DIGITAL EMPOWERMENT PAKISTAN FOUNDATION

DEP

C++ Programming

MUHAMMAD SHAHZAIB

DEP TASK 2

User Manual

Table of Contents

Contact Management System

1. Introduction

The Contact Management System is a console-based C++ application designed to manage personal contacts efficiently. This application allows users to add, view, delete, edit, and search contacts, all from a simple and user-friendly interface. The contacts are stored persistently in a text file, ensuring that they are available even after the application is closed and reopened.

2. Functions

1. Add Contact

Functionality: This function allows the user to add a new contact to the system. The user must provide a unique ID, the contact's name, and their phone number.

Steps:

- The system prompts the user to enter a unique contact ID.
- The user is then asked to input the contact's name.
- Finally, the user must enter the contact's phone number.
- The new contact is saved to the contacts.txt file.

2. View Contacts

Functionality: This function displays all the contacts currently stored in the system in a tabular format.

Steps:

- The system reads the contact data from the contacts.txt file.
- It displays the contact ID, name, and phone number for each contact.

3. Delete Contact

Functionality: This function allows the user to delete a contact by entering the contact's ID.

Steps:

- The system prompts the user to enter the contact ID they wish to delete.
- If the contact is found, it is removed from the system and the contacts.txt file is updated.
- If the contact is not found, an appropriate message is displayed.

4. Edit Contact

Functionality: This function allows the user to edit the details of an existing contact.

Steps:

- The system prompts the user to enter the contact ID they wish to edit.
- If the contact is found, the user is asked to enter the new name and phone number.
- The contact details are updated in the system and the `contacts.txt` file is updated.
- If the contact is not found, an appropriate message is displayed.

5. Search Contact

Functionality: This function allows the user to search for contacts by ID, name, or phone number.

Steps:

- The system prompts the user to enter a search query.
- It searches through the contacts and displays any contacts that match the query.
- If no contacts match the query, an appropriate message is displayed.

6. Exit

Functionality: This function allows the user to exit the application.

Steps:

- The system exits the loop and terminates the application.

3. Instructions to Run the Program on a Windows System

Prerequisites

- A C++ compiler (e.g., GCC, MSVC)
- A terminal or command prompt

Steps to Compile and Run the Program

1. **Open Command Prompt:**
   - Press `Win + R`, type `cmd`, and press `Enter`.
2. **Navigate to the Directory:**
   - Use the `cd` command to navigate to the directory where the source code file is located.
3. **Compile the Program:**
   - Use a C++ compiler to compile the source code. For example, if using GCC:

     ```
     g++ -o ContactManager ContactManager.cpp
     ```

4. **Run the Program:**
   - Execute the compiled program:

ContactManager

How the Code Works

1. **Initialization:**
    - When the program starts, it initializes a ContactManager object which attempts to load contacts from the contacts.txt file.
2. **Main Menu:**
    - The main menu is displayed, allowing the user to choose an action (add, view, delete, edit, search contacts, or exit).
3. **User Interaction:**
    - Depending on the user's choice, the corresponding function is called.
    - After completing the chosen action, the system prompts the user to press any key to continue, clears the screen, and re-displays the main menu.
4. **File Handling:**
    - Contacts are stored in a file named contacts.txt. The file is read when the program starts and updated whenever contacts are added, edited, or deleted.
5. **Error Handling:**
    - The system checks for errors such as duplicate IDs, invalid input, and file handling issues. Appropriate error messages are displayed to guide the user.
6. **Exiting the Program:**
    - When the user chooses to exit, the program terminates gracefully.

CODE:

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <iomanip>
#include <algorithm>
#include <limits>

using namespace std;

class Contact {
public:
    int id;
    string name;
    string phoneNumber;

    Contact(int id, string name, string phoneNumber) : id(id), name(name),
phoneNumber(phoneNumber) {}
};

class ContactManager {
private:
    vector<Contact> contacts;
    const string fileName = "contacts.txt";

    void saveContacts() {
        ofstream file(fileName, ofstream::trunc);
        if (!file.is_open()) {
            cerr << "Error opening file for writing.\n";
            return;
        }
        for (const auto& contact : contacts) {
            file << contact.id << endl;
            file << contact.name << endl;
            file << contact.phoneNumber << endl;
        }
        file.close();
    }

    bool isUniqueId(int id) {
        return find_if(contacts.begin(), contacts.end(), [id](const Contact& c) {
return c.id == id; }) == contacts.end();
    }
    int getValidatedId() {
        int id;
        while (true) {
            cout << "Enter contact ID: ";
            cin >> id;
            if (cin.fail() || !isUniqueId(id)) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << "Invalid or duplicate ID. Please enter a unique numeric
ID.\n";
            } else {
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                break;
            }
        }
        return id;
    }

public:
    ContactManager() {
        loadContacts();
    }

    void loadContacts() {
        ifstream file(fileName);
        if (!file.is_open()) {
            cerr << "Error opening file for reading.\n";
            return;
        }

        int id;
        string name;
        string phoneNumber;
        while (file >> id >> ws && getline(file, name) && getline(file,
phoneNumber)) {
            contacts.emplace_back(id, name, phoneNumber);
        }
        file.close();
    }

    void addContact() {
        int id = getValidatedId();
        string name, phoneNumber;
        cout << "Enter contact name: ";
        getline(cin, name);
        cout << "Enter contact phone number: ";
        getline(cin, phoneNumber);

        contacts.emplace_back(id, name, phoneNumber);
        saveContacts();
        cout << "Contact added successfully.";
        pressAnyKeyToContinue();
    }

    void viewContacts() {
        if (contacts.empty()) {
            cout << "No contacts available.\n";
        } else {
            cout << setw(5) << "ID" << setw(20) << "Name" << setw(20) << "Phone
Number" << endl;
            for (const auto& contact : contacts) {
                cout << setw(5) << contact.id << setw(20) << contact.name <<
setw(20) << contact.phoneNumber << endl;
            }
        }
        pressAnyKeyToContinue();
    }

    void deleteContact() {
        int id;
        cout << "Enter contact ID to delete: ";
        cin >> id;

        auto it = remove_if(contacts.begin(), contacts.end(), [id](const Contact&
c) { return c.id == id; });
        if (it != contacts.end()) {
            contacts.erase(it, contacts.end());
            saveContacts();
            cout << "Contact deleted successfully.";
        } else {
            cout << "Contact not found.";
        }
        pressAnyKeyToContinue();
    }

    void editContact() {
        int id;
        cout << "Enter contact ID to edit: ";
        cin >> id;
        cin.ignore();

        for (auto& contact : contacts) {
            if (contact.id == id) {
                string newName, newPhoneNumber;
                cout << "Enter new name: ";
                getline(cin, newName);
                cout << "Enter new phone number: ";
                getline(cin, newPhoneNumber);

                contact.name = newName;
                contact.phoneNumber = newPhoneNumber;
                saveContacts();
                cout << "Contact updated successfully.";
                pressAnyKeyToContinue();
                return;
            }
        }
        cout << "Contact not found.";
        pressAnyKeyToContinue();
    }

    void searchContact() {
        cout << "Enter ID, name, or phone number to search: ";
        cin.ignore();
        string query;
        getline(cin, query);

        bool found = false;
        cout << setw(5) << "ID" << setw(20) << "Name" << setw(20) << "Phone 
Number" << endl;
        for (const auto& contact : contacts) {
            if (to_string(contact.id) == query || contact.name.find(query) != 
string::npos || contact.phoneNumber.find(query) != string::npos) {
                found = true;
                cout << setw(5) << contact.id << setw(20) << contact.name << 
setw(20) << contact.phoneNumber << endl;
            }
        }

        if (!found) {
            cout << "No contacts found matching the query.";
        }
        pressAnyKeyToContinue();
    }

    void pressAnyKeyToContinue() {
        cout << "\nPress any key to continue...";
        cin.ignore();
        cin.get();
        #if defined(_WIN32) || defined(_WIN64)
            system("CLS");
        #else
            system("clear");
        #endif
    }
};

int main() {
    ContactManager manager;
    int choice;

    do {
        #if defined(_WIN32) || defined(_WIN64)
            system("CLS");
        #else
            system("clear");
        #endif
        cout << "\nContact Management System\n";
        cout << "1. Add Contact\n";
        cout << "2. View Contacts\n";
        cout << "3. Delete Contact\n";
        cout << "4. Edit Contact\n";
        cout << "5. Search Contact\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                manager.addContact();
                break;
            case 2:
                manager.viewContacts();
                break;
            case 3:
                manager.deleteContact();
                break;
            case 4:
```

```cpp
                manager.editContact();
                break;
            case 5:
                manager.searchContact();
                break;
            case 6:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice. Please try again.";
                manager.pressAnyKeyToContinue();
        }
    } while (choice != 6);

    return 0;
}
```