#include <iostream>

#include <string>

#include <vector>

#include <ctime>

#include <algorithm> // Include the algorithm header for sorting

class Contact {

public:

std::string name;

std::string phoneNumber;

struct tm birthday;

Contact(const std::string& name, const std::string& phoneNumber, const struct tm& birthday)

: name(name), phoneNumber(phoneNumber), birthday(birthday) {}

};

// Define a custom comparison function for sorting contacts by name

bool compareByName(const Contact& a, const Contact& b) {

return a.name < b.name;

}

class PhoneDirectory {

private:

std::vector<Contact> contacts;

public:

void addContact(const std::string& name, const std::string& phoneNumber, const struct tm& birthday) {

contacts.push\_back(Contact(name, phoneNumber, birthday));

}

void displayDirectory() {

std::cout << "Phone Directory (Sorted by Name):\n";

// Sort the contacts by name before displaying

std::sort(contacts.begin(), contacts.end(), compareByName);

for (const Contact& contact : contacts) {

std::cout << "Name: " << contact.name << ", Phone: " << contact.phoneNumber;

std::cout << ", Birthday: " << contact.birthday.tm\_mon + 1 << "/" << contact.birthday.tm\_mday << "\n";

}

}

void displayUpcomingBirthdays() {

std::time\_t now = std::time(0);

struct tm today = \*std::localtime(&now);

std::cout << "Upcoming Birthdays:\n";

for (const Contact& contact : contacts) {

if (contact.birthday.tm\_mon == today.tm\_mon && contact.birthday.tm\_mday > today.tm\_mday) {

std::cout << "Name: " << contact.name;

std::cout << ", Birthday: " << contact.birthday.tm\_mon + 1 << "/" << contact.birthday.tm\_mday << "\n";

}

}

}

void searchContact(const std::string& name) {

bool found = false;

for (const Contact& contact : contacts) {

if (contact.name == name) {

std::cout << "Contact found:\n";

std::cout << "Name: " << contact.name << ", Phone: " << contact.phoneNumber;

std::cout << ", Birthday: " << contact.birthday.tm\_mon + 1 << "/" << contact.birthday.tm\_mday << "\n";

found = true;

}

}

if (!found) {

std::cout << "Contact with name '" << name << "' not found.\n";

}

}

void editContact(const std::string& name) {

for (Contact& contact : contacts) {

if (contact.name == name) {

std::cout << "Editing contact: " << contact.name << "\n";

std::cout << "Enter new phone number: ";

std::cin.ignore(); // Clear the newline character from the input buffer.

std::getline(std::cin, contact.phoneNumber);

std::cout << "Enter new birthday (MM/DD): ";

std::string bdayStr;

std::getline(std::cin, bdayStr);

int month, day;

sscanf(bdayStr.c\_str(), "%d/%d", &month, &day);

contact.birthday.tm\_mon = month - 1; // Adjust for 0-based month

contact.birthday.tm\_mday = day;

std::cout << "Contact information updated.\n";

return;

}

}

std::cout << "Contact with name '" << name << "' not found.\n";

}

void deleteContact(const std::string& name) {

for (auto it = contacts.begin(); it != contacts.end(); ++it) {

if (it->name == name) {

contacts.erase(it);

std::cout << "Contact '" << name << "' deleted.\n";

return;

}

}

std::cout << "Contact '" << name << "' not found.\n";

}

};

// Function to clear the input buffer

void clearInputBuffer() {

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

int main() {

PhoneDirectory directory;

while (true) {

std::cout << "\nPhone Directory Menu:\n";

std::cout << "1. Add Contact\n";

std::cout << "2. Display Directory (Sorted by Name)\n";

std::cout << "3. Display Upcoming Birthdays\n";

std::cout << "4. Delete Contact\n";

std::cout << "5. Search for Contact\n";

std::cout << "6. Edit Contact\n";

std::cout << "7. Exit\n";

int choice;

std::cout << "Enter your choice: ";

std::cin >> choice;

clearInputBuffer(); // Clear the input buffer.

if (choice == 7) {

break;

}

switch (choice) {

case 1: {

std::string name, phoneNumber;

struct tm birthday;

std::cout << "Enter Name: ";

std::getline(std::cin, name);

std::cout << "Enter Phone Number: ";

std::getline(std::cin, phoneNumber);

std::cout << "Enter Birthday (MM/DD): ";

std::string bdayStr;

std::getline(std::cin, bdayStr);

int month, day;

sscanf(bdayStr.c\_str(), "%d/%d", &month, &day);

birthday.tm\_mon = month - 1; // Adjust for 0-based month

birthday.tm\_mday = day;

directory.addContact(name, phoneNumber, birthday);

std::cout << "Contact added.\n";

break;

}

case 2:

directory.displayDirectory();

break;

case 3:

directory.displayUpcomingBirthdays();

break;

case 4: {

std::string nameToDelete;

std::cout << "Enter the name of the contact to delete: ";

std::getline(std::cin, nameToDelete);

directory.deleteContact(nameToDelete);

break;

}

case 5: {

std::string nameToSearch;

std::cout << "Enter the name of the contact to search for: ";

std::getline(std::cin, nameToSearch);

directory.searchContact(nameToSearch);

break;

}

case 6: {

std::string nameToEdit;

std::cout << "Enter the name of the contact to edit: ";

std::getline(std::cin, nameToEdit);

directory.editContact(nameToEdit);

break;

}

default:

std::cout << "Invalid choice. Please try again.\n";

break;

}

}

std::cout << "Goodbye!\n";

return 0;

}