ADDRESS BOOK MANGEMENT SYSTEM

Presented By

B REDDY GANESH

TABLE OF CONTENTS

Abstract

The **Address Book Management System** is a C-based console application designed to manage and organize contact details such as name, phone number, and email. The system provides essential features like creating new contacts, searching for existing ones, editing details, deleting unwanted entries, and listing all stored contacts in a tabular format.

The application ensures data accuracy and consistency by implementing strong input validation rules—for example, enforcing a 10-digit format for phone numbers and checking for valid email patterns. Duplicate entries are also prevented.

To maintain contact data even after the program exits, the system reads from and writes to a **CSV file**, demonstrating the use of file handling in C. The project is modular, with clear separation of logic using header and source files, and utilizes structures (struct) to group and manage related data efficiently.

This project is ideal for beginners learning C programming, as it covers key concepts like structures, file operations, modular coding, and user input validation—all through a practical, real-world application.

Introduction

The Address Book Management System aims to simulate a basic contact management application using the C programming language. This project helps students and beginners understand how to build a structured application that manages data effectively, with features that mimic those found in real-world contact apps.

The primary motivation behind this project is to gain hands-on experience with:

- Structured programming using struct.
- File handling for persistent storage.
- Modular code architecture (separation of logic into .c and .h files).
- Robust user input validation.

The system supports a user-friendly, menu-driven interface, making it easy to perform operations like adding, editing, and deleting contacts. It also offers persistent data storage in a CSV file format, allowing contacts to be preserved between sessions.

System Architecture

The system follows a modular architecture, dividing functionality into multiple source and header files for maintainability and clarity. Each module has a specific responsibility:

File Structure:

- **main.c**: Entry point, displays the menu and handles user choices.
- **contact.c**: Core logic for contact operations (create, edit, delete, search and validate).
- **file.c**: Handles reading from and writing to the contact.csv file.
- **contact.h**: Defines the Contact and AddressBook structures and declares contact- related functions.
- **file.h**: Declares file handling functions.

Function Flow:

```
User → Menu (main.c)
        → Create/Search/Edit/Delete/List (contact.c)
        → Validation (within contact.c)
        → File Read/Write (file.c ↔ contact.csv)
```

Data Handling:

- **Storage**: Contact information is stored in an array of struct Contact inside the AddressBook structure.

- **Persistence**: Data is saved in a CSV file and loaded automatically when the program starts.

Modularity:

- Changes in file or contact logic can be made independently.
- Header files ensure all modules can share data and function declarations cleanly.

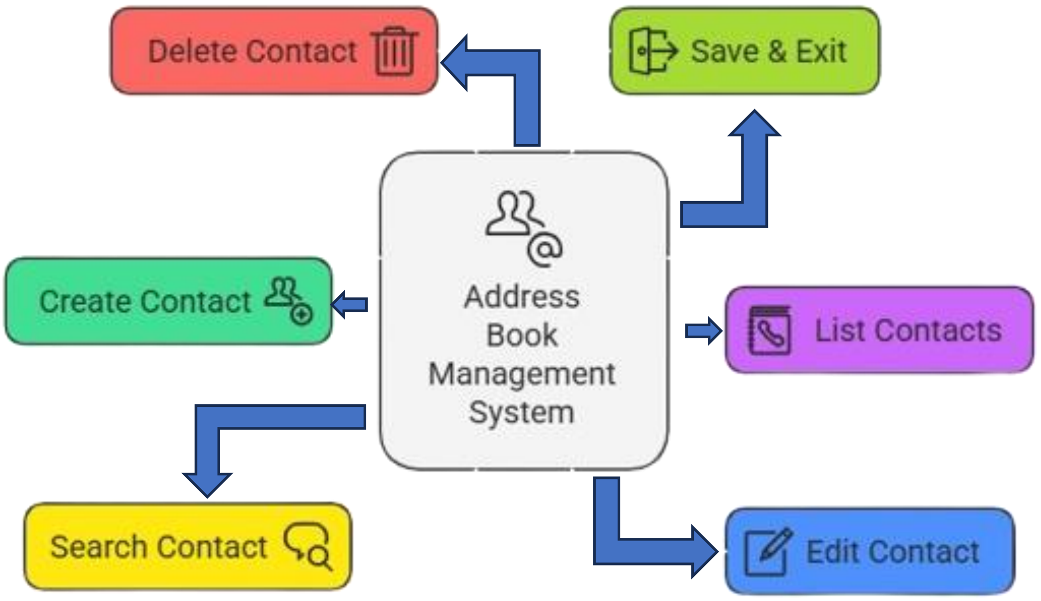
Requirements

Software Requirements:

- **Operating System**: Windows, Linux, or macOS
- **Compiler**: GCC (GNU Compiler Collection) or any standard C compiler
- **Editor/IDE**: Code::Blocks, Turbo C, Visual Studio Code, or any text editor with terminal support
- **Terminal**: Command Prompt, Terminal, or an IDE-integrated console

Features

The Address Book Management System is equipped with a comprehensive set of features to support effective contact management. Each feature is designed with usability, validation, and data persistence in mind:



1. Create Contact

- Add new contact details, including name, phone number, and email.
- Ensures phone number contains exactly 10 digits.
- Verifies email format and prevents duplication of contact entries.

2. Search Contact

- Search for contacts using one of the following identifiers:
    - Full name
    - Phone number
    - Email address
- Displays matched contact details in a user-friendly format.

3. Edit Contact

- Update an existing contact's:
    - Name
    - Phone number
    - Email
    - All fields (if necessary)
- Re-validates all new input before updating the record.

4. Delete Contact

- Delete a contact using a known:
    - Name
    - Phone number
    - Email
- Automatically updates the contact list and adjusts index positions.

5. List All Contacts

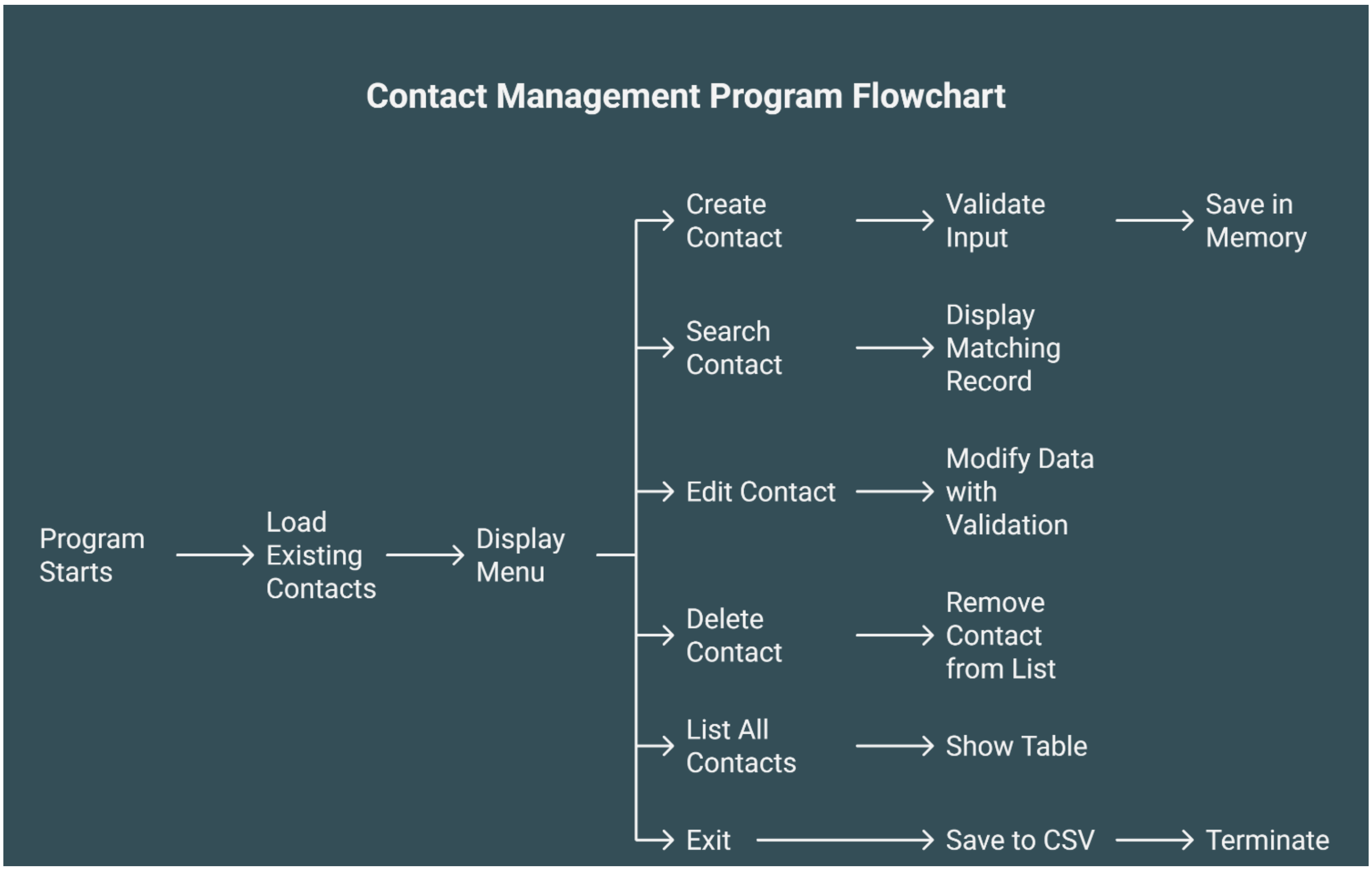
- Lists all stored contacts in a structured, tabular format.
- Useful for reviewing or verifying existing entries.

6. Save and Exit

- Saves all contact data to an external CSV file (contact.csv).
- Ensures data is preserved between program sessions for long-term usage.

These features demonstrate the use of core C programming concepts like structured data (struct), modular coding, dynamic input validation, and persistent file storage.

Workflow/Working



Visual workflow of the Address Book Management System

The Address Book Management System operates through a clear and intuitive menu-driven interface. Below is a step-by-step explanation of how the user interacts with the application and how the program processes input:

User Workflow:

1. **Program Start**
    - The application begins by initializing the address book structure.
    - It attempts to load existing contact data from a file named contact.csv. If the file does not exist, the program starts with an empty address book.
    - Once initialization is complete, a user-friendly menu is displayed in the terminal, listing available operations.
2. **User Menu Options**
    - The menu provides the user with the following functional choices:
        - Create Contact: Add a new entry with validated input.
        - Search Contact: Look up a contact by name, phone, or email.
        - Edit Contact: Modify existing contact details.
        - Delete Contact: Remove a contact entry.
        - List All Contacts: View all saved contacts.
        - Exit: Save all data and close the application.
3. **Action Handling**
    - Based on the selected option, the program guides the user through appropriate prompts.
    - Input is validated at every step (e.g., phone must be 10 digits, email must be correctly formatted).
    - The contact list held in memory is dynamically updated.
    - Feedback is provided for every action, including success or error messages such as "Contact not found" or "Phone number already exists."
4. **Data Persistence**
    - When the user chooses to exit the program, all current contact data is written back to the contact.csv file.
    - This mechanism ensures that no data is lost between sessions, providing a consistent and persistent user experience**.**

Data Structures Used:

The system leverages struct in C to model and manage contact information. These structures allow grouping related fields together, improving organization and simplifying access to contact data.

Contact Structure:

```
typedef struct Contact
 {
         char name[50];
         char phone[20];
         char email[50];
} Contact;
```

- Stores individual contact details.
- Used as the basic unit of the address book.

AddressBook Structure:

```
typedef struct
{
    Contact contacts[MAX_CONTACTS];
    int contactCount;
 } AddressBook;
```

- Holds an array of contacts (up to a predefined maximum).
- Tracks the current number of stored contacts.

These structures form the foundation for all operations including search, edit, delete, and file save/load. The modular approach ensures that data management remains clean and easy to extend.

Validation:

To maintain accurate and reliable contact information, the system implements strict input validation for both phone numbers and email addresses during creation and editing of contacts.

Phone Number Validation

- Length Check: Phone numbers must contain exactly 10 digits.
- Digit Check: All characters must be numerical digits (0–9).
- Uniqueness Check: A phone number already present in the address book cannot be reused.

Examples:

- ✅ Valid: 9876543210
- ❌ Invalid: 98765 → "Phone number must contain exactly 10 digits"
- ❌ Invalid: 98ab543210 → "Phone number must contain digits only"
- ❌ Duplicate: Already exists → "This phone number is already present"

Email Validation

- Minimum Length: Must be longer than 5 characters.
- Format Check:
  - Must contain the @ symbol.
  - Must end with .com.
  - Only lowercase letters, digits, and specific symbols (@, .) are allowed.
- Uniqueness Check: Email must not be duplicated in the address book.

Examples:

- ✅ Valid: example123@gmail.com
- ❌ Invalid: abc@ → "Email is too short"
- ❌ Invalid: testemail.com → "Email must contain '@'"
- ❌ Invalid: User@domain.com → "Only lowercase letters allowed"
- ❌ Duplicate: Already exists → "This email already exists in the address book"

These validations are enforced automatically when the user adds or updates a contact, ensuring that the address book maintains consistent and accurate records.

File Handling:

The Address Book Management System ensures data persistence using standard C file I/O operations. All contacts are saved in a file named contact.csv, allowing users to retain their contact information between sessions.

Loading Contacts from File:

- On program start, the system attempts to open and read from contact.csv.
- It reads the contact count from the first line and loads each contact into memory using fscanf().
- If the file does not exist or is empty, the program initializes with zero contacts.

Saving Contacts to File:

- Before exiting, the system writes all contact data to contact.csv using fprintf().
- The first line stores the number of contacts.

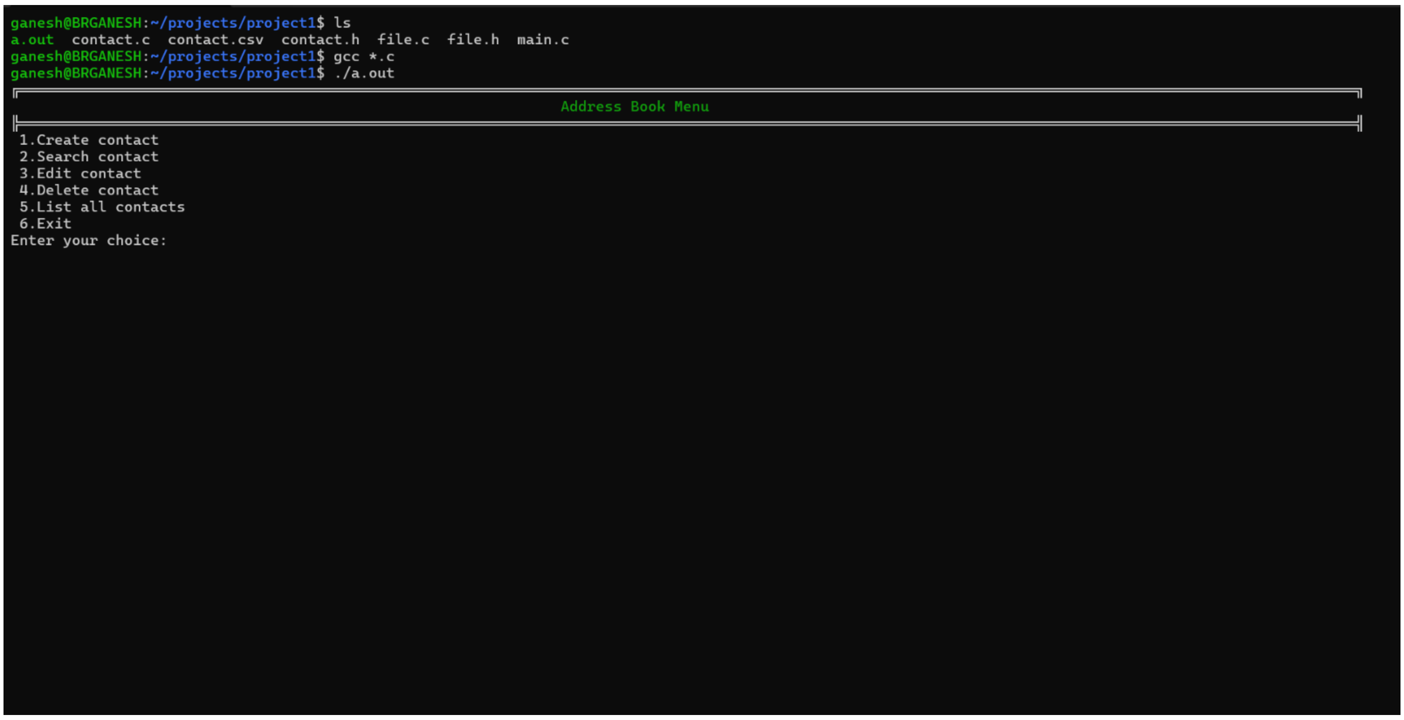
- Each contact is written in CSV format: name,phone,email.

File Operations Used:

- fopen(): Opens the file in read ("r") or write ("w") mode.
- fscanf(): Reads formatted input from the file.
- fprintf(): Writes formatted output to the file.
- fclose(): Closes the file after reading or writing.

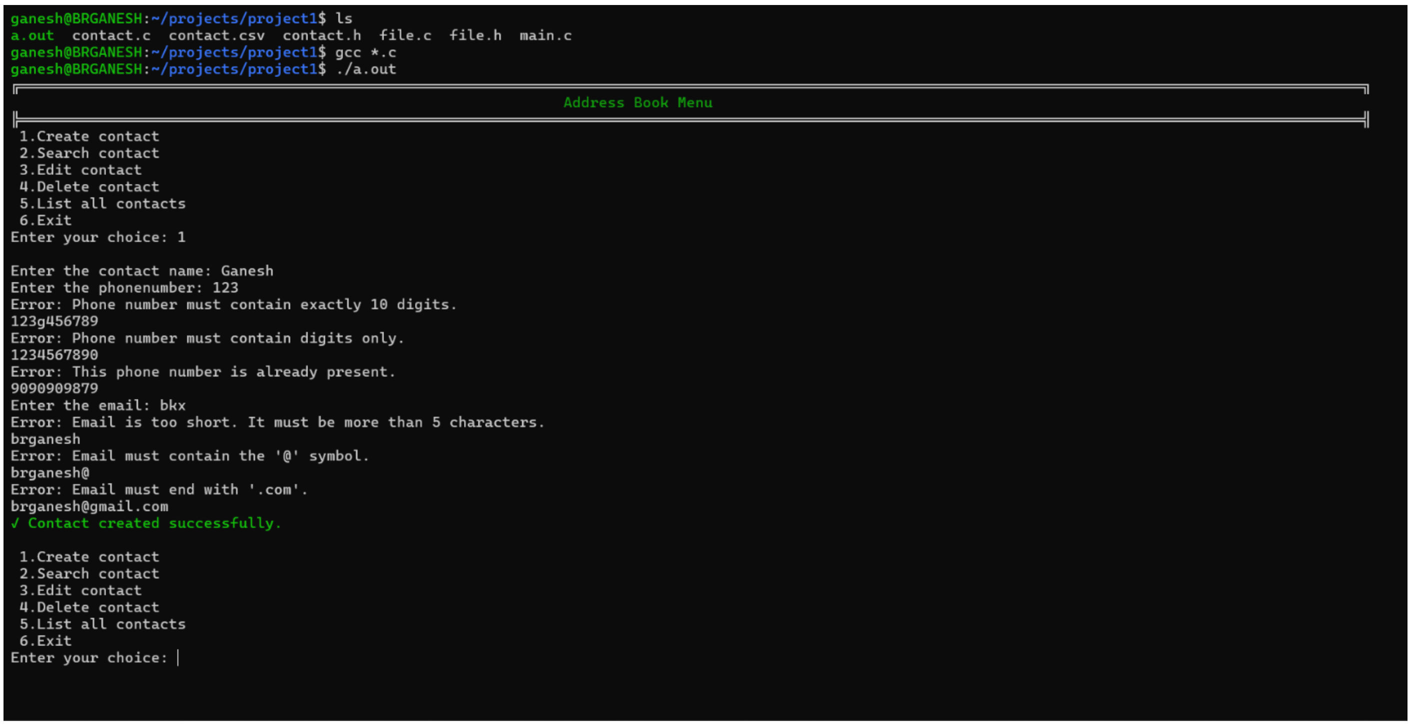
This file handling mechanism ensures the user's contact data is not lost and is automatically restored in the next session.

Sample Outputs



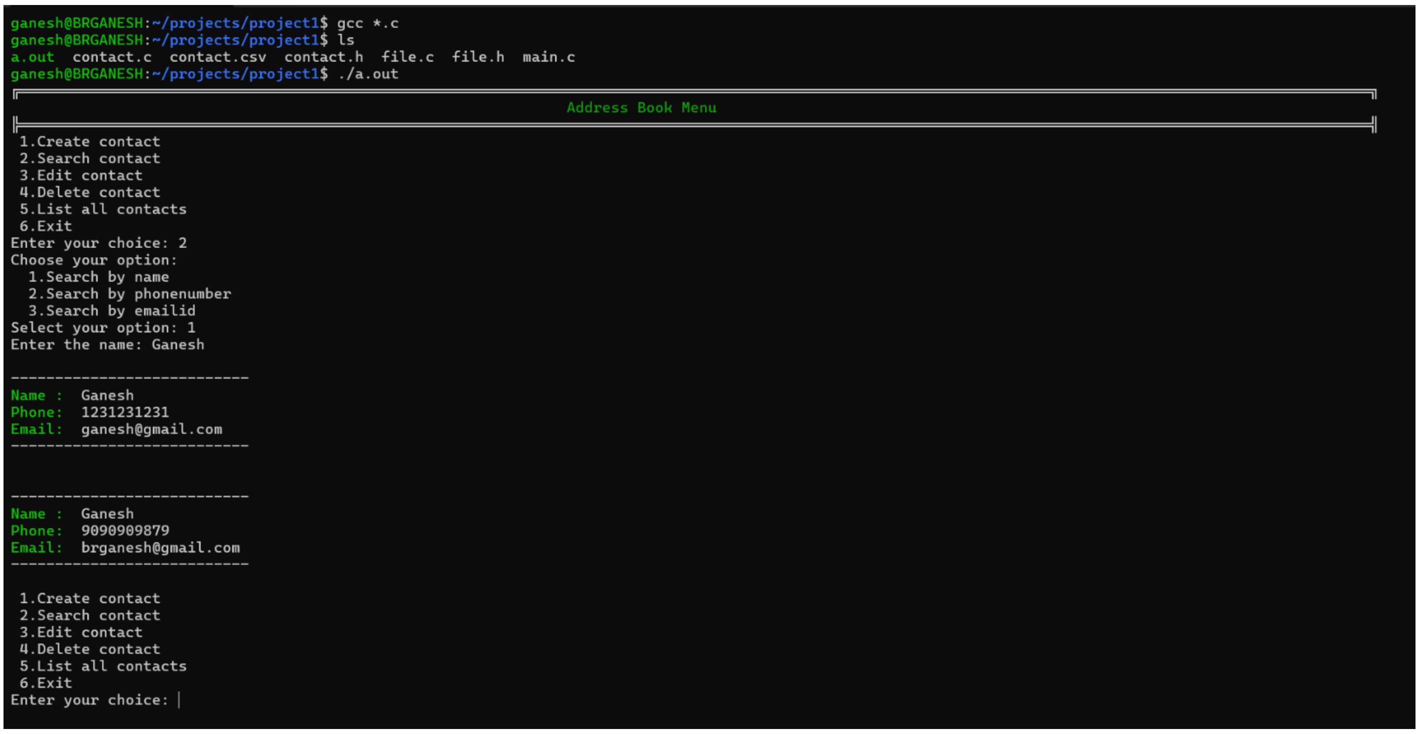
```
ganesh@BRGANESH:~/projects/project1$ ls
a.out  contact.c  contact.csv  contact.h  file.c  file.h  main.c
ganesh@BRGANESH:~/projects/project1$ gcc *.c
ganesh@BRGANESH:~/projects/project1$ ./a.out
                                          Address Book Menu
 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice:
```

Fig1: Displaying the menu



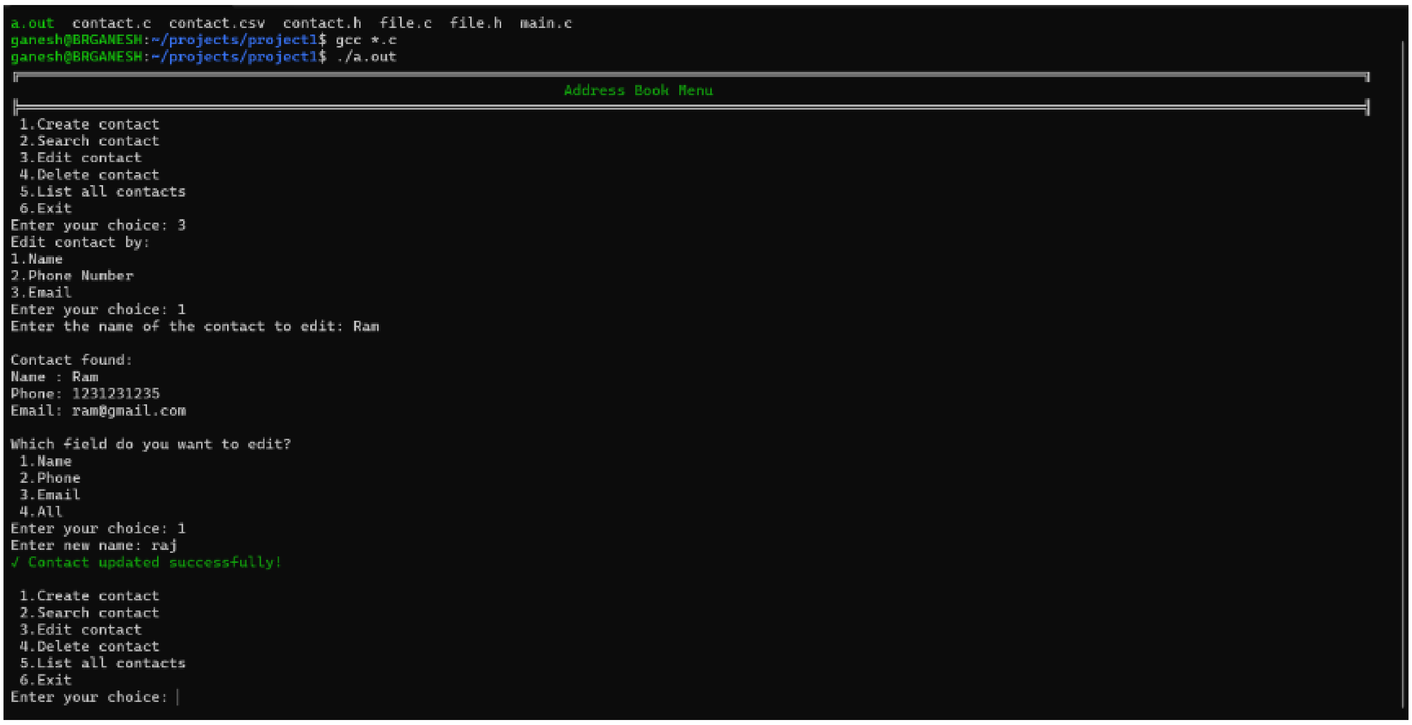
```
ganesh@BRGANESH:~/projects/project1$ ls
a.out  contact.c  contact.csv  contact.h  file.c  file.h  main.c
ganesh@BRGANESH:~/projects/project1$ gcc *.c
ganesh@BRGANESH:~/projects/project1$ ./a.out
                                                          Address Book Menu

 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice: 1

Enter the contact name: Ganesh
Enter the phonenumber: 123
Error: Phone number must contain exactly 10 digits.
123g456789
Error: Phone number must contain digits only.
1234567890
Error: This phone number is already present.
9090909879
Enter the email: bkx
Error: Email is too short. It must be more than 5 characters.
brganesh
Error: Email must contain the '@' symbol.
brganesh@
Error: Email must end with '.com'.
brganesh@gmail.com
✓ Contact created successfully.

 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice: |
```

Fig2: Create contact



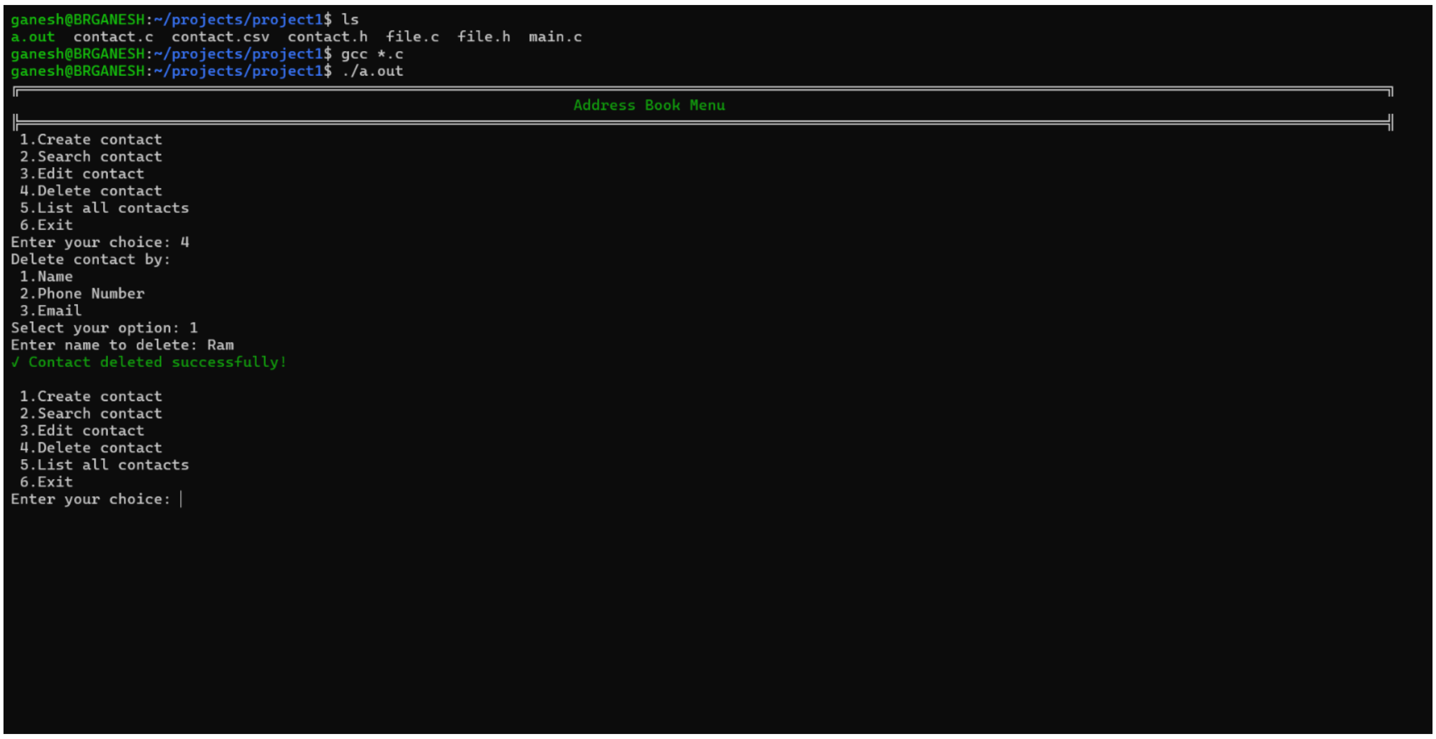
```
ganesh@BRGANESH:~/projects/project1$ gcc *.c
ganesh@BRGANESH:~/projects/project1$ ls
a.out  contact.c  contact.csv  contact.h  file.c  file.h  main.c
ganesh@BRGANESH:~/projects/project1$ ./a.out
                                                          Address Book Menu

 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice: 2
Choose your option:
  1.Search by name
  2.Search by phonenumber
  3.Search by emailid
Select your option: 1
Enter the name: Ganesh

---------------------------
Name :  Ganesh
Phone:  1231231231
Email:  ganesh@gmail.com
---------------------------


---------------------------
Name :  Ganesh
Phone:  9090909879
Email:  brganesh@gmail.com
---------------------------

 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice: |
```

Fig3: Search contact



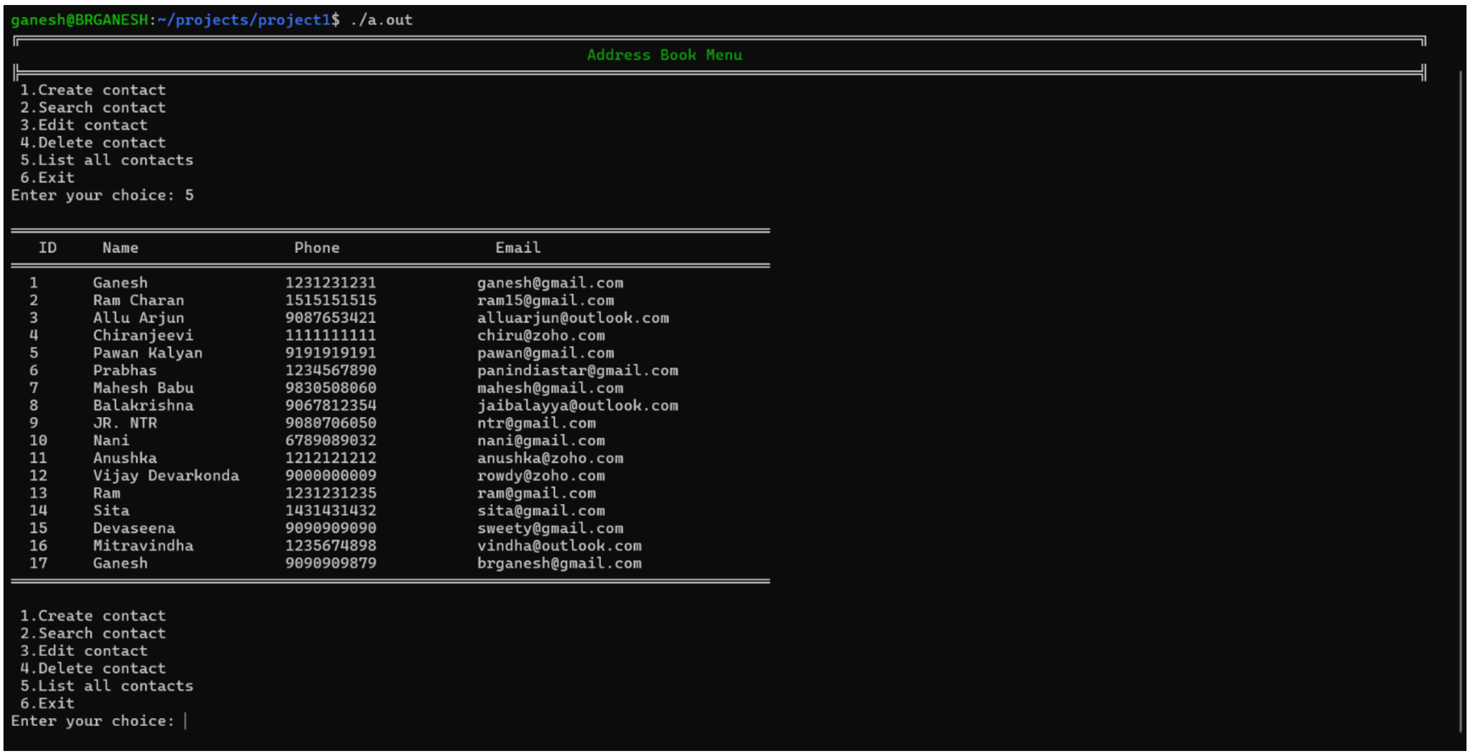
```
a.out  contact.c  contact.csv  contact.h  file.c  file.h  main.c
ganesh@BRGANESH:~/projects/project1$ gcc *.c
ganesh@BRGANESH:~/projects/project1$ ./a.out
                                        Address Book Menu
 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice: 3
Edit contact by:
1.Name
2.Phone Number
3.Email
Enter your choice: 1
Enter the name of the contact to edit: Ram

Contact found:
Name : Ram
Phone: 1231231235
Email: ram@gmail.com

Which field do you want to edit?
 1.Name
 2.Phone
 3.Email
 4.All
Enter your choice: 1
Enter new name: raj
✓ Contact updated successfully!

 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice:
```

Fig4: Edit contact



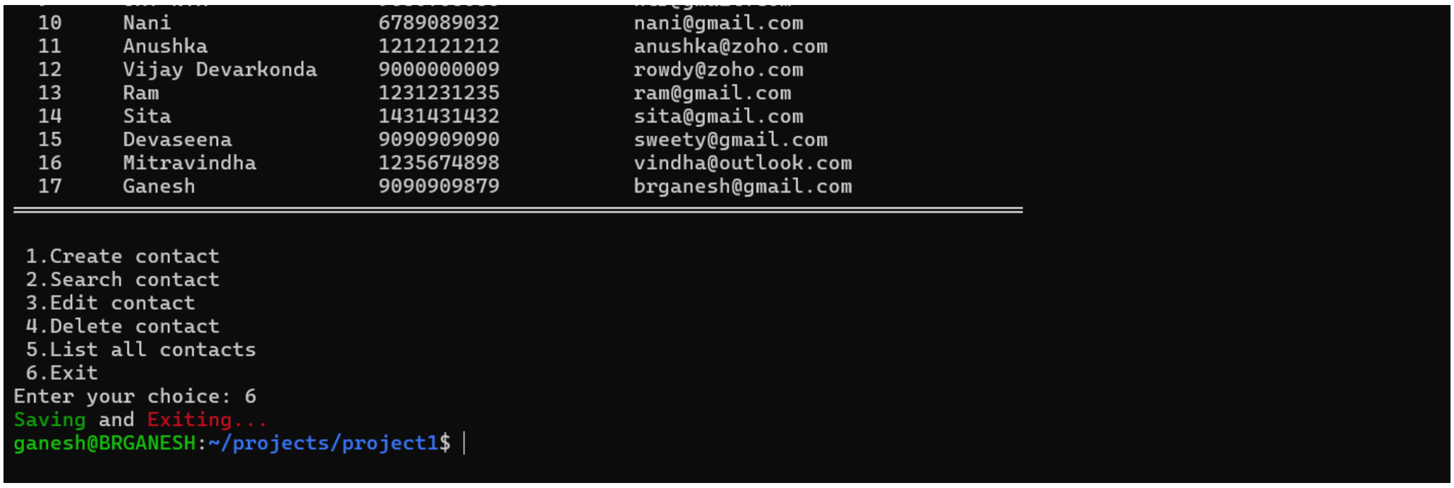
```
ganesh@BRGANESH:~/projects/project1$ ls
a.out  contact.c  contact.csv  contact.h  file.c  file.h  main.c
ganesh@BRGANESH:~/projects/project1$ gcc *.c
ganesh@BRGANESH:~/projects/project1$ ./a.out
                                        Address Book Menu
 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice: 4
Delete contact by:
 1.Name
 2.Phone Number
 3.Email
Select your option: 1
Enter name to delete: Ram
✓ Contact deleted successfully!

 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice:
```

Fig5: Delete contact



```
ganesh@BRGANESH:~/projects/project1$ ./a.out
                                    Address Book Menu
 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice: 5

   ID      Name                 Phone                 Email

   1       Ganesh               1231231231            ganesh@gmail.com
   2       Ram Charan           1515151515            ram15@gmail.com
   3       Allu Arjun           9087653421            alluarjun@outlook.com
   4       Chiranjeevi          1111111111            chiru@zoho.com
   5       Pawan Kalyan         9191919191            pawan@gmail.com
   6       Prabhas              1234567890            panindiastar@gmail.com
   7       Mahesh Babu          9830508060            mahesh@gmail.com
   8       Balakrishna          9067812354            jaibalayya@outlook.com
   9       JR. NTR              9080706050            ntr@gmail.com
   10      Nani                 6789089032            nani@gmail.com
   11      Anushka              1212121212            anushka@zoho.com
   12      Vijay Devarkonda     9000000009            rowdy@zoho.com
   13      Ram                  1231231235            ram@gmail.com
   14      Sita                 1431431432            sita@gmail.com
   15      Devaseena            9090909090            sweety@gmail.com
   16      Mitravindha          1235674898            vindha@outlook.com
   17      Ganesh               9090909879            brganesh@gmail.com


 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice:
```

Fig6: List all contacts



```
   10      Nani                 6789089032            nani@gmail.com
   11      Anushka              1212121212            anushka@zoho.com
   12      Vijay Devarkonda     9000000009            rowdy@zoho.com
   13      Ram                  1231231235            ram@gmail.com
   14      Sita                 1431431432            sita@gmail.com
   15      Devaseena            9090909090            sweety@gmail.com
   16      Mitravindha          1235674898            vindha@outlook.com
   17      Ganesh               9090909879            brganesh@gmail.com


 1.Create contact
 2.Search contact
 3.Edit contact
 4.Delete contact
 5.List all contacts
 6.Exit
Enter your choice: 6
Saving and Exiting...
ganesh@BRGANESH:~/projects/project1$
```

Fig7: Save & Exit

Conclusion

The Address Book Management System successfully demonstrates how foundational concepts of the C programming language—such as structures, file handling, modular design, and validation—can be applied to solve real-world problems. It offers a simple yet effective way to manage and store contact information in a persistent, user-friendly manner.

This project not only enhances programming logic but also improves understanding of data organization and system design. It serves as a valuable exercise for students and beginners to transition from theory to hands-on implementation.

Future Scope

Although this system performs well as a basic contact manager, it can be enhanced in several meaningful ways:

- 🔐 **Add Password Protection**: Secure the address book with authentication.
- ▯ **Sorting Feature**: Allow users to sort contacts alphabetically by name or email.
- 🔍 **Advanced Search**: Implement partial match search (e.g., substrings).
- 📦 **Export Formats**: Support exporting to formats like JSON, XML, or PDF.
- 🖥 **GUI Interface**: Develop a graphical user interface using tools like C++ Qt or Python Tkinter.
- 🌐 **Database Integration**: Connect the system to a SQL or NoSQL database for scalable storage.

These improvements would increase the project's usability, security, and professionalism, making it suitable for more advanced real-world applications.

THANK YOU