**Contact Book**

**OOP Project**

**23F-0515 & 23F-0738**

**Code**

**FIRST DELIVERABLE**

**And**

**SECOND DELIVERABLE**

**ADDRESS.H:**

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Address

{

private:

string house;

string street;

string city;

string country;

public:

Address();

Address(string h, string s, string ci, string c);

string getHouse();

string getStreet();

string getCity();

string getCountry();

void setHouse(string h);

void setStreet(string s);

void setCity(string ci);

void setCountry(string c);

bool equals(const Address& address);

void print\_address();

Address copy\_address();

};

**ADDRESS.CPP:**

#include "Address.h"

Address::Address()

{

house = "";

street = "";

city = "";

country = "";

}

Address::Address(string h, string s, string ci, string c)

{

house = h;

street = s;

city = ci;

country = c;

}

string Address::getHouse() {

return house;

}

string Address::getStreet() {

return street;

}

string Address::getCity() {

return city;

}

string Address::getCountry() {

return country;

}

void Address::setHouse(string h) {

house = h;

}

void Address::setStreet(string s) {

street = s;

}

void Address::setCity(string ci) {

city = ci;

}

void Address::setCountry(string c) {

country = c;

}

bool Address::equals(const Address& address)

{

if (this->house == address.house &&

this->street == address.street &&

this->city == address.city &&

this->country == address.country)

{

return true;

}

else

{

return false;

}

}

void Address::print\_address()

{

cout << "House No. " << house << ", Street " << street << ", " << city << ", " << country;

}

Address Address::copy\_address()

{

Address\* temp = new Address;

temp->house = this->house;

temp->street = this->street;

temp->city = this->city;

temp->country = this->country;

return \*temp;

}

**CONTACT.H:**

#pragma once

#ifndef BASIC\_LIB

#define BASIC\_LIB

#include <iostream>

#include <string>

#endif // !BASIC\_LIB

#include "Address.h"

using namespace std;

class Contact

{

private:

string first\_name;

string last\_name;

string mobile\_number;

string email\_address;

Address\* address;

public:

Contact();

Contact(string first, string last, string mobile, string email, Address\* add);

Contact(const Contact& other);

string getFirstName();

string getLastName();

string getMobileNumber();

string getEmailAddress();

Address\* getAddress();

void setFirstName(string first);

void setLastName(string last);

void setMobileNumber(string mobile);

void setEmail(string email);

void setAddress(Address\* add);

bool equals(Contact contact);

Contact\* copy\_contact();

void printContactInfo();

};

**CONTACT.CPP:**

#include "Contact.h"

Contact::Contact()

{

first\_name = "";

last\_name = "";

mobile\_number = "";

email\_address = "";

address = nullptr;

}

Contact::Contact(string first, string last, string mobile, string email, Address\* add)

{

if (!first.empty()) {

first\_name = first;

}

if (!last.empty()) {

last\_name = last;

}

if (mobile.size() == 11) {

mobile\_number = mobile;

}

email\_address = email;

address->setHouse(add->getHouse());

address->setStreet(add->getStreet());

address->setCity(add->getCity());

address->setCountry(add->getCountry());

}

Contact::Contact(const Contact& other)

{

this->first\_name = other.first\_name;

this->last\_name = other.last\_name;

this->mobile\_number = other.mobile\_number;

this->email\_address = other.email\_address;

this->address = new Address;

/\*this->address = other.address;\*/

this->address->setCity(other.address->getCity());

this->address->setCountry(other.address->getCountry());

this->address->setHouse(other.address->getHouse());

this->address->setStreet(other.address->getStreet());

}

string Contact::getFirstName() { return first\_name; }

string Contact::getLastName() { return last\_name; }

string Contact::getMobileNumber() { return mobile\_number; }

string Contact::getEmailAddress() { return email\_address; }

Address\* Contact::getAddress() { return address; }

void Contact::setFirstName(string first) {

if (!first.empty()) {

first\_name = first;

}

}

void Contact::setLastName(string last) {

if (!last.empty()) {

last\_name = last;

}

}

void Contact::setMobileNumber(string mobile) {

if (mobile.size() == 11) {

mobile\_number = mobile;

}

}

void Contact::setEmail(string email) {

email\_address = email;

}

void Contact::setAddress(Address\* add) {

address = add;

}

bool Contact::equals(Contact contact) {

if (this->first\_name == contact.first\_name &&

this->last\_name == contact.last\_name &&

this->mobile\_number == contact.mobile\_number &&

this->email\_address == contact.email\_address &&

this->address->equals(\*(contact.address)))

{

return true;

}

return false;

}

Contact\* Contact::copy\_contact() {

Contact\* newContact = new Contact;

newContact->first\_name = this->first\_name;

newContact->last\_name = this->last\_name;

newContact->mobile\_number = this->mobile\_number;

newContact->email\_address = this->email\_address;

newContact->address = this->address;

return newContact;

}

void Contact::printContactInfo()

{

cout << "Name: " << first\_name << " " << last\_name << endl;

cout << "Mobile Number: " << mobile\_number << endl;

cout << "Email Address: " << email\_address << endl;

cout << "Address: ";

address->print\_address();

cout << endl;

}

**CONTACTSBOOK.H:**

#pragma once

#include "Address.h"

#include "Contact.h"

#include <iostream>

#include <string>

using namespace std;

#ifndef BASIC\_LIB

#define BASIC\_LIB

#endif // !BASIC\_LIB

class Group {

private:

string groupName;

Contact\*\* groupContacts;

int capacity;

int size;

public:

Group();

Group(string name, int cap);

~Group();

bool checkGroup(const Group& others);

string getGroupName();

void addContact(Contact\* contact);

void removeContact(int num);

int findContactIndex(string contactName);

void displayContacts();

};

class ContactsBook {

private:

static const int size\_of\_contacts = 100;

Contact\* contacts\_list[size\_of\_contacts];

static int contacts\_count;

Group\* groups[100];

int groups\_count = 0;

bool full();

void resize\_list();

void sort\_contacts\_list(int choice);

public:

ContactsBook();

void add\_contact(const Contact& contact);

int total\_contacts();

Contact\* search\_contact(string first\_name, string last\_name);

Contact\* search\_contact(string mobile);

Contact\* search\_contact(Address\* add);

Contact\* general\_search(string general);

void printAllContacts();

void printSpecificContact(int num);

void print\_contacts\_sorted(int choice);

void deleteContact(int num);

void updateContact(int num, Contact contact);

void merge\_duplicates();

void save\_to\_file();

void input\_from\_file();

void advance\_search();

void createGroup(string groupName);

void addContactToGroup(Contact contact, string groupName);

void removeContactfromGroup();

void printGroupContacts();

void deleteGroup(string groupName);

};

class SearchHistory

{

string history[100];

static int historyCount;

public:

void SearchBox(string hist);

void displayHistory();

};

**CONTACTSBOOK.CPP:**

#include "ContactsBook.h"

#include <fstream>

Group::Group() {

groupName = " ";

capacity = 0;

size=0;

}

Group::Group(string name, int cap = 20) : capacity(cap) {

groupName = name;

groupContacts = new Contact \* [capacity];//array of pointers to contact objects

}

Group::~Group() {

for (int i = 0; i < size; ++i) {

delete groupContacts[i];

groupContacts[i] = nullptr; // Set pointer to nullptr after deletion

}

delete[] groupContacts;

}

bool Group::checkGroup(const Group& others)

{

if (this->groupName == others.groupName)

{

return true;

}

}

string Group::getGroupName() { return groupName; }

void Group::addContact(Contact\* contact) {

if (contact != nullptr) {

if (size < capacity) {

groupContacts[size++] = new Contact(\*contact);

cout << "Contact added to group '" << groupName << "' successfully." << endl;

}

else {

cout << "Group capacity reached. Cannot add more contacts." << endl;

}

}

else {

cout << "Invalid contact. Cannot add to group." << endl;

}

}

void Group::removeContact(int num) {

if (num >= 1 && num <= capacity && groupContacts[num - 1] != nullptr) {

delete groupContacts[num - 1];

for (int i = num - 1; i < size - 1; i++) {

groupContacts[i] = groupContacts[i + 1];

}

groupContacts[size - 1] = nullptr; // Update the last element to nullptr

size--;

cout << "Contact removed from group '" << groupName << "' successfully." << endl;

}

else {

cout << "Invalid contact number or contact not found." << endl;

}

}

int Group::findContactIndex(string contactName) {

for (int i = 0; i < size; ++i) {

if (groupContacts[i]->getFirstName() == contactName) {

return i;

}

}

return -1; // Return -1 if contact not found

}

void Group::displayContacts() {

cout << "Contacts in group '" << groupName << "':" << endl;

for (int i = 0; i < size; i++) {

cout << "\nContact " << i + 1 << ":\n";

groupContacts[i]->printContactInfo();

cout << endl;

}

}

bool ContactsBook::full()

{

if (contacts\_count == size\_of\_contacts) {

return true;

}

else

{

return false;

}

}

void ContactsBook::resize\_list() {

//double the size of array

int newSize = size\_of\_contacts \* 2;

Contact\* new\_contacts\_list = new Contact[newSize];

//copy existing contacts to the new array

for (int i = 0; i < contacts\_count; i++) {

new\_contacts\_list[i] = \*contacts\_list[i];

}

// Free memory allocated for the old contacts list

\*contacts\_list = new\_contacts\_list;

delete[] contacts\_list;

}

void ContactsBook::sort\_contacts\_list(int choice)

{

for (int i = 0; i < contacts\_count - 1; ++i)

{

for (int j = 0; j < contacts\_count - i - 1; ++j)

{

// if sort by first name

if (choice == 1)

{

if (contacts\_list[j]->getFirstName() > contacts\_list[j + 1]->getFirstName())

{

//swap contacts

Contact\* temp = contacts\_list[j];

contacts\_list[j] = contacts\_list[j + 1];

contacts\_list[j + 1] = temp;

}

}

// if sort by last name

else if (choice == 2)

{

if (contacts\_list[j]->getLastName() > contacts\_list[j + 1]->getLastName())

{

// swap contacts

Contact\* temp = contacts\_list[j];

contacts\_list[j] = contacts\_list[j + 1];

contacts\_list[j + 1] = temp;

}

}

}

}

}

ContactsBook::ContactsBook()

{

for (int i = 0; i < 100; ++i) {

groups[i] = nullptr;

}

for (int i = 0; i < size\_of\_contacts; i++)

{

contacts\_list[i] = nullptr;

}

}

void ContactsBook::add\_contact(const Contact& contact) {

if (full()) {

resize\_list();

}

contacts\_list[contacts\_count] = new Contact(contact);

cout << "\nContact has been added successfully\n";

contacts\_count++;

}

int ContactsBook::total\_contacts() {

return contacts\_count;

}

Contact\* ContactsBook::search\_contact(string first\_name, string last\_name) {

bool flag = false;

for (int i = 0; i < contacts\_count; i++) {

if (contacts\_list[i]->getFirstName() == first\_name && contacts\_list[i]->getLastName() == last\_name) {

flag = true;

//return a copy of the found contact

return new Contact(\*(contacts\_list[i]));

}

}

if (!flag)

{

cout << "\nNo Contact found with such name\n";

return nullptr; //contact not found

}

}

Contact\* ContactsBook::search\_contact(string mobile) {

bool flag = false;

for (int i = 0; i < contacts\_count; ++i) {

if (contacts\_list[i]->getMobileNumber() == mobile) {

flag = true;

//return a copy of the found contact

return new Contact(\*(contacts\_list[i]));

}

}

if (!flag)

{

cout << "\nNo Contact found with such number\n";

//no cont found

return nullptr;

}

}

Contact\* ContactsBook::search\_contact(Address\* add) {

bool flag = false;

for (int i = 0; i < contacts\_count; i++) {

if (contacts\_list[i]->getAddress()->equals(\*add)) {

flag = true;

//return a copy of the found contact

return new Contact(\*(contacts\_list[i]));

}

}

if (!flag) {

cout << "\nNo Contact found with such address\n";

return nullptr; // Contact not found

}

}

Contact\* ContactsBook::general\_search(string general)

{

for (int i = 0; i < contacts\_count; i++)

{

if (contacts\_list[i]->getFirstName() == general || contacts\_list[i]->getLastName() == general || contacts\_list[i]->getEmailAddress() == general || contacts\_list[i]->getMobileNumber() == general)

{

return new Contact(\*(contacts\_list[i]));

}

else if (contacts\_list[i]->getAddress()->getHouse() == general || contacts\_list[i]->getAddress()->getStreet() == general || contacts\_list[i]->getAddress()->getCity() == general || contacts\_list[i]->getAddress()->getCountry() == general)

{

return new Contact(\*(contacts\_list[i]));

}

}

cout << "\nNo Contact found with such detail\n";

return nullptr;

}

void ContactsBook::printAllContacts()

{

for (int i = 0; i < contacts\_count; i++)

{

cout << "\nContact " << i + 1 << ":\n";

contacts\_list[i]->printContactInfo();

cout << endl;

}

}

void ContactsBook::printSpecificContact(int num)

{

if (contacts\_list[num - 1] == nullptr)

{

cout << "\nNo contact found at " << num << endl;

return;

}

cout << "\nContact: " << num << ":\n";

contacts\_list[num - 1]->printContactInfo();

}

void ContactsBook::print\_contacts\_sorted(int choice) {

sort\_contacts\_list(choice);

for (int i = 0; i < contacts\_count; i++)

{

cout << "\nContact " << i + 1 << ":\n";

contacts\_list[i]->printContactInfo();

cout << endl;

}

}

void ContactsBook::deleteContact(int num)

{

cout << "\nContact number " << num << " has been deleted successfully\n";

delete contacts\_list[num - 1];

//tp fill the space

for (int i = num - 1; i < contacts\_count - 1; i++) {

contacts\_list[i] = contacts\_list[i + 1];

}

contacts\_count--;

}

void ContactsBook::updateContact(int num, Contact contact)

{

delete contacts\_list[num - 1];

contacts\_list[num - 1] = new Contact(contact);

cout << "\nContact " << num << " has been updated\n";

}

void ContactsBook::merge\_duplicates()

{

bool flag = false;

for (int i = 0; i < contacts\_count; i++)

{

for (int j = i + 1; j < contacts\_count; j++)

{

if (contacts\_list[i]->equals(\*contacts\_list[j]))

{

cout << "\nA duplicate contact has been found and both are merged\n";

delete contacts\_list[j];

flag = true;

//to fill up the space

for (int k = j; k < contacts\_count - 1; k++) {

contacts\_list[k] = contacts\_list[k + 1];

}

contacts\_count--;

j--;

}

}

}

if (!flag)

{

cout << "\nNo duplicate contact found!\n";

}

}

void ContactsBook::save\_to\_file() {

ofstream outfile("out.txt");

outfile.open("out.txt");

if (!outfile.is\_open()) {

cout << "\nError! File can not be openned" << endl;

return;

}

for (int i = 0; i < contacts\_count; i++) {

outfile << "Contact " << i + 1 << ":\n";

outfile << "Name: " << contacts\_list[i]->getFirstName() << " " << contacts\_list[i]->getLastName() << endl;

outfile << "Mobile Number: " << contacts\_list[i]->getMobileNumber() << endl;

outfile << "Address: House No. " << contacts\_list[i]->getAddress()->getHouse() << ", Street ";

outfile << contacts\_list[i]->getAddress()->getStreet() << ", " << contacts\_list[i]->getAddress()->getCity();

outfile << ", " << contacts\_list[i]->getAddress()->getCountry() << endl << endl;

}

outfile.close();

cout << "\nContacts saved to file successfully\n";

}

void ContactsBook::input\_from\_file() {

ifstream infile("in.txt");

infile.open("in.txt");

string line;

if (!infile.is\_open()) {

cout << "\nError! File can not be openned" << endl;

return;

}

cout << "\nContacts from file:" << endl;

while (getline(infile, line)) {

cout << line << endl;

}

infile.close();

}

void ContactsBook::advance\_search() {

string advance\_string, store\_string;

int count = 0, k = 0, x = 0, contact\_count = 0;

cout << "Enter your input (without space): ";

cin >> advance\_string;

for (int n = 0; n < contacts\_count; n++)

{

store\_string = contacts\_list[n]->getFirstName();

for (int i = 0; i < strlen(advance\_string.c\_str()); i++)

{

for (int j = 0; j < strlen(store\_string.c\_str()); j++)

{

if (store\_string[j] == advance\_string[i] && k < j) {

count++;

k = j;

x = i;

}

}

}

store\_string = contacts\_list[n]->getLastName();

k = 0;

for (int i = x; i < strlen(advance\_string.c\_str()); i++)

{

for (int j = 0; j < strlen(store\_string.c\_str()); j++)

{

if (store\_string[j] == advance\_string[i] && k < j) {

count++;

k = j;

}

}

}

if (count == strlen(advance\_string.c\_str()))

{

cout << "\nFirst Name: " << contacts\_list[n]->getFirstName() << endl;

cout << "Last Name: " << contacts\_list[n]->getLastName() << endl;

cout << "Mobile no.: " << contacts\_list[n]->getMobileNumber() << endl;

cout << "Email: " << contacts\_list[n]->getEmailAddress() << endl;

cout << "Address: " << endl;

cout << "House: " << contacts\_list[n]->getAddress()->getHouse() << endl;

cout << "Street: " << contacts\_list[n]->getAddress()->getStreet() << endl;

cout << "City: " << contacts\_list[n]->getAddress()->getCity() << endl;

cout << "Country: " << contacts\_list[n]->getAddress()->getCountry() << endl;

// cout << "Group: " << contacts\_list[n]->getGroup() << endl << endl;

cout << endl;

contact\_count++;

}

}

count = 0;

for (int n = 0; n < contacts\_count; n++)

{

store\_string = contacts\_list[n]->getMobileNumber();

for (int i = 0; i < strlen(advance\_string.c\_str()); i++)

{

for (int j = 0; j < strlen(store\_string.c\_str()); j++)

{

if (store\_string[j] == advance\_string[i] && k < j) {

count++;

k = j;

x = i;

}

}

}

if (count == strlen(advance\_string.c\_str()))

{

cout << "\nFirst Name: " << contacts\_list[n]->getFirstName() << endl;

cout << "Last Name: " << contacts\_list[n]->getLastName() << endl;

cout << "Mobile no.: " << contacts\_list[n]->getMobileNumber() << endl;

cout << "Email: " << contacts\_list[n]->getEmailAddress() << endl;

cout << "Address: " << endl;

cout << "House: " << contacts\_list[n]->getAddress()->getHouse() << endl;

cout << "Street: " << contacts\_list[n]->getAddress()->getStreet() << endl;

cout << "City: " << contacts\_list[n]->getAddress()->getCity() << endl;

cout << "Country: " << contacts\_list[n]->getAddress()->getCountry() << endl;

// cout << "Group: " << contacts\_list[n]->getGroup() << endl << endl;

cout << endl;

contact\_count++;

}

}

count = 0;

for (int n = 0; n < contacts\_count; n++)

{

store\_string = contacts\_list[n]->getEmailAddress();

for (int i = 0; i < strlen(advance\_string.c\_str()); i++)

{

for (int j = 0; j < strlen(store\_string.c\_str()); j++)

{

if (store\_string[j] == advance\_string[i] && k < j) {

count++;

k = j;

x = i;

}

}

}

store\_string = contacts\_list[n]->getLastName();

k = 0;

if (count == strlen(advance\_string.c\_str()))

{

cout << "\nFirst Name: " << contacts\_list[n]->getFirstName() << endl;

cout << "Last Name: " << contacts\_list[n]->getLastName() << endl;

cout << "Mobile no.: " << contacts\_list[n]->getMobileNumber() << endl;

cout << "Email: " << contacts\_list[n]->getEmailAddress() << endl;

cout << "Address: " << endl;

cout << "House: " << contacts\_list[n]->getAddress()->getHouse() << endl;

cout << "Street: " << contacts\_list[n]->getAddress()->getStreet() << endl;

cout << "City: " << contacts\_list[n]->getAddress()->getCity() << endl;

cout << "Country: " << contacts\_list[n]->getAddress()->getCountry() << endl;

// cout << "Group: " << contacts\_list[n]->getGroup() << endl << endl;

cout << endl;

contact\_count++;

}

}

cout << contact\_count << " Contacts Found!" << endl;

}

void ContactsBook::createGroup(string groupName) {

if (groups\_count < 100) {

groups[groups\_count] = new Group(groupName);

groups\_count++;

cout << "\nGroup '" << groupName << "' created successfully." << endl;

}

else {

cout << "\nMaximum number of groups reached. Cannot create more groups." << endl;

}

}

void ContactsBook::addContactToGroup(Contact contact, string groupName) {

for (int i = 0; i < groups\_count; i++) {

if (groups[i]->checkGroup(groupName)) {

groups[i]->addContact(&contact);

return;

}

}

cout << "Group '" << groupName << "' not found." << endl;

}

void ContactsBook::removeContactfromGroup()

{

string name;

int num;

bool flag = false;

cout << "Enter name of group you want to remove contacts from: ";

cin.ignore();

getline(cin, name);

for (int i = 0; i < groups\_count; i++)

{

if (groups[i]->getGroupName() == name) {

system("cls");

groups[i]->displayContacts();

flag = true;

}

}

if (!flag)

{

cout << "\nNo gorup found with name " << name << endl;

return;

}

cout << "\nEnter number of contact you want to delete in the group: ";

cin >> num;

for (int i = 0; i < groups\_count; i++)

{

if (groups[i]->checkGroup(name)) {

groups[i]->removeContact(num);

return;

}

}

cout << "Group '" << name << "' not found." << endl;

}

void ContactsBook::printGroupContacts()

{

for (int i = 0; i < groups\_count; i++)

{

cout << "Group " << i + 1 << ":\n";

groups[i]->displayContacts();

}

}

void ContactsBook::deleteGroup(string groupName) {

int index = -1;

for (int i = 0; i < groups\_count; i++) {

if (groups[i]->getGroupName() == groupName) {

index = i;

break;

}

}

if (index != -1) {

delete groups[index];

for (int i = index; i < groups\_count - 1; ++i) {

groups[i] = groups[i + 1];

}

groups\_count--;

cout << "Group '" << groupName << "' deleted successfully." << endl;

}

else {

cout << "Group '" << groupName << "' not found." << endl;

}

}

int ContactsBook::contacts\_count = 0;

void SearchHistory::SearchBox(string hist)

{

if (historyCount < 100)

{

history[historyCount] = hist;

historyCount++;

}

if (historyCount >= 6)

{

for (int i = 0; i < 5; i++) {

history[i] = history[i + 1];

}

historyCount--;

}

}

void SearchHistory::displayHistory()

{

int n, count = 1;

if (historyCount < 6)

{

n = historyCount - 1;

}

else if (historyCount >= 5)

{

n = 4;

}

cout << "\nSearch history:\n";

for (int i = n; i >= 0; i--)

{

cout << count << ". ";

cout << history[i] << endl;

count++;

}

count = 1;

}

int SearchHistory::historyCount = 0;

**MAIN:**

#include "Address.h"

#include "Contact.h"

#include "ContactsBook.h"

void GroupMenu()

{

cout << "\nGROUP MENU\n";

cout << "\n\n1. Create a new Group\n";

cout << "2. Add a Contact to a Group\n";

cout << "3. Remove a Contact from a Group\n";

cout << "4. Display All Contacts in a Group\n";

cout << "5. Delete a Group\n";

cout << "6. Return to Main Menu\n";

cout << "\nEnter choice: ";

}

void Menu()

{

cout << "\nCONTACT MENU\n";

cout << "\n\n1. Add Contact\n2. Merge Duplicates\n";

cout << "3. Store to file\n4. Load from file\n";

cout << "5. Print Contacts Sorted\n6. Print / delete / update contacts\n";

cout << "7. Search contacts\n8. Display count of contacts\n";

cout << "9. Manage Groups\n10. Search History\n11. Exit Program\n";

cout << "\nEnter choice: ";

}

Contact takeInputs(Contact& tempContact)

{

string first, last, mobile, email;

string house, street, city, country;

Address tempAddress;

cout << "First Name: ";

cin.ignore();

getline(cin, first);

tempContact.setFirstName(first);

cout << "Last Name: ";

getline(cin, last);

tempContact.setLastName(last);

cout << "Mobile Number: ";

getline(cin, mobile);

if (mobile.size() > 11 || mobile.size() < 11)

{

do

{

cout << "\nInvalid Number!\nEnter a valid Number: ";

getline(cin, mobile);

} while (mobile.size() != 11);

}

tempContact.setMobileNumber(mobile);

cout << "Email Address: ";

getline(cin, email);

tempContact.setEmail(email);

cout << "House Number: ";

getline(cin, house);

tempAddress.setHouse(house);

cout << "Street: ";

getline(cin, street);

tempAddress.setStreet(street);

cout << "City: ";

getline(cin, city);

tempAddress.setCity(city);

cout << "Country: ";

getline(cin, country);

tempAddress.setCountry(country);

tempContact.setAddress(&tempAddress);

return tempContact;

}

int main()

{

bool ProgramEnd = false;

int menuChoice, searchChoice, delContChoice;

int groupChoice;

string first, last, mobile, email;

string house, street, city, country;

SearchHistory history;

ContactsBook contactbook;

Address tempAddress, findAddress;

Contact tempContact;

while (!ProgramEnd)

{

Menu();

cin >> menuChoice;

if (menuChoice == 1)

{

cout << "\nEnter details of contact you want to add:\n";

contactbook.add\_contact(takeInputs(tempContact));

cout << endl;

system("pause");

system("cls");

}

else if (menuChoice == 2)

{

contactbook.merge\_duplicates();

cout << endl;

system("pause");

system("cls");

}

else if (menuChoice == 3)

{

contactbook.save\_to\_file();

cout << endl;

system("pause");

system("cls");

}

else if (menuChoice == 4)

{

contactbook.input\_from\_file();

cout << endl;

system("pause");

system("cls");

}

else if (menuChoice == 5)

{

cout << "\n1. Sort All by first names\n2. Sort All by last name\n";

cout << "\nEnter your chocie: ";

cin >> searchChoice;

contactbook.print\_contacts\_sorted(searchChoice);

cout << endl;

system("pause");

system("cls");

}

else if (menuChoice == 6)

{

contactbook.printAllContacts();

cout << "\n1. To view/update/delete a specific contact\n";

cout << "2. Continue";

cout << "\n\nEnter choice : ";

cin >> searchChoice;

if (searchChoice == 1)

{

cout << "Enter contact number you want the details of: ";

cin >> delContChoice;

contactbook.printSpecificContact(delContChoice);

cout << "\n1. To delete contact";

cout << "\n2. To update contact";

cout << "\n3. Continue\n\n";

cout << "Enter your choice: ";

cin >> menuChoice;

if (menuChoice == 1)

{

contactbook.deleteContact(delContChoice);

cout << endl;

system("pause");

system("cls");

continue;

}

else if (menuChoice == 2)

{

cout << "\nEnter the new updated info:\n";

contactbook.updateContact(delContChoice, takeInputs(tempContact));

cout << endl;

system("pause");

system("cls");

continue;

}

else if (menuChoice == 3)

{

system("cls");

continue;

}

else

{

cout << "\nInvalid input\n";

}

}

else if (searchChoice == 2)

{

system("cls");

continue;

}

else

{

cout << "Invalid input\n";

}

cout << endl;

system("pause");

system("cls");

}

else if (menuChoice == 7)

{

system("cls");

cout << "\n1. Search by Name\n2. Search by Mobile number";

cout << "\n3. Search by Address\n4. General Search\n5. Advance Search\n\nEnter your choice: ";

cin >> searchChoice;

if (searchChoice == 1)

{

cout << "\nEnter the name of person you want to find:\n";

cout << "First Name: ";

cin >> first;

cout << "Last Name: ";

cin >> last;

string combinedName = first + " " + last;

history.SearchBox(combinedName);

cout << endl;

if (contactbook.search\_contact(first, last) != nullptr)

{

contactbook.search\_contact(first, last)->printContactInfo();

}

cout << endl;

system("pause");

system("cls");

}

else if (searchChoice == 2)

{

cout << "Enter mobile number of the person you want to find: ";

cin.ignore();

getline(cin, mobile);

history.SearchBox(mobile);

cout << endl;

if (contactbook.search\_contact(mobile) != nullptr)

{

contactbook.search\_contact(mobile)->printContactInfo();

}

cout << endl;

system("pause");

system("cls");

}

else if (searchChoice == 3)

{

cout << "Enter the address of person you want to find:\n";

cout << "House Number: ";

cin >> house;

findAddress.setHouse(house);

cout << "Street: ";

cin >> street;

findAddress.setStreet(street);

cout << "City: ";

cin >> city;

findAddress.setCity(city);

cout << "Country: ";

cin >> country;

findAddress.setCountry(country);

string combineAddress = "House No. " + house + ", Street " + street + ", " + city + ", " + country;

history.SearchBox(combineAddress);

cout << endl;

if (contactbook.search\_contact(&findAddress) != nullptr)

{

contactbook.search\_contact(&findAddress)->printContactInfo();

}

cout << endl;

system("pause");

system("cls");

}

else if (searchChoice == 4)

{

cout << "Enter any info of contact to find: ";

cin.ignore();

getline(cin, first);

history.SearchBox(first);

if (contactbook.general\_search(first) != nullptr)

{

contactbook.general\_search(first)->printContactInfo();

}

cout << endl;

system("pause");

system("cls");

}

else if (searchChoice == 5)

{

contactbook.advance\_search();

cout << endl;

system("pause");

system("cls");

}

else

{

cout << "\nInvalid Input\n";

system("pause");

system("cls");

}

}

else if (menuChoice == 8)

{

cout << "\nTotal number of contacts saved in contact book = " << contactbook.total\_contacts() << endl;

cout << endl;

system("pause");

system("cls");

}

else if (menuChoice == 9)

{

while (true)

{

system("cls");

GroupMenu();

cin >> groupChoice;

if (groupChoice == 1)

{

cout << "\nEnter name of your group: ";

cin.ignore();

getline(cin, first);

contactbook.createGroup(first);

cout << endl;

system("pause");

system("cls");

}

else if (groupChoice == 2)

{

cout << "Enter name of group you want to add contacts in: ";

cin.ignore();

getline(cin, first);

cout << "\nEnter details of contact you want to add:\n";

contactbook.addContactToGroup((takeInputs(tempContact)), first);

cout << endl;

system("pause");

system("cls");

}

else if (groupChoice == 3)

{

contactbook.removeContactfromGroup();

cout << endl;

system("pause");

system("cls");

}

else if (groupChoice == 4)

{

system("cls");

contactbook.printGroupContacts();

cout << endl;

system("pause");

system("cls");

}

else if (groupChoice == 5)

{

cout << "Enter name of group you want to delete: ";

cin.ignore();

getline(cin, first);

contactbook.deleteGroup(first);

cout << endl;

system("pause");

system("cls");

}

else if (groupChoice == 6)

{

system("cls");

break;

}

else

{

cout << "\nInvalid Input\n";

system("pause");

system("cls");

}

system("cls");

}

}

else if (menuChoice == 10)

{

history.displayHistory();

cout << endl;

system("pause");

system("cls");

}

else if (menuChoice == 11)

{

ProgramEnd = true;

}

else

{

cout << "\nInvalid Input\n";

system("pause");

system("cls");

}

}

cout << "\nProgram Ended\n";

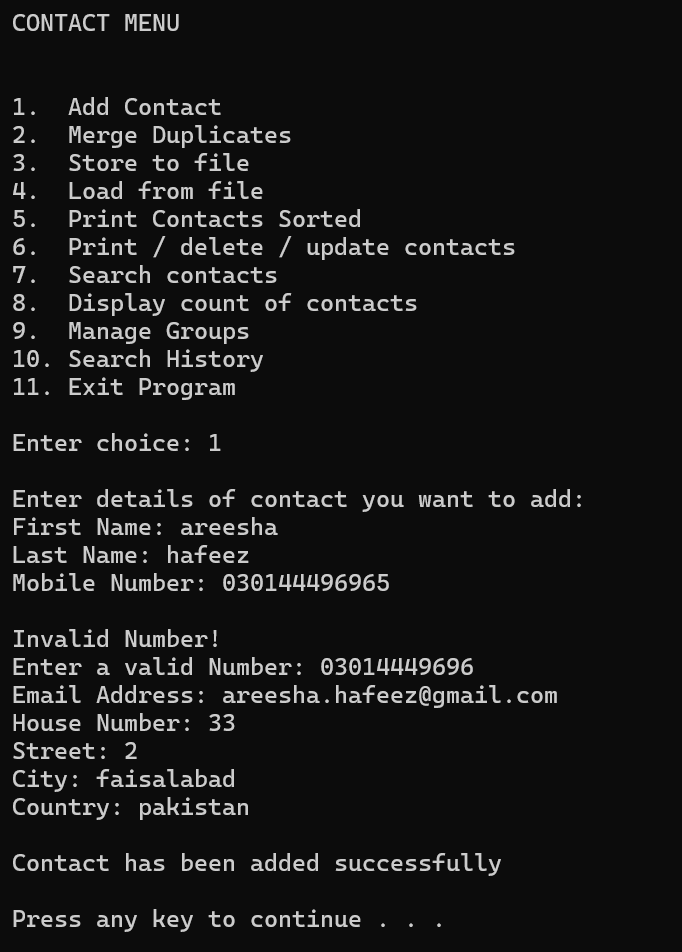
cout << endl;

system("pause");

return 0;

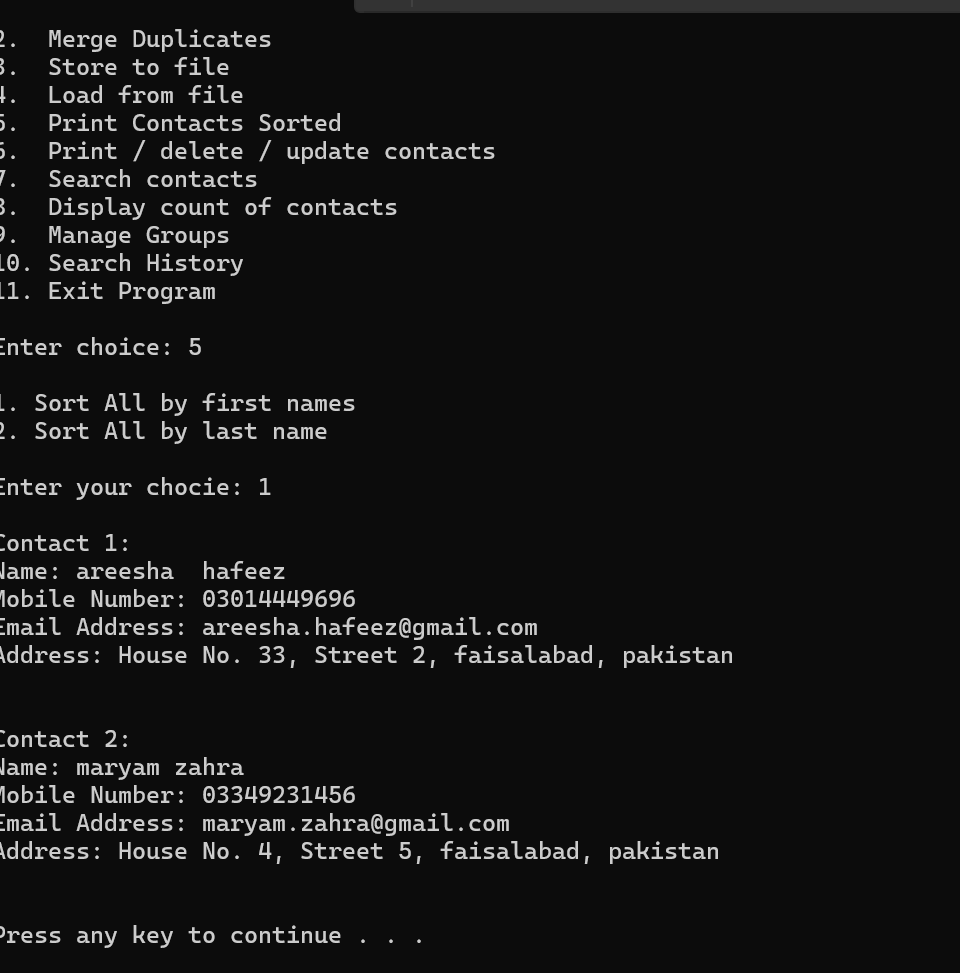
}

**OUTPUTS:**



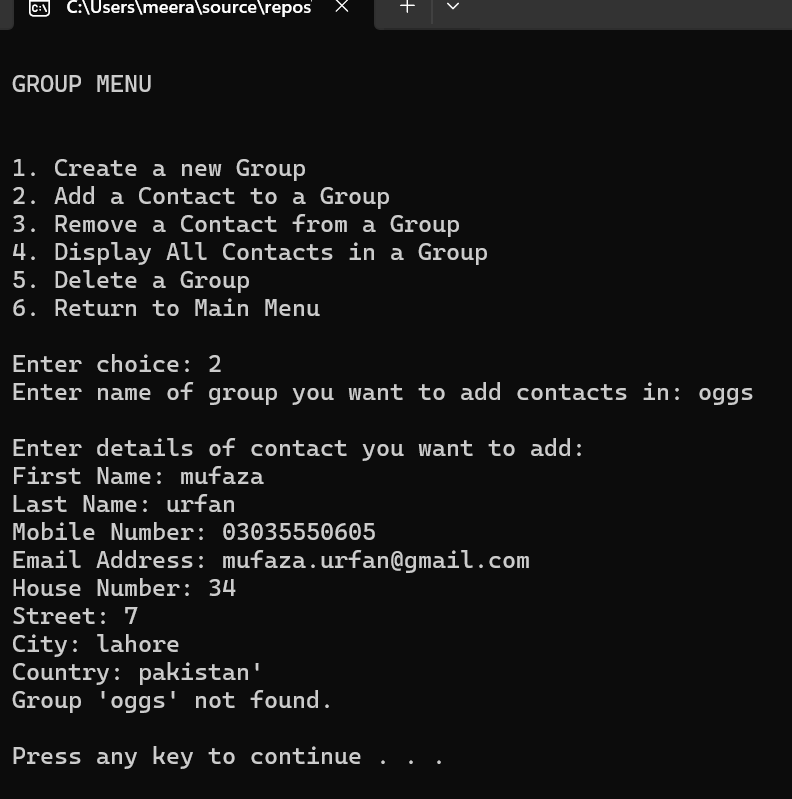
A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated



A screenshot of a computer program

Description automatically generated