Department of
Aeronautical and Aviation Engineering
航空及民航工程學系

FACULTY OF
ENGINEERING
工程學院
WHERE CONCEPTS BECOME REALITY

ENG2002 Application Development Assignment Report

Group 7:
LIN Ju | 21106434D
QIN Qijun | 21101279D
NI Rouheng | 21102803D

The Hong Kong Polytechnic University
November 2022

ENG2002 Application Development Assignment

Report

Group 7 • November 2022

ENG2002 Application Development Assignment Report

Abstract

The project comprehensively incorporate the basic Python knowledge learned in class to ultimate in creating a multi-functional Python program, which achieves functions that fulfils and even exceeds the distinction's requirements of the instructions. It is worth noticing that the main program integrates a dual system of a "user registration platform" and a "task implementation platform" in a modular way, ensuring the complete functionality and efficient data retrieval of the database, while affirming each module's independency. Furthermore, GitHub is used to assure efficient collaboration of members in the completion of tasks online.

1. Introduction

1.1. Objective and Requirements

This assignment examines the wide range of programming knowledge learned in class. The instruction requires us to implement a phonebook to manage phone records using Jupyter Notebook. After the user performs the corresponding registration or login command, an interactive menu that recognizes the user's instructions and executes the corresponding module is populated, which will then navigate to the phonebook's main program. The phonebook requires two main objects called phoneBK and phoneRec. PhoneRec storages the main keys and items such as name, email, nickname, last call date, and phone number that the user has specified, and phoneBK contains several different modules, which is also the content of tasks, to realize the functions included but not limited to add, delete, check, and display phoneRec.

Our own interpretation of this assignment is that we want to create a user-friendly phonebook management system, which has two sub-systems, one for managing user accounts and one for managing each user's phone book, and these two systems are independent but interconnected. The previous sub-system gets the key to a certain user, which then be used to get the data of that user in the phonebook database. On this basis, we have implemented the interaction between the user and the program, which in turn has constructed this complete phonebook management system.

1.2. Methodology

We have assigned tasks according to our specialties and established a healthy seminar mechanism to facilitate efficient communication. We use collaborative platforms such as GitHub to facilitate simultaneous work. We primarily obtain the comprehensive knowledge required to the tasks' solutions from textbooks, supplemented by online encyclopedias, Google, GitHub and other forums. See *2.1. Research Approach* for more details.

For the design of the program, we have adopted a highly modular programming approach and created two systems, User Management System and Phone Book System, to assure the independence of each while maintaining the integrity of the entire program. This approach also enhances the readability of the code and reduces the cost of subsequent upgrades and debugging (if any). We have also done a lot of feasibility checks after the completion of the program to ensure its integrity and security. See *2.2. Architecture of Design, and 2.4. Demonstration of Feasibility* for more details.

We also encountered many issues during the compilation of the code. We have given the optimal solution within our ability through continuous learning and experimentation. See *2.3. Problem Solving* for more details.

2. Methodology

2.1. Research Approach

2.1.1. Task Distribution

Before each of us worked individually, we met several times to give our views on the main elements of the program. We discussed several key issues, such as specific processes and ways of collaboration, thoroughly before starting our respective programming.

QIN Qijun plays a decisive and primary role in the design and programming of the overall framework, as well as in the expansion regarding functionality of the application.

LIN Ju is mainly responsible for the tasks' accomplishment. He also tested the execution of the code and found some hard-to-find bugs.

NI Rouheng also contributes to the tasks, and offers financial support to our dining needs.

2.1.2. Methods of Collaboration

We had a discussion quickly after the task was delegated to break down the required items to be completed. We then established a logical sequence for our programming based on the listed items and drew a flow chart. In addition to this, we established a set of uniform naming rules to ensure that different modules could be cross-referenced rather than duplicated.

After the task assignment and discussion, we faced the main challenge of how to get everyone to collaborate efficiently on the code editing phase simultaneously. The solution we adopted was to use GitHub and Visual Studio for online collaboration. All three team members had their own branch, and we committed our progress to the master branch after finishing part of our coding, thus achieving efficient collaboration and synchronization of the codes.

At the initial stage of coding, we worked on building the structural framework of the entire program and then fill in functions accordingly. This highly modular approach to code editing makes our program independent and ready to be checked, updated, and expanded.

2.1.3. Methods of Research

Our team has established a "seminar" mechanism, which represented a innovative alternative to facilitate mutual communication among members. After the tasks were assigned, we researched the basic structure, construction and operation of databases using internet sources. We shared what we had learned and exchanged our progress at regular intervals each week. We also learned from other GitHub repositories and translated them into our knowledge and experience, and finally converted them into code output.

We have been constantly solving bugs and expanding the functionality of our code through countless errors, and learning workarounds by searching the Internet based on every Jupyter Notebook error report. We tested the stability and simplicity of the code through extensive trial and error and brute force decoding. We also tested the readability and organization of the code, and ensured that all the knowledge adopted were purely course-based by demonstrating it to python learners of similar level with us.

2.2. Architecture Design

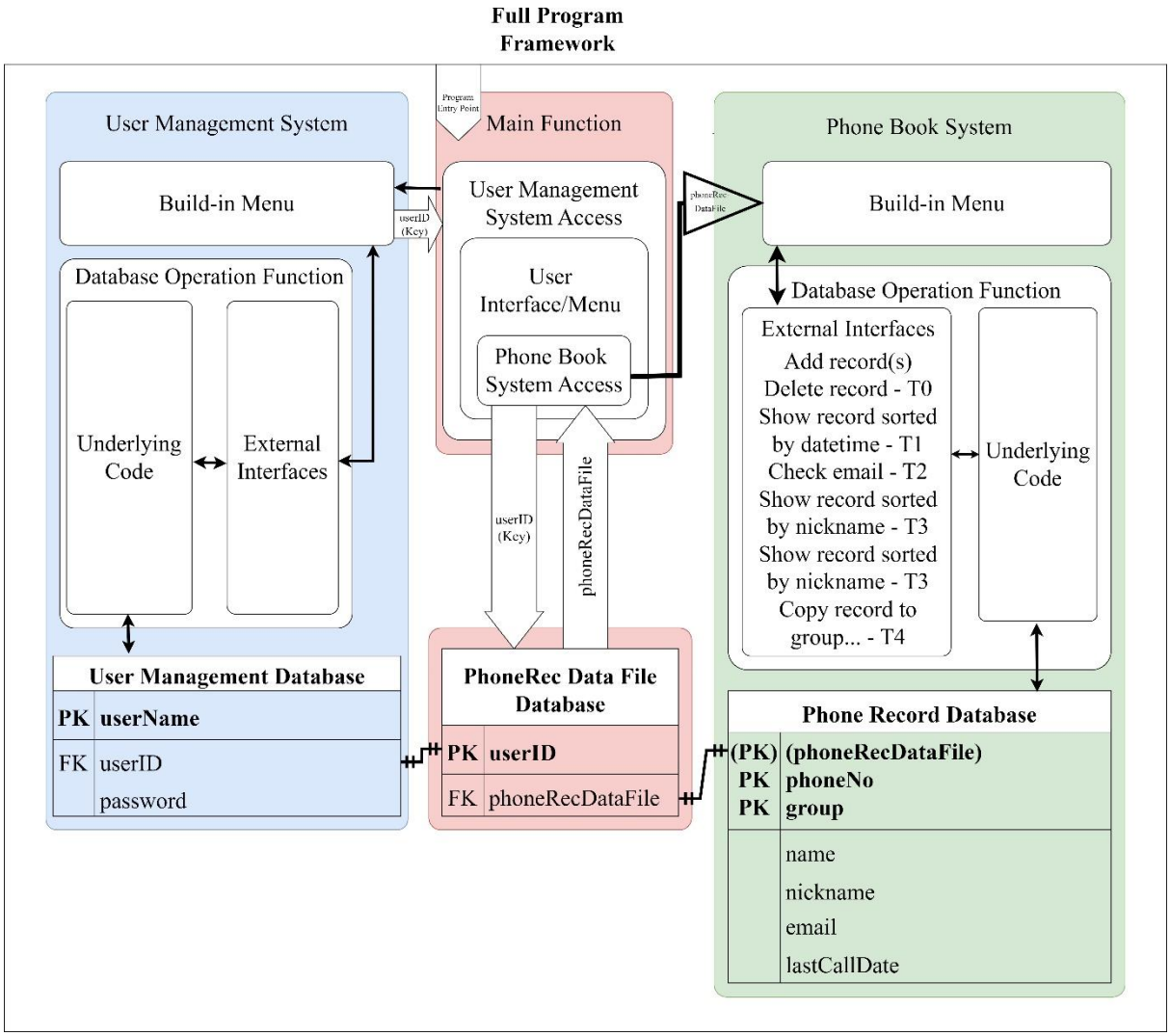

Figure 1 Program Framework

2.2.1. Introduction of the Architecture

- ***Overall introduction***

  The project consists of a main function, the entry point of the project, and two modules, the User Management System and the Phone Book System, which are connected to each other by the main function that takes the user information from the User Management System and connects it to the Phone Book System. Each module corresponds to a database and can operate on each database and process the data. In addition, the Main function in Figure 1 above also corresponds to a database, which is actually converted into different file names/paths based on different user

information, and each file is treated as a separate Phone Book Database, which simply means that there are many files in a folder, and this folder is equivalent to a database.

- ***Database Structure***

  There are three databases in this project, they are User Management Database, PhoneRec Data File Database, and Phone Record Database, and the Relationships between them are of 1:1 type, i.e., each user (ID) corresponds to one Phone Record Database (File). Their primary and foreign key relationships are as follows:

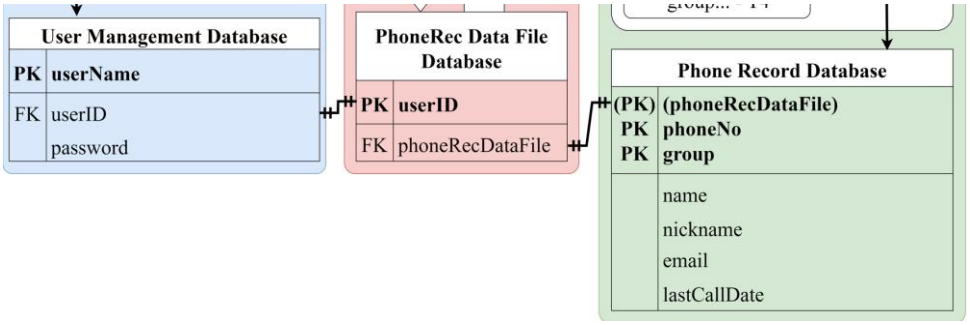

Figure 2 Database Relationship ER Graph

- ***Main Function***

  Main function is the entrance of the project. It is mainly used to call two modules.

- ***User Management System and Phone Book System***

  Both modules have a similar structure, i.e., they have a build-in menu, external interfaces and underlying code.

  - Build-in menu is the menu that comes with both mods, making it easy for users to select and manipulate. build-in menu is connected to External Interfaces.

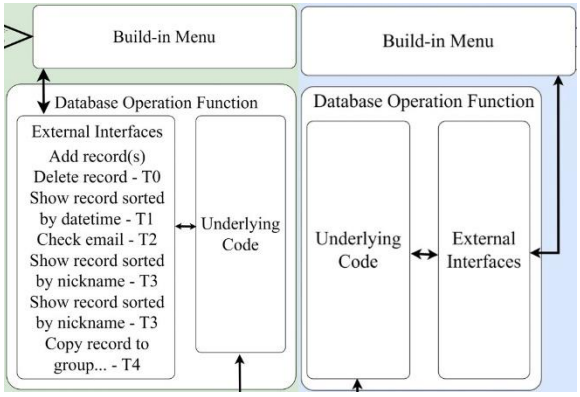

Figure 3 Inner Structure of the two systems

- External Interfaces is used to provide an interface for other developers to operate the system. It integrates many complex operating instructions, which greatly simplifies the system and reduces the risk of errors, which improves the independence and standardization of the modules and makes it easier and safer for developers to use the system.

- The underlying code is the part of the system that operates directly on the database or performs complex processing of data. Due to the complexity and low robustness of the functions in this part, it requires a high level of developer knowledge to ensure proper operation of the entire system, and inappropriate operations may lead to system errors. Therefore, this part of the function is integrated into the external interface function to improve its usability.

2.2.2. Classes

2.2.2.1. User Management System (user_management_sys.py):

(class) userInfo

This class is used to store various information about the user. It currently includes the user's ID and name, and more can be added if required, such as gender, age, email, etc.

(class) UserManageSys

- ***Briefing***:

  This class is mainly used to manipulate and manage users and user database. An inheritance of the UserManageSys class corresponds to a particular user (including unknown user status). This class is highly independent and can be embedded independently in other applications, and provides a comprehensive set of member functions to ensure that user management can be implemented in different projects.

- ***Variables***:

  Member variables mainly include:

  - the current user Info(userInfo),

- the database address(uDBRootPath),
- the current user status(userAccess),
- the current system status(sys_status).

- ***Functions***:

  It contains 7 main aspect member functions as well as some helper functions.

  - Build-in menu/Display (.built_in_menu()): Implementation of menus to enable users to select and operate relevant functions. Enabling user-oriented applications.

  - User sign in/sign up/log out function (.user_sign_up()/.user_log_in()/.user_log_out()): Depending on the information provided about the user, these functions implement the user login, registration or logout.

  - Function user_check(): Go and check if the user exists or if the password is correct based on the information given.

  - Database read/write (.user_database_read(), .user_database_write()): Perform uniformly formatted database read and write operations to enable standardised storage and fetching of user data.

  - Function database_built_up(): Create a blank User Management System-specific database if the database does not exist.

  - Function file_location_detect(): This function is used to find the database location. By default, this function will automatically search for the default path in the different disks and connect when found. If the location of the database is

provided externally, it will check and connect to that database.

- Other helper functions:
  This part of the function is mainly for the main function to achieve better functionality. .isfile()/.isfolder(), which is used as a replacement for os.path.isfile() as we are concerned that this function is not allowed;

  .user_name_check()/.user_id_create() for checking that the name matches the specification and creating the ID; .ums_encryption()/.ums_decryption( ) are used to implement encryption and decryption of user information, which in this project we just simulate simply.

In these member functions,

- Database read/write, user_check(), file_location_detect() and other helper functions are part of the underlying code, which directly manipulates the database, and generally do not need to be called from external sources.

- User sign in/sign up/log out, database_built_up() functions are external interfaces, which are provided directly to external callers and have a higher degree of object orientation.

2.2.2.2. Phone Book System (phonebook.py)

(class) phoneRec

This class is used to store various information of a phone record. It currently includes phoneNo, group, name, nickname, email, and lastCallDate.

(class) phoneBk

- ***Brief:***

  This class is mainly used to manipulate and manage phone book/database of a certain user. An inheritance of the phoneBk class corresponds to a phone book of a particular user (including unknown user status). This class is highly independent and can be embedded independently in other applications, and provides a comprehensive set of member functions to ensure that the phone book management can be implemented in different projects.

- ***Variables***:

  Its member variables mainly include:

  - 3 phone record group lists, i.e. family, friend and junk, which is to record the class phoneRec,
  - a list of string ‘family’, ‘friend’ and ‘junk’ to make display more conveniently,
  - .filePath to record the path of the database, and
  - .ph_status to record the status of the phone book system.

- ***Functions***:

  The member functions of this class can be divided into the following categories:

  - Build-in menu/Display (.built_in_menu()): Implementation of menus to enable users to select and operate relevant functions. Enabling user-oriented applications.
  - Core functions: Core functions: add, delete, display records (sorted by time, nickname), check emails, copy records. This is the main function provided to the user to operate.

- Database synchronization, difference checking: .ph_syncing_to_database(), .ph_syncing_to_database() function integrates standardized database write/read jobs and variable record operations, enabling direct data exchange between the system and the database, greatly simplifying read and write operations to the database. .compare_database() function helps to compare the differences between the database and the current records of the system, making it very convenient for developers to develop projects.

- Database access function(ph_database_access()): Dedicated functions for connecting to databases, standardizing database connection mechanisms and improving object-orientation.

- Database read/write and conflict checking: Performing direct operations on the database.

- Other functions: This part of the function is mainly for the main function to achieve better functionality. Including but not limited to string processing, time conversion, data print formatting, grouping, record retrieving, etc.

In these member functions,

- Database read/write and conflict checking, and other helper functions are part of the underlying code, which directly manipulates the database, and generally do not need to be called from external sources.

- Core functions, Database synchronization, difference checking, Database access functions are external interfaces, which are provided directly to external callers and have a higher degree of object orientation. This allows for increased

module independence, standardization and greatly simplifies the complexity of project development.

2.2.3. Reasoning

Our team set the goal of achieving an independent implementation of each module when architecting the project. Therefore, we built the User Management System and Phone Book System separately and used the Main Function to connect them. This resulted in a clear, highly structured and professional program. For the structure within each module, we set up external interfaces in addition to built-in menus, making the whole module more independent and well embedded in different projects. In addition, the complex underlying code is also integrated into the external interface, which is more in line with object-oriented standards, i.e. more convenient for developers to use.

2.2.4. Execution Flow

The program will first enter the main function through the entry point. Then, the main function will call the User Management System module and carry out the UserManageSys class inheritance. After entering class __init__(), it will call the built-in menu and carry out the user login/logout or registration operation. Once the initialization is completed, the main function will inherit the UserManageSys class that contains the user ID for the user. At this point, the main program will create or search the corresponding Phone Record database file by the user ID. Next, the program calls the Phone Book System and passes the path of the file to the class phoneBK for initialization and returns a phoneBK class that corresponds to the user's Phone Record Data. Finally, this class will enter the built-in menu and perform the relevant Phone Book operations.

2.3. Problems Solving

2.3.1. Variable synchronization problem of class

- ***Description***

  During the execution of Task 4, when we copy a class to another variable, if the value of one of the variables is changed, the value of the original class will

also be altered, causing the variables in the class of the original group to be changed to the corresponding value of the target group after copying.

- ***Analysis***

  We concurred that the driver of the problem was the fact that after copying the class to another variable, the address remained the same between the two classes due to the inherent properties of the class itself. Therefore, when we modified the newly copied class, the original class was changed accordingly, affecting the data in the database.

- ***Solution***

  We have re-inherited the phoneRec class with a new variable to obviate the problems that occurred above.

2.4. Demonstration of Feasibility

We conducted a large number of manual data comparison inspections, analyzed each data modification, further discussed and analyzed according to the problems that emerged during this process, which thereby came up with a corresponding solution, then further improved the code according to that solution. Finally, it formed a virtuous positive feedback loop to achieve the usual upward spiral of product quality.

3. Results

3.1. Architecture

3.1.1. User Management System

- Main Menu

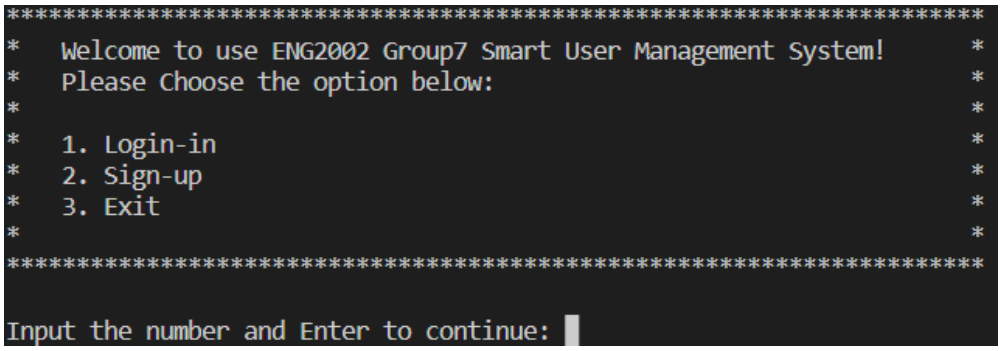

Figure 4 Main Menu of the UMS

- Log-in/Sign-up Interface

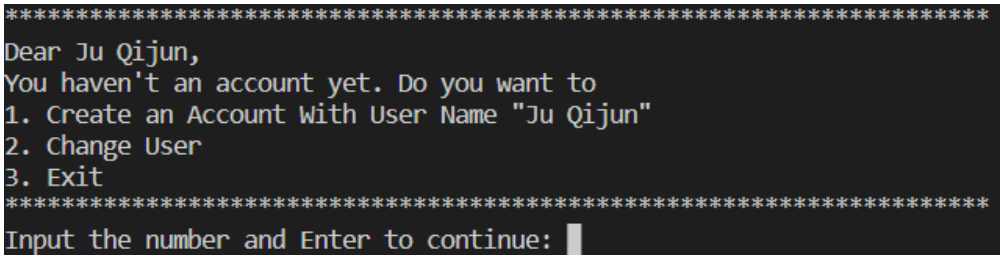

Figure 5 Log-in/Sign-up Interface of UMS

- Successful Sign-up Message*

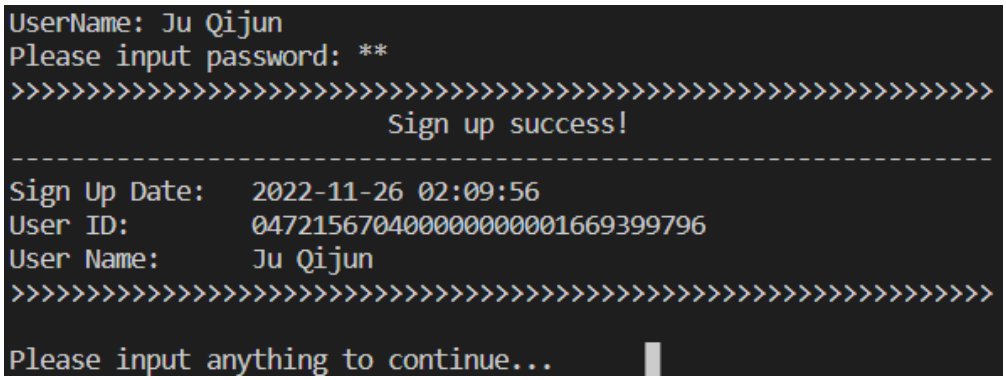

Figure 6 Successful Sign-up Message of UMS

- Successful Log-in Message

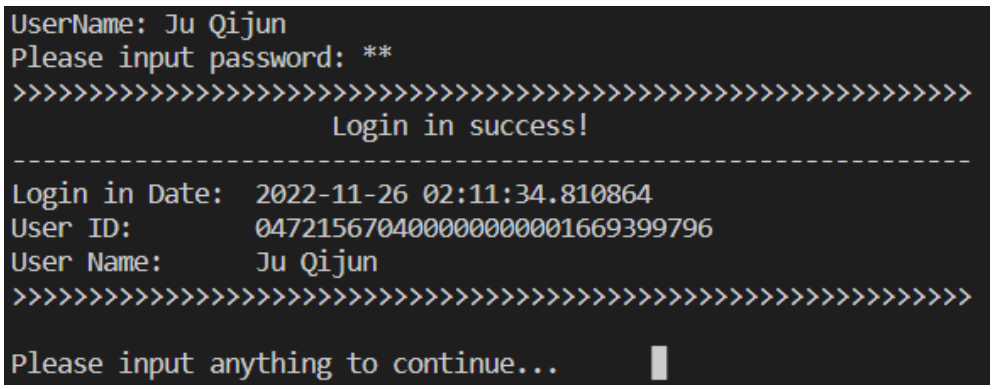

Figure 7 Successful Log-in Message of UMS

**Please be noted that a unique user id will be automatically generated.*

3.1.2. Phonebook System

- Main Menu:

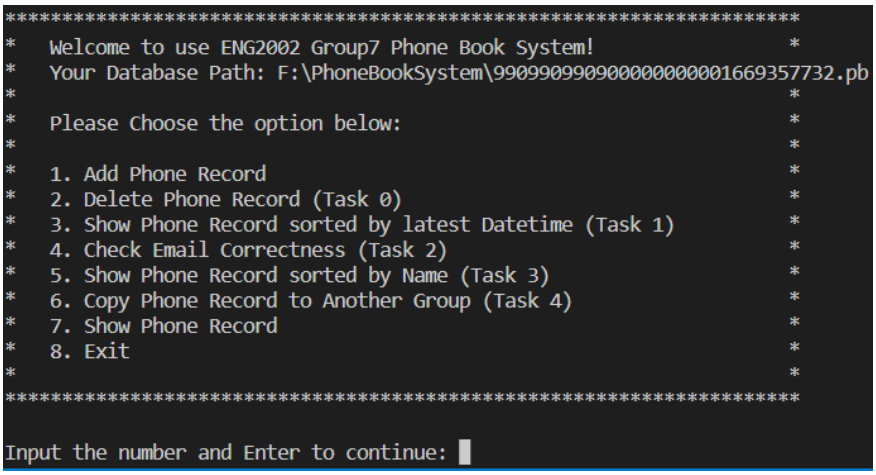

Figure 8 Main Menu

3.2. Task 0

3.2.1. Objectives

The program will delete the corresponding phone records based on the phone numbers entered by the user under the given group. After a phone record is deleted, the valid records should be stored consecutively by the fact that the elements following the deleted record have all been moved forward. Accordingly, the number of valid elements in the list will be updated.

If the deletion cannot be performed for any acceptable reason, an error message will be displayed.

3.2.2. Execution Results

After entering the corresponding sub-menu from the main application, the user can determine the corresponding group to be deleted:

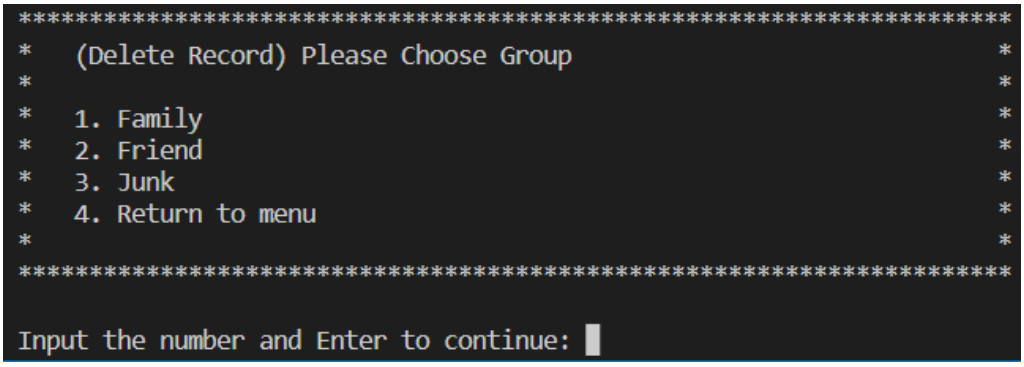

Figure 9 Task 0 Sub-menu

The program required the user to input the specific phone number as a key to delete the corresponding call log information:

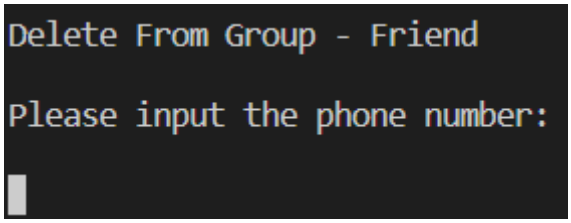

Figure 10 Request to Specify Phone Number

The program will delete the specific dial log in the specific group according to the user's instruction:

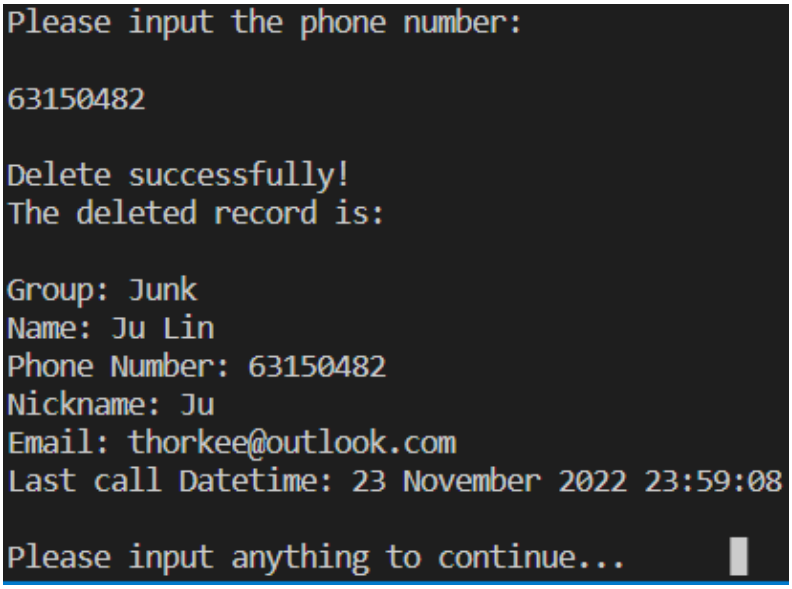

Figure 11 Message of Successful Deletion of a Designated Record

The program will return to the main menu after the successful deletion.

Otherwise, an error message will be displayed if the program fails to retrieve the specific record:

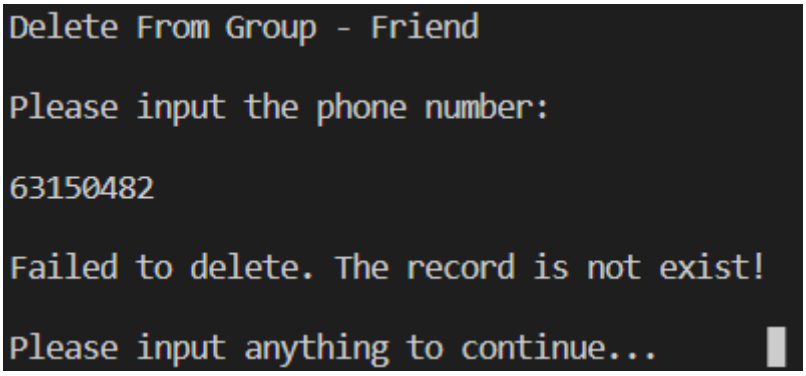

Figure 12 Error Message of Unsuccessful Deletion

3.3. Task 1

3.3.1. Objectives

The task requires the program to present a sorted list of phone records in a predetermined format based on a specified group given by the user, along with the corresponding information of a dial log sorted by time.

3.3.2. *Execution Results*

After entering the corresponding sub-menu from the main application, the user can determine the corresponding group to be displayed:

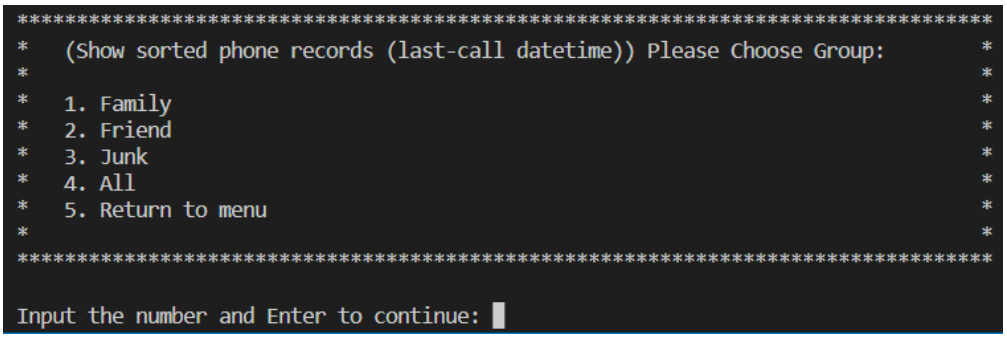

Figure 13 Task 1 Sub-menu

The program will display call logs and other required information according to the group selected by the user:

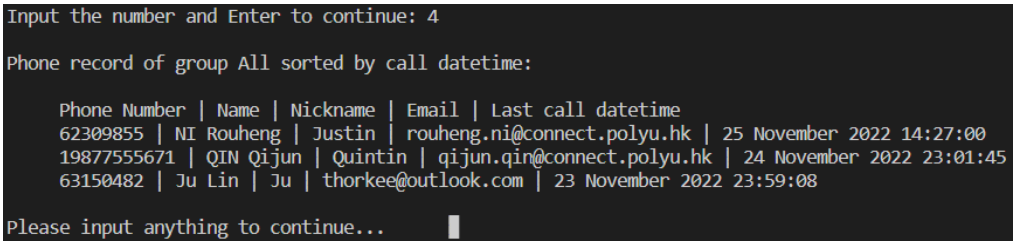

Figure 14 Sorted-by-datetime Dial Log Display According to User Instruction

The program will return to main menu if 5 is detected as user input.

The program will display an error message then display the sub-menu when invalid input is detected.

3.4. Task 2

3.4.1. *Objectives*

The task requires the program to perform a validity check on the email information in the database.

3.4.2. *Execution Results*

After log-in and the selection of 4. Check Email Correctness (Task 2) in the main menu, the sub-menu of Task 2 will be displayed as follow:

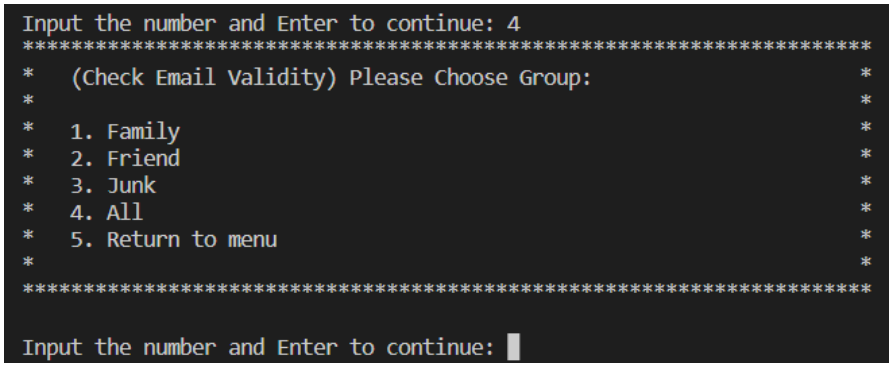

Figure 15 Sub-menu of Task 2

The system will perform email compliance checks on the appropriate groups based on the numbers input by the user.

If no invalid email addresses found, the program will generate a corresponding indicative message:

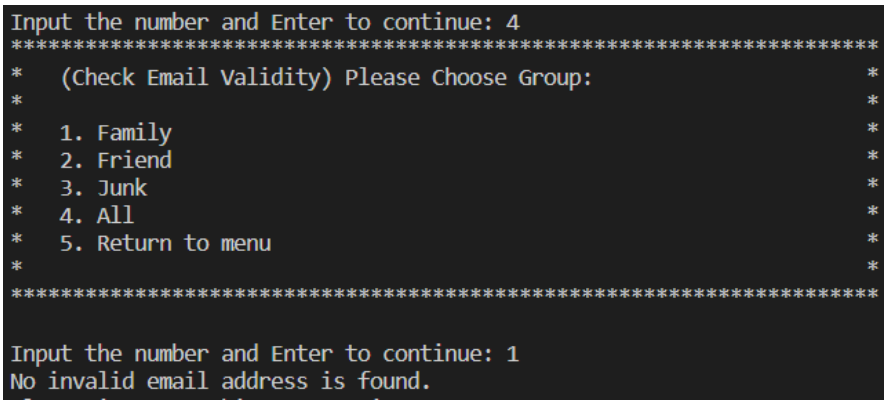

Figure 16 No Invalid Email Found Message

If invalid emails found in the corresponding groups, the program will generate a corresponding message indicating the invalid email and the user's nickname of it:

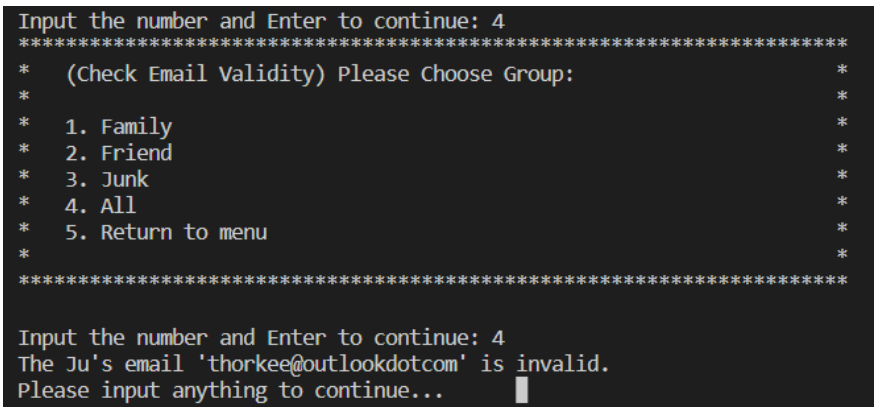

Figure 17 Indicative Message of Invalid Emails

The program will return to main menu if 5 is detected as user input.

The program will display an error message then display the sub-menu when invalid input is detected.

3.5. Task 3

3.5.1. Objective

The task requires the program to display a sorted list of phone records based on the nickname under a user-given group. It also allows users to specify whether the sorting is done in ascending order or descending order. The items in the displayed sorted list include the nickname and phone number.

3.5.2. Execution Result

After log-in and the selection of 5. Show Phone Record sorted by Name (Task 3) in the main menu, the sub-menu of Task 3 will be displayed as follow:

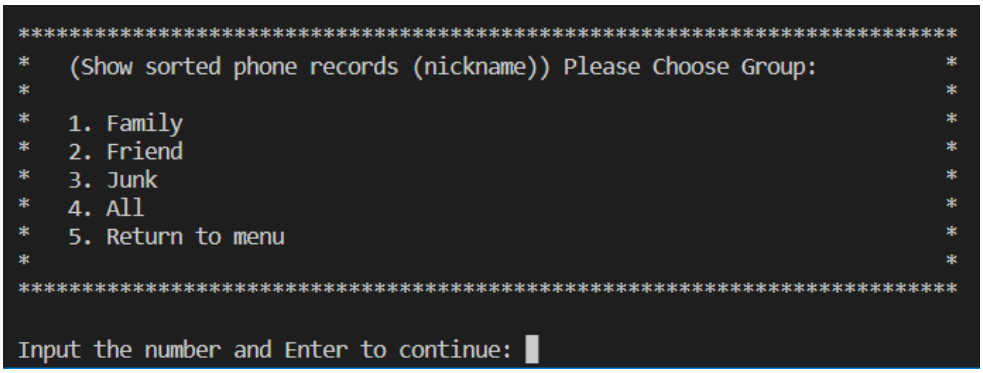

Figure 18 Sub-menu of Task 3

The system will display corresponding phone records on the appropriate groups based on the numbers input by the user.

Sorted order of ascending or descending can be selected as follows:

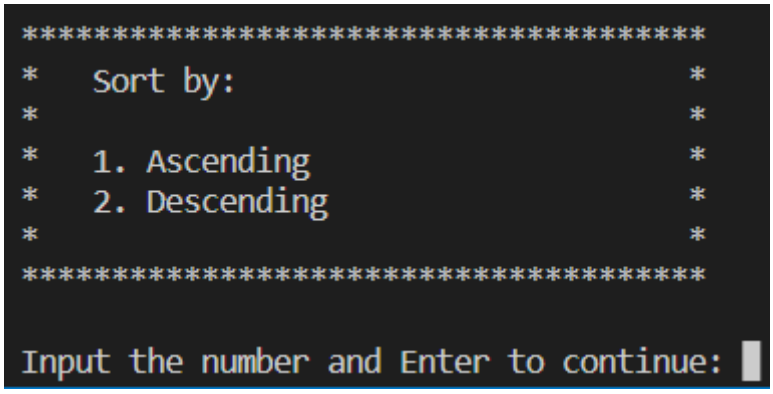

Figure 19 Sorted Order Selection

Phone record of a specific group in a designated sorted order is the displayed as follows:

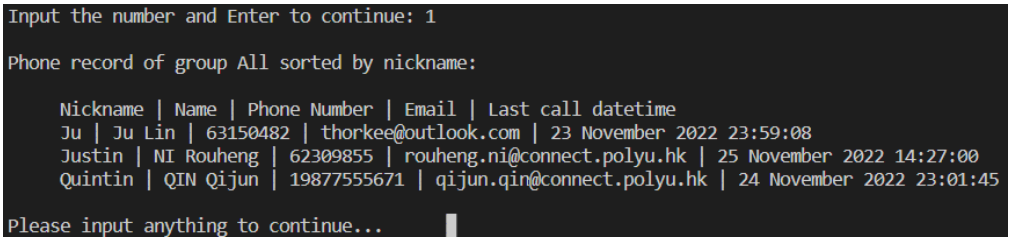

Figure 20 Phone Record in Ascending Order

Or,

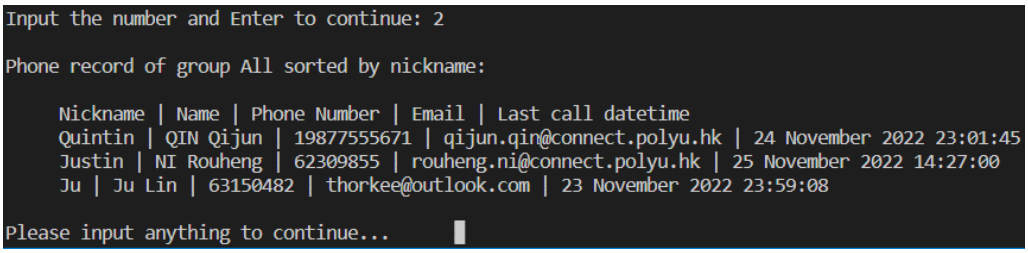

Figure 21 Phone Book in Descending Order

3.6. Task 4

3.6.1. Objective

The task requires the program to relocate the latest contact information of a user-selected person with the phone number as identifier from one group to another.

3.6.2. Execution Result

After log-in and the selection of 6. Copy Phone Record to Another Group (Task 4) in the main menu, the sub-menu of Task 4 will be displayed as follow:

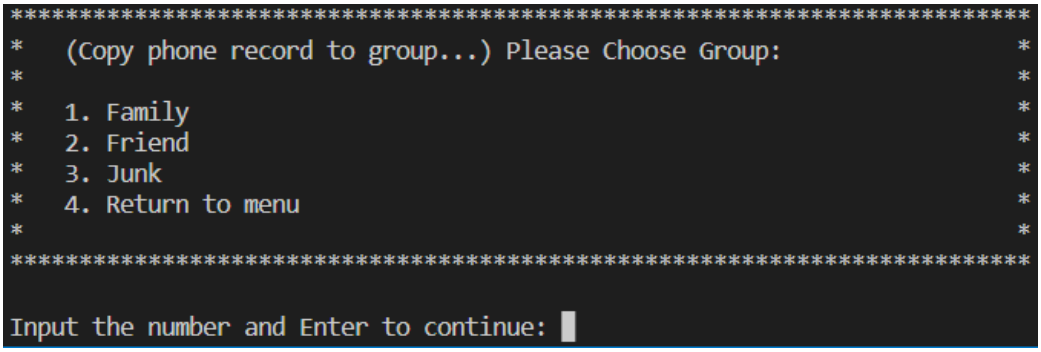

Figure 22 Task 3 Sub-menu

The system will enter the database of the specified group, then request users to input phone number as an identifier to sort out the latest record of the targeted contact:

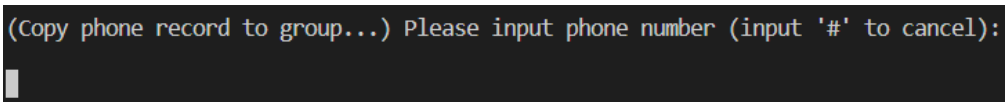

Figure 23 Task 4 Phone Number Request Message

Specific contact information will be displayed, and user will be requested to input the designated group to relocate this record:

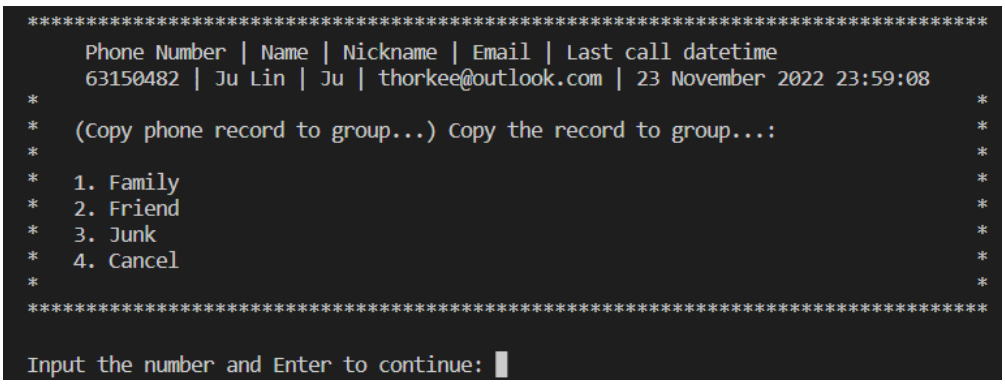

Figure 24 Message Requesting for Targeted Destination

A message indicating successful relocation will be shown upon successful operation:

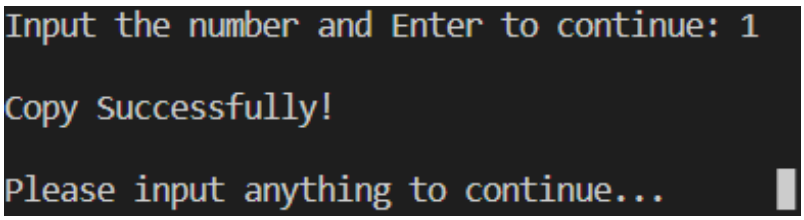

Figure 25 Message of Successful Relocation

Since the method used in the copying process is append as shown below, by definition, it meets the requirements of the new item location in the question:

```
76      def add_rec(self, phRec):       # add a record only, return a list with an element that cannot add in
77                                                  #group 1-Family 2-Friend "3"-Junk
78          if(self.ph_conflict_check(phRec.phoneNo, phRec.group)):  # Check whether the phRec has exist
79              return [phRec]
80          if(phRec.group == 1):
81              self.family.append(phRec)
82          if(phRec.group == 2):
83              self.friend.append(phRec)
84          if(phRec.group == 3):
85              self.junk.append(phRec)
86          return []
```

Figure 26 Append Code

Otherwise, an error message will be displayed if

- The number input by user can not be found in the group specified:

```
(Copy phone record to group...) Please input phone number (input '#' to cancel):

63150482

Cannot find phone record '63150482' in group Friend

Please input anything to continue...
```

Figure 27 Task 4 Number-Not-Found Error Message

- The dial record already exists in the designated group:

```
Input the number and Enter to continue: 3
The phone record

     Phone Number | Name | Nickname | Email | Last call datetime
     63150482 | Ju Lin | Ju | thorkee@outlook.com | 23 November 2022 23:59:08

has exist in group Junk

Please input anything to continue...
```

Figure 28 Task 4 Dial-Record-Already-Exists Error Message

4. Conclusion and Future Development

4.1. Summary

In this project, we developed a complete Phonebook System through self-learning and collaborative teamwork, giving the most optimal solution within our capabilities on the basis of the distinction requirements of the subject.

4.2. Limitations

In this project, we, using our intelligence, have developed many sophisticated program structures and algorithms in a restricted environment. However, we still face some problems that for the time being cannot coincide with our ideal of perfectionism.

First, we developed an encryption algorithm in our program to implement encryption of database information. However, this algorithm is a simple transformation of characters and no more advanced algorithm encryption (such as hash encryption algorithm) is implemented

Second, we did not restrict the database of the User Management System with reasonable permissions, which led to a certain risk of database content being tampered with or illegally read.

Third, there is no find-my-password function, which is likely to bring bad experience to users.

Fourth, our current database is only applicable to the storage and retrieval of computer local disk, not quite compatible with cloud database for the time being, which needs a little time for improvement.

4.3. Future Development

In order to enhance the professionalism of the whole project, we may, firstly, upgrade the encryption algorithm to make it more in line with modern encryption requirements. Second, improve the security, stability, and robustness of the database. Third, develop more practical user-oriented functions to effectively improve user

experience and satisfaction. Fourth, develop an online platform to break time and space restrictions and promote world peace.