

COntact management system

BRAINANZAA

From:

AP22110010753 KARISHMA PATHAN

AP22110010755 JAAYNY MITHRI DASARI   
AP22110010757 NIKHILESWAR KOTHA   
AP2110010777 SUDARSHAN REDDY PALLE

AP22110010746 ASHWIN S

# **Table of Contents**

[Abstract](#_bookmark3) 3

[List of Figures 4](#_bookmark8)

[Introduction 5](#_bookmark8)

[Methodology 6](#_bookmark8)

[Discussion](#_bookmark8) 7

[Concluding Remarks](#_bookmark8) 9

[Future Work](#_bookmark8) 10

[Code](#_bookmark8) 11

[References](#_bookmark19) 18

# **Abstract**

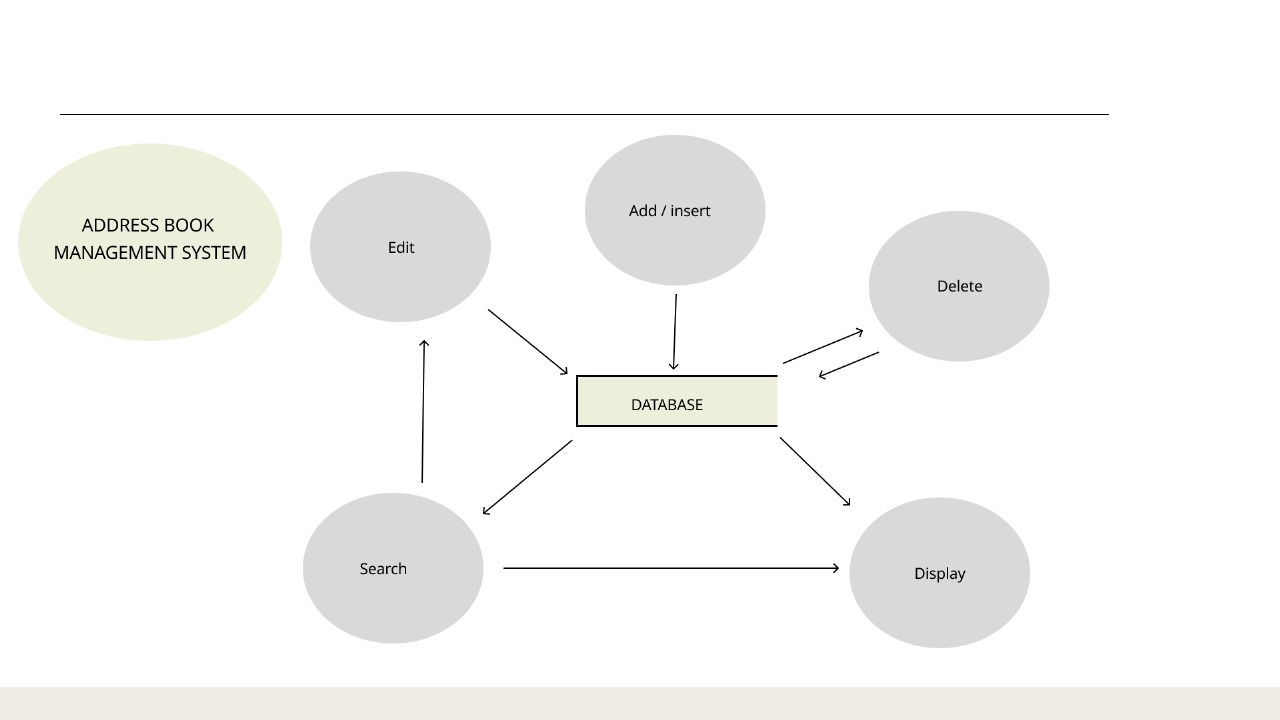
# The provided C++ code presents a comprehensive implementation of an address book program, designed to offer users a robust platform for managing their contacts. The program encompasses a range of essential functionalities, including adding, editing, deleting, viewing, and searching contacts. These operations are seamlessly integrated into a user-friendly menu-driven interface, allowing for intuitive interaction.

# The underlying data structure facilitating contact storage is a map, a key-value container that pairs unique contact identifiers with their corresponding information. This choice enables efficient retrieval and management of contact details. To ensure data persistence across sessions, the program incorporates file input/output operations, utilizing a text file named "contacts.csv" for storing and retrieving contact information.

# The menu-driven interface guides users through various options, fostering a straightforward and accessible user experience. Each menu option corresponds to a specific contact management task, such as adding a new contact, editing an existing one, deleting a contact, viewing all contacts, or searching the address book based on different criteria.

The code implements a structured Contact structure, consisting of attributes such as first name, last name, address, phone number, and email address. These attributes provide a holistic view of each contact, enhancing the completeness of the address book.

# **List of Figures**

Data Flow Diagaram(DFD):

Flow Chart:



**INTRODUCTION:**

This C++ code implements a console-based address book application, offering users the ability to manage their contacts efficiently. Key features include:

Users are presented with a menu offering options to add, view, search, edit, and delete contacts, providing intuitive interaction.

Contacts are stored in a map data structure, allowing for quick retrieval and manipulation. Each contact contains essential details such as first name, last name, address, phone number, and email address.

Contact information is saved to a text file (contacts.csv) upon program exit and loaded back when the program starts, ensuring data persistence across sessions.

Basic input validation is implemented to handle user errors, such as invalid menu choices or duplicate contact entries.

The code includes error-handling mechanisms to gracefully manage exceptions, enhancing the program's reliability and stability.

The code provides a solid foundation that can be extended with additional features or enhancements, such as advanced search options or integration with external services.

This code offers a practical solution for organizing and managing contact information, suitable for personal or small-scale business use. Its simplicity, efficiency, and modularity make it a valuable tool for users seeking a straightforward address book application in C++.

**Methodology :**

Structuring the Data: The Contact struct is defined to encapsulate the attributes of each contact, including first name, last name, address, phone number, and email address. This struct provides a convenient way to organize and manipulate contact information.

A map<string, Contact> is chosen to store the contacts. This data structure allows for efficient retrieval and storage of contacts, with the key being a combination of the first name and last name. Using a map ensures that contacts can be easily looked up and managed.

The program provides a menu-driven interface to interact with users. Users are presented with a set of options and prompted to choose an action based on their requirements. The cin and cout functions are used to handle user input and output.

The program utilizes file input/output operations to persist contact information between program executions. Contacts are saved to a text file (contacts.csv) when the program exits and loaded from the file when the program starts. This ensures that contact data is retained across sessions.

The core functionalities of adding, viewing, searching, editing, and deleting contacts are implemented as separate functions. These functions encapsulate the logic for performing each operation, promoting code modularity and maintainability.

Basic input validation is incorporated to handle potential user errors. For example, the

program ensures that the user enters a valid menu option and checks for duplicate contacts

adding new contacts.

E The code includes error handling mechanisms to gracefully handle scenarios such as file reading errors or contacts not found during editing or deletion operations. This ensures that the program behaves predictably even in unexpected situations.

The code is adequately documented with comments to explain the purpose and functionality of each section. This helps improve code readability and makes it easier for other developers to understand and maintain the codebase.

**Discussion:**

The code implements a basic address book management system that allows users to perform various operations like adding, viewing, searching, editing, and deleting contacts. It employs a command-line interface (CLI) where users interact through a menu-driven system, selecting options to execute specific tasks related to contact management.

Each functionality is encapsulated within separate functions (addContacts(), viewContacts(), searchContact(), editContact(), deleteContact()), ensuring modularity and clear separation of concerns. The main() function serves as the orchestrator, presenting a menu to the user and invoking the appropriate function based on the selected option.

The addContacts() function enables users to input contact details such as first name, last name, address, and contact number. It validates input, avoiding spaces in names with multiple words, and checks for a maximum limit of 100 contacts. Contacts are appended to the 'Contacts.csv' file.

The viewContacts() function displays all stored contacts in a formatted manner by reading the data from the 'AddressBook.txt' file and presenting it in a tabulated format.

The searchContact() function allows users to search for specific contacts based on criteria like first name, last name, address, or contact number. It reads the 'Contacts.csv' file, performs searches according to the user's choice, and displays matching entries.

The editContact() and deleteContact() functions enable editing and deletion of specific contacts respectively. They interact with the 'Contacts.csv' file, enabling users to modify or remove contacts based on user input and update the file accordingly.

The code lacks comprehensive error handling mechanisms for all possible input scenarios, and certain functionalities may not cover edge cases, potentially leading to unexpected behavior in extreme scenarios. Furthermore, the code doesn't implement sophisticated data structures or algorithms for efficient data manipulation, which could be considered for scalability and optimization in handling larger contact databases.

Overall, while this code provides a functional foundation for an address book management system, further improvements in error handling, robustness, and scalability can enhance its reliability and usability in real-world scenarios.

**Concluding Remarks:**

The provided C++ code offers a rudimentary implementation of an address book management system with basic functionalities like adding, viewing, searching, editing, and deleting contacts. It structures these functionalities into separate functions, enhancing modularity and maintainability. However, the code lacks certain robustness and extensibility features that could improve its reliability and usability.

One notable strength of the code lies in its menu-driven approach, providing users with a clear interface to navigate through various operations. The functions for adding, viewing, searching, editing, and deleting contacts operate on a text file ('Contacts.csv') to store and manipulate contact data. The code effectively reads and writes data to this file, allowing users to interact with their contact list persistently across different sessions.

Nevertheless, the code could benefit from several enhancements. Firstly, error handling is minimal, and the code does not comprehensively handle edge cases, potentially leading to unexpected behavior or crashes if users input invalid or unexpected data. Improving input validation and error handling mechanisms could make the system more robust and user-friendly.

Additionally, the code lacks scalability for larger datasets. As it relies on text files for storing contacts, it might face performance issues when handling a considerable volume of data. Implementing more efficient data structures or considering the use of a database could enhance performance and scalability.

Furthermore, the code repetition in certain sections, particularly within functions like editContact() and deleteContact(), could be minimized. Repeated blocks of code should be refactored into reusable functions to adhere to the principle of Don't Repeat Yourself (DRY), improving code readability and maintainability.

In conclusion, while the provided code offers a functional basis for an address book management system, there's ample room for improvement. Strengthening error handling, optimizing for scalability, reducing code redundancy, and enhancing user input validation would significantly enhance the reliability, efficiency, and usability of the system. These improvements would make the system more suitable for real-world applications and larger datasets.

**Future Work**

While the current iteration of the Address Book Management System accomplishes its intended functionalities, there exist several potential avenues for further enhancement and expansion. The following outlines potential areas of future development:

1. User Interface Refinement: Enhancing the user interface (UI) can significantly improve the system's accessibility and user-friendliness. Implementing a graphical user interface (GUI) with more intuitive controls, visual feedback, and interactive elements could make navigation and data management more efficient.
2. Data Encryption and Security Measures: Introducing robust encryption mechanisms and authentication protocols can bolster the security of stored contact information. Implementing features such as password protection or encryption algorithms can ensure data privacy and prevent unauthorized access.
3. Advanced Search and Sorting Capabilities: Expanding the search functionality to support more complex queries based on multiple parameters (e.g., combining first name and last name searches) can enhance the user's ability to find specific contacts quickly. Additionally, incorporating sorting options for contacts based on various criteria (alphabetical, date added, etc.) can further streamline data retrieval.
4. Integration with Cloud Services or Synchronization: Enabling synchronization with cloud storage or online services can facilitate data backup, allowing users to access their address book across multiple devices. Integration with platforms like Google Contacts or Microsoft Outlook could broaden the system's usability and convenience.
5. Contact Grouping and Categorization: Implementing features to categorize contacts into groups (e.g., friends, family, work contacts) could aid in better organizing and managing large contact lists. Offering functionalities to create custom groups and apply filters for easier retrieval would add value to the system.
6. Expansion to Mobile Platforms: Developing a mobile application for iOS and Android devices would extend the system's accessibility and usability to users on the go. A mobile app could offer similar functionalities and sync seamlessly with the desktop version.
7. Performance Optimization and Error Handling: Conducting performance analysis and optimizing code to handle larger datasets efficiently can enhance the system's overall performance. Additionally, implementing robust error-handling mechanisms to manage unexpected user inputs or system errors is essential for a more stable application.

These potential areas for future work present opportunities to elevate the Address Book Management System, adding functionalities, security enhancements, and usability improvements that would further benefit its users.

**CODE:**

#include <iostream>

#include <map>

#include <string>

#include <fstream>

#include <sstream>

#include <iomanip>

using namespace std;

struct Contact {

    string firstName;

    string lastName;

    string address;

    string phoneNumber;

    string emailAddress;

};

map<string, Contact> addressBook;

void addContacts();

void viewContacts();

void searchContact();

void editContact();

void deleteContact();

void loadContacts();

void saveContacts();

int main() {

    loadContacts();

    bool run = true;

    do {

        int option;

        while (true) {

        cout << "\n\n---Search Address Book---" << endl;

        cout << "\n--- Address Book Menu ---" << endl;

        cout << "1. Add Contact" << endl;

        cout << "2. Edit Contact" << endl;

        cout << "3. Delete Contact" << endl;

        cout << "4. View All Contacts" << endl;

        cout << "5. Search Address Book" << endl;

        cout << "6. Exit" << endl;

        cout << "Please enter your choice (1-6): ";

        cin >> option;

        cin.ignore();

        if ( option >= 1 && option <= 6) {

            break;

        } else {

            cout << "Invalid input. Please enter a number from 1 to 6." << endl;

            saveContacts();

            return 1;

        }

        }

        switch (option) {

            case 1:

                cout << "\n--- Add a New Contact ---" << endl;

                addContacts();

                break;

            case 2:

                cout << "\n--- Edit an Existing Contact ---" << endl;

                editContact();

                break;

            case 3:

                cout << "\n--- Delete a Contact ---" << endl;

                deleteContact();

                break;

            case 4:

                cout << "\n--- View All Contacts ---" << endl;

                viewContacts();

                break;

            case 5:

                cout << "\n--- Search in Address Book ---" << endl;

                searchContact();

                break;

            case 6:

                run = false;

                break;

            default:

                cout << "\nInvalid choice. Please choose between 1 to 6." << endl;

        }

    } while (run);

    saveContacts();

    cout << "\nProgram Terminated. All changes have been saved." << endl;

}

void loadContacts() {

    ifstream file("contacts.csv");

    string line;

    while (getline(file, line)) {

        stringstream ss(line);

        string firstName, lastName, address, phoneNumber, emailAddress;

        getline(ss, firstName, ',');

        getline(ss, lastName, ',');

        getline(ss, address, ',');

        getline(ss, phoneNumber, ',');

        getline(ss, emailAddress, ',');

        Contact contact = {firstName, lastName, address, phoneNumber, emailAddress};

        string key = firstName + " " + lastName;

        addressBook[key] = contact;

    }

    file.close();

}

void saveContacts() {

    ofstream file("contacts.csv");

    for (const auto &pair : addressBook) {

        const Contact &contact = pair.second;

        file << contact.firstName << ',';

        file << contact.lastName << ',';

        file << contact.address << ',';

        file << contact.phoneNumber << ',';

        file << contact.emailAddress << '\n';

    }

    file.close();

}

void addContacts() {

    Contact contact;

    cout << "Enter First Name: ";

    getline(cin, contact.firstName);

    cout << "Enter Last Name: ";

    getline(cin, contact.lastName);

    string key = contact.firstName + " " + contact.lastName;

    if (addressBook.find(key) != addressBook.end()) {

        cout << "Contact already exists. Please enter a unique first name and last name." << endl;

        cout<<"Would you like to edit the existing contact? (Y/N): ";

        char ch;

        cin>>ch;

        cin.ignore();

        if(ch=='Y' || ch=='y'){

            editContact();

        }

        return;

    }

    cout << "Enter Address: ";

    getline(cin, contact.address);

    cout << "Enter Contact Number: ";

    getline(cin, contact.phoneNumber);

    cout << "Enter Email Address: ";

    getline(cin, contact.emailAddress);

    addressBook[key] = contact;

    cout << "Contact added successfully!" << endl;

}

void viewContacts() {

    cout << left << setw(20) << "First Name" << setw(20) << "Last Name" << setw(20) << "Phone Number" << setw(30) << "Email Address" << setw(30) << "Address" << endl;

    cout << "-----------------------------------------------------------------------------------------------------------" << endl;

    for (const auto &pair : addressBook) {

        const Contact &contact = pair.second;

        cout << left << setw(20) << contact.firstName << setw(20) << contact.lastName << setw(20) << contact.phoneNumber << setw(30) << contact.emailAddress << setw(30) << contact.address << endl;

    }

}

void searchContact() {

    int choice;

    string searchCriteria;

    while (true) {

        cout << "\n\n---Search Address Book---" << endl;

        cout << "1.) First name" << endl;

        cout << "2.) Last name" << endl;

        cout << "3.) Address" << endl;

        cout << "4.) Contact " << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        cin.ignore();

        if (choice >= 1 && choice <= 4) {

            break;

        } else {

            cout << "Please Enter choice from 1 to 4" << endl;

        }

    }

    switch (choice) {

        case 1:

            cout << "Enter First Name: ";

            break;

        case 2:

            cout << "Enter Last Name: ";

            break;

        case 3:

            cout << "Enter Address: ";

            break;

        case 4:

            cout << "Enter Contact: ";

            break;

    }

    getline(cin, searchCriteria);

    bool visited=false;

    bool found = false;

    for (const auto &pair : addressBook) {

        const Contact &contact = pair.second;

        switch (choice) {

            case 1:

                if (contact.firstName == searchCriteria) {

                    found = true;

                }

                break;

            case 2:

                if (contact.lastName == searchCriteria) {

                    found = true;

                }

                break;

            case 3:

                if (contact.address == searchCriteria) {

                    found = true;

                }

                break;

            case 4:

                if (contact.phoneNumber == searchCriteria) {

                    found = true;

                }

                break;

        }

        if (found) {

            if(visited==false){

                cout << "\n--- Search in Address Book ---" << endl;

                cout << left << setw(20) << "First Name" << setw(20) << "Last Name" << setw(20) << "Phone Number" << setw(30) << "Email Address" << setw(30) << "Address" << endl;

                cout << "-----------------------------------------------------------------------------------------------------------------------------------------------" << endl;

                visited = true;

                found=false;

            }

            cout << left << setw(20) << contact.firstName << setw(20) << contact.lastName << setw(20) << contact.phoneNumber << setw(30) << contact.emailAddress << setw(30) << contact.address << endl;

        }

    }

    if (!visited) {

        cout << "No matches found." << endl;

    }

}

void editContact() {

    char ch;

    string firstName, lastName;

    cout << "Enter First Name of the contact you want to edit: ";

    getline(cin, firstName);

    cout << "Enter Last Name of the contact you want to edit: ";

    getline(cin, lastName);

     string oldKey = firstName + " " + lastName;

    string key = firstName + " " + lastName;

    if (addressBook.find(key) != addressBook.end()) {

        Contact contact = addressBook[oldKey];

        cout<<"Do you wish to edit the First Name? (Y/N): ";

        cin>>ch;

        cin.ignore();

        if (ch == 'Y' || ch == 'y')

        {

            cout << "Enter new First Name: ";

            getline(cin, contact.firstName);

        }

        cout<<"Do you wish to edit the last Name? (Y/N): ";

        cin>>ch;

        cin.ignore();

        if (ch == 'Y' || ch == 'y')

        {

            cout << "Enter new Last Name: ";

            getline(cin, contact.lastName);

        }else{

            contact.lastName = lastName;

        }

        cout<<"Do you wish to edit the address? (Y/N): ";

        cin>>ch;

        cin.ignore();

        if (ch == 'Y' || ch == 'y')

        {

            cout << "Enter new Address: ";

            getline(cin, contact.address);

        }else{

            contact.address = addressBook[oldKey].address;

        }

        cout<<"Do you wish to edit the contact number? (Y/N): ";

        cin>>ch;

        cin.ignore();

        if (ch == 'Y' || ch == 'y')

        {

            cout << "Enter new Contact Number: ";

            getline(cin, contact.phoneNumber);

        }else{

            contact.phoneNumber = addressBook[oldKey].phoneNumber;

        }

        cout<<"Do you wish to edit the e-mail address? (Y/N): ";

        cin>>ch;

        cin.ignore();

        if (ch == 'Y' || ch == 'y')

        {

        cout << "Enter new Email Address: ";

        getline(cin, contact.emailAddress);

        }else{

            contact.emailAddress = addressBook[oldKey].emailAddress;

        }

        addressBook.erase(oldKey); // Remove the old contact

        string newKey = contact.firstName + " " + contact.lastName;

        addressBook[newKey] = contact; // Add the updated contact with the new key

        cout << "Contact updated." << endl;

    } else {

        cout << "Contact not found." << endl;

    }

}

void deleteContact() {

    string firstName, lastName;

    cout << "Enter First Name of the contact you want to delete: ";

    getline(cin, firstName);

    cout << "Enter Last Name of the contact you want to delete: ";

    getline(cin, lastName);

    string key = firstName + " " + lastName;

    if (addressBook.find(key) != addressBook.end()) {

        addressBook.erase(key);

        cout << "Contact deleted." << endl;

    } else {

        cout << "Contact not found." << endl;

    }

}

**References:**

1. **Online Resources**:
   * GeeksforGeeks. "File Handling in C++." <https://www.geeksforgeeks.org/file-handling-c-classes/>
   * Stack Overflow. "C++ String Input." <https://stackoverflow.com/questions/16523686/c-cin-input-validation>
2. **Documentation**:
   * C++ Reference. <https://en.cppreference.com/>
   * Standard C++ Library reference. <https://www.cplusplus.com/reference/>
3. **Websites and Tutorials**:
   * W3Schools. "C++ File Handling." <https://www.w3schools.com/cpp/cpp_files.asp>
   * Tutorialspoint. "C++ String Manipulation." <https://www.tutorialspoint.com/cplusplus/cpp_strings.htm>

*THANK YOU*