**CS 311 - Project Milestone 3**

**Term 1 – 2018**

**Project Report**

**Class Group #: 3                                                                    Project Group #: 3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Name** | **ID** | **Role Participated In** | **Sign** |
| 1 | Jumana AlGhamdi | 2180006622 | -Shelve class .  -participate in writing the code in the main class.  -debugging the code. -writing the project report. |  |
| 2 | Sumaya Aljaafary | 2170004887 | -Book class  -participate in writing the code in the main class.  - debugging the code.  -writing the project report. |  |
| 3 | Maryam Jamal | 2170008191 | -Person superclass.  -participate in writing the code in the main class.  - debugging the code.  -writing the project report |  |
| 4 | Zainah Alhaddad | 2170001944 | -Borrower subclass  -participate in writing the code in the main class.  - debugging the code.  -writing the project report. |  |
| 5 | Latifa Alolayan | 2170000580 | -Owner subclass  -participate in writing the code in the main class.  - debugging the code.  -writing the project report. |  |
| 6 | Doaa Saleh – Saleh | 2170007785 | -Buyer subclass  -participate in writing the code in the main class.  - debugging the code.  -writing the project report. |  |



Kaff platform

(A platform for exchanging books

between CCSIT students).

**Introduction**

We are a team of six members decided to design and code a system for (Kaff Platform) which is an initiative of Student Activity Unit in Computer Science College - female section. The initiative is "exchange books". Its goal is to help students each other to get the books they need. Especially, the prices of university books are generally considered high and many of the students are unable to buy books. So, the students can get the books by this platform with affordable.  
In addition, another goal is to recycle books through this platform, where these books are in excess of the need of some students. So it is best to get rid of them in an appropriate way by giving them to students. Through this platform, students can buy and sell books as well as borrow and then return it. Our system contains six class tree of them are subclasses of (Person) superclass, every one of us had designed a class herself then we compared our work together then we made connections between them and finally, we wrote the main method.  
Our system contains six classes, three of them are subclasses of (Person) superclass, every one of us had designed a class herself then we combined our work together then we made connections between them and finally, we wrote the main method.

**UML Diagrams:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Book** | | - title: String  - bookID: String  - owner: Person  + level: int  - availability: Boolean  +NUMBER\_OF\_BOOKS: int  + courseID : String  Status : String  Price : double | | + Book( )  + Book(title: String, owner: Person, level: int, edition: int, availability: boolean)  + setTitle(title: String): void  + getTitle( ):String  + setBookID(bookID: String): void  + getBookID( ): String  + setOwner(owner: Person): void  + getOwner( ):Person  + setAvailability (availability: boolean): void  + getAvailability ( ) : boolean  + toString( ): String | | |  | | --- | | **Shelve** | | +booksForBorrow:ArrayList< Book>  +booksForSale: ArrayList<Book> | | +Shevle()  +addBook(book:Book) : boolean  +removeBook(book:Book) : boolean  +isBookInShelve(Book b) : boolean  +getBooksOwnedBy(student:Person) : ArrayList<Book>  +displayAllBooks() : void  +getSaleBooks() : ArrayList<Book>  +getBorrowBooks() : ArrayList<Book>  +getBookByLevel(level:int) : ArrayList<Book>  +getBookByName(name:String) : ArrayList<Book>  +getBook(bookID:String) : ArrayList<Book> | |

|  |
| --- |
| **Person** |
| -name : String  -academicNum : String  -level : int  -phone : String  -email : String |
| +Person(name:String, academicNum:String, level:int, phone:String, email:String)  +getName() : String  +getAcademicNum(): String  +getLevel(): int  +getPhone(): String  +getEmail() : String  +setName(sName : String) : void  +setAcademicNum(academicNum:String) : void  +setLevel(level:int) : void  +setPhone(phone: String) : void  +setEmail(email : String) : void  +toString():String |



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Owner** | | -gotPaid: boolean  -graduating: boolean  - bookCount: int | | +Owner(name: String, ownerID: String, phone: String, level: int, email: String, gotPaid:boolean, graduating:boolean, bookCount: int)  +setGotPaid(gotPaid: boolean): void  +getGotPaid(): boolean  +getBookCount(): int  +setBookCount(x: int): void  +incrmntBookCounts():void  +decrmnntBookCount(): void  +toString: String | | |  | | --- | | **Buyer** | | + pay : Boolean | | +Buyer(BuyerName : String, IDnumber : int, level : int, phone: String, email:String, pay:boolean)  +setPay(p:boolean):void  +getPay(): Boolean  +toString( ): String | | |  | | --- | | **Borrower** | | - returnedState : Boolean  - returnDate : Date | | + Borrower(BorrowerName: String, IDnumber:int, level:int, phone:String, email:String, returnDate:Date)  +getReturnedState(): Boolean  +getReturnDate(): Date  +setReturnedState(s:boolean): void  + setReturnDate(d:Date): void  + toString( ): String | |

**DIFFICULTIES & HOW PROBLEMS WERE SOLVED:**

Because we’ve implemented polymorphism as an array list of type Person that’s name user:

ArrayList<Person> user = new ArrayList<>();

we have faced some difficulties regarding getting only owners or borrowers or buyers as needed, we also had an issue regarding determining the position of the current owner or buyer or borrower in the list. However, we were able to overcome the first issuer by creating three functions that receive the user array list and then use the instanceof operator to search for only objects of type owner or borrower or buyer and returns them as an array list.

public ArrayList<Owner> getOwners(ArrayList<Person> user)

ArrayList<Buyer> getBuyers(ArrayList<Person> user)

public ArrayList<Borrower> getBorrowers(ArrayList<Person> user)

As for the second issue, we used three variables to determine the index of the current owner or buyer or borrower that were all initialized by -1 when the user starts the User System,

int currentOwner= -1

int currentBuyer= -1

int currentBorrower= -1

and the updated to the correct index when creating an object of each class, for example, when creating an object of type Owner the currentOwner variable will be updated to user.size()-1: currentOwner=user.size()-1

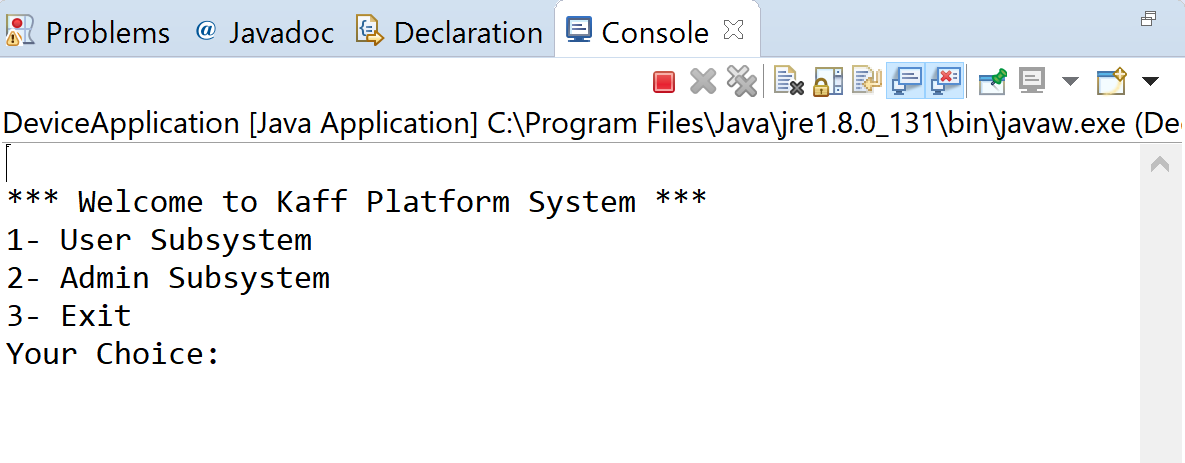
We also had a problem to determine the book id to make sure its unique enough so we make this method to generate it :

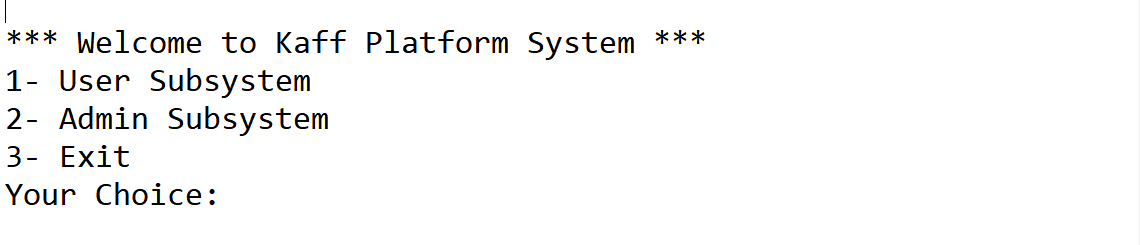
bookID=noBooks+courseID;

**CHECKLIST OF REQUIREMENTS FULFILLED IN THE PROJECT:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Function**  **/Application Aspect** | **UML Datagram** | **Classes Management** | **Inheritance & Polymorphism** | **Array Of Objects** | **Program Structure** | **Error Handling** | **Analysis** |
| **Person superclass: Buyer, Borrower, Owner subclasses.**  **Book class**  **Shelve class** | **✔** | **✔** |  | **✔** |  |  |  |
| **KaffPlatformSystem (Main) class** |  |  | **✔** | **✔** | **✔** | **✔** |  |
| **Get User Information** |  |  |  |  | **✔** | **✔** | **Asking the user to enter his information when he enters the user system.** |
| **The user adds a book** |  |  | **✔** | **✔** | **✔** | **✔** | **If the current user adds a book for the first time then creates an object of type Owner and add it to user arraylist.**  **Then the user is asked to enter the book information and an object of type Book is stored in the shelve.** |
| **User buys a book** |  |  | **✔** | **✔** | **✔** | **✔** | **If a current user buys a book for the first time then creates an object of type Buyer and add it to user ArrayList.**  **Book availability is set to false.** |
| **User borrows a book** |  |  | **✔** | **✔** | **✔** | **✔** | **If current user borrows a book for the first time then creates an object of type Borrower and add it to user ArrayList.**  **Book availability is set to false.** |
| **Borrowers return a book** |  |  |  |  | **✔** |  | **Book availability is set to true.** |
| **Display books** |  |  |  |  | **✔** |  | **Books are displayed by status(sale/borrowed), level,title.** |
| **User updates his/her info** |  |  |  |  | **✔** | **✔** | **Update the variable where the user was asked to enter at the first of the program, them checks if this user has been added to the list as an Owner or Borrower Or Buyer and updates the object.** |
| **The owner received his/her money** |  |  |  |  | **✔** | **✔** | **Admin enters the owner ID and then SetGotPaid=true** |
| **Admin removes a book** |  |  |  |  | **✔** | **✔** |  |
| **Display books owned by the user** |  |  | **✔** | **✔** |  | **✔** |  |
| **Update Book information** |  |  |  |  | **✔** | **✔** |  |
| **Display users information** |  |  |  |  | **✔** |  |  |

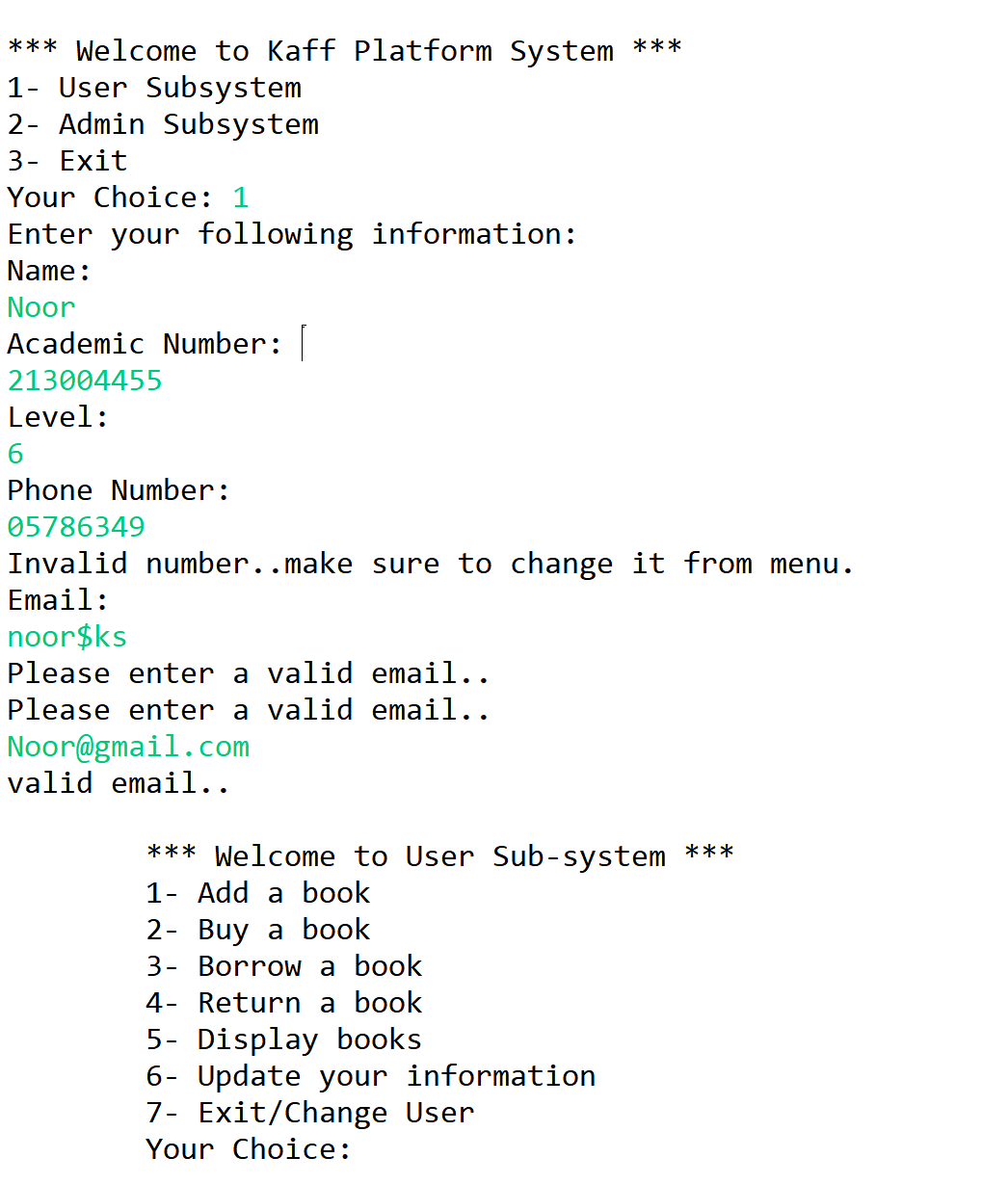
**Print screen of each interface**



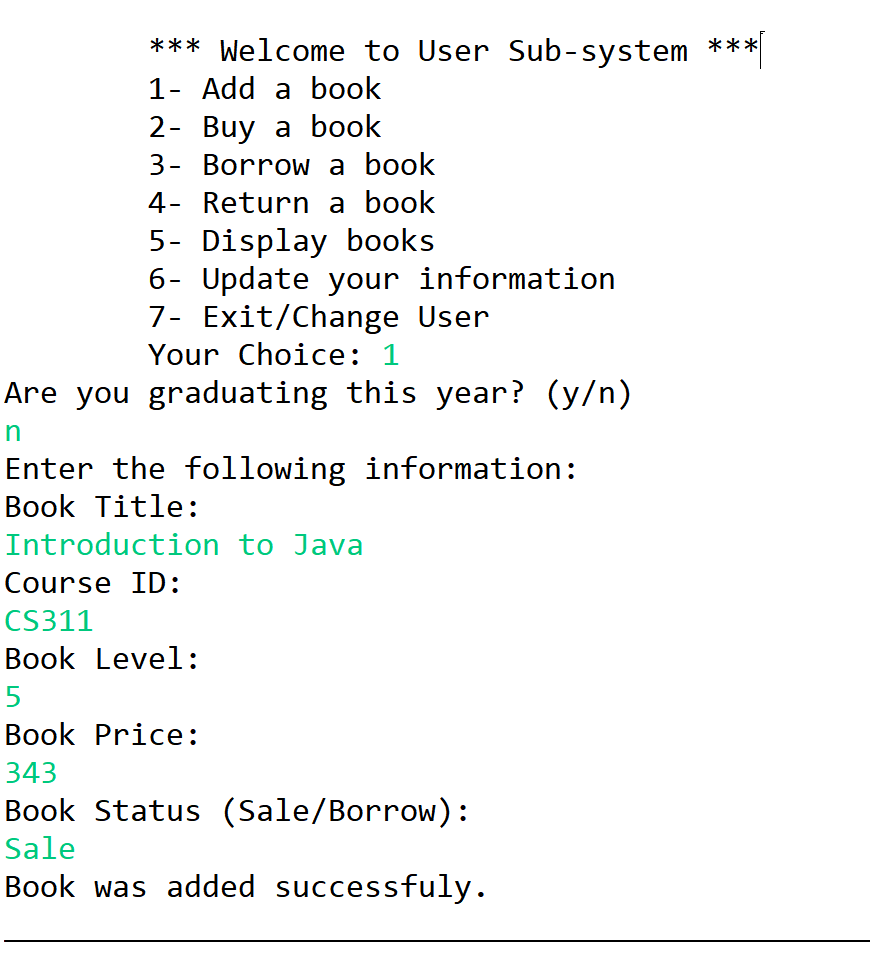


The system is running and the first menu is displayed.

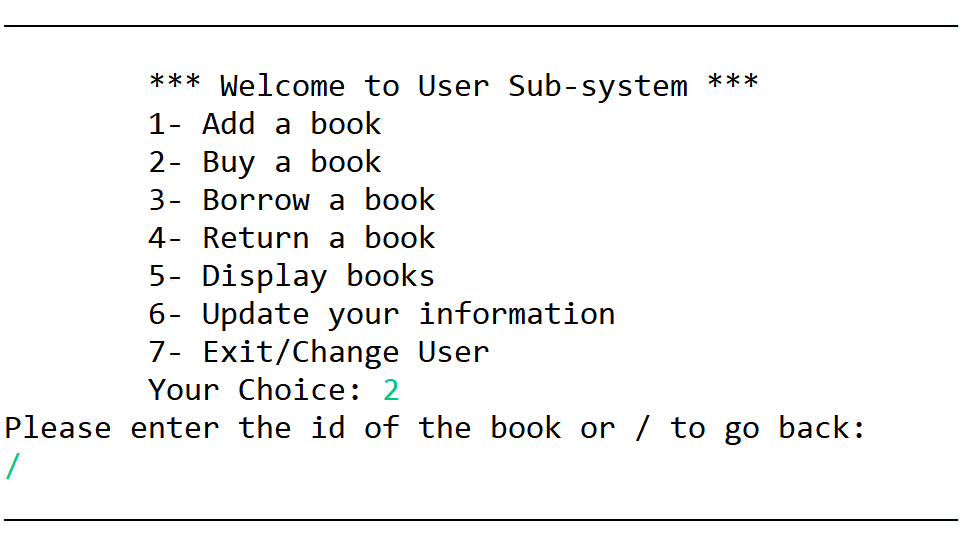
Then we go to the User Subsystem:



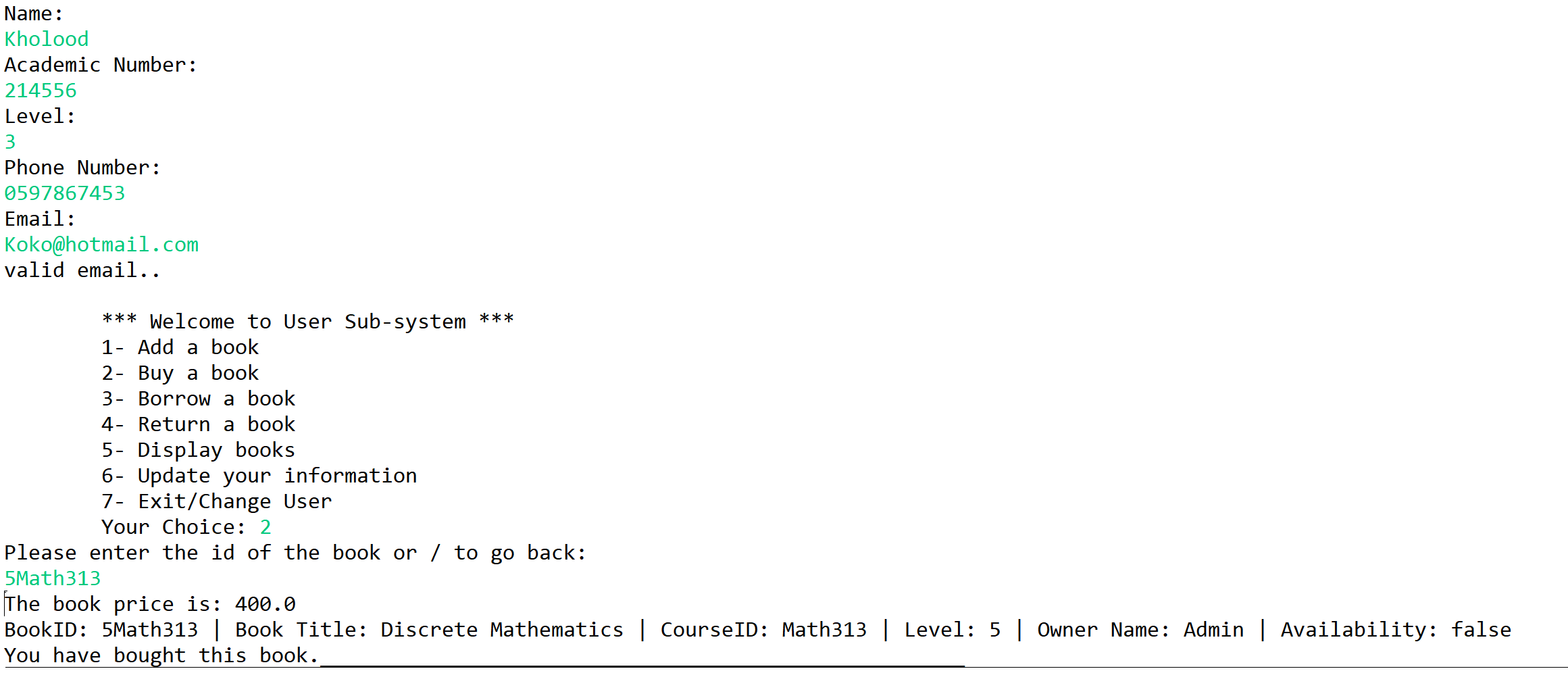
A user information is saved while error handling different user inputs.



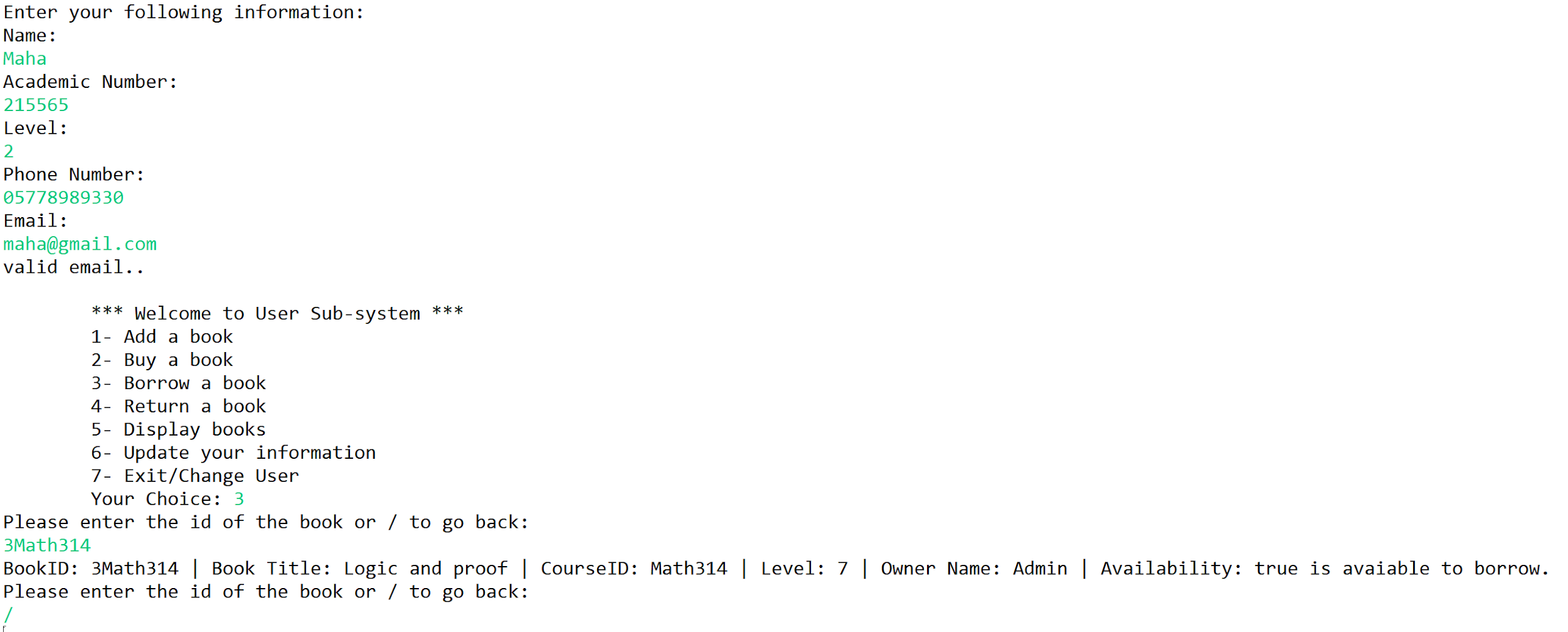
The user chose to add a book.



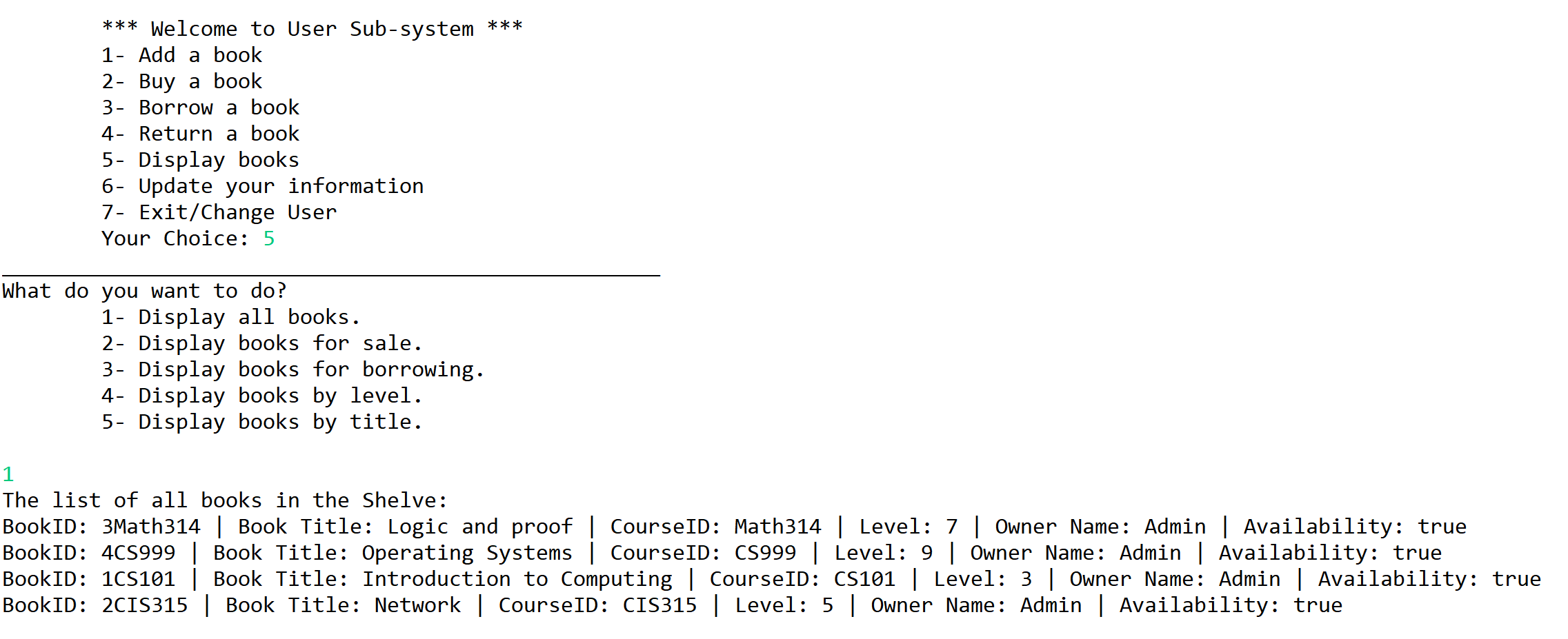
The user can get back if changes her mind from buying a book.



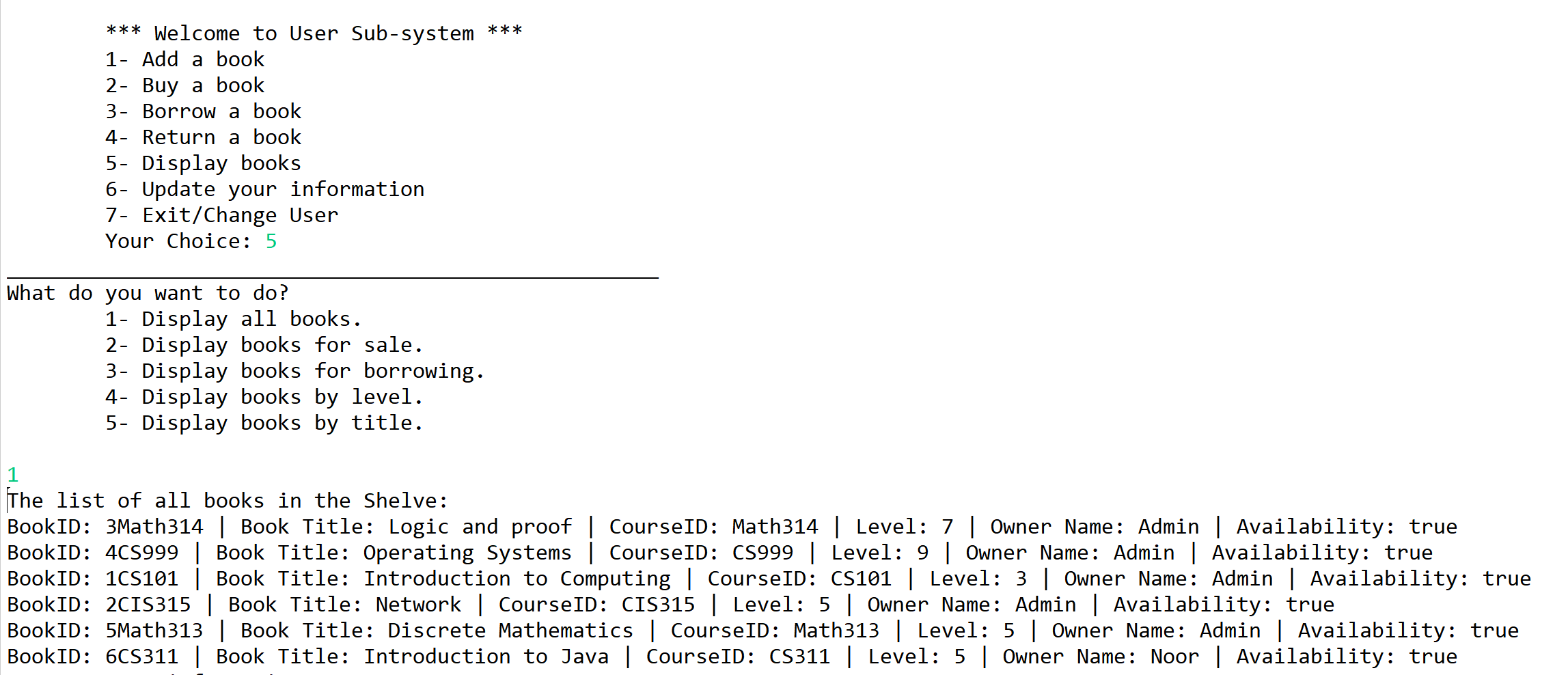
New user, buying a book.



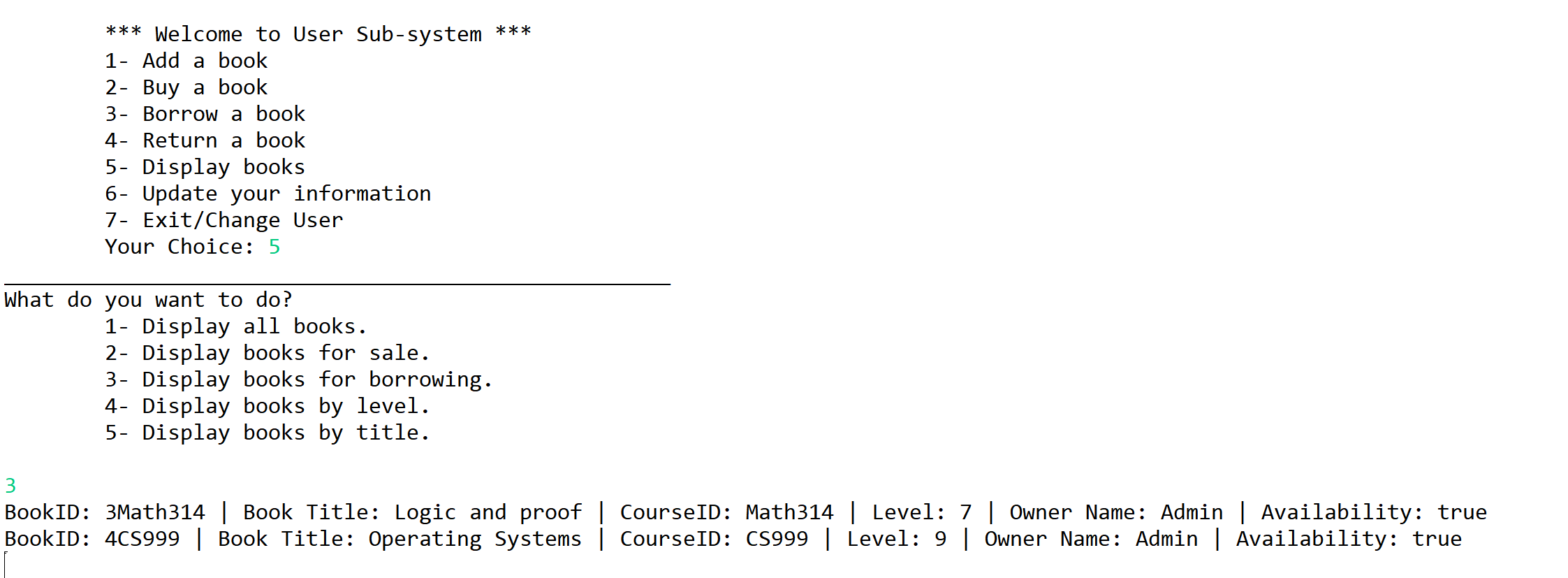
Another user borrowing a book.



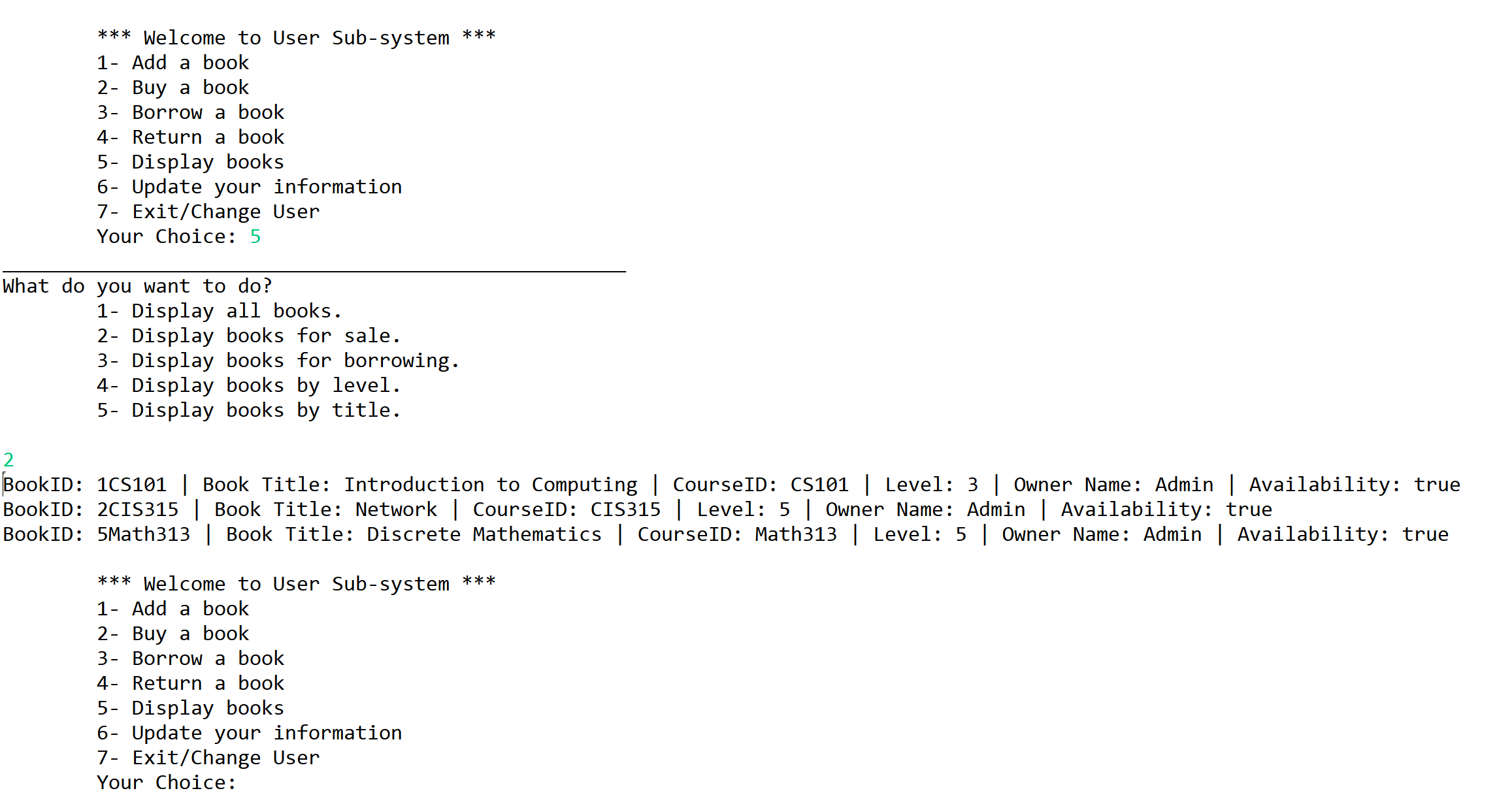
Display of all books before user adding any book.

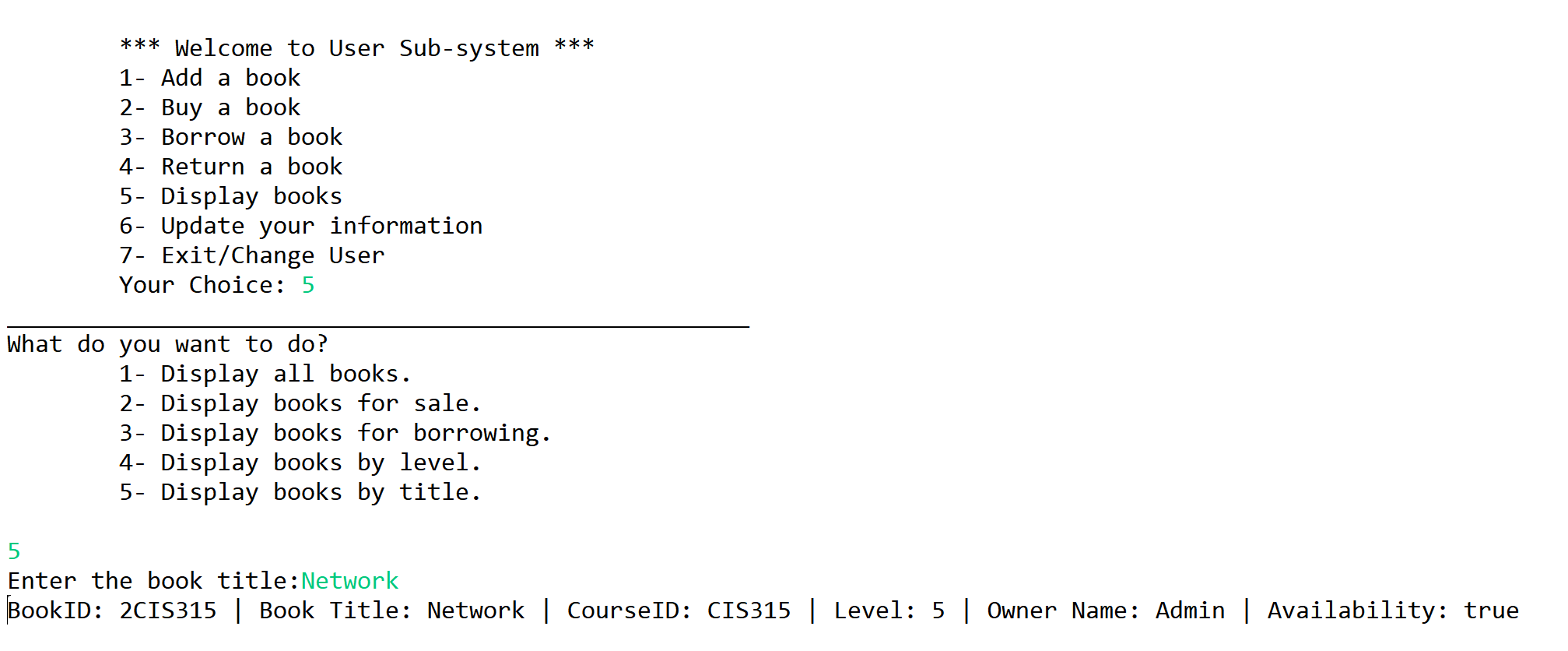
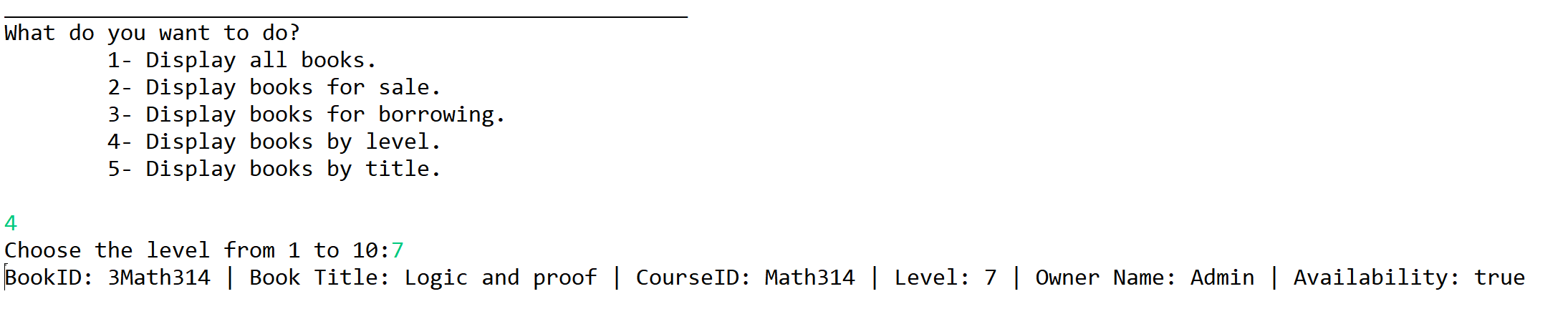


Display of all books after user addition.

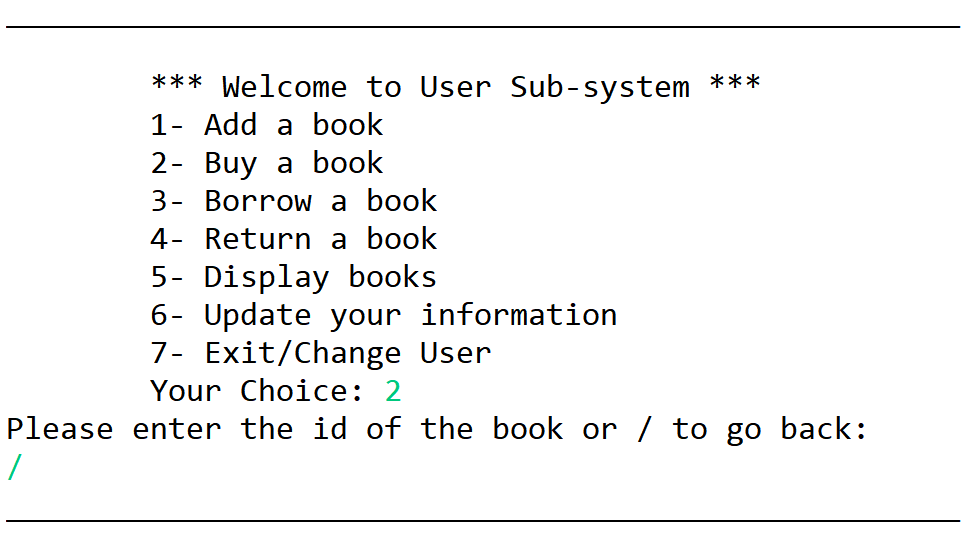


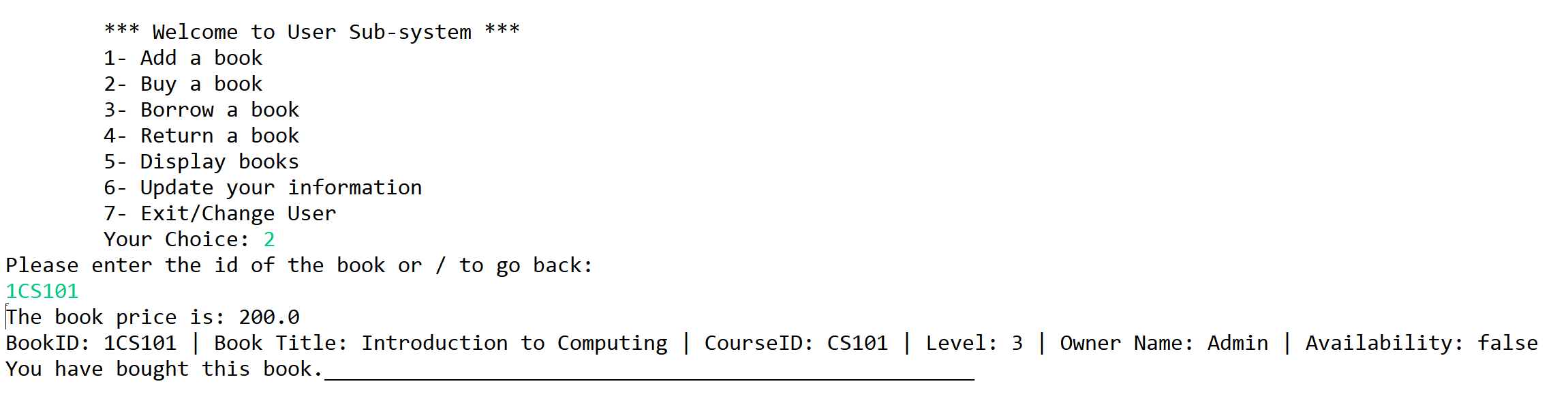
Display books for Borrow.

Display books for Sale.

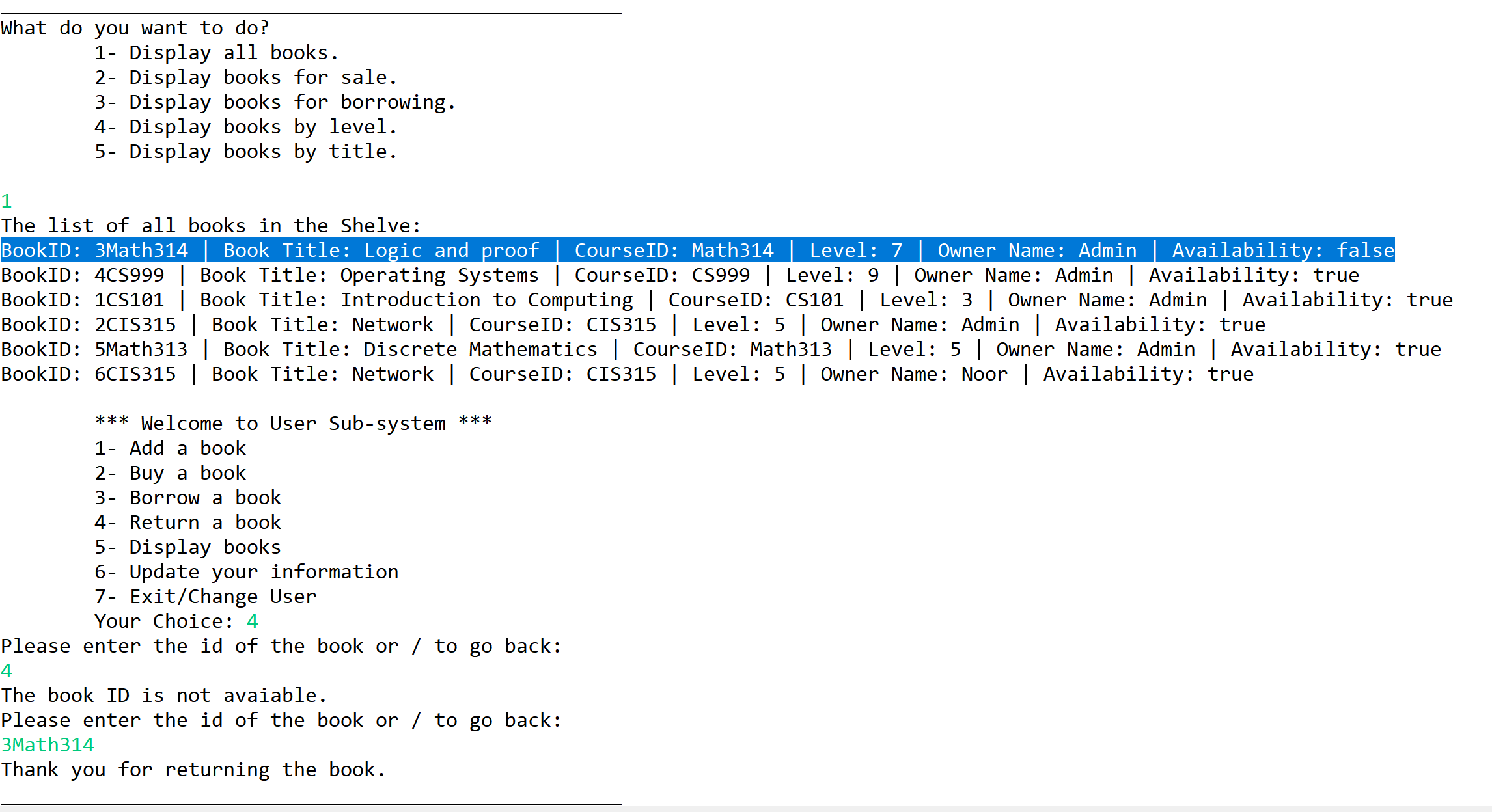


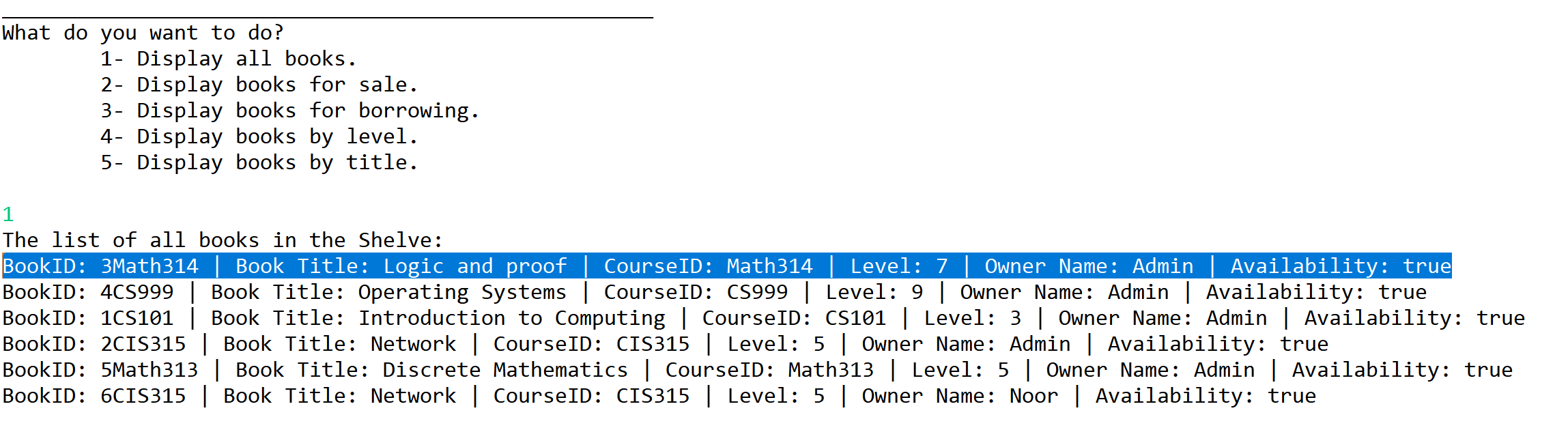
Display books based on title, and based on level for the book.



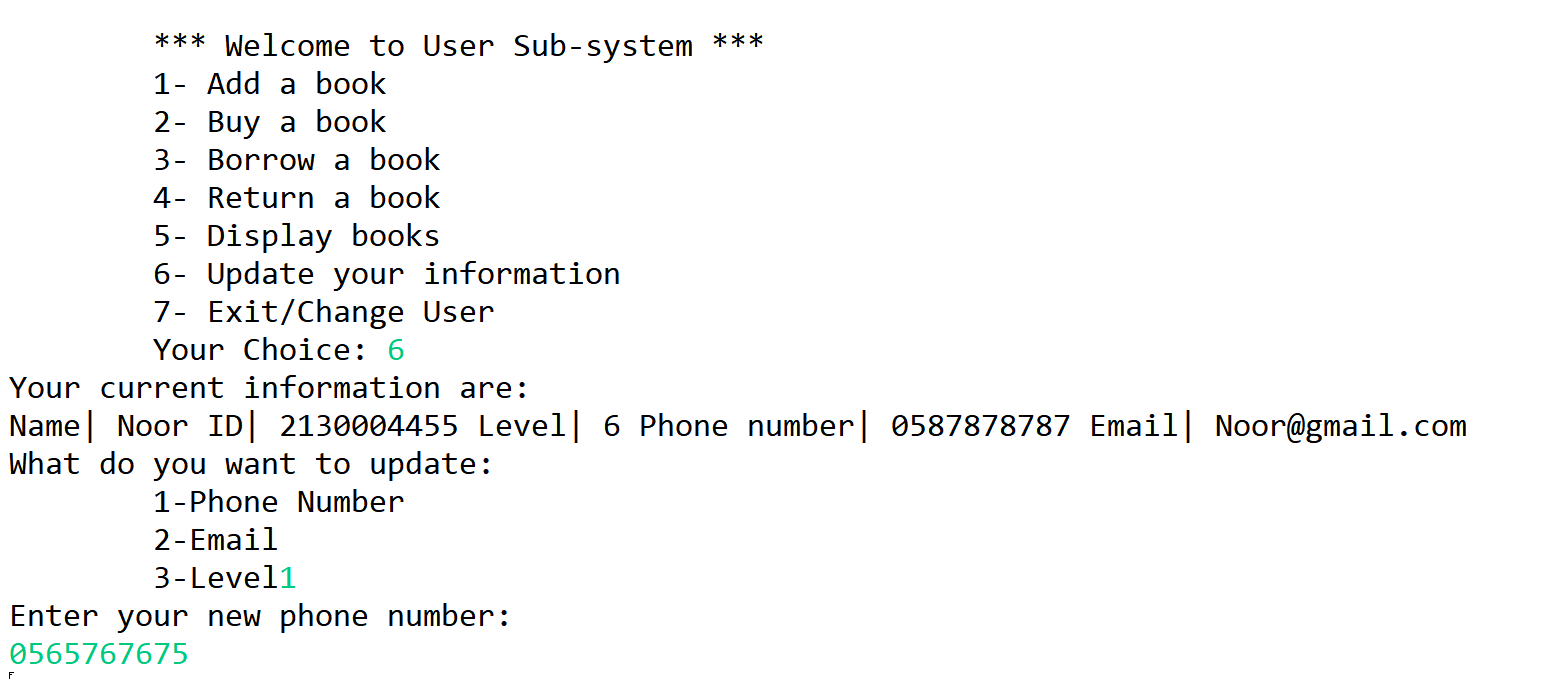


Buy a book have the option of going back and based on the book ID the user can choose to buy a book.

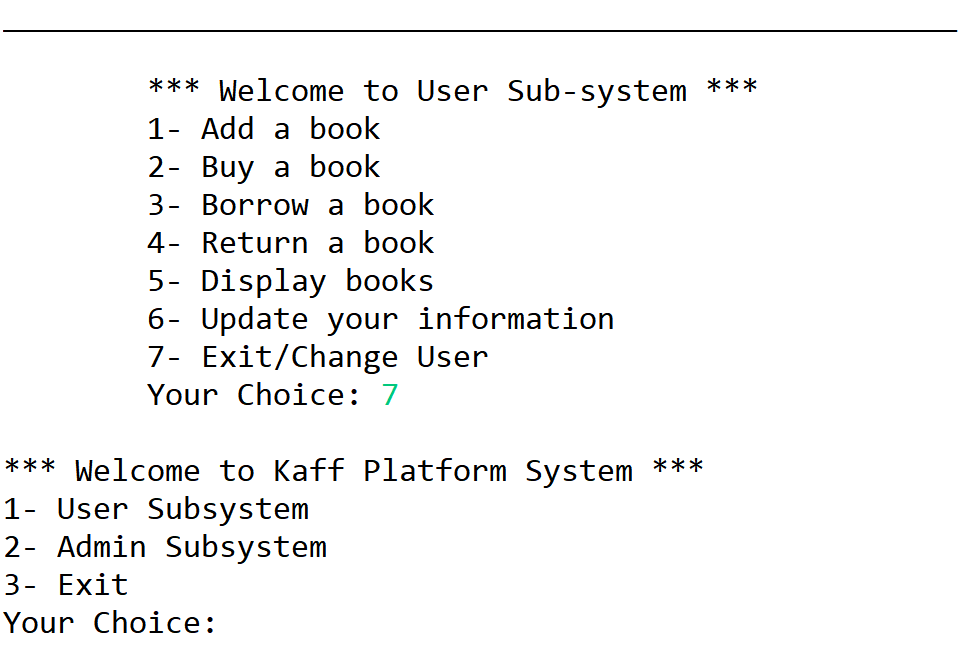




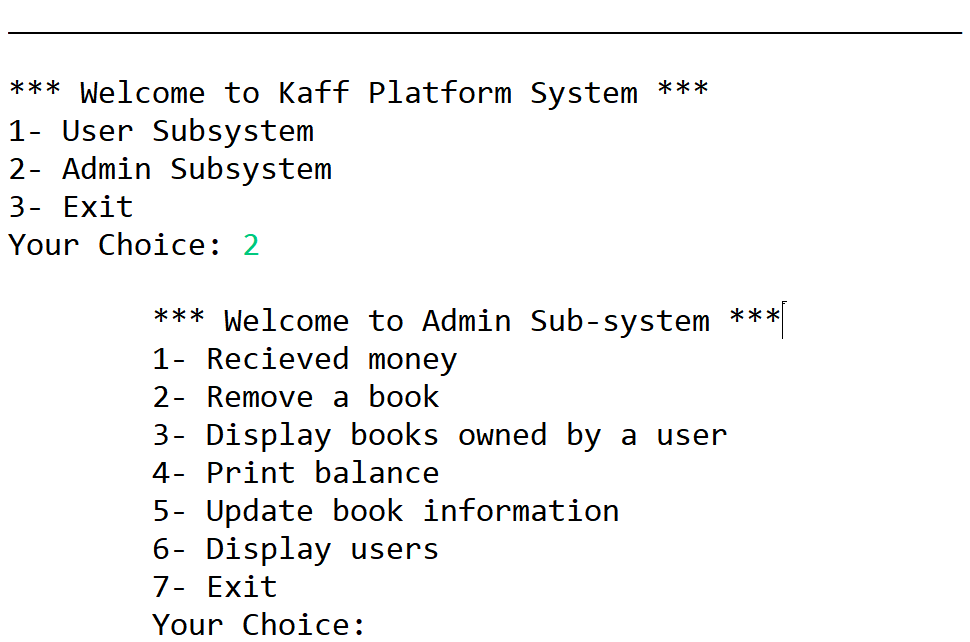
Here is a sample where the book was not available and after the user returns it, it changes its availability.



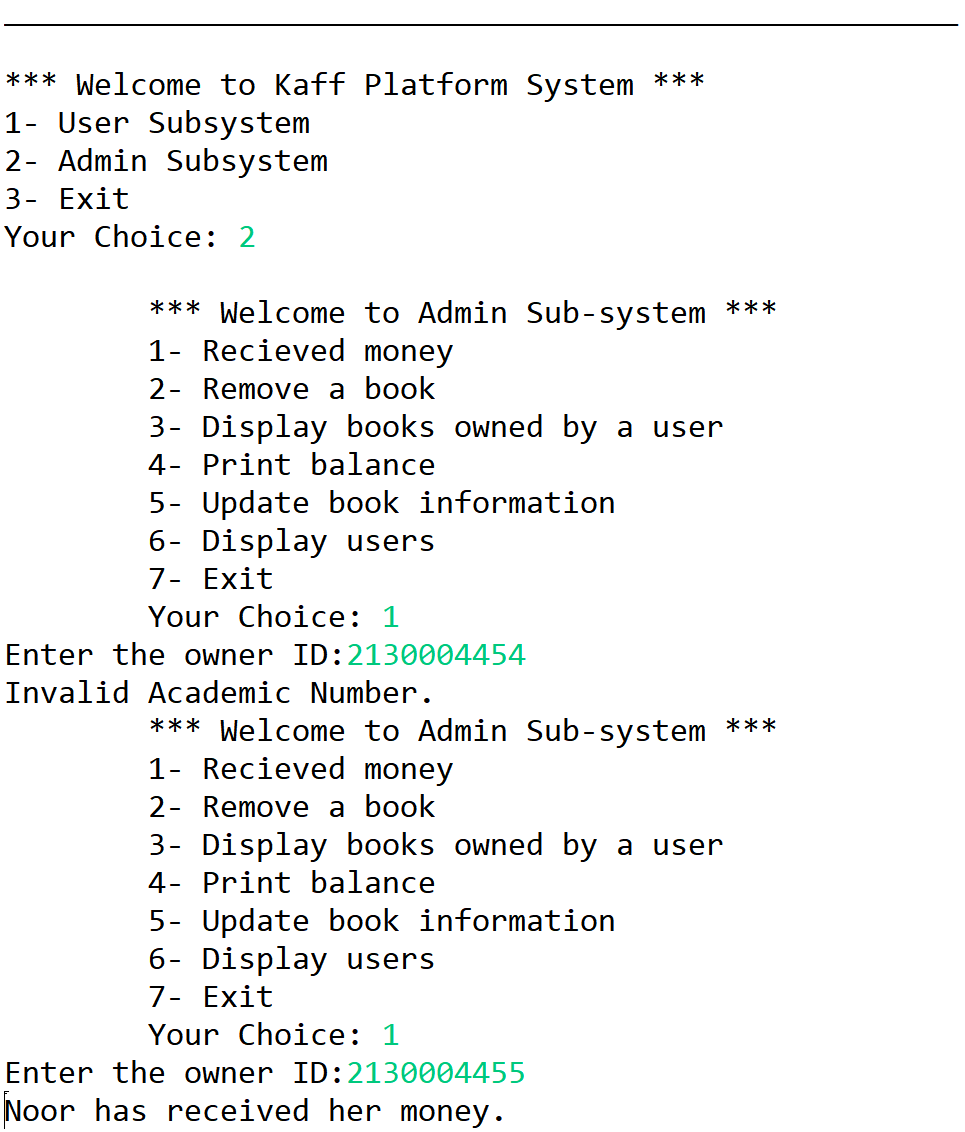
A user decides to update her information, in this sample the phone number is changed.



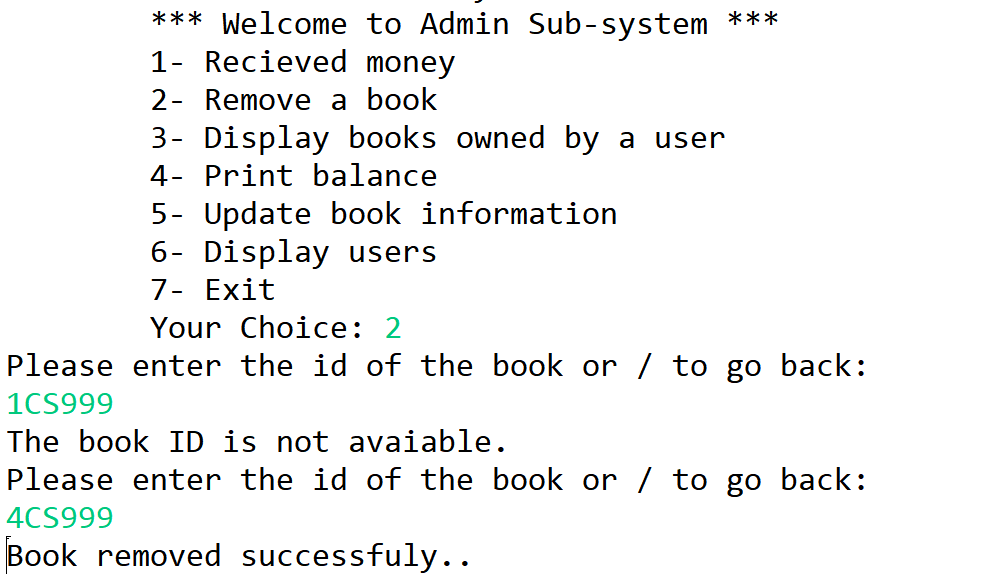
The user exited the Sub-System.



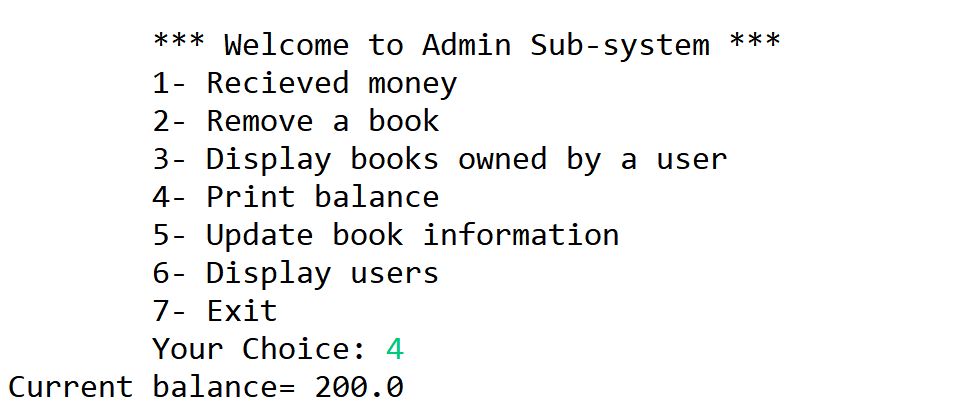
Menu of the Admin Sub-System.



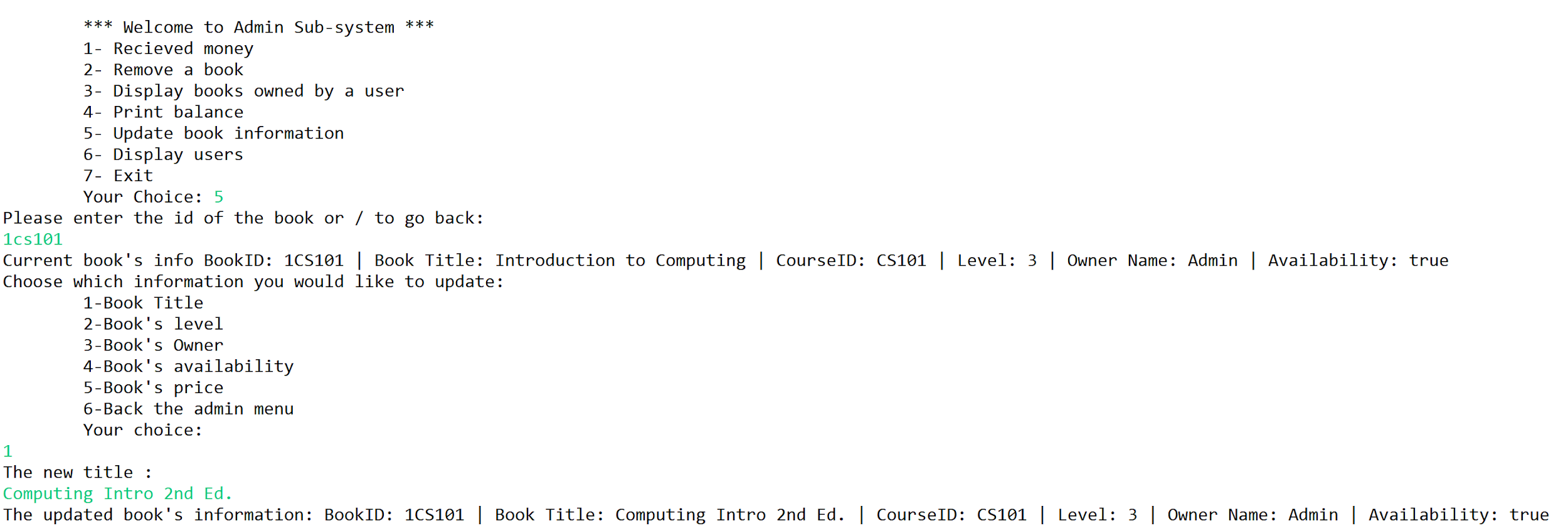
Admin Checks if money is received for a specific owner and the ID must be correct.

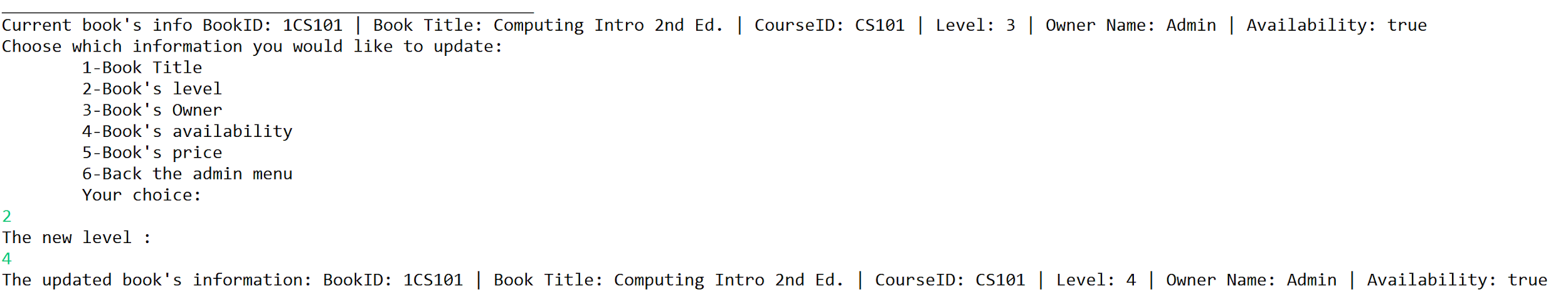


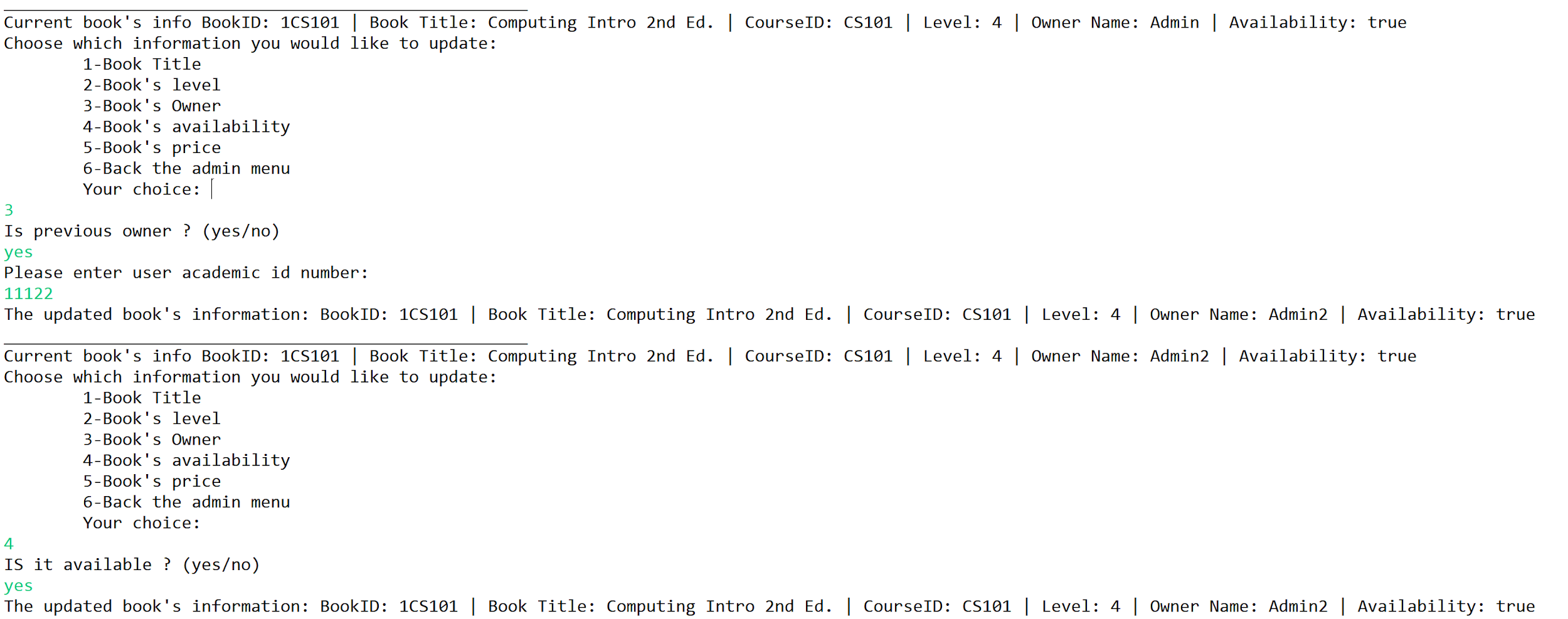
The admin removes a book from the shelve based on Book ID and availability.

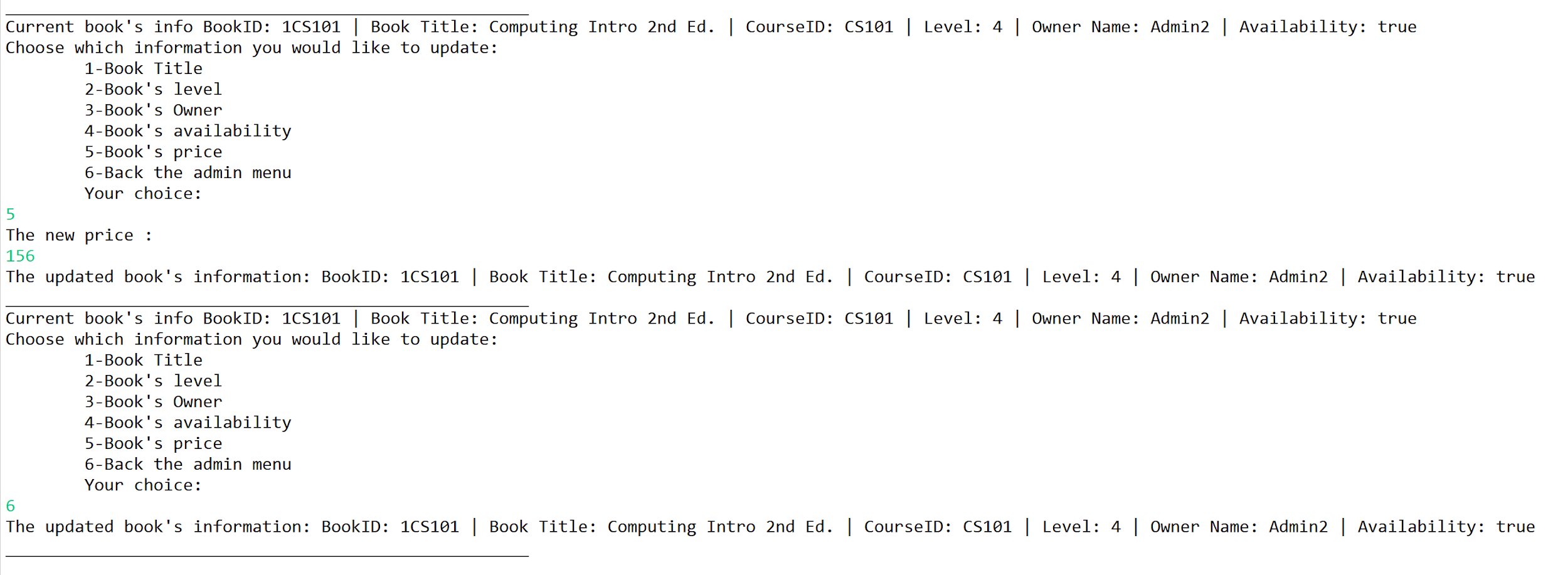


This option will return the amount of buy for the bought books in balance.

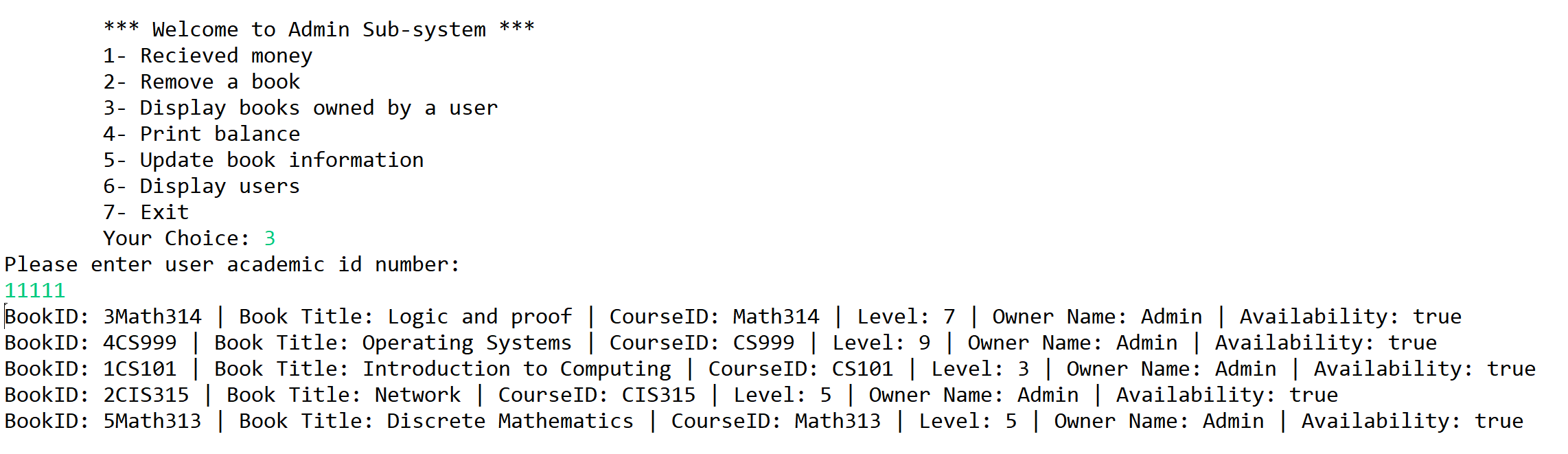




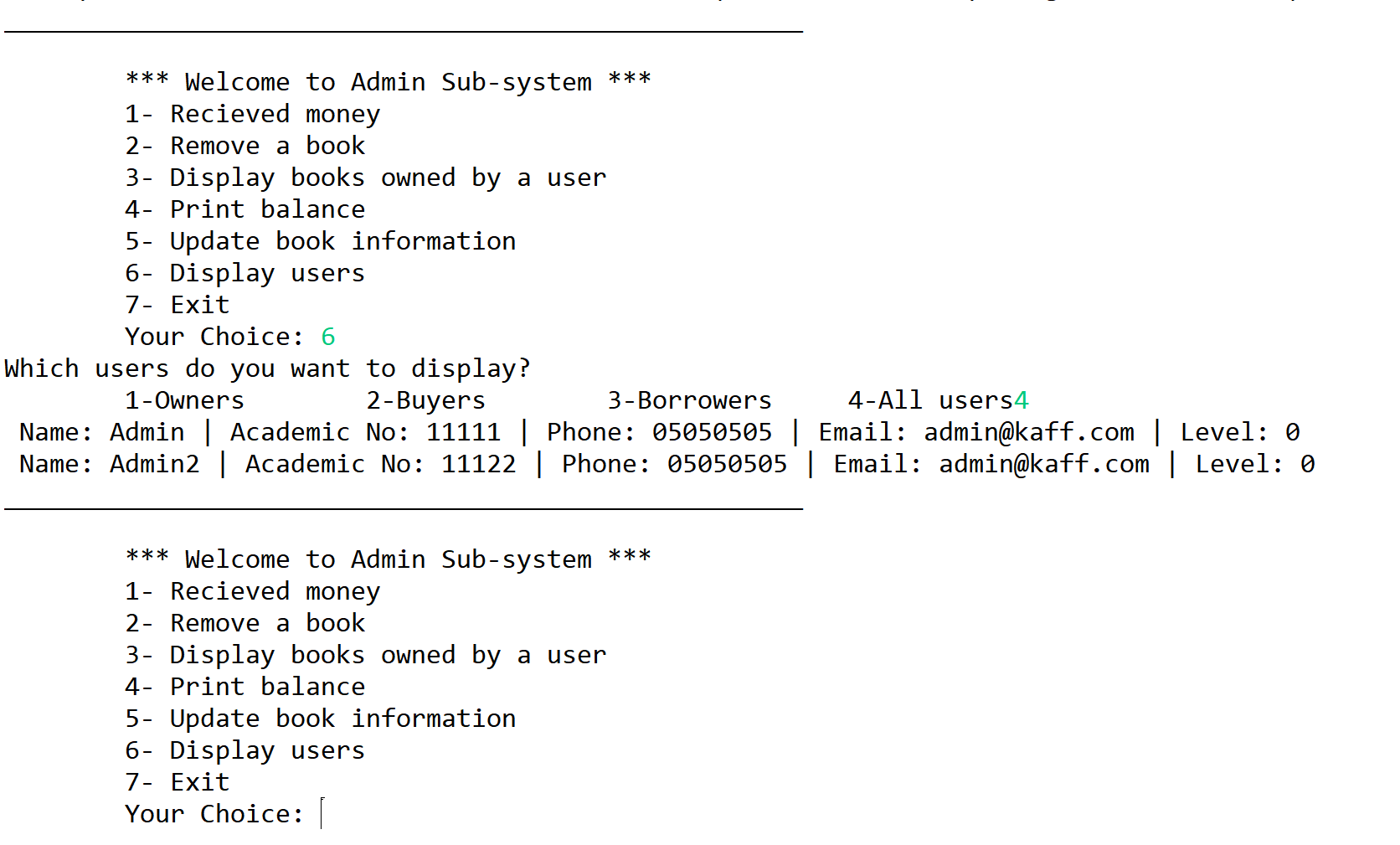


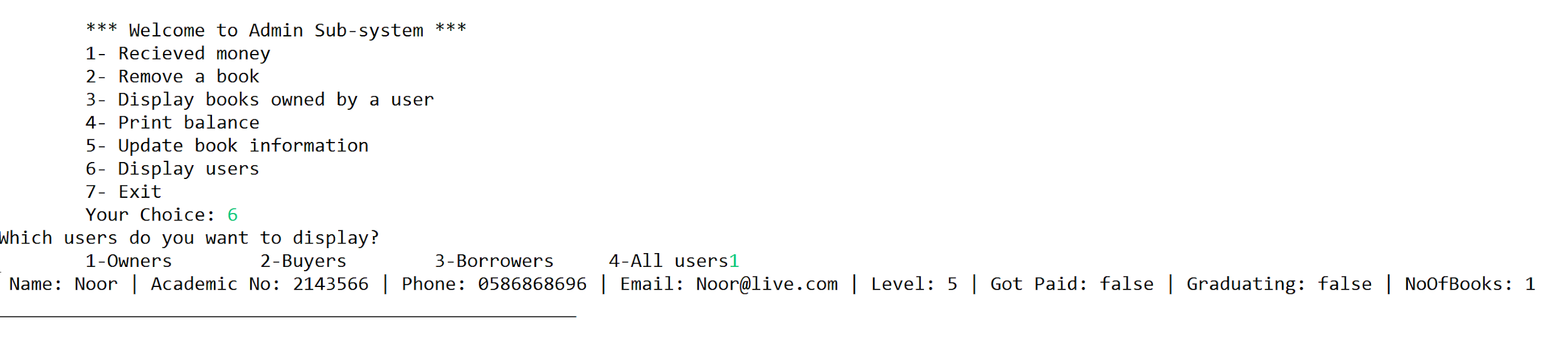
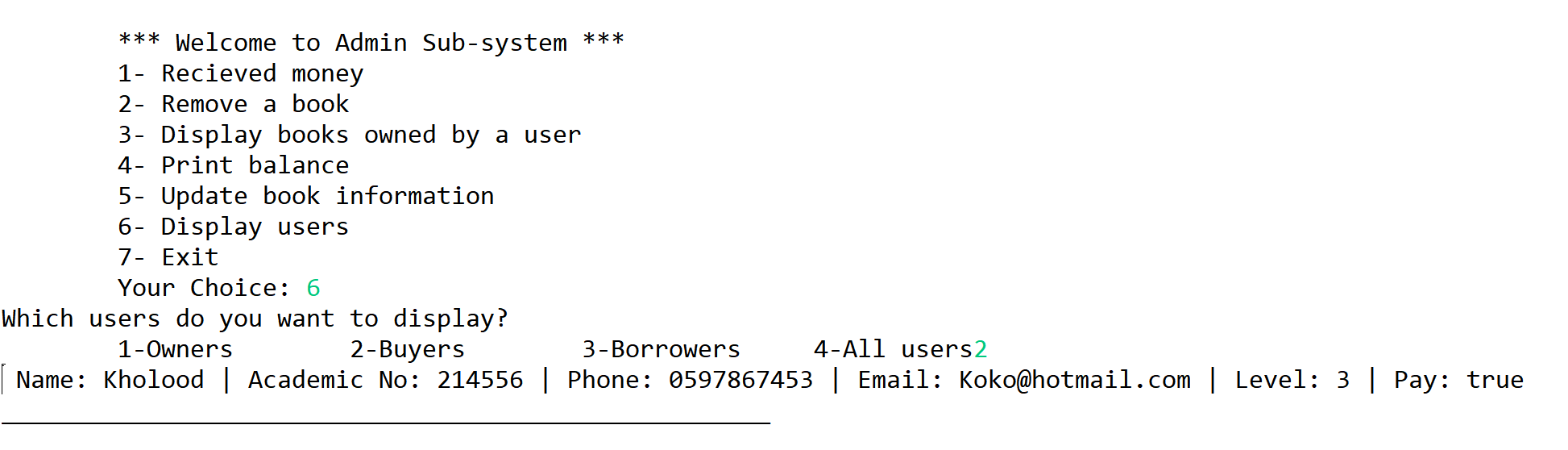
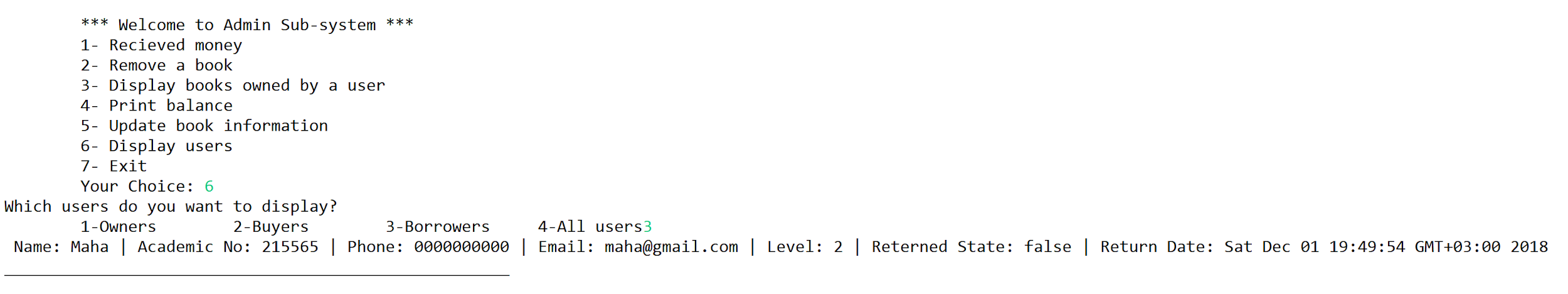
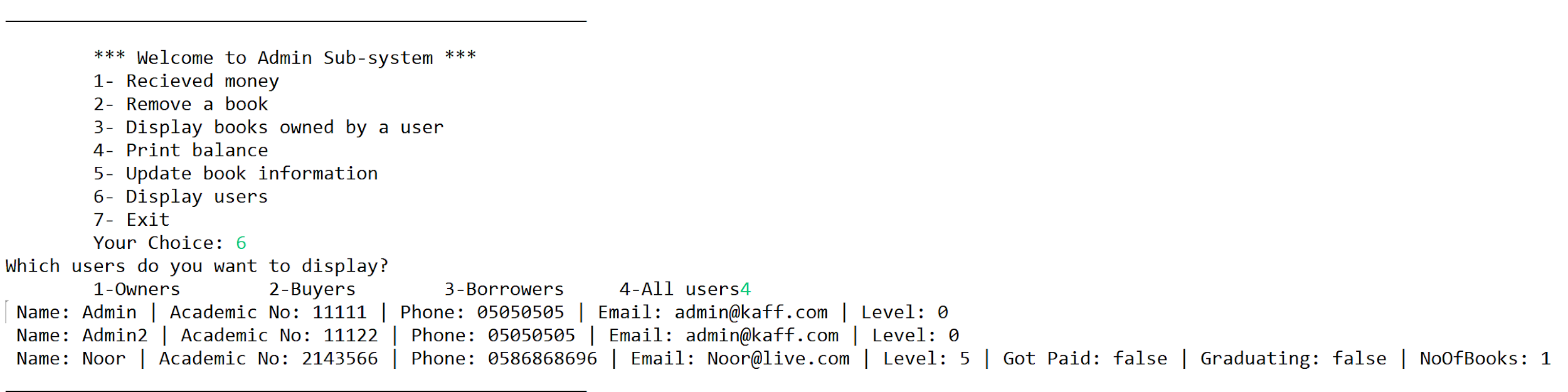


For updating the book information, title, level, owner, availability, and price. This code handels errors as well. Also it prints the book info before and after the changes.

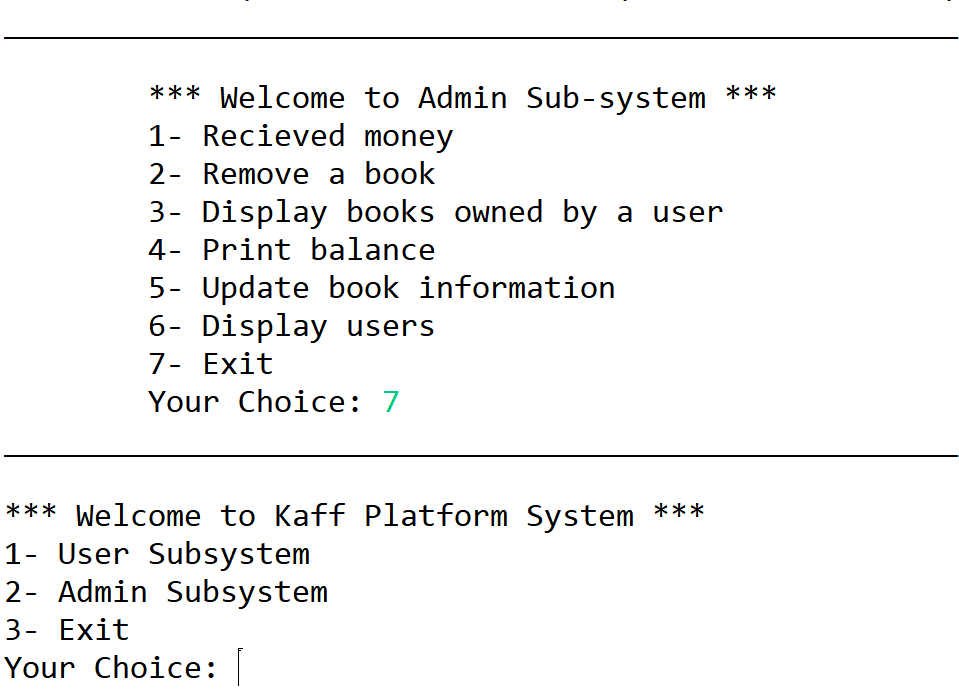


Here code display books based on owner of them entering the ID.

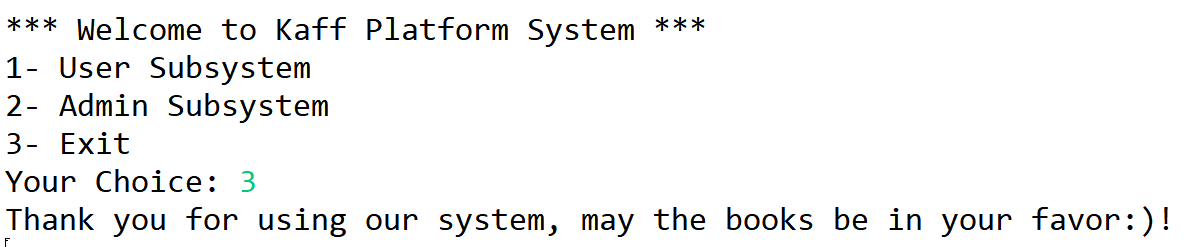




The admin can display the users based on the type of all of them.



Admin exits the Admin Sub-System.



Exiting the System.

**GENERAL COMMENTS:**

* We are really proud of the result we have of the project.
* Although we are proud of the result and the complexity of the system, we think we can further develop it with OOP2 using GUI and other skills and tools.
* The project helps us understand OOP by implementing it.
* We hope that one day it becomes a perfect system that we can use in the activity unit.