Budapesti Műszaki és Gazdaságtudományi Egyetem

Address Book

Final project

Ana Silkin, QL8SD3

Basics of Programming 2, BMEVIIIAA03

Lab instructor: Vaitkus Márton

May 2024

Contents

CHAPTER 1

Introduction

In today's digital landscape, managing personal and professional contacts efficiently is crucial. Traditional methods can be cumbersome, especially as the volume of contact information grows. This project introduces a streamlined digital solution for storing and managing contact details such as names, surnames, phone numbers, addresses, email addresses, and other relevant attributes.

The application, built using the C++ programming language, offers a comprehensive suite of functionalities designed to enhance the user experience. Key features include the ability to open existing address books, create new ones, and perform various contact management operations such as adding, searching, editing, and deleting contacts. Each functionality is carefully developed to ensure ease of use and robust data handling.

This document provides an overview of the problem and a detailed description of the solution, including the system's architecture and user interface. The subsequent sections will delve deeper into the technical implementations and the interaction design, ensuring that the user can efficiently navigate and utilize the application to its full potential. Through this introduction and the detailed discussions that follow, it is aimed to provide a clear and comprehensive understanding of how this digital address book can resolve common contact management issues.

CHAPTER 2

Program Interface

This section outlines the primary methods through which users interact with the digital address book application. It details the initial setup, execution, and termination processes essential for operating the program effectively.

2.1 Setting Up the Programming Environment

To run the digital address book application, users must first ensure that their system has the appropriate development environment. For Windows, an environment such as MinGW or Microsoft Visual Studio is suitable. For macOS and Linux, compilers like g++ can be utilized, which are typically pre-installed or available through package managers.

For instance, to compile the program using g++, the user would enter the following command:

```
g++ Main.cpp Contact.cpp FileFunctions.cpp Functions.cpp HelpFunctions.cpp -o main
```

This command compiles the program and links all the necessary source files, resulting in an executable named main. To start the program, users simply need to execute the following command in the terminal:

```
./main
```

This application does not require any startup parameters under normal operations.

2.2 Terminating the program

The program can be terminated in several ways:

2.2.1 Normal Termination

The application provides an option to exit gracefully through its user interface. Usually, this is done by selecting a quit option from the main menu, which safely closes the application without any adverse effects on the system or the data.

2.2.2 Forced Termination

In cases where the application may become unresponsive, users can force-quit the program using system-specific commands, such as Ctrl+C in the command line interface or using task manager utilities provided by the operating system.

2.2.3 Command Line Exit

When running from a command line interface, closing the command line window or terminating the session will also close the application. This method is generally not recommended as it may not ensure the proper saving of data or clean closure of system resources.

CHAPTER 3

Program Execution

This section provides a comprehensive guide on using the address book application from an end-user's perspective. It details the types of inputs required, the expected outputs, functionalities available, and navigational aspects of the program interface. This guide acts as the core part of the user manual, designed to assist users in interacting with the application effectively.

3.1 Starting the Program

Upon launching the application, users are greeted with the main menu which presents the initial choices for interaction:

- Open an existing address book
- Create a new address book

Users input is 1 or 2 to select their preferred option, followed by entering the name of the address book file when prompted. It is important to note that if the new user will decide to create a new address book, he will be asked to input a contact as soon as the address book is created.

3.2 Main Menu Options

After opening or creating a new address book, the program displays the menu, which includes the following options:

- Display Contacts
- Search Contact
- Add Contact
- Edit Contact
- Delete Contact
- Delete Address Book
- Quit

Each option can be selected by entering the corresponding number.

3.3 Detailed Options Description

3.3.1 Display Contacts

The "Display Contacts" function is an integral part of the digital address book application, allowing users to view all stored contacts or filter them based on specific criteria. Users can choose to view all contacts or display them based on a specific category. Each option can be selected by entering the corresponding number. In the case when there is no contact of the specified type, the program will show an error message.

3.3.2 Search contact

When the user chooses to search for a contact, they are prompted to specify the search criterion(e.g.: last name, first name, phone number). Depending on the input, the program searches the address book and displays any matching entries.

3.3.3 Add contact

Selecting "Add contact", the program prompts the user to enter details such as name, surname, phone number, email address, and other relevant information. The user inputs are takes as series of prompts. Upon completion, the application will confirm the addition of the contact to the address book. In the case when there is an existing contact with the same attributes, the user will be asked whether he wants to edit the contact, or to cancel adding it. The user will be returned to the main menu.

3.3.4 Edit contact

If editing is selected, the program first asks for the identifier of the contact (e.g., name or phone number) and then allows the user to change the contact's details through a series of prompts similar to those used for adding a contact. In the instance when the contact does not exist, the user will see an error message: "Contact not found" and will be returned to the main menu.

3.3.5 Delete contact or Address Book

To delete a contact or the entire address book, the user selects the appropriate option, confirms their choice, and the program performs the deletion. For safety, a confirmation prompt ensures the user does not accidentally delete important data.

3.3.6 Exiting the program

To exit the application, the user selects Quit from the main menu. The program will then terminate, ensuring all data is saved properly.

CHAPTER 4

Input and Output

This section of the user manual outlines the input and output formats utilized by the address book application. Detailed descriptions of both interactive inputs (user-entered data) and file-based operations (input and output files) are provided to ensure clarity and proper usage.

4.1 Interactive Inputs

The application primarily interacts with the user through a CLI (command-line interface). Here, users enter data through prompts that guide them to provide the necessary information for each functionality, such as adding, searching, or editing contacts.

Example: Adding a contact

- Prompt: Enter the last name of the person:
- User Input: Smith
- Prompt: Enter the first name of the person:
- User Input: John
- Prompt: Enter the phone number of the person:
- User Input: 555-1234
- Prompt: Enter the email address of the person:
- User Input: john.smith@email.com

Input format: textual data entered through the keyboard, following instructions.

4.2 File-Based Inputs and Outputs

The application supports reading from and writing to files, specifically for loading and saving contact lists. These files are formatted to ensure compatibility.

4.2.1 Input File Format:

- File Type: CSV (Comma-Separated Values)
- Content: Each line represents a contact with fields separated by commas.
- Fields: Last Name, First Name, Phone Number, Email Address, Additional Attributes (optional).

4.2.2 Output File Format:

- File Type: CSV
- Content: Mirrors the input file format, with any changes made during the session reflected. The last and first names are saved in capital letters.

CHAPTER 5

Program Structure

This section serves as the technical manual for the digital address book application, detailing its architecture and internal components. It provides a clear breakdown of the program's overall structure and its constituent parts, including subprograms, modules, and classes. This guide facilitates understanding of the functional and algorithmic logic used throughout the application.

5.1 Overview of Program Architecture

The address book is structures around a modular design, enabling easy maintenance. The main components of the application are:

- Main Module: handles user interaction, including input processing and menu navigation.
- Contact Management: covers classes and functions for creating, modifying, searching, and deleting contact information.
- File Handling: responsible for reading and writing to files, ensuring data persistence.
- Utility functions: helper functions that perform various tasks as string manipulation, input validation and error handling.

5.2 Data Structures

The program is written using a variety of data structures, in order to manage contact information and user interactions effectively:

- Data Types: custom classes (Contact, PersonalContact, etc.), standard library types (std::string, std::list, etc.).
- Variables: store individual data points such as contact names, addresses, phone numbers, etc.
- Containers
  - `std::list<Contact*>`

    used to store collections of contacts, allowing for efficient insertion and deletion. The choice to use a list container was based on the ease of reordering contacts in alphabetical order, which enhances the user experience.

– `std::vector<std::string>`

utilized for temporary storage of file lines or processing multi-part data.

5.3 Subprograms and modules

Each module of the program is designed to handle specific aspects pf the application's functionality. Here's a breakdown of the key modules:

1. Main Module (Main.cpp):
   - Purpose: Manages the application's startup, user commands, and menu interactions.
   - Functionality: Initializes the program, displays the main menu, and routes the user's choices to the appropriate functionality in other modules.
2. Contact Module (Contact.h, Contact.cpp)
   - Classes:
     – Contact: base class for all contact types.
     – PersonalContact, BusinessContact, FamilyContact: derived/child classes that extend the base/parent class for specific contact types.
   - Purpose: defines the properties and methods common to all contacts, as well as specific attributes for different types of contacts.
   - Functionality: includes methods for displaying contact details, updating contact information , and converting data to and from CSV format for file storage. Since we have contacts with specific attributes, it was made the decision to write for each type its toCSV() method.
3. FileFunctions Module (FileFunctions.cpp, FileFunctions.h):
   - Purpose: file input and output management.
   - Functionality: Reads and writes contact data to files, handles file existence checks, and manages file creation and deletion.
4. HelpFunctions Module (HelpFunctions.cpp, HelpFunctions.h)
   - Purpose: provides utilities that support other modules.
   - Functionality: includes methods for comparing last and first names, transforming text to uppercase, validating inputs (numerical, character).

5.3.1 Detailed Explanation of Class "Contact"

Base Class: Contact

Purpose: the class "Contact" serves as the base class for all types of contacts. It provides the fundamental attributes and methods that are common across different kinds of contacts.

Attributes:

- lastName

- firstName
- phoneNumber
- emailAddress

Methods:

- display(): displays the contact's information.
- add(): adds the contact to the address book.
- updateLastName(newLastName): updates the last name of the contact.
- updateFirstName(newFirstName): updates the first name of the contact.
- updatePhoneNumber(newPhoneNumber): updates the phone number.
- updateEmailAddress(newEmailAddress): updates the email address.
- toCSV(): converts the contact's information into a comma-separated values (CSV) string.

Constructor: initializes a contact with the provided last name, first name, phone number, and email address.

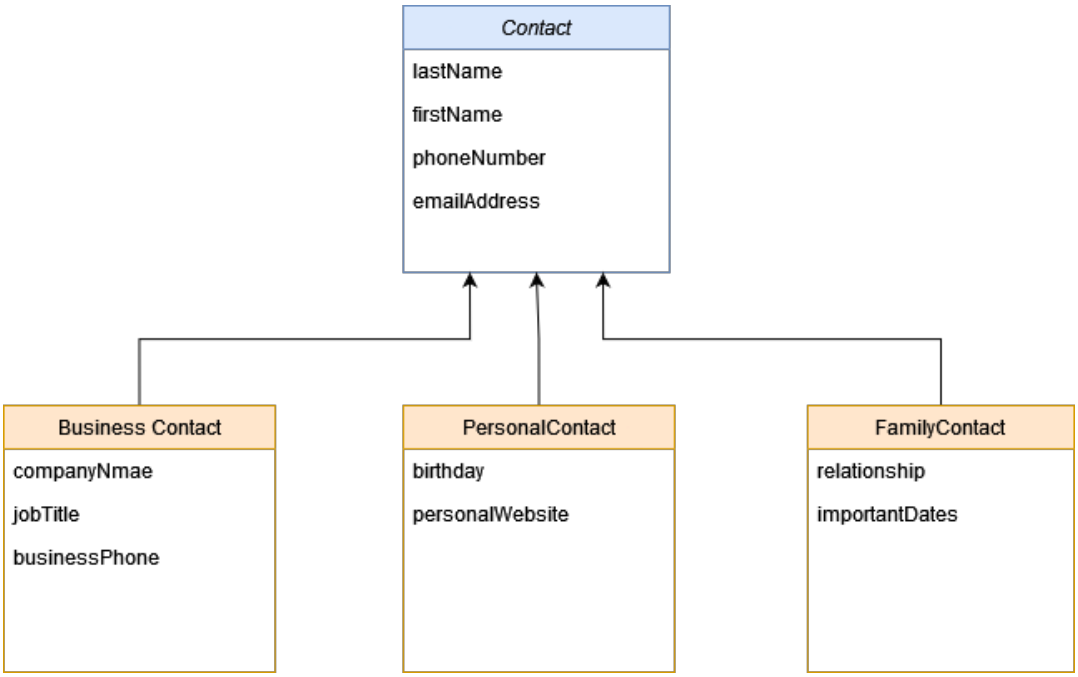
Derived Classes: Specific Types of Contacts

1. Business Contact (derived from based/parent class Contact)
    - Additional Attributes:
        - companyName: the name of the company where the contact works.
        - jobTitle: the job title of the contact at their company.
        - businessPhone: a business-specific contact number.
    - Specialized methods:
        - updateCompanyName(newCompanyName): updates the contact's company name.
        - updateJobTitle(newJobTitle): updates the contact's job title.
        - updateBusinessPhone(newBusinessPhone): updates the contact's business phone number.
        - toCSV(): includes additional business-related information in the CSV output.
    - Constructor: constructs a BusinessContact with details specific to business contexts.
2. Personal Contact (derived from based/parent class Contact)
    - Additional attributes:
        - birthday: the birthday of the contact.
        - personalWebsite: a URL to the personal website of the contact.

- Specialized Methods:
  - updateBirthday(newBirthday): updates the contact's birthday.
  - updatePersonalWebsite(newPersonalWebsite): updates the URL of the contact's personal website.
  - toCSV(): includes personal details in the CSV output.
- Constructor: initializes a PersonalContact with the corresponding attributes.

3. Family Contact (derived from based/parent class Contact)
   - Additional Attributes:
     - relationship: defines the relationship of the contact to the user (e.g., parent, sibling).
     - importantDates: important dates related to the contact (e.g., anniversaries) (Note: it is possible to have just one important date).
   - Specialized Methods:
     - updateRelationship(newRelationship): updates the description of the relationship.
     - updateImportantDates(newImportantDates): updates important dates associated with the contact.
     - toCSV(): outputs family-related information in the CSV format.
   - Constructor: construct a FamilyContact object with the specific details that correspond to this class.

This diagram could enhance the understanding of the class inheritance structure of the system. At the top, we have the Base Class - Contact from which three other classes derive. Each derived class, labeled as in the program: BusinessContact, Personal Contact, and Family Contact, extends the Contatc Class with specific functionalities and attributes. This structure showcases the object-oriented design principles we've implemented, emphasizing modularity and reusability.



5.3.2 Detailed Explanation of Class "FileFunctions"

The "FileFunctions" class is responsible for all file handling tasks, ensuring that data is stored correctly to and loaded from persistent storage, allowing for reliable data management and recovery.

Methods of the "FileFunctions" Class

1. `bool fileExists(const string& filename):`

   - Purpose: checks for file existence in the file system, using the "stat" system call.
   - Return: true if the file exists, false - otherwise.

2. `bool isFileEmpty(const string& filename)`

   - Purpose: determines whether a file is empty.
   - Return: true if the file is empty, false - otherwise.

3. `bool isFileEmpty(const string& filename)`

   - Purpose: creates a new file.
   - Return: true if the file is successfully created and opened; otherwise, returns false with an error message.

4. `bool handleFileAccess(const string& filename, int choice)`

   - Purpose: manages access to a specified file, offering options based on whether the file exists or not.
   - Return: true if the file is ready for use (either already existed or was created); otherwise, handles user choices for retrying or exiting.

5. `void readFromFile(const string& fileName, list<Contact*>& addressBook)`

   - Purpose: loads contacts from a specified CSV file into the address book.
   - Return: -.

6. `void writeContactsToFile(const string& fileName, const list<Contact*>& addressBook)`

   - Purpose: writes all contacts from the address book into a specified file in CSV format.
   - Return: -.

5.3.3 Detailed Explanation of Class "Functions"

The "Functions" class encapsulates crucial functionalities related to contact management. Ir operates on a list of contacts, facilitating operations like adding, displaying, searching, deleting contacts as well as managing entire address books. The breakdown of the methods of the module are presented below.

Methods of the "Functions" Class

1. ```
   void addContactAlphabetically(const string &
      filename, list<Contact*>& addressbook)
   ```

   - Purpose: adds a new contact to the address book in alphabetical order based on last and first names, by prompting the user to enter the contact's details.

2. ```
   void displayContacts(const string &fileName, list
      <Contact*>& adrressBook)
   ```

   - Purpose: displays all contacts or a filtered group of contacts from the address book.

3. ```
   void editContacts(const string &fileName, list<
      Contact*>& adrressBook)
   ```

   - Purpose: Allows editing of a selected contact's details.

4. ```
   editContactDirectly(Contact* contactToEdit, const
       string &fileName, list<Contact*>& addressBook
      )
   ```

   - Purpose: in cases where a user attempts to add a contact whose attributes match those of an existing contact, they are presented with the option to either edit the existing contact directly or cancel the addition. This functionality allows for immediate modification of the existing contact, thus bypassing the more extensive steps involved in the standard "editContact" method, avoiding redundant navigation through multiple editing steps.

5. ```
   void searchContact(const string &fileName, list<
      Contact*>& addressBook)
   ```

   - Purpose: searches for contacts based on a specific attribute (last name, first name, phone number, or email address).

6. ```
   void deleteContact(const string &fileName, list<
      Contact*>& addressBook)
   ```

   - Purpose: deletes a specified contact from the address book.

7. `void deleteAddressBook(const string &fileName, list<Contact*>& addressBook)`

   - Purpose: deletes the entire address book file and clears all contacts in memory.
   - Return: -.

5.3.4 Detailed Explanation of Class "HelpFunctions"

The "HelpFunctions" class provides utility functions that facilitate various common operations such as string manipulation, input validation, and contact comparison; it enhances the application's usability and reliability by ensuring consistent and error-free user interactions.

Methods of the "FileFunctions" Class

1. `std::string toUpperCase(const string& str)`

   - Purpose: converts a given string to uppercase.
   - Return: the uppercase version of the input string.

2. `bool compareContacts(const Contact* a, const Contact* b)`

   - Purpose: compares two contacts primarily by last name and, if necessary, by first name to facilitate alphabetical ordering.
   - Return: true if the first contact should come before the second contact in alphabetical order; otherwise, returns false.

3. `int getValidChoice(int from, int to)`

   - Purpose: validates user input to ensure it falls within a specified range and is of the correct type. It gives to user 4 trials of wrong inputs. On the 5th one, the user is returned to the main menu or exited on the program (depending on which part of the program is the user now).
   - Return: the user's choice if valid; -1 if invalid inputs persist after several attempts.

4. `char getYesOrNo()`

   - Purpose: retrieves a 'yes' or 'no' response from the user.
   - Return: the character 'y' or 'n' if a valid input is entered; returns a space character ' ' as a default or error value if the maximum number of attempts is reached. It gives to user 4 trials of wrong inputs. On the 5th one, the user is returned to the main menu or exited on the program (depending on which part of the program is the user now).

5.4 Conclusion

In wrapping up our discussion on the program structure of address book project, it's imperative to highlight how the application fully leverages the principles and features of object-oriented programming (OOP).

Utilization of OOP Principles:

- Encapsulation: The application effectively encapsulates data and functionality within well-defined classes such as Contact, BusinessContact, PersonalContact, and FamilyContact. Each class manages its own state and behaviors, hiding the internal details from other parts of the application. This encapsulation ensures that data manipulation is restricted to appropriate methods within each class.

- Inheritance: The use of inheritance is prominently seen in the hierarchical structure of contact types. The base class Contact provides common attributes and methods, which are then extended by more specific types of contacts. This approach not only eliminates redundancy but also eases the introduction of new contact types with minimal changes to existing code.

- Polymorphism: Polymorphism is utilized through methods that are overridden in derived classes and through interfaces that allow objects to be manipulated generically. For instance, the method display() is implemented in each derived class to handle specific details about how contact information is presented to the user, enabling the same call to behave differently depending on the object's class type.

- Abstraction: The application abstracts complex operations into simple interfaces. Methods like addContactAlphabetically or editContactDirectly provide a high-level interface for complex operations, hiding the intricate details of how contacts are managed and stored. This abstraction makes the application easier to use and modify.

In conclusion, the application not only demonstrates effective use of object-oriented programming but also showcases how OOP can lead to creating robust, efficient, and user-friendly software by incorporating the use of friend classes, we've enhanced the security and encapsulation of the system, allowing controlled access to internal data of classes only where absolutely necessary. This strategic use of OOP principles and friend classes underscores our dedication to quality and innovation, ensuring that the digital address book remains adaptable, secure, and up-to-date as user needs.

CHAPTER 6

Testing and Verification

6.1 Objective

The primary goal of testing and verification is to confirm that:

- The application performs as expected under normal and boundary conditions.
- All user inputs are handled correctly.
- The application is free from defects that affect user operations.
- Data integrity is maintained throughout operations such as addition, deletion, and modification of contacts.

6.2 Testing Environment Setup

- Software: C++ development environment with GCC compiler.
- Data: Sample contact data in CSV format for testing file operations.

6.3 Example of test

The following sequence of commands can be used to check and verify the program. The inputs are designed for an existing address book. After completing the steps in Section 2.1, proceed with the following steps, pressing the "Enter" key after each input. The comments enhance the understanding of what is being checked at each step.

```
1
ab.txt // Enter the name of the existing address book.
//-------------Check the display function
1
1
1
2
9 // Check for error handling with an invalid menu option
//-------------Check the add function
3
1
silkin
```

```
ana
+36202262496
ana.silkin@mail.ru
2 // Similar contact exists, return to the main menu
3
1
doe
john
555-555-5555
john.doe@yahoo.com
//-------------Check the search function
2
1
doe // Search for contact by last name
//-------------Check the edit function
4
1
doe // Select contact by last name
jackson // Change the last name to 'Jackson'
enter // Press Enter to skip changing the first name
enter // Press Enter to skip changing the phone number
enter // Press Enter to skip changing the email
//-------------Check the delete function
5
2
john // Choose to delete by first name
2 // Choose the second 'John' in the list for deletion
k // Mistaken input, should prompt for correction
y // Confirm deletion of the contact
//-------------Check that the contact was deleted
1
1 // Display all contacts to verify deletion
//-------------Exit the program
0 // Exit the application
```

This test serves as a comprehensive verification of the application's main functionalities, including displaying, adding, searching, editing, and deleting contacts within an address book. Each step is designed to not only test the basic operations but also to ensure that the program handles errors and user inputs appropriately, such as responding to invalid options and confirming deletions. Following this test procedure should reveal any fundamental issues with the program's operations or its user interface. Developers and testers are encouraged to use these results to further refine and debug the program, ensuring a robust and user-friendly application. If any step fails, review the associated code sections for logical or implementation errors, adjust accordingly, and retest to confirm the effectiveness of any modifications.

CHAPTER 7

Improvements and Extensions

This chapter addresses potential areas for improvement and outlines future extensions for the program.

7.1 Current Limitations

1. User Interface: currently, the program operates with a basic console-based interface, which can be less intuitive for users who are accustomed to more interactive graphical interfaces.

2. Data Storage: the application uses a flat-file system for storing data, which might not scale well as the data grows or as operations become more complex.

3. Error Handling: while basic error handling is implemented, the program could improve in how it manages and reports errors back to the user, especially under unexpected conditions.

7.2 Proposed Extensions

1. Graphical User Interface (GUI): introducing a graphical user interface (GUI) could provide a more intuitive way for users to manage their contacts, incorporating elements such as clickable buttons, form inputs, and visual feedback that are not possible in a console-based interface.

2. Database Integration: transitioning from a flat-file system to a relational database management system (RDBMS), such as MySQL, would offer numerous benefits. These include improved data integrity, easier management of large datasets, and the ability to perform complex queries and reports.

3. Verification Features:

    - Email Verification: implementing email verification would ensure that the contact information is accurate (it contains 1 "@" character and at least 1 dot after it).
    - Phone Number Validation: Adding phone number validation could improve data quality by verifying that phone numbers entered are in a valid format (the input string is made up of numbers, "-" and maybe "+" at start).

CHAPTER 8

Difficulties Encountered

Throughout the development of this project, several challenges were encountered that influenced the final outcome.

- New Language Proficiency: the project was written in C++, a language that was relatively new. The initial learning impacted the development speed.
- Debugging Complexity: particularly challenging as the complexity of the application increased. The lack of deep familiarity with the language's debugging tools sometimes led to longer-than-expected downtimes in development.
- Time Management: balancing the time between learning to use the language effectively and making progress on the application was a persistent challenge.

The difficulties faced during the project were substantial, yet they provided valuable learning experiences. The necessity to adapt to these challenges not only enhanced our problem-solving skills but also deepened our understanding of the practical applications of C++ and OOP features.

CHAPTER 9

Conclusion

This project successfully created a functional address book application, meeting many of initial goals and providing essential learning experiences in software development and problem-solving. Achievements

- Functional Application: a working address book that can add, edit, search, and delete contacts.
- Problem-Solving: developed better problem-solving abilities by addressing various unexpected challenges during the project.

Lessons Learned

- Early Research: understanding the tools and programming languages before starting can help make the development process smoother.

Future Directions

- Enhancements: There is room to add more features, such as adding a graphical user interface, or using a database for better data management.
- User Feedback: Implementing a system to gather user feedback will help improve the application based on what users like and need.

Final Thoughts

In conclusion, this project not only met its basic goals but also laid a foundation for future enhancements.

CHAPTER 10

References

The following resources provided insights and guidance on how to design the project:

[1] https://objectintrospection.org/docs/addrbook-intro
[2] https://stackoverflow.com/questions/29734258/c-address-book
[3] https://cplusplus.com/forum/beginner/281719/
[4] https://www.codewithrandom.com/2023/04/05/address-book-in-c/