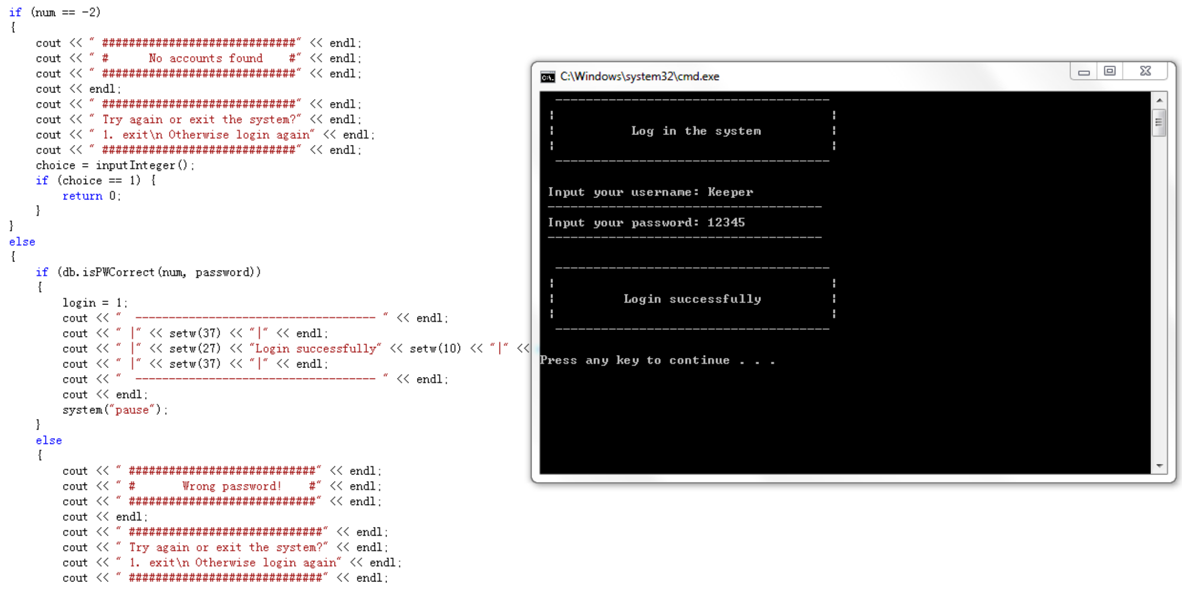
```
#include"DataBase.h"

int inputInteger(); //this external funtion ask user to input a integer by storing in string and return converted integer

int main() {
    cout << "  ------------------------------------------------------------------ " << endl;
    cout << " |" << setw(67) << "|" << endl;
    cout << " |" << setw(67) << "|" << endl;
    cout << " |" << setw(48) << "WareHouse Managing System 3.0" << setw(19) << "|" << endl;
    cout << " |" << setw(67) << "|" << endl;
    cout << " |" << setw(67) << "|" << endl;
    cout << "  ------------------------------------------------------------------ " << endl;

    DataBase db;
    cout << " ----------------------------------" << endl;
    system("pause");

    User* u;
    int choice;
    bool exit = false;
    string username, password;
    int login = 0, num;


    while (login == 0)
    {
        system("cls");
        cout << "  ------------------------------------ " << endl;
        cout << " |" << setw(37) << "|" << endl;
        cout << " |" << setw(27) << "Log in the system" << setw(10) << "|" <
        cout << " |" << setw(37) << "|" << endl;
        cout << "  ------------------------------------ " << endl;
        cout << endl;
        cout << " Input your username: ";
        getline(cin,username);
        cout << " ------------------------------------" << endl;
        cout << " Input your password: ";
        getline(cin,password);
        cout << " ------------------------------------" << endl;
        cout << endl;
        num = db.userExist(username);
```

C:\Windows\system32\cmd.exe

```
 --------------------------------------------------------------------
 |                                                                  |
 |                                                                  |
 |                   WareHouse Managing System 3.0                  |
 |                                                                  |
 |                                                                  |
 --------------------------------------------------------------------


 Please input the System Time with format YYYY/MM/DD: 2002/13/1
 ----------------------------------
 Invalid date. please input again

 Please input the System Time (format YYYY/MM/DD): 2002/2/29
 ----------------------------------
 Invalid date. please input again

 Please input the System Time (format YYYY/MM/DD): 2001/10/25
 ----------------------------------
Press any key to continue . . .
```

Figure 53: main function test system time set

As shown above (Figure 53), at the beginning of the program, the creation of Database will require user to input the System time. When user inputs 2002/13/1, the system rejects input because there are only 12 months in a year. When user inputs 2002/2/29, the system also rejects input because year 2002 is not a leap year, and there are only 28 days in February. User input 2001/10/25 is accepted and set by the system.



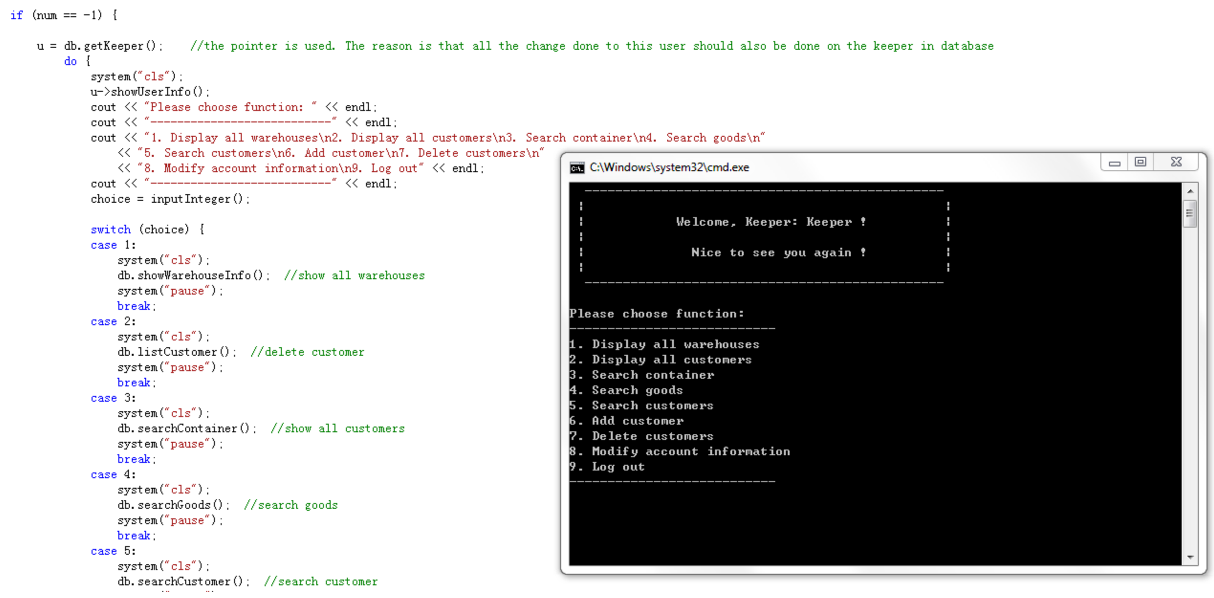
```
if (num == -2)
{
    cout << " ############################" << endl;
    cout << " #      No accounts found    #" << endl;
    cout << " ############################" << endl;
    cout << endl;
    cout << " ############################" << endl;
    cout << " Try again or exit the system?" << endl;
    cout << " 1. exit\n Otherwise login again" << endl;
    cout << " ############################" << endl;
    choice = inputInteger();
    if (choice == 1) {
        return 0;
    }
}
else
{
    if (db.isPWCorrect(num, password))
    {
        login = 1;
        cout << "  ------------------------------------ " << endl;
        cout << " |" << setw(37) << "|" << endl;
        cout << " |" << setw(27) << "Login successfully" << setw(10) << "|" <<
        cout << " |" << setw(37) << "|" << endl;
        cout << "  ------------------------------------ " << endl;
        cout << endl;
        system("pause");
    }
    else
    {
        cout << " ############################" << endl;
        cout << " #       Wrong password!    #" << endl;
        cout << " ############################" << endl;
        cout << endl;
        cout << " ############################" << endl;
        cout << " Try again or exit the system?" << endl;
        cout << " 1. exit\n Otherwise login again" << endl;
        cout << " ############################" << endl;
```

C:\Windows\system32\cmd.exe

```
 ------------------------------------
 |                                  |
 |         Log in the system        |
 |                                  |
 ------------------------------------

 Input your username: Keeper
 ------------------------------------
 Input your password: 12345
 ------------------------------------

 ------------------------------------
 |                                  |
 |        Login successfully        |
 |                                  |
 ------------------------------------

Press any key to continue . . .
```

Figure 54: main function test Keeper log in

Because there are two types of users in the system: keeper and customer. Therefore, testing will be carried on these two types respectively. When user inputs keeper's username and password, the system will carry on to keeper functions.

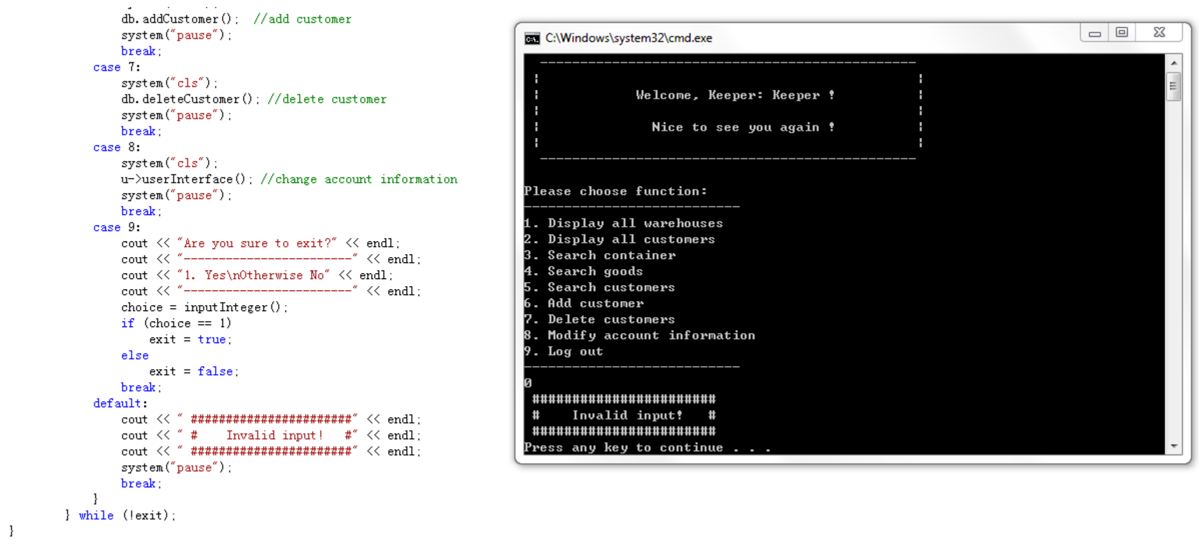


```
if (num == -1) {

    u = db.getKeeper();    //the pointer is used. The reason is that all the change done to this user should also be done on the keeper in database
        do {
            system("cls");
            u->showUserInfo();
            cout << "Please choose function: " << endl;
            cout << "---------------------------" << endl;
            cout << "1. Display all warehouses\n2. Display all customers\n3. Search container\n4. Search goods\n"
                << "5. Search customers\n6. Add customer\n7. Delete customers\n"
                << "8. Modify account information\n9. Log out" << endl;
            cout << "---------------------------" << endl;
            choice = inputInteger();

            switch (choice) {
            case 1:
                system("cls");
                db.showWarehouseInfo();  //show all warehouses
                system("pause");
                break;
            case 2:
                system("cls");
                db.listCustomer();  //delete customer
                system("pause");
                break;
            case 3:
                system("cls");
                db.searchContainer();  //show all customers
                system("pause");
                break;
            case 4:
                system("cls");
                db.searchGoods();  //search goods
                system("pause");
                break;
            case 5:
                system("cls");
                db.searchCustomer();  //search customer
```

```
 ------------------------------------------------
 |                                              |
 |          Welcome, Keeper: Keeper !           |
 |                                              |
 |            Nice to see you again !           |
 |                                              |
 ------------------------------------------------

Please choose function:
---------------------------
1. Display all warehouses
2. Display all customers
3. Search container
4. Search goods
5. Search customers
6. Add customer
7. Delete customers
8. Modify account information
9. Log out
---------------------------
```

Figure 55: main function test Keeper

As shown in the cmd (Figure 55), there are eight functions in keeper access. Because keeper has higher access to the warehouse, container, customer and his personal information, the operation will transfer to the database level, directly modifying the database vectors.

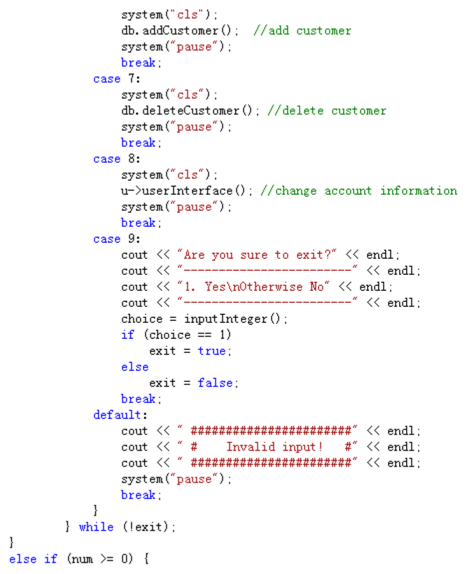


```
                db.addCustomer();  //add customer
                system("pause");
                break;
            case 7:
                system("cls");
                db.deleteCustomer(); //delete customer
                system("pause");
                break;
            case 8:
                system("cls");
                u->userInterface(); //change account information
                system("pause");
                break;
            case 9:
                cout << "Are you sure to exit?" << endl;
                cout << "------------------------" << endl;
                cout << "1. Yes\nOtherwise No" << endl;
                cout << "------------------------" << endl;
                choice = inputInteger();
                if (choice == 1)
                    exit = true;
                else
                    exit = false;
                break;
            default:
                cout << " #######################" << endl;
                cout << " #    Invalid input!   #" << endl;
                cout << " #######################" << endl;
                system("pause");
                break;
            }
        } while (!exit);
}
```

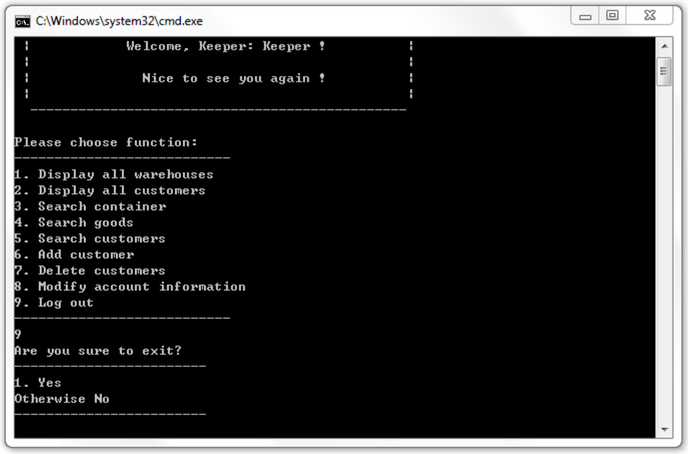
```
 ------------------------------------------------
 |                                              |
 |          Welcome, Keeper: Keeper !           |
 |                                              |
 |            Nice to see you again !           |
 |                                              |
 ------------------------------------------------

Please choose function:
---------------------------
1. Display all warehouses
2. Display all customers
3. Search container
4. Search goods
5. Search customers
6. Add customer
7. Delete customers
8. Modify account information
9. Log out
---------------------------
0
 #######################
 #    Invalid input!   #
 #######################
Press any key to continue . . .
```

Figure 56: main function test reject invalid function choose

When user inputs wrong input, system will display invalid information and require inputs again (Figure 56).



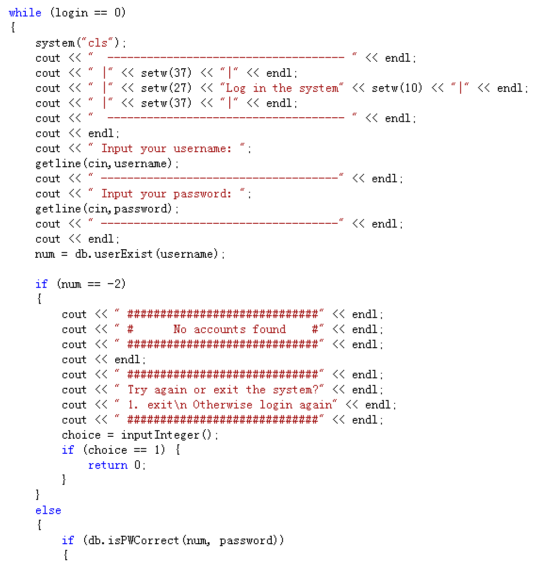
```
                system("cls");
                db.addCustomer();  //add customer
                system("pause");
                break;
            case 7:
                system("cls");
                db.deleteCustomer(); //delete customer
                system("pause");
                break;
            case 8:
                system("cls");
                u->userInterface(); //change account information
                system("pause");
                break;
            case 9:
                cout << "Are you sure to exit?" << endl;
                cout << "------------------------" << endl;
                cout << "1. Yes\nOtherwise No" << endl;
                cout << "------------------------" << endl;
                choice = inputInteger();
                if (choice == 1)
                    exit = true;
                else
                    exit = false;
                break;
            default:
                cout << " #######################" << endl;
                cout << " #    Invalid input!   #" << endl;
                cout << " #######################" << endl;
                system("pause");
                break;
            }
        } while (!exit);
    }
    else if (num >= 0) {
```



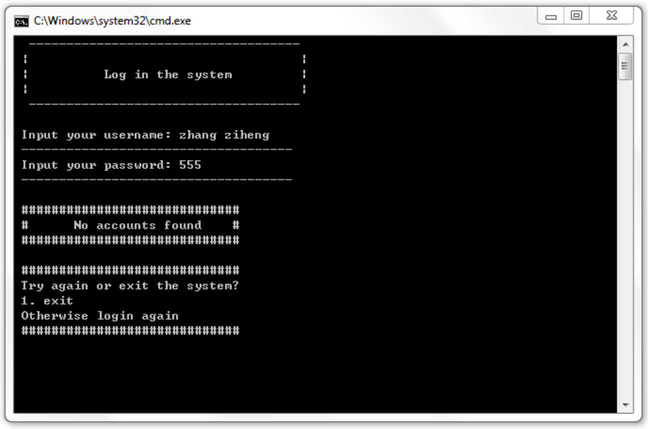
```
 |            Welcome, Keeper: Keeper !            |
 |                                                 |
 |              Nice to see you again !            |
 |                                                 |
  -------------------------------------------------

Please choose function:
----------------------------
1. Display all warehouses
2. Display all customers
3. Search container
4. Search goods
5. Search customers
6. Add customer
7. Delete customers
8. Modify account information
9. Log out
----------------------------
9
Are you sure to exit?
------------------------
1. Yes
Otherwise No
------------------------
```

Figure 57: main function test log out

When user tends to exit, system will ask user whether sure to exit in case of mistakes. When user input 1, the system exits, and when user input otherwise, the system will return to the function choosing part (Figure 57).



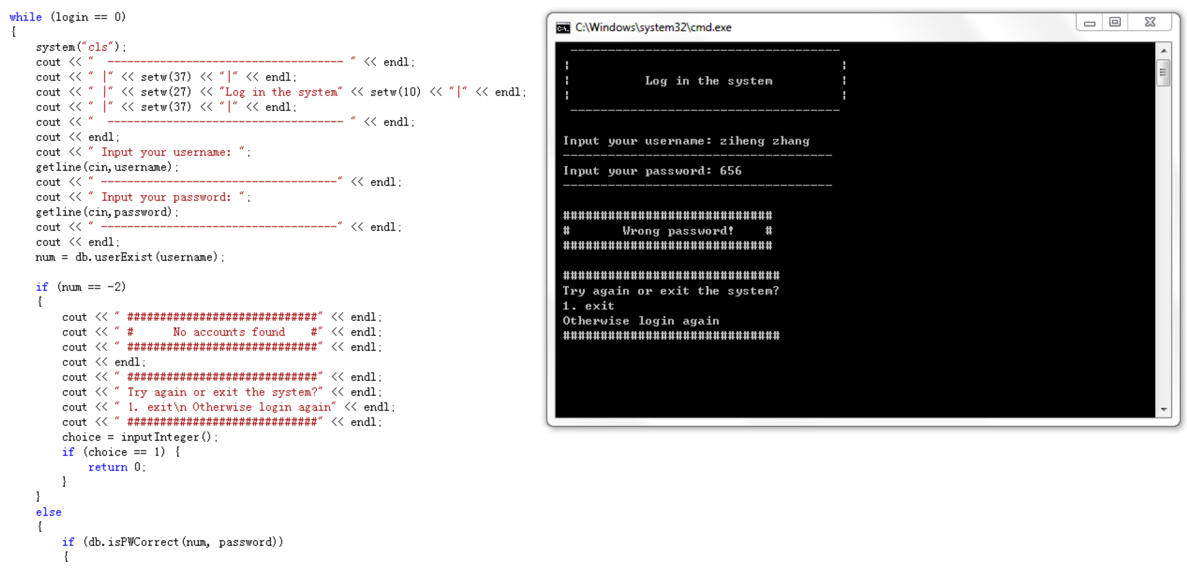
```
while (login == 0)
{
    system("cls");
    cout << "  ------------------------------------- " << endl;
    cout << " |" << setw(37) << "|" << endl;
    cout << " |" << setw(27) << "Log in the system" << setw(10) << "|" << endl;
    cout << " |" << setw(37) << "|" << endl;
    cout << "  ------------------------------------- " << endl;
    cout << endl;
    cout << " Input your username: ";
    getline(cin,username);
    cout << " -------------------------------------" << endl;
    cout << " Input your password: ";
    getline(cin,password);
    cout << " -------------------------------------" << endl;
    cout << endl;
    num = db.userExist(username);

    if (num == -2)
    {
        cout << " ##############################" << endl;
        cout << " #      No accounts found     #" << endl;
        cout << " ##############################" << endl;
        cout << endl;
        cout << " ##############################" << endl;
        cout << " Try again or exit the system?" << endl;
        cout << " 1. exit\n Otherwise login again" << endl;
        cout << " ##############################" << endl;
        choice = inputInteger();
        if (choice == 1) {
            return 0;
        }
    }
    else
    {
        if (db.isPWCorrect(num, password))
        {
```



```
  -------------------------------------
 |                                     |
 |          Log in the system          |
 |                                     |
  -------------------------------------

 Input your username: zhang ziheng
 -------------------------------------
 Input your password: 555
 -------------------------------------

 ##############################
 #      No accounts found     #
 ##############################

 ##############################
 Try again or exit the system?
 1. exit
 Otherwise login again
 ##############################
```

Figure 58: main function test invalid account

Next user is customer, who has access to his goods and his personal information. However, the username and password of one customer are important to check according to the file. When the username is not listed in the file, the system will display “No accounts found” (Figure 58).



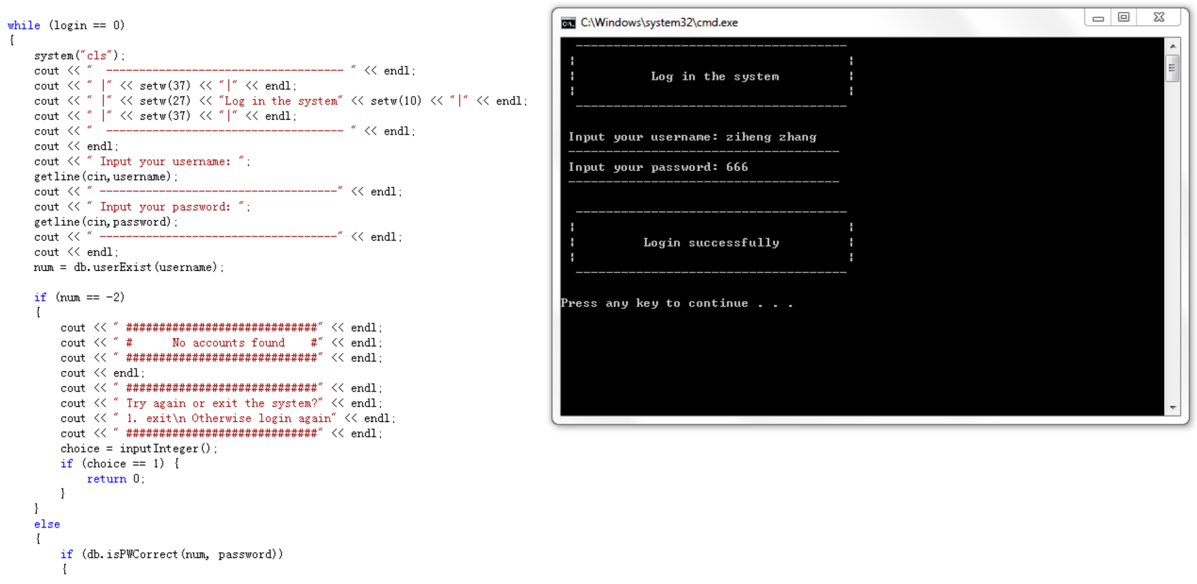
```
while (login == 0)
{
    system("cls");
    cout << "  ------------------------------------ " << endl;
    cout << " |" << setw(37) << "|" << endl;
    cout << " |" << setw(27) << "Log in the system" << setw(10) << "|" << endl;
    cout << " |" << setw(37) << "|" << endl;
    cout << "  ------------------------------------ " << endl;
    cout << endl;
    cout << " Input your username: ";
    getline(cin,username);
    cout << " ------------------------------------" << endl;
    cout << " Input your password: ";
    getline(cin,password);
    cout << " ------------------------------------" << endl;
    cout << endl;
    num = db.userExist(username);

    if (num == -2)
    {
        cout << " ##############################" << endl;
        cout << " #       No accounts found    #" << endl;
        cout << " ##############################" << endl;
        cout << endl;
        cout << " ##############################" << endl;
        cout << " Try again or exit the system?" << endl;
        cout << " 1. exit\n Otherwise login again" << endl;
        cout << " ##############################" << endl;
        choice = inputInteger();
        if (choice == 1) {
            return 0;
        }
    }
    else
    {
        if (db.isPWCorrect(num, password))
        {
```

C:\Windows\system32\cmd.exe

```
  ------------------------------------
 |                                    |
 |          Log in the system         |
 |                                    |
  ------------------------------------

Input your username: ziheng zhang
------------------------------------
Input your password: 656
------------------------------------

############################
#       Wrong password!    #
############################

#############################
Try again or exit the system?
1. exit
Otherwise login again
#############################
```

Figure 59: main function test invalid password

When the password is not correct, the system will also display “Wrong password” to require user to login again (Figure 59).



```
while (login == 0)
{
    system("cls");
    cout << "  ------------------------------------ " << endl;
    cout << " |" << setw(37) << "|" << endl;
    cout << " |" << setw(27) << "Log in the system" << setw(10) << "|" << endl;
    cout << " |" << setw(37) << "|" << endl;
    cout << "  ------------------------------------ " << endl;
    cout << endl;
    cout << " Input your username: ";
    getline(cin,username);
    cout << " ------------------------------------" << endl;
    cout << " Input your password: ";
    getline(cin,password);
    cout << " ------------------------------------" << endl;
    cout << endl;
    num = db.userExist(username);

    if (num == -2)
    {
        cout << " ##############################" << endl;
        cout << " #       No accounts found    #" << endl;
        cout << " ##############################" << endl;
        cout << endl;
        cout << " ##############################" << endl;
        cout << " Try again or exit the system?" << endl;
        cout << " 1. exit\n Otherwise login again" << endl;
        cout << " ##############################" << endl;
        choice = inputInteger();
        if (choice == 1) {
            return 0;
        }
    }
    else
    {
        if (db.isPWCorrect(num, password))
        {
```

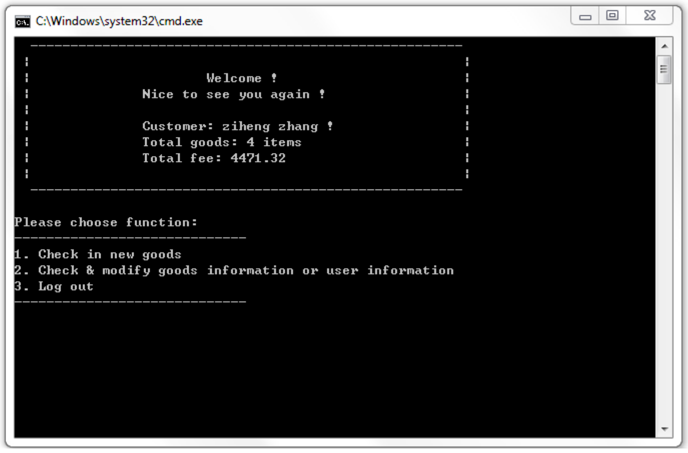
C:\Windows\system32\cmd.exe

```
  ------------------------------------
 |                                    |
 |          Log in the system         |
 |                                    |
  ------------------------------------

 Input your username: ziheng zhang
 ------------------------------------
 Input your password: 666
 ------------------------------------

  ------------------------------------
 |                                    |
 |        Login successfully          |
 |                                    |
  ------------------------------------

Press any key to continue . . .
```

Figure 60: main function test Customer log in

Only when username and password are both correct, the system will accept the customer and carry on to the customer functions (Figure 60).

```
    u = db.getCustomer(num);  //the pointer is used. The reason is that all the change done to this user should also be done on the customer in database

    do {

        system("cls");
        u->showUserInfo();
        cout << "Please choose function: " << endl;
        cout << "------------------------------" << endl;
        cout << "1. Check in new goods\n2. Check & modify goods information or user information\n3. Log out" << endl;
        cout << "------------------------------" << endl;
        choice = inputInteger();

        switch (choice) {
        case 1:
            system("cls");
            db.addNewGoods(u->getID());  //add goods
            break;
        case 2:
            system("cls");
            u->userInterface(); //other function
            break;
        case 3:
            cout << endl;
            cout << "Are you sure to exit?" << endl;
            cout << "------------------------" << endl;
            cout << "1.Yes\n0therwise No" << endl;
            cout << "------------------------" << endl;
            choice = inputInteger();
            if (choice == 1)
                exit = true;
            else
                exit = false;
            break;
        default:
            cout << " ######################" << endl;
            cout << " #    Invalid input!   #" << endl;
            cout << " ######################" << endl;
```

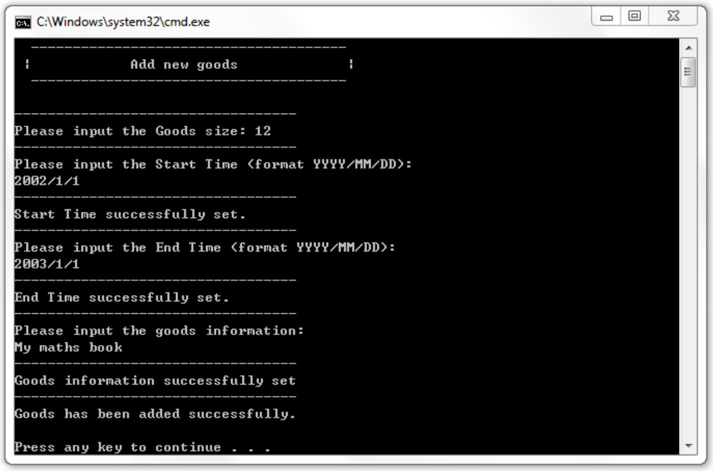


```
C:\Windows\system32\cmd.exe
 ------------------------------------------------------------
 |                                                          |
 |                       Welcome !                          |
 |                Nice to see you again !                   |
 |                                                          |
 |                Customer: ziheng zhang !                  |
 |                Total goods: 4 items                      |
 |                Total fee: 4471.32                        |
 |                                                          |
 ------------------------------------------------------------

Please choose function:
------------------------------
1. Check in new goods
2. Check & modify goods information or user information
3. Log out
------------------------------
```

Figure 61: main function test Customer

The first operation level of customer is displayed above, which includes "Check in new goods", "Check & modify goods information or user information" and "Log out". Moreover, all user information is displayed above the function choosing: name, goods number and total fee. As shown in code, when customer wants to check in new goods, the system will transfer directly to the database function. When customer wants to conduct other operations, the system will transfer to the customer interface, so all these function are operated in class Customer (Figure 61).

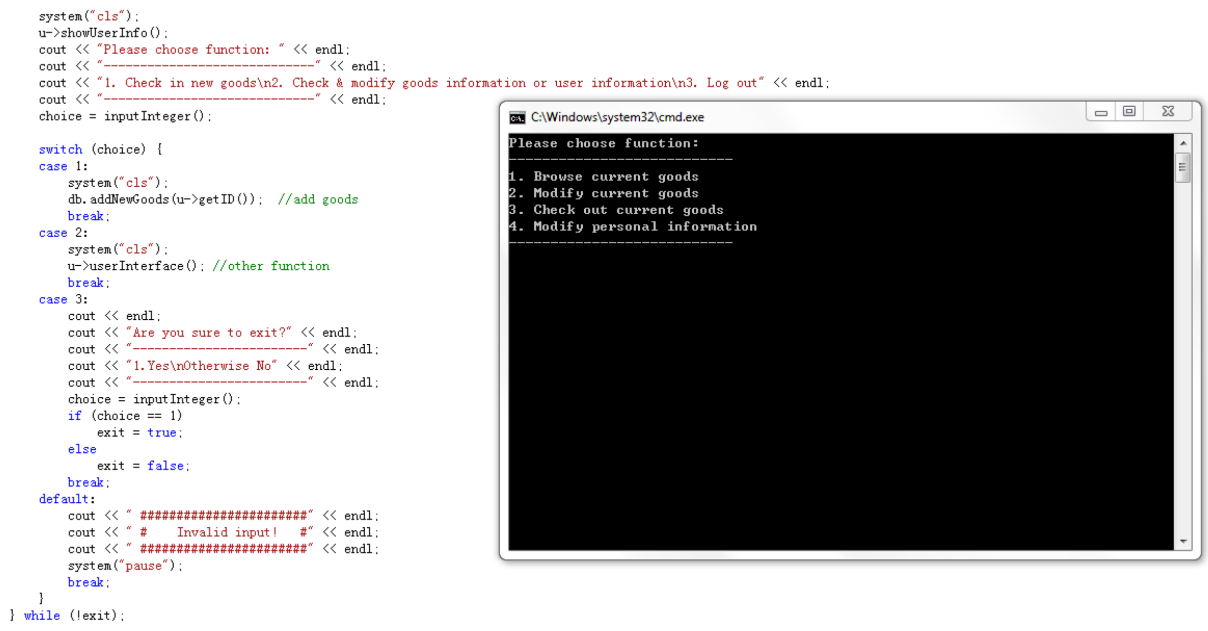
```
    system("cls");
    u->showUserInfo();
    cout << "Please choose function: " << endl;
    cout << "------------------------------" << endl;
    cout << "1. Check in new goods\n2. Check & modify goods information or user information\n3. Log out" << endl;
    cout << "------------------------------" << endl;
    choice = inputInteger();

    switch (choice) {
    case 1:
        system("cls");
        db.addNewGoods(u->getID());  //add goods
        break;
    case 2:
        system("cls");
        u->userInterface(); //other function
        break;
    case 3:
        cout << endl;
        cout << "Are you sure to exit?" << endl;
        cout << "------------------------" << endl;
        cout << "1.Yes\n0therwise No" << endl;
        cout << "------------------------" << endl;
        choice = inputInteger();
        if (choice == 1)
            exit = true;
        else
            exit = false;
        break;
    default:
        cout << " ######################" << endl;
        cout << " #    Invalid input!   #" << endl;
        cout << " ######################" << endl;
        system("pause");
        break;
    }
} while (!exit);
```



```
C:\Windows\system32\cmd.exe
  ----------------------------------------
 |              Add new goods             |
  ----------------------------------------

------------------------------
Please input the Goods size: 12
------------------------------
Please input the Start Time (format YYYY/MM/DD):
2002/1/1
------------------------------
Start Time successfully set.
------------------------------
Please input the End Time (format YYYY/MM/DD):
2003/1/1
------------------------------
End Time successfully set.
------------------------------
Please input the goods information:
My maths book
------------------------------
Goods information successfully set
------------------------------
Goods has been added successfully.

Press any key to continue . . .
```

Figure 62: main function test Customer add new goods

When customer wants to check in new goods, these functions are operated in class DataBase and all these inputs are also necessary to check whether invalid (Figure 62).

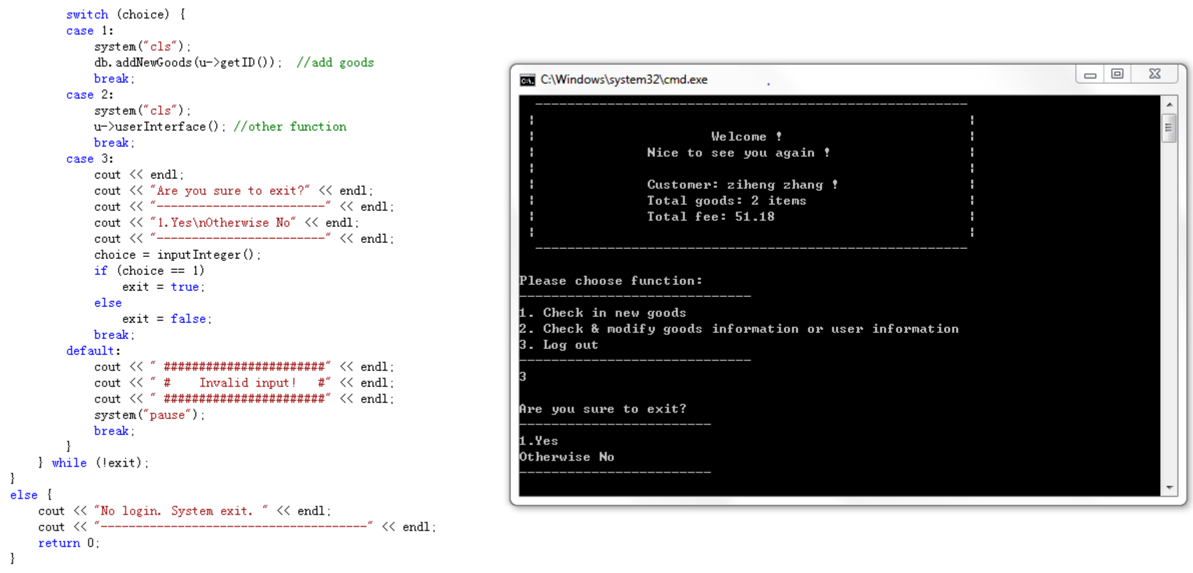


```
        system("cls");
        u->showUserInfo();
        cout << "Please choose function: " << endl;
        cout << "------------------------------" << endl;
        cout << "1. Check in new goods\n2. Check & modify goods information or user information\n3. Log out" << endl;
        cout << "------------------------------" << endl;
        choice = inputInteger();

        switch (choice) {
        case 1:
            system("cls");
            db.addNewGoods(u->getID());  //add goods
            break;
        case 2:
            system("cls");
            u->userInterface(); //other function
            break;
        case 3:
            cout << endl;
            cout << "Are you sure to exit?" << endl;
            cout << "------------------------" << endl;
            cout << "1.Yes\nOtherwise No" << endl;
            cout << "------------------------" << endl;
            choice = inputInteger();
            if (choice == 1)
                exit = true;
            else
                exit = false;
            break;
        default:
            cout << " #######################" << endl;
            cout << " #    Invalid input!   #" << endl;
            cout << " #######################" << endl;
            system("pause");
            break;
        }
    } while (!exit);
```

```
C:\Windows\system32\cmd.exe
Please choose function:
---------------------------
1. Browse current goods
2. Modify current goods
3. Check out current goods
4. Modify personal information
---------------------------
```

Figure 63: main function test Customer user Interface

When customer chooses to check and modify goods information or user information, the system will transfer to class Customer, and interface will also be provided by class Customer (Figure 63).

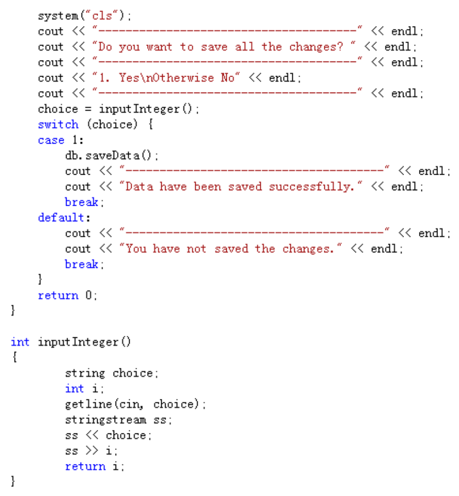


```
        switch (choice) {
        case 1:
            system("cls");
            db.addNewGoods(u->getID());  //add goods
            break;
        case 2:
            system("cls");
            u->userInterface(); //other function
            break;
        case 3:
            cout << endl;
            cout << "Are you sure to exit?" << endl;
            cout << "------------------------" << endl;
            cout << "1.Yes\nOtherwise No" << endl;
            cout << "------------------------" << endl;
            choice = inputInteger();
            if (choice == 1)
                exit = true;
            else
                exit = false;
            break;
        default:
            cout << " #######################" << endl;
            cout << " #    Invalid input!   #" << endl;
            cout << " #######################" << endl;
            system("pause");
            break;
        }
    } while (!exit);
}
else {
    cout << "No login. System exit. " << endl;
    cout << "--------------------------------------" << endl;
    return 0;
}
```

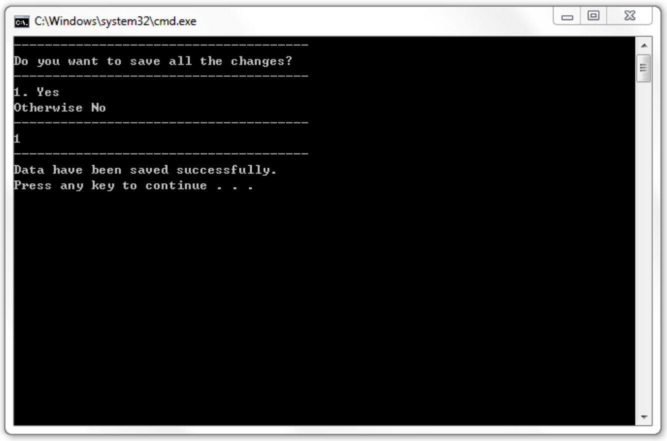
```
C:\Windows\system32\cmd.exe
 ----------------------------------------------------------
 |                                                        |
 |                       Welcome !                        |
 |               Nice to see you again !                  |
 |                                                        |
 |               Customer: ziheng zhang !                 |
 |               Total goods: 2 items                     |
 |               Total fee: 51.18                         |
 |                                                        |
 ----------------------------------------------------------

Please choose function:
------------------------------
1. Check in new goods
2. Check & modify goods information or user information
3. Log out
------------------------------
3

Are you sure to exit?
------------------------
1.Yes
Otherwise No
------------------------
```

Figure 64: main function test Customer log out

When customer wants to log out the system, the system will require customer to ensure exit in case of mistakes (Figure 64).



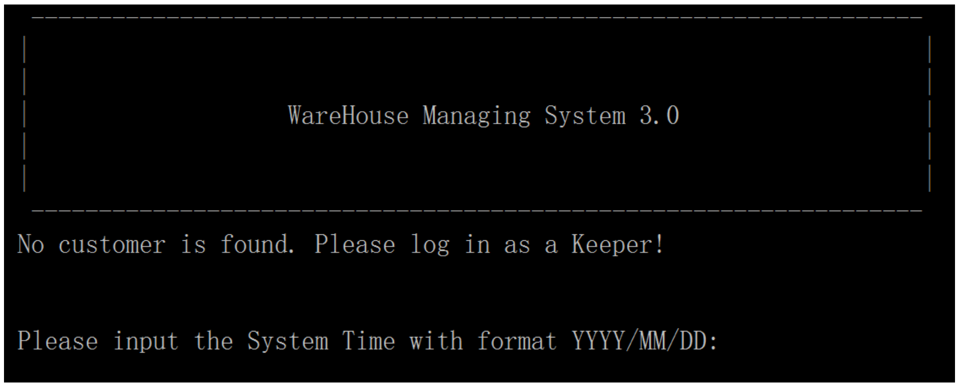
```
    system("cls");
    cout << "--------------------------------------" << endl;
    cout << "Do you want to save all the changes? " << endl;
    cout << "--------------------------------------" << endl;
    cout << "1. Yes\nOtherwise No" << endl;
    cout << "--------------------------------------" << endl;
    choice = inputInteger();
    switch (choice) {
    case 1:
        db.saveData();
        cout << "--------------------------------------" << endl;
        cout << "Data have been saved successfully." << endl;
        break;
    default:
        cout << "--------------------------------------" << endl;
        cout << "You have not saved the changes." << endl;
        break;
    }
    return 0;
}

int inputInteger()
{
        string choice;
        int i;
        getline(cin, choice);
        stringstream ss;
        ss << choice;
        ss >> i;
        return i;
}
```



```
C:\Windows\system32\cmd.exe
--------------------------------------
Do you want to save all the changes?
--------------------------------------
1. Yes
Otherwise No
--------------------------------------
1
--------------------------------------
Data have been saved successfully.
Press any key to continue . . .
```

Figure 65: main function test Customer data saving

After user is determined to exit the system, the system will ask user whether to save all changes and write to the files. This function will be operated in class DataBase, which includes opening the files, writing to the files and closing the files. If user chooses 1, the operation will be conducted and changes will be saved to the files. Otherwise, the system will reject to save changes (Figure 65).



```
 -------------------------------------------------------------------
|                                                                   |
|                                                                   |
|                   WareHouse Managing System 3.0                   |
|                                                                   |
|                                                                   |
 -------------------------------------------------------------------
No customer is found. Please log in as a Keeper!


Please input the System Time with format YYYY/MM/DD:
```

Figure 66: main function test no customer

Figure 66 shows the instruction to log in as a Keeper if there is no customer being read. It may be caused by the missing file when read. It may also because all the customers are deleted be Keeper.

Debugs report

The sub pointer vector and inheritance render demanding debug job. Throughout test, all the bugs found were successfully solved. There were 4 typical bugs that worth emphasizing.

Pointer and vector

This bug happened when the sub pointer vector, of which the elements are pointers pointing to the elements of original vector stored in database. When the original vector is altered, the memory address may be changed as well, which will also affect the sub pointer vector. To solve that, whenever the original vector is modified, the corresponding sub pointer vector will be cleared and reset.

0 or delete

This bug happens when showing customer/goods that has been deleted. The rule defines the delete of customer and goods is to set their id to 0. The real erase them from the original vector is when the file is saved. To solve that, some statements were written to reject the items with id equals to 0.

Moreover, erase customer and goods with 0 id at the beginning of addNewGoods and the end of deleteCustomer function.

Endl

When writing a file, the endl at the last line will cause an empty line, which will cause trouble when read the file next time. To solve that, endl is removed for the last line to be written of each file.

Include

This bug happens when original decide to realize all the user functionalities through their own class, which will include database class to user class. It causes loop to include itself, which is end up with the error that the User is not declared. To solve that, functionalities are realized by both customer/keeper class and database class.

User Manual

Login

(For the convenience of testing the program, the user can type in the system time in the format of “2012/12/25” at the beginning) Before getting started, the user should login the system by typing in the username and the password. The user is allowed to exit the program when they input wrong information. If there is no customer account, the system will show the recommendations to log in as a Keeper. The keeper can create accounts if needed.

Keeper

The keeper (Username: Keeper, password: 12345) has the authority to

1. Create an account,

The keeper can directly type in the name and the password of the user to create a new account

2. Delete customer,

By typing in the corresponding ID of a user, the keeper can remove the account of a user directly.

3. Modify account information,

The keeper has the authority to change the username or the password of himself/herself.

4. Display all the costumer’s information,

The keeper could check all the customer information, but could not modify them.

5. Display all the warehouse information,

The keeper could also check all the warehouse information, seeing how the containers are occupied in the four warehouses.

6. Search container,

The keeper could search a container to see whether it is empty or not. If it is not empty, it would display the id of the goods which is stored in it.

7. Search goods,

The keeper could search goods to read their information, while has no authority to modify them.

8. Search customer.

The keeper could also search customer information, without authority to modify them.

Customer

The customer has the authority to

1. Check in new goods,

The customer could check in new items by typing in the size, the start time, end time and the good information.

2. Check out current goods,

The customer could check out current items by typing in the corresponding goods ID.

3. Modify the current goods,

The customers could change the goods information by themselves.

4. Browse the current goods,

The customer could display all the current goods stored in the warehouses.

5. Modify account information,

The customer could modify his/her own account information.

Exit and data saving

The user can exit the program by choosing log out. The system will let the user choose whether to save the data.