Address Book System

The Address Book System is a program designed to manage a collection of contacts using basic data structures and file handling in the C programming language. The system is built around the concept of an address book, which is essentially a collection of contacts. Each contact contains information such as a name, phone number, and email address. The system allows users to perform various operations on these contacts, such as adding, editing, deleting, searching, and listing them. Additionally, the system supports saving and loading contacts from a file to ensure data persistence.

Below is a theoretical breakdown of the system's components and functionality:

1. Core Concepts

1.1 Contact

A contact is the fundamental unit of the address book. It represents a single entry in the address book and contains the following attributes:

Name: A string representing the contact's name. It is validated to ensure it contains only alphabetic characters and spaces.

Phone Number: A string representing the contact's phone number. It is validated to ensure it is exactly 10 digits long.

Email Address: A string representing the contact's email address. It is validated to ensure it contains an "@" symbol and ends with ".com".

1.2 Address Book

The address book is a collection of contacts. It is implemented as a structure that contains:
An array of Contact structures (up to a maximum of 100 contacts).
A counter (contactCount) to keep track of the number of contacts currently stored.
A flag (modified) to indicate whether the address book has been modified since the last save.

2. Functionality

The system provides the following core functionalities:

2.1 Contact Management

Adding a Contact: Users can add a new contact to the address book. The system validates the input (name, phone number, and email) to ensure it meets the required format. If the input is valid, the contact is added to the address book.

Editing a Contact: Users can edit an existing contact's details. The system ensures that the edited details are valid and do not conflict with existing contacts (e.g., no duplicate phone numbers or email addresses).

Deleting a Contact: Users can delete a contact from the address book by specifying the contact's name. The system removes the contact and adjusts the remaining contacts to fill the gap.

Searching for a Contact: Users can search for a contact by name, phone number, or email. The system performs a caseinsensitive search and displays matching contacts.

Listing Contacts: Users can view all contacts stored in the address book. The system displays the contacts in a formatted list.

2.2 File Handling

Loading Contacts: When the program starts, it loads contacts from a file (contact.txt) into the address book. This ensures that the address book retains its state between program runs.

Saving Contacts: Users can save the current state of the address book to the file. This ensures that any changes made during the program's execution are persisted.

2.3 Input Validation

The system includes validation functions to ensure that user input meets specific criteria:

Name Validation: Ensures that the name contains only alphabetic characters and spaces.

Phone Validation: Ensures that the phone number is exactly 10 digits long.

Email Validation: Ensures that the email address contains an "@" symbol and ends with ".com".

3. Theoretical Workflow

3.1 Initialization

When the program starts, it initializes the address book by loading contacts from the contact.txt file. This file contains a list of contacts in a commaseparated format (e.g., John Doe,1234567890,john@example.com). The system reads this file, parses each line, and populates the address book with the contacts.

3.2 User Interaction

The program provides a menudriven interface for users to interact with the address book. Users can choose from the following options:

1. List Contacts: Displays all contacts in the address book.
2. Create Contact: Allows users to add a new contact after validating the input.
3. Search Contact: Allows users to search for a contact by name, phone number, or email.
4. Edit Contact: Allows users to modify an existing contact's details.
5. Delete Contact: Allows users to remove a contact from the address book.
6. Save Contacts: Saves the current state of the address book to the file.
7. Exit: Exits the program, prompting the user to save any unsaved changes.

3.3 Data Persistence

The system ensures data persistence by saving the address book to a file (contact.txt) whenever the user chooses to save or exit the program. This allows the address book to retain its state between program runs.

4. Theoretical Design

4.1 Modular Design

The system is designed in a modular fashion, with separate files for different components:

contact.h: Contains the definitions of data structures and function prototypes.

contact.c: Implements the core functionality for managing contacts.
main.c: Implements the user interface and main program logic.
contact.txt: Stores the contacts in a persistent format.

4.2 Data Structures

The system uses two primary data structures:

Contact: Represents a single contact with attributes for name, phone number, and email.
AddressBook: Represents the collection of contacts and includes metadata such as the number of contacts and a modification flag.

4.3 Error Handling

The system includes basic error handling to ensure data integrity:

Invalid inputs (e.g., names with numbers, phone numbers that are not 10 digits, or invalid email addresses) are rejected.
Duplicate contacts (e.g., contacts with the same name, phone number, or email) are prevented.

5. Theoretical Use Cases

5.1 Adding a Contact

The user selects the "Create Contact" option from the menu.
The system prompts the user to enter the contact's name, phone number, and email.
The system validates the input and adds the contact to the address book if it is valid.

5.2 Searching for a Contact

The user selects the "Search Contact" option from the menu.
The system prompts the user to choose a search criterion (name, phone number, or email).
The user enters the search term, and the system displays matching contacts.

5.3 Editing a Contact

The user selects the "Edit Contact" option from the menu.
The system prompts the user to search for the contact to edit.
The user selects which details to edit (name, phone number, or email) and enters the new values.
The system validates the input and updates the contact if it is valid.

5.4 Deleting a Contact

The user selects the "Delete Contact" option from the menu.
The system prompts the user to enter the name of the contact to delete.
The system removes the contact from the address book.

5.5 Saving and Exiting

The user selects the "Save Contacts" option to save the current state of the address book to the file.
When exiting the program, the system prompts the user to save any unsaved changes.

6. Theoretical Advantages

Simplicity: The system is easy to understand and use, making it suitable for beginners.
Modularity: The system is divided into separate modules, making it easy to extend or modify.
Data Persistence: Contacts are saved to a file, ensuring that data is retained between program runs.

Input Validation: The system ensures that all inputs are valid, preventing errors and maintaining data integrity.

7. Theoretical Limitations

Fixed Size: The address book has a fixed maximum size (100 contacts), which may not be sufficient for all use cases.
Basic Validation: The validation rules (e.g., email must end with ".com") are simplistic and may not cover all edge cases.
No Advanced Features: The system lacks advanced features such as sorting, grouping, or exporting contacts to other formats.

8. Conclusion

The Address Book System is a straightforward yet powerful tool for managing contact information. It highlights essential programming concepts such as data structures, efficient file handling, and structured input validation. The modular design ensures code reusability and maintainability, making it easier to extend the system with additional features.

By implementing case-insensitive search and duplicate checks, the system enhances usability and data integrity. Error handling mechanisms improve stability, reducing the risk of crashes due to invalid inputs.

The use of functions for key operations promotes better organization and readability. Although it lacks advanced features like cloud synchronization or GUI support, it serves as a strong educational project. The program offers hands-on experience with memory management, string manipulation, and persistent storage. Future enhancements could include encryption, a graphical interface, or integration with external databases.

Overall, this project provides a solid foundation for developing more sophisticated contact management systems in C.