***OOP Final project***

***Naveed Hassan , Mustafa Ahmad  
23F\_0599,23F\_0541***

***Main.cpp***#include <iostream>

#include "ContactsBook.h"

using namespace std;

int main()

{

int choice, choice\_2, count = 0, num\_contact, ind = 0, user\_contact\_choice, group\_id, id\_in\_group, del\_grp, search\_grp;

Contact\* found, new\_contact;

Address new\_contact\_add;

string new\_contact\_info, new\_contact\_address, file\_name, first, last, phone, house, street, city, country, group\_name;

cout << "Enter Number of Contacts you want: ";

cin >> num\_contact;

ContactsBook Contact\_list(num\_contact);

while (true)

{

system("cls");

cout <<"0.To print total number of contacts\n" "1. Add New Contact\n2. Merge Duplicates\n"

<< "3. Store To file (input file name)\n4. Load from file\n"

<< "5. Print Contacts Sorted(by first\_name, last\_name)\n"

<< "6. Print Contacts Unsorted\n7. Search Contacts\n"

<< "8. Display Count of Contacts\n9. Manage Contacts\n10. View Search History\n11. Manage Groups\n12. Exit\nEnter Your Choice: ";

cin >> choice;

cin.ignore();

switch (choice)

{

case 0:

cout << "Total number of Contacts are : " << Contact\_list.total\_contacts() << '\n';

break;

case 1:

cout << "Enter First Name of Contact: ";

getline(cin, new\_contact\_info);

new\_contact.set\_first\_name(new\_contact\_info);

cout << "Enter Last Name of Contact: ";

getline(cin, new\_contact\_info);

new\_contact.set\_last\_name(new\_contact\_info);

cout << "Enter Mobile Number(11 digits): ";

getline(cin, new\_contact\_info);

new\_contact.set\_mobile\_number(new\_contact\_info);

cout << "Enter Email Address of Contact: ";

getline(cin, new\_contact\_info);

new\_contact.set\_email\_address(new\_contact\_info);

cout << "Address:\n";

cout << "Enter House: ";

getline(cin, new\_contact\_address);

new\_contact\_add.set\_house(new\_contact\_address);

cout << "Enter Street: ";

getline(cin, new\_contact\_address);

new\_contact\_add.set\_street(new\_contact\_address);

cout << "Enter City: ";

getline(cin, new\_contact\_address);

new\_contact\_add.set\_city(new\_contact\_address);

cout << "Enter Country: ";

getline(cin, new\_contact\_address);

new\_contact\_add.set\_country(new\_contact\_address);

new\_contact.set\_address(new\_contact\_add);

Contact\_list.add\_contact(new\_contact);

break;

case 2:

Contact\_list.merge\_duplicates();

break;

case 3:

cout << "Enter File Name: ";

getline(cin, file\_name);

Contact\_list.save\_to\_file(file\_name);

break;

case 4:

cout << "Enter File Name: ";

getline(cin, file\_name);

Contact\_list.load\_from\_file(file\_name);

break;

case 5:

int choice\_sort;

cout << "Want to sort By\n";

cout << "1. First Name\t2. Last Name";

while (1)

{

cout << "\nEnter you choice(1/2): ";

cin >> choice\_sort;

if (choice\_sort)

{

Contact\_list.print\_contacts\_sorted("first name");

break;

}

else if (choice\_sort == 2)

{

Contact\_list.print\_contacts\_sorted("last name");

break;

}

else

cout << "Wrong Input, Enter Again: ";

}

break;

case 6:

Contact\_list.print\_contacts();

break;

case 7:

cout << "Enter a string: ";

getline(cin, first);

found = Contact\_list.search\_contact(first, ind);

if (found == nullptr)

cout << "Contact Not Found!\n";

else

{

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

cout << "[ " << found[i].get\_first\_name() << " ]";

cout << "[ " << found[i].get\_last\_name() << " ] ";

cout << "[ " << found[i].get\_mobile\_number() << " ] ";

cout << "[ " << found[i].get\_email\_address() << " ] ";

cout << std::endl;

cout << "[ ";

found[i].get\_address()->print\_address();

cout << " ]\n";

}

}

break;

case 8:

cout << "Number of contacts: " << Contact\_list.total\_contacts() << endl;

break;

case 9:

Contact\_list.print\_name\_of\_contacts();

cout << "Enter ID of contact to show all information: ";

cin >> user\_contact\_choice;

Contact\_list.manage\_contacts(user\_contact\_choice);

break;

case 10:

cout << "1. Print all string searched\n2. Print top 5 searched contacts\n";

cout << "Enter Your Choice: ";

cin >> choice;

if (choice == 1)

Contact\_list.print\_search\_history();

else if (choice == 2)

Contact\_list.view\_top\_search();

else

cout << "Wrong Input!\n";

break;

case 11:

cout << "1. Create a group\n2. Available Groups\n"

<< "3. Add id in Group\n4. Delete ID from group\n5. Print All Contacts in Group\n"

<< "6. Delete a Group\n7. Search In a group\n";

cout << "Enter Your Choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter Name of group: ";

cin.ignore();

getline(cin, group\_name);

Contact\_list.create\_group(group\_name);

break;

case 2:

if (Contact\_list.get\_group\_count() == 0)

cout << "No Group Found!\n";

else {

Contact\_list.print\_groups();

}

break;

case 3:

if (Contact\_list.get\_group\_count() == 0)

cout << "No Group Found!\n";

else

{

Contact\_list.print\_groups();

cout << "In which Group You want to add Contact: ";

cin >> group\_id;

cout << "Available Contacts:\n";

Contact\_list.print\_name\_of\_contacts();

cout << "Which ID you want to add in Group: ";

cin >> id\_in\_group;

if (Contact\_list.total\_contacts() < id\_in\_group)

cout << "ID not Found!\n";

else

Contact\_list.add\_group(group\_id - 1, id\_in\_group - 1);

}

break;

case 4:

if (Contact\_list.get\_group\_count() == 0)

cout << "No Group Found!\n";

else {

Contact\_list.print\_groups();

cout << "In which Group You want to Delete Contact: ";

cin >> group\_id;

Contact\_list.print\_cont\_grps(group\_id);

cout << "Enter ID of Contact You want to delete: ";

cin >> id\_in\_group;

if (Contact\_list.total\_contacts() < id\_in\_group)

cout << "ID not Found!\n";

else

Contact\_list.remove\_contact(group\_id - 1, id\_in\_group - 1);

}

break;

case 5:

if (Contact\_list.get\_group\_count() == 0)

cout << "No Group Found!\n";

else

{

Contact\_list.print\_groups();

cout << "Enter ID of group whose contacts you want to see: ";

cin >> group\_id;

Contact\_list.print\_cont\_grps(group\_id);

}

break;

case 6:

if (Contact\_list.get\_group\_count() == 0)

cout << "No Group Found!\n";

else

{

Contact\_list.print\_groups();

cout << "Which Group You want to Delete: ";

cin >> del\_grp;

Contact\_list.remove\_group(del\_grp);

}

break;

case 7:

Contact\_list.print\_name\_of\_contacts();

cout << "Enter ID you want to search in all groups: ";

cin >> search\_grp;

Contact\_list.search\_group(search\_grp);

break;

default:

cout << "Wrong Input!\n";

break;

}

break;

case 12:

exit(0);

break;

default:

cout << "Wrong Input, Enter Again!\n";

}

system("pause");

}

}

***Address.h***

#pragma once

#include <iostream>

#include <string>

class Address {

private:

std::string house;

std::string street;

std::string city;

std::string country;

public:

//equals function

bool equals(const Address& address);

//print function

void print\_address();

//copy function

Address copy\_address();

//setters and getters

std::string get\_house() const;

void set\_house(std::string house);

std::string get\_street() const;

void set\_street(std::string street);

std::string get\_city() const;

void set\_city(std::string city);

std::string get\_country() const;

void set\_country(std::string country);

//constructors

Address(std::string house, std::string street, std::string city, std::string country);

Address();

};

***Address.cpp***

#include "Address.h"

//constructors

Address::Address() :house(""), street(""), city(""), country(""){}//default constructor

Address::Address(std::string house, std::string street, std::string city, std::string country)//parameterized constructor

{

this->house = house;

this->street = street;

this->city = city;

this->country = country;

}

//setter And getters

std::string Address::get\_house() const

{

return house;

}

void Address::set\_house(std::string house)

{

this->house = house;

}

std::string Address::get\_street() const

{

return street;

}

void Address::set\_street(std::string street)

{

this->street = street;

}

std::string Address::get\_city() const

{

return city;

}

void Address::set\_city(std::string city)

{

this->city = city;

}

std::string Address::get\_country() const

{

return country;

}

void Address::set\_country(std::string country)

{

this->country = country;

}

//equals function

bool Address::equals(const Address& address)

{

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

{

if (this->street == address.street)

{

if (this->city == address.city)

{

if (this->country == address.country)

{

return true;

}

}

}

}

return false;

}

//print function

void Address::print\_address()

{

std::cout << "Address: " << house << ", " << street << ", " << city << ", " << country;

}

//copy of this

Address Address::copy\_address()

{

Address copy\_this\_address(this->house, this->street, this->city, this->country);

return copy\_this\_address;

}

***Contact.h***

#pragma once

#include"contact\_history.h"

#include"history.h"

#ifndef BASIC\_LIB

#define BASIC\_LIB

#include <iostream>

#include <string>

#endif // !BASIC\_LIB

#include "Address.h"

class Contact {

private:

std::string first\_name;

std::string last\_name;

std::string mobile\_number;

std::string email\_address;

Address\* address;

int search\_times;

public:

bool equals(Contact contact);

Contact\* copy\_contact();

void set\_first\_name(std::string first\_name);

std::string get\_first\_name() const;

void set\_last\_name(std::string last\_name);

std::string get\_last\_name() const;

void set\_mobile\_number(std::string);

std::string get\_mobile\_number() const;

void set\_email\_address(std::string email\_address);

std::string get\_email\_address() const;

void set\_address(Address& address);

Address\* get\_address() const;

int get\_search\_times() const;

void inc\_search\_count();

Contact();

Contact(std::string first\_name, std::string last\_name, std::string mobile\_number, std::string email\_address, Address\* address);

};

***Contact.cpp***

#include "Contact.h"

#include "Address.h"

#include <string>

/\*

\* Implement a constructor that initializes all the Contact parts, including address. Call the setter to set values

\* as they have the logic to check correct input

\* Declaration is given in Contact.h

\*/

Contact::Contact() :first\_name(""), last\_name(""), mobile\_number(""), email\_address("")

{

address = nullptr;

search\_times = 0;

}

int Contact::get\_search\_times() const

{

return search\_times;

}

Contact::Contact(std::string first\_name, std::string last\_name, std::string mobile\_number, std::string email\_address, Address\* address)

{

set\_first\_name(first\_name);

set\_last\_name(last\_name);

set\_mobile\_number(mobile\_number);

set\_email\_address(email\_address);

set\_address(\*address);

}

void Contact::inc\_search\_count()

{

search\_times++;

}

/\*

\* Implement getter/setters for all the member variables of Contact. Also declare them in the Contact.h header file

\* In each setter, do not set value if string is empty ("" 0 size).

\* Check in phone number setter, Phone number must be 11 digits

\*/

void Contact::set\_first\_name(std::string first\_name)

{

int length = first\_name.length();

if (length != 0)

{

this->first\_name = first\_name;

}

}

std::string Contact::get\_first\_name() const

{

return first\_name;

}

void Contact::set\_last\_name(std::string last\_name)

{

int length = last\_name.length();

if (length != 0)

{

this->last\_name = last\_name;

}

}

std::string Contact::get\_last\_name() const

{

return last\_name;

}

void Contact::set\_email\_address(std::string email\_address)

{

int length = last\_name.length();

if (length != 0)

{

this->email\_address = email\_address;

}

}

std::string Contact::get\_email\_address() const

{

return email\_address;

}

void Contact::set\_mobile\_number(std::string mobile\_number)

{

int length = mobile\_number.length();

if (length == 11)

{

this->mobile\_number = mobile\_number;

}

}

std::string Contact::get\_mobile\_number() const

{

return mobile\_number;

}

void Contact::set\_address(Address& address)

{

this->address = new Address;

int length = address.get\_house().length();

if (length != 0)

{

this->address->set\_house(address.get\_house());

}

length = address.get\_street().length();

if (length != 0)

{

this->address->set\_street(address.get\_street());

}

length = address.get\_city().length();

if (length != 0)

{

this->address->set\_city(address.get\_city());

}

length = address.get\_country().length();

if (length != 0)

{

this->address->set\_country(address.get\_country());

}

}

Address\* Contact::get\_address() const

{

return address;

}

/\*

\* Implement the equals function(as declared in header file) that takes a

\* Contact object and checks if it is equal to current object refered by (this pointer).

\* Compare all the attributes.

\* For comparing address of contact you can call the equals function of the Address object;

\* For example: this.address.equals

\*/

bool Contact::equals(Contact contact)

{

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

{

if (this->last\_name == contact.last\_name)

{

if (this->mobile\_number == contact.mobile\_number)

{

if (this->email\_address == contact.email\_address)

{

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

{

return true;

}

}

}

}

}

return false;

}

Contact\* Contact::copy\_contact()

{

Contact\* new\_ptr;

new\_ptr = new Contact;

new\_ptr->set\_first\_name(first\_name);

new\_ptr->set\_last\_name(last\_name);

new\_ptr->set\_mobile\_number(mobile\_number);

new\_ptr->set\_email\_address(email\_address);

Address\* new\_address;

new\_address = new Address(\*address);

new\_ptr->set\_address(\*new\_address);

return new\_ptr;

}

***ContactsBook.h***

#pragma once

#include "Address.h"

#include "Contact.h"

#include "contact\_history.h"

#include"groups.h"

#ifndef BASIC\_LIB

#define BASIC\_LIB

#include <iostream>

#include <string>

#endif // !BASIC\_LIB

class ContactsBook

{

private:

groups\* groups\_obj;

Contact\* contacts\_list; // array of contacts

int size\_of\_contacts; // storage limit

int contacts\_count; // total contacts currently stored, next contact will be

contact\_history\* history;// stored at this count plus 1 index

int group\_count;

public:

void add\_contact(const Contact& contact);

int total\_contacts();

void search\_group(int);

int get\_group\_count();

void print\_groups();

void create\_group(string);

void add\_group(int, int);

void print\_cont\_grps(int);

Contact\* search\_contact(std::string user\_input, int&);

void print\_search\_history();

ContactsBook(int size\_of\_list);

void print\_by\_id(int);

void print\_name\_of\_contacts();

void manage\_contacts(int);

void edit\_contact(int, Contact\*);

void print\_contacts();

void remove\_contact(int, int);

void remove\_group(int);

void print\_contacts\_sorted(std::string choice); // Only two choices first\_name or last\_name

void merge\_duplicates(); // Implement this function that finds and merges all the duplicates

// Duplicate means exact equal, this means if

void view\_top\_search();

Contact\* copy\_contact();

/\*

\* This function loads contacts from the given file (see details in ContactsBook.cpp)

\*/

void load\_from\_file(std::string file\_name);

void save\_to\_file(std::string file\_name);

private:

bool full();

void resize\_list();

void sort\_contacts\_list(Contact\* contacts\_list, std::string choice);

};

***ContactsBook.cpp***

; #include "ContactsBook.h"

#include<string>

#include<fstream>

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

{

/\*

- Check if the list is full, if it is full call the resize function

- If list is not full add the contact to the end of the array

- (end means the place where last contact was inserted)

- At start it will be 0th index as no contact has been inserted before so

- count is zero (contacts\_count member)

- Increment the count

- As inserted successfully, return 1

\*/

if (full())

{

resize\_list();

}

contacts\_list[contacts\_count] = contact;

contacts\_count++;

}

int ContactsBook::total\_contacts()

{

return contacts\_count;

}

bool ContactsBook::full()

{

/\*

\* Return true if list is full, false otherwise

\*/

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

return true;

return false;

}

void ContactsBook::resize\_list()

{

/\*

\* Here you will resize the contact list, see example code given in lab manual to see how to resize arrays

\* You will allocate a temporary new array of double the current size and copy the contacts from

\* previous array to this array one by one, get the copy of each contact using copy\_contact

\* function of Contact class

\* Delete the previous array

\* Assign the new temporary array to the contacts\_list pointer

\* Updated the this->size\_of\_contacts with new size

\*/

int temp\_size = size\_of\_contacts \* 2;

Contact\* new\_contacts\_list = new Contact[temp\_size];

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

{

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

}

delete[] contacts\_list;

contacts\_list = new\_contacts\_list;

size\_of\_contacts = temp\_size;

}

void ContactsBook::view\_top\_search()

{

int top\_times, top\_index[5], count\_var = 0;

bool flag;

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

{

flag = false;

if (count\_var == 5)

{

break;

}

for (int j = 0; j < count\_var; j++)

{

if (i == top\_index[j])

{

flag = true;

break;

}

}

if (flag == false)

{

if (contacts\_list[i].get\_search\_times() != 0)

{

top\_times = contacts\_list[i].get\_search\_times();

top\_index[count\_var] = i;

}

else

{

continue;

}

}

else

{

continue;

}

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

{

flag = false;

if (top\_times < contacts\_list[j].get\_search\_times())

{

for (int k = 0; k < count\_var; k++)

{

if (j == top\_index[k])

{

flag = true;

break;

}

}

if (flag == false)

{

top\_times = contacts\_list[j].get\_search\_times();

top\_index[count\_var] = j;

}

}

}

count\_var++;

}

if (count\_var == 0)

{

std::cout << "No searched contact! \n";

}

else

{

std::cout << "Top " << count\_var << " searched contacts are: \n";

for (int i = 0; i < count\_var; i++)

{

std::cout << "[ " << contacts\_list[i].get\_first\_name() << " ]";

std::cout << "[ " << contacts\_list[i].get\_last\_name() << " ] ";

std::cout << "[ " << contacts\_list[i].get\_mobile\_number() << " ] ";

std::cout << "[ " << contacts\_list[i].get\_email\_address() << " ] ";

std::cout << std::endl;

std::cout << "[ ";

contacts\_list[i].get\_address()->print\_address();

std::cout << " ]\n";

std::cout << "Searched Count: " << contacts\_list[i].get\_search\_times() << endl;

}

}

}

void ContactsBook::print\_search\_history()

{

if (history->get\_search\_count() < 0)

{

cout << "No History Found!\n";

}

for (int i = history->get\_search\_count() - 1; i >= 0; i--)

{

history->get\_history(i);

}

}

Contact\* ContactsBook::search\_contact(std::string user\_input, int& ind)

{

/\*

\* In all search functions perform the searching according the given parameter and return a copy of the contact using copy func

\* Remove this return nullptr; before writing your code

\*/

history->input\_history(user\_input);

ind = 0;

Contact\* ptr = new Contact[size\_of\_contacts];

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

{

if (contacts\_list[i].get\_first\_name().find(user\_input) != std::string::npos)

{

ptr[ind] = contacts\_list[i];

contacts\_list[i].inc\_search\_count();

ind++;

}

else if (contacts\_list[i].get\_last\_name().find(user\_input) != std::string::npos)

{

ptr[ind] = contacts\_list[i];

contacts\_list[i].inc\_search\_count();

ind++;

}

else if (contacts\_list[i].get\_mobile\_number().find(user\_input) != std::string::npos)

{

ptr[ind] = contacts\_list[i];

contacts\_list[i].inc\_search\_count();

ind++;

}

else if (contacts\_list[i].get\_email\_address().find(user\_input) != std::string::npos)

{

ptr[ind] = contacts\_list[i];

contacts\_list[i].inc\_search\_count();

ind++;

}

else if (contacts\_list[i].get\_address()->get\_house().find(user\_input) != std::string::npos)

{

ptr[ind] = contacts\_list[i];

contacts\_list[i].inc\_search\_count();

ind++;

}

else if (contacts\_list[i].get\_address()->get\_street().find(user\_input) != std::string::npos)

{

ptr[ind] = contacts\_list[i];

contacts\_list[i].inc\_search\_count();

ind++;

}

else if (contacts\_list[i].get\_address()->get\_city().find(user\_input) != std::string::npos)

{

ptr[ind] = contacts\_list[i];

contacts\_list[i].inc\_search\_count();

ind++;

}

else if (contacts\_list[i].get\_address()->get\_country().find(user\_input) != std::string::npos)

{

ptr[ind] = contacts\_list[i];

contacts\_list[i].inc\_search\_count();

ind++;

}

}

if (ind == 0)

return nullptr;

else

{

return ptr;

}

}

ContactsBook::ContactsBook(int size\_of\_list) :size\_of\_contacts(size\_of\_list), contacts\_count(0)

{

/\*

\* Initialize the contacts\_list array, also initialize the size and count members accordingly

\*/

contacts\_list = new Contact[size\_of\_contacts];

history = new contact\_history(size\_of\_contacts);

groups\_obj = nullptr;

group\_count = 0;

}

void ContactsBook::print\_name\_of\_contacts()

{

std::string user\_contact\_choice;

bool contact\_found{ true };

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

{

std::cout << i + 1 << ". [ " << contacts\_list[i].get\_first\_name() << " ]";

std::cout << "[ " << contacts\_list[i].get\_last\_name() << " ] ";

std::cout << std::endl;

}

}

void ContactsBook::manage\_contacts(int user\_choice)

{

user\_choice--;

print\_by\_id(user\_choice);

int choice;

std::string new\_contact\_info, new\_contact\_address;

Contact edited\_contact;

Address edited\_contact\_add;

std::cout << "1. to edit contact" << std::endl;

std::cout << "2. to delete contact" << std::endl;

std::cout << "3. to exit" << std::endl;

std::cout << "Your choice: ";

std::cin >> choice;

switch (choice)

{

case 1:

{

std::cout << "Enter First Name of Contact: ";

std::cin.ignore();

getline(std::cin, new\_contact\_info);

edited\_contact.set\_first\_name(new\_contact\_info);

std::cout << "Enter Last Name of Contact: ";

getline(std::cin, new\_contact\_info);

edited\_contact.set\_last\_name(new\_contact\_info);

std::cout << "Enter Mobile Number(11 digits): ";

getline(std::cin, new\_contact\_info);

edited\_contact.set\_mobile\_number(new\_contact\_info);

std::cout << "Enter Email Address of Contact: ";

getline(std::cin, new\_contact\_info);

edited\_contact.set\_email\_address(new\_contact\_info);

std::cout << "Address: ";

std::cout << "Enter House: ";

getline(std::cin, new\_contact\_address);

edited\_contact\_add.set\_house(new\_contact\_address);

std::cout << "Enter Street: ";

getline(std::cin, new\_contact\_address);

edited\_contact\_add.set\_street(new\_contact\_address);

std::cout << "Enter City: ";

getline(std::cin, new\_contact\_address);

edited\_contact\_add.set\_city(new\_contact\_address);

std::cout << "Enter Country: ";

getline(std::cin, new\_contact\_address);

edited\_contact\_add.set\_country(new\_contact\_address);

edited\_contact.set\_address(edited\_contact\_add);

contacts\_list[user\_choice] = edited\_contact;

break;

}

case 2:

{

Address obj;

if (!(user\_choice == contacts\_count - 1)) {

for (int k = user\_choice + 1; k < contacts\_count; k++)

{

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

}

}

contacts\_list[contacts\_count - 1].set\_first\_name("");

contacts\_list[contacts\_count - 1].set\_last\_name("");

contacts\_list[contacts\_count - 1].set\_email\_address("");

contacts\_list[contacts\_count - 1].set\_mobile\_number("");

contacts\_list[contacts\_count - 1].set\_address(obj);

contacts\_count--;

break;

}

case 3:

{

break;

}

default: {

std::cout << "Wrong input!";

break;

}

}

}

int ContactsBook::get\_group\_count()

{

return group\_count;

}

void ContactsBook::search\_group(int id)

{

id--;

bool flag = false;

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

{

for (int j = 0; j < groups\_obj[i].get\_count(); j++)

{

if (id == groups\_obj[j].get\_contact\_id(j))

{

cout << "Found in Group " << i + 1;

print\_by\_id(groups\_obj[j].get\_contact\_id(j));

flag = true;

}

}

}

if (!flag)

cout << "No such Contact Found in group!\n";

}

void ContactsBook::remove\_group(int group\_number)

{

for (int i = group\_number - 1; i < group\_count - 1; i++)

{

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

}

group\_count--;

std::cout << "Group removed! \n";

}

void ContactsBook::print\_groups()

{

cout << "Available Groups:\n";

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

{

cout << i + 1 << ". " << groups\_obj[i].get\_group\_name() << endl;

}

cout << "\n";

}

void ContactsBook::remove\_contact(int group\_id, int contact\_ind)

{

groups\_obj[group\_id].remove\_id(contact\_ind);

}

void ContactsBook::create\_group(string name)

{

if (group\_count == 0)

{

groups\_obj = new groups[group\_count + 1];

groups\_obj->set\_group\_name(name);

groups\_obj->set\_id\_size(size\_of\_contacts);

group\_count++;

}

else

{

groups\* temp\_obj = new groups[group\_count];

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

{

temp\_obj[i] = groups\_obj[i];

}

delete[] groups\_obj;

groups\_obj = nullptr;

groups\_obj = new groups[group\_count + 1];

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

{

groups\_obj[i] = temp\_obj[i];

}

delete[] temp\_obj;

temp\_obj = nullptr;

groups\_obj[group\_count].set\_group\_name(name);

groups\_obj[group\_count].set\_id\_size(size\_of\_contacts);

group\_count++;

}

}

void ContactsBook::add\_group(int group\_id, int contact\_ind)

{

groups\_obj[group\_id].set\_contact\_id(contact\_ind);

}

void ContactsBook::print\_cont\_grps(int group\_id)

{

for (int i = 0; i < groups\_obj[group\_id - 1].get\_count(); i++)

{

int id\_index = groups\_obj[group\_id - 1].get\_contact\_id(i);

std::cout << "Contact " << i + 1 << ":\n";

print\_by\_id(id\_index);

}

}

void ContactsBook::print\_contacts()

{

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

{

std::cout << "[ " << contacts\_list[i].get\_first\_name() << " ]";

std::cout << "[ " << contacts\_list[i].get\_last\_name() << " ] ";

std::cout << "[ " << contacts\_list[i].get\_mobile\_number() << " ] ";

std::cout << "[ " << contacts\_list[i].get\_email\_address() << " ] ";

std::cout << std::endl;

std::cout << "[ ";

contacts\_list[i].get\_address()->print\_address();

std::cout << " ]\n";

}

}

void ContactsBook::print\_by\_id(int user\_choice)

{

std::cout << "[ " << contacts\_list[user\_choice].get\_first\_name() << " ]";

std::cout << "[ " << contacts\_list[user\_choice].get\_last\_name() << " ] ";

std::cout << "[ " << contacts\_list[user\_choice].get\_mobile\_number() << " ] ";

std::cout << "[ " << contacts\_list[user\_choice].get\_email\_address() << " ] ";

std::cout << std::endl;

std::cout << "[ ";

contacts\_list[user\_choice].get\_address()->print\_address();

std::cout << " ]\n";

}

void ContactsBook::print\_contacts\_sorted(std::string choice)

{

/\*

\* Create a copy of this->contacts\_list array here (do it by creating a new function that returns copy)

\* Call the sort function sort\_contacts\_list to sort the newly created copy

\* This copy is created to avoid changing the original insertion order of contact list

\* Then print the contacts in following format:

\* [First Name] [Last Name] [Mobile] [Email]

\* Call the address print function to print address on next line

\*/

Contact\* new\_contact\_list = copy\_contact();

sort\_contacts\_list(new\_contact\_list, choice);

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

{

if (new\_contact\_list[i].get\_address() == nullptr)

{

continue;

}

std::cout << "[ " << new\_contact\_list[i].get\_first\_name() << " ]";

std::cout << "[ " << new\_contact\_list[i].get\_last\_name() << " ] ";

std::cout << "[ " << new\_contact\_list[i].get\_mobile\_number() << " ] ";

std::cout << "[ " << new\_contact\_list[i].get\_email\_address() << " ] ";

std::cout << std::endl;

std::cout << "[ ";

new\_contact\_list[i].get\_address()->print\_address();

std::cout << " ]\n";

}

}

Contact\* ContactsBook::copy\_contact()

{

Contact\* new\_contact\_list = new Contact[size\_of\_contacts];

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

{

new\_contact\_list[i] = contacts\_list[i];

}

return new\_contact\_list;

}

void ContactsBook::sort\_contacts\_list(Contact\* contacts\_list, std::string choice)

{

/\*

You should not duplicate the code to sort based on choices

Follow the strategy provided in manual/tutorial to avoid duplicating the code (Section B & E only)

Sort by the fist letter of first name or last name as given in choice

\*/

if (choice == "First Name" || choice == "first name")

{

for (int i = 0; i < size\_of\_contacts - 1; i++)

{

for (int j = i + 1; j < size\_of\_contacts; j++)

{

if (contacts\_list[i].get\_first\_name() > contacts\_list[j].get\_first\_name())

{

Contact temp = contacts\_list[i];

contacts\_list[i] = contacts\_list[j];

contacts\_list[j] = temp;

}

}

}

}

if (choice == "Last Name" || choice == "last name")

{

for (int i = 0; i < size\_of\_contacts - 1; i++)

{

for (int j = i + 1; j < size\_of\_contacts; j++)

{

if (contacts\_list[i].get\_last\_name() > contacts\_list[j].get\_last\_name())

{

Contact temp = contacts\_list[i];

contacts\_list[i] = contacts\_list[j];

contacts\_list[j] = temp;

}

}

}

}

}

void ContactsBook::merge\_duplicates()

{

// Implement this function that finds and merges all the duplicates

// Duplicate means exact equal

// If there are three copies of a Contact, then only one should remain

// For example: If total contact are 10, and 2 contacts are same, after removing duplicates

// 9 contacts will be left; and the this->contacts\_count of list will 9

// At the end display contacts mernged, and count of merged contact

// Iterate through the contacts list to merge duplicates

Address obj;

int merged\_contacts = 0;

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]))

{

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

{

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

contacts\_list[k].set\_first\_name("");

contacts\_list[k].set\_last\_name("");

contacts\_list[k].set\_email\_address("");

contacts\_list[k].set\_mobile\_number("");

contacts\_list[k].set\_address(obj);

}

contacts\_count--;

++merged\_contacts;

}

}

}

cout << "Merged contacts are : " << merged\_contacts << '\n';

}

void ContactsBook::load\_from\_file(std::string file\_name)

{

/\*

Read contacts back from file in the same format

Read the contacts\_count, and run a loop for this contacts\_count and read the

\* contacts in the same format as you stored

\* Add them to contact book (this.add\_contact function can be used)

\* Finally, close the file

\*/

std::ifstream input;

input.open(file\_name);

if (!input)

std::cout << "Error in opening file\n";

else

{

std::string str\_1, str\_2, str\_3, str\_4, str\_5, str\_6, str\_7;

input >> this->contacts\_count;

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

{

input >> str\_1 >> str\_2 >> str\_3 >> str\_4 >> str\_5 >> str\_6 >> str\_7;

contacts\_list[i].set\_first\_name(str\_1);

contacts\_list[i].set\_last\_name(str\_3);

contacts\_list[i].set\_mobile\_number(str\_5);

contacts\_list[i].set\_email\_address(str\_7);

input >> str\_1 >> str\_2 >> str\_3 >> str\_4 >> str\_5 >> str\_6 >> str\_7;

contacts\_list[i].get\_address()->set\_house(str\_1);

contacts\_list[i].get\_address()->set\_street(str\_3);

contacts\_list[i].get\_address()->set\_city(str\_5);

contacts\_list[i].get\_address()->set\_country(str\_7);

}

}

input.close();

}

void ContactsBook::save\_to\_file(std::string file\_name)

{

/\*

\* In this function you will store all the contacts to a file

\* On first line store contacts\_count

\* On next lines store contacts in the following format:

\* 2 lines for each contact

\* On oneline write contact attributes(except address) in comma separated format. On second line

\* store address atributes in the same format

\* first\_name,last\_name,mobile\_number,email\_address

\* house,street,city,country

\* Close the file

\*/

std::ofstream output1;

output1.open(file\_name, std::ios::app);

if (!output1)

std::cout << "Error !" << std::endl;

else

{

Address\* obj;

output1 << contacts\_count << std::endl;

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

{

obj = contacts\_list[i].get\_address();

output1 << contacts\_list[i].get\_first\_name() << " , " << contacts\_list[i].get\_last\_name() << " , " << contacts\_list[i].get\_mobile\_number() << " , " << contacts\_list[i].get\_email\_address() << std::endl;

output1 << obj->get\_house() << " , " << obj->get\_street() << " , " << obj->get\_city() << " , " << obj->get\_country() << std::endl;

}

}

output1.close();

}

Contact\_history.h

#pragma once

#include"History.h"

using namespace std;

class contact\_history

{

int search\_size;

History\* search\_history;

int search\_count;

public:

contact\_history(int);

void input\_history(string);

void resize\_Obj();

int get\_search\_count();

void get\_history(int);

};

Contacthistory.cpp

#include<iostream>

#include"contact\_history.h"

contact\_history::contact\_history(int search\_size)

{

this->search\_size = search\_size;

search\_history = new History[search\_size];

search\_count = 0;

}

void contact\_history::input\_history(string search)

{

if (search\_count >= search\_size)

{

resize\_Obj();

}

search\_history[search\_count].set\_search\_words(search);

search\_history[search\_count].set\_time\_date();

search\_count++;

}

void contact\_history::get\_history(int search\_index)

{

tm search\_time = search\_history[search\_index].get\_time\_date();

cout << "Search " << search\_index + 1 << endl;

cout << "Time: " << search\_time.tm\_hour << ":" << search\_time.tm\_min << ":" << search\_time.tm\_sec << ", Date: " << search\_time.tm\_mday << "/" << search\_time.tm\_mon + 1 << "/" << search\_time.tm\_year + 1900 << endl;

cout << "string: " << search\_history[search\_index].get\_search\_words() << endl;

}

void contact\_history::resize\_Obj()

{

History\* temp\_obj = new History[search\_size];

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

{

temp\_obj[i] = search\_history[i];

}

delete[]search\_history;

search\_history = nullptr;

search\_size \*= 2;

search\_history = new History[search\_size];

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

{

search\_history[i] = temp\_obj[i];

}

delete[]temp\_obj;

temp\_obj = nullptr;

}

int contact\_history::get\_search\_count()

{

return search\_count;

}

History.h

#pragma once

#include<iostream>

using namespace std;

class History

{

tm get\_time;

string search\_words;

time\_t time\_date;

public:

void set\_search\_words(string);

void set\_time\_date();

string get\_search\_words() const;

tm get\_time\_date() const;

};

History.cpp

#include"History.h"

void History::set\_search\_words(std::string search\_words)

{

this->search\_words = search\_words;

}

void History::set\_time\_date()

{

time\_date = time(0);

localtime\_s(&get\_time, &time\_date);

}

std::string History::get\_search\_words() const

{

return search\_words;

}

tm History::get\_time\_date() const

{

return get\_time;

}

Groups.h

#pragma once

#include<iostream>

using namespace std;

class groups

{

string group\_name;

int\* contact\_id;

int id\_size;

int id\_count;

public:

groups();

void set\_id\_size(int);

void set\_contact\_id(int);

void set\_group\_name(string);

string get\_group\_name() const;

int get\_count() const;

void remove\_id(int);

void resize\_list();

int get\_contact\_id(int);

};

Groups.cpp

#include"groups.h"

groups::groups()

{

id\_size = 0;

id\_count = 0;

contact\_id = nullptr;

}

void groups::set\_group\_name(string name)

{

this->group\_name = name;

}

int groups::get\_contact\_id(int index)

{

return contact\_id[index];

}

void groups::set\_contact\_id(int id)

{

if (id\_count >= id\_size)

resize\_list();

contact\_id[id\_count] = id;

id\_count++;

}

string groups::get\_group\_name() const

{

return group\_name;

}

void groups::set\_id\_size(int size)

{

this->id\_size = size;

contact\_id = new int[id\_size];

}

int groups::get\_count() const

{

return id\_count;

}

void groups::resize\_list()

{

int\* temp\_id = new int[id\_size];

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

{

temp\_id[i] = contact\_id[i];

}

delete[] contact\_id;

id\_size = id\_size \* 2;

contact\_id = new int[id\_size];

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

{

contact\_id[i] = temp\_id[i];

}

delete[] temp\_id;

temp\_id = nullptr;

}

void groups::remove\_id(int id)

{

if (!(id\_count - 1 == id))

{

for (int i = id; i < id\_count; i++)

{

contact\_id[i] = contact\_id[i + 1];

}

}

contact\_id[id\_count - 1] = 0;

id\_count--;

}