Digital Empowerment Pakistan

Assignment 02

Name: Yashfa Mir

C++ Programming Internship

<u>Task 2</u>

Contact Management System

Contact Management System Code:

```
#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <limits>
#include <algorithm>
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() const
{         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 manager;     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);             manager.addContact(name,
phoneNumber);             break;         case 2:
manager.viewContacts();             break;
case 3:

            cout << "Enter name to delete: ";
getline(cin, name);
manager.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 a tool to effectively manage and organize contact information. It offers functions to add, view, and remove contacts, with data saved in a file named `Contact.txt`. This application is written in C++ and uses standard libraries for file operations and user interaction.

2. **Setup Instructions**

2.1 **Preparing Files**

1. **Create the Contact File:**
   - Ensure a file named `Contact.txt` exists in the same directory as the source code. This file will store contact names and phone numbers, each on separate lines.

2.2 **Environment Configuration**

Ensure your development environment is ready for C++ development. The following steps provide a general guide:

2. **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: Install MinGW or use an IDE like Visual Studio.
3. **Set Up an Integrated Development Environment (IDE):**
   - Visual Studio Code: A lightweight IDE with extensions for C++ development.
   - CLion: A powerful IDE for C++ by JetBrains.
   - Visual Studio: A full-featured IDE for Windows.
3. **Running the Program**

3.1 **Compiling and Executing**

Follow these steps to compile and run the program:
1. **Compile the Application:**
   - Use a C++ compiler to compile the source code file `main.`.
   - Example (using `g++` on Linux):

```
bash  g++ -o contact_manager
main.
```

2. **Run the Application:**

        - Execute the compiled program (e.g., `contact_manager` or the equivalent name).
        - Example (on Linux):

```
bash

./contact_manager
```

4. Features

4.1 **Adding a Contact**

The `addContact` function allows users 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 contacts stored in the `Contact.txt` file:

```
void viewContacts()
```

4.3 **Removing a Contact**

The `deleteContact` function removes a contact by name:

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

5. Error Management

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

5.1 **File Handling Errors**

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

5.2 **User Input Errors**

- **Invalid Choices:**
    - The application prompts the user to enter a valid option if the input is incorrect.

6. Conclusion

The "Contact Management System" is a straightforward yet efficient tool for managing contacts. It demonstrates key concepts in file handling and user interaction in C++. The application can be further customized and extended based on specific needs, such as adding more contact fields or enhancing error-handling mechanisms.