Name: Noor Akhtar

C++ Programming Internship

Task 2

Contact Management System

Table of Contents

Contact Management System:

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

using namespace std;

class Contact {
public:
    string name;
    string phoneNum;

    Contact(string n, string p) {
        name = n;
        phoneNum = p;

    }
};

class ContactManager {
private:
    vector<Contact> contacts;

    void loadContacts() {
        ifstream infile("Contact.txt");
        if (!infile.is_open()) {
            cerr << "Unable to open file contact.txt" << endl;
            return;
        }

        string name, phoneNumber;
        while (getline(infile, name) && getline(infile, phoneNumber)) {
            contacts.emplace_back(name, phoneNumber);
        }

        infile.close();
    }

    void saveContactsToFile() {
        ofstream outfile("Contact.txt");
        if (!outfile.is_open()) {
            cerr << "Unable to open file " << endl;
            return;
        }

        for (const auto& contact : contacts) {
            outfile << contact.name << endl;
            outfile << contact.phoneNum << endl;
        }

        outfile.close();
    }

public:
    ContactManager() {
        loadContacts();
    }
```

```
    ~ContactManager() {
        saveContactsToFile();
    }

    void addContact(const string& name, const string& phoneNumber) {
        contacts.emplace_back(name, phoneNumber);
        cout << "Contact added successfully!" << endl;
        saveContactsToFile();
    }

    void viewContacts() {
        if (contacts.empty()) {
            cout << "No contacts available." << endl;
            return;
        }

        cout << "Contacts:" << endl;
        for (const auto& contact : contacts) {
            cout << "Name: " << contact.name << ", Phone Number: " <<
contact.phoneNum << endl;
        }
    }

    void deleteContact(const string& name) {
        auto it = remove_if(contacts.begin(), contacts.end(), [&](const Contact&
contact) {
            return contact.name == name;
            });

        if (it != contacts.end()) {
            contacts.erase(it, contacts.end());
            cout << "Contact deleted !" << endl;
            saveContactsToFile();
        }
        else {
            cout << "Contact not found." << endl;
        }
    }
};

void showMenu() {
    cout << "\nContact Management System" << endl;
    cout << "1. Add Contact" << endl;
    cout << "2. View Contacts" << endl;
    cout << "3. Delete Contact" << endl;
    cout << "4. Exit" << endl;
    cout << "Choose an option: ";
}

int main() {
    ContactManager manage;
    int choice;
    string name, phoneNumber;

    while (true) {
        showMenu();
        cin >> choice;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        switch (choice) {
        case 1:
            cout << "Enter name: ";
            getline(cin, name);
```

```
            cout << "Enter phone number: ";
            getline(cin, phoneNumber);
            manage.addContact(name, phoneNumber);
            break;
        case 2:
            manage.viewContacts();
            break;
        case 3:
            cout << "Enter name to delete: ";
            getline(cin, name);
            manage.deleteContact(name);
            break;
        case 4:
            cout << "Exiting..." << endl;
            return 0;
        default:
            cout << "Invalid choice. Please try again." << endl;
        }
    }
}
```

Documentation

1. Introduction

The Contact Management System is designed to efficiently manage and organize contacts. It provides functionalities to add, view, and delete contacts, ensuring the data is persisted in a file named Contact.txt. The application is built using C++ and leverages standard libraries for file handling and user interaction.

2. Setup Instructions

2.1 File Setup

1. **Create the Contact File:**
    - Ensure a file named Contact.txt is present in directory as the source code. This file will store the contacts in a text format, with each contact's name and phone number on different lines.

2.2 Environment Setup

Ensure your development environment is set up for C++ development. The following steps provide a general guide for setting up the environment:

1. **Install a C++ Compiler:**
    - **Linux:** Use the package manager (e.g., sudo apt-get install g++).
    - **macOS:** Use Homebrew (e.g., brew install gcc).
    - **Windows:** Download and install MinGW or use an IDE like Visual Studio.
2. **Set Up an Integrated Development Environment (IDE):**
    - **Visual Studio Code:** A lightweight IDE with extensions for C++ development.
    - **CLion:** A powerful IDE for C++ development by JetBrains.
    - **Visual Studio:** A full-featured IDE for Windows.

3. Running the Application

3.1 Compilation and Execution

Follow these steps to compile and run the application:

1. **Compile the Application:**
    - Use a C++ compiler to compile the source code file main.cpp.
    - Example (using g++ on Linux):

      bash
      Copy code
      g++ -o contact_manager main.cpp

2. **Run the Application:**

- Execute the compiled executable (contact_manager or equivalent based on compilation).
- Example (on Linux):

  bash
  Copy code
  ./contact_manager

4. Functionality

4.1 Adding a Contact

The addContact function allows the user to add a new contact by providing a name and phone number:

```
void addContact(const string& name, const string& phoneNumber)
```

4.2 Viewing Contacts

The viewContacts function displays all the contacts stored in the Contact.txt file:

```
void viewContacts()
```

4.3 Deleting a Contact

The deleteContact function removes a contact by the specified name:

```
void deleteContact(const string& name)
```

5. Error Handling

The application includes mechanisms to handle errors that may occur during file operations and user input:

5.1 File Operation Errors

- **File Opening Errors:**
  - The application checks if the file Contact.txt can be opened for reading or writing. If the file cannot be opened, an error message is displayed.

5.2 User Input Errors

- **Invalid Choices:**
  - The application verifies the user's choice and ask them to enter a valid option if the input is wrong.

6. Conclusion

The “Contact Management System” is a simple effective tool for managing contacts. It demonstrates key concepts in file handling and user interaction in C++. We can further customize and extend the application functionality based on their specific, such as adding additional contact fields or enhancing error-handling mechanisms.