Database of Hospital

Documentation

Name: Khongmeng Kormoua

Neptun Code: I3MLPQ

Basic of Programming 2

Final project

**INTRODUCTION**

A database, mainly store a huge amount of information that to be used later when needed. Database is used in every organizations, to store any kind of information. Since in this era, Data has been a very important aspect for development like AI, marketing, research, etc. This project will demonstrate how a database of a hospital could be and this documentation contains an explaination, description in detail and this documentation as a data could be referred and used later by other programmer as an idea and instruction for their further development.

**CHAPTER 1: PROGRAM OVERVIEW**

This program mainly can store a list of Doctor, Nurse and Patient. All those lists can be access and modify anytime. This program can show a list of Doctor, Nurse and Patient. A user can add a new Person (Doctor, Nurse, Patient) into a list and also can remove a Person (Doctor, Nurse, Patient) from a list anytime too. Program also allow user to search and get all information of a particular Person (Doctor, Nurse, Patient). The lists of Person (Doctor, Nurse, Patient) are stored permanently, even a program is shutdown, all information in the lists will remain the same as before the program shutdown.

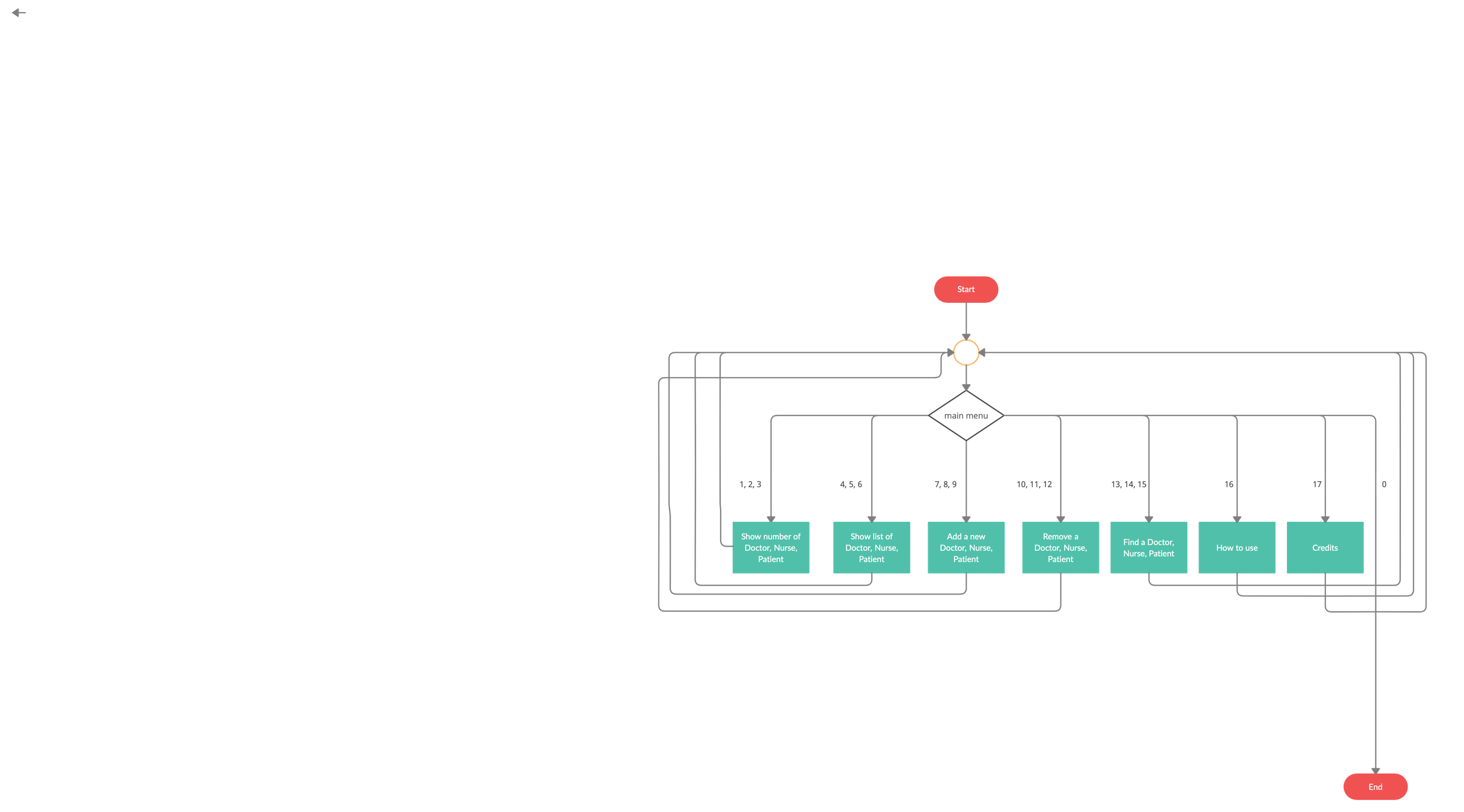
**CHAPTER 2: SOLUTION DESIGN**

Database of hospital is implemented base on Object Oriented Programming.

In this program, I added several techniques for instance: exception handling to check for some possible error, file management for permanent storing data, dynamic memory management, inheritance and polymorphism for uniform handling and operator overloading for a simpler implementation inside other function. I create a list using container called multimap.

This program consists of several files working together, doing so will separate a code into different source file and each of them will give a functionality which related to them only, this make a code much more understandable:

* 1. **main.cpp:** which works as switch function, switch from this functionality to others for instance: call function to load database, call function to add a new doctor into a database, save this database in permanent place which in this case is a file.
  2. **menu.h:** a header file which contains all main function declarations for instance: add doctor into a database, remove doctor from a database, show a list of doctor, etc.
  3. **menu.cpp:** a source file which contains an implementation of all functions at menu.h
  4. **person.h:** a header file for class Person which will be a base class for other class like class Doctor, class Nurse, class Patient. This header file will contain all declaration of this class.
  5. **person.cpp:** a source file which contains all implementation of function, method that is declared in person.h
  6. **doctor.h:** a header file for class Doctor which contains all declaration of function, method and operator overloading of this class.
  7. **doctor.cpp:** a source file which has all implementation of all functions, methods, operator overloading from doctor.h
  8. **nurse.h:** a header file for class Nurse which contains all declaration of function, method and operator overloading of this class.
  9. **nurse.cpp:** a source file which has all implementation of all functions, methods and operator overloading from nurse.h
  10. **patient.h:** a header file for class Patient which has all declaration of function, method and operator overloading of this class.
  11. **patient.cpp:** a source file which has all implementation of all functions, methods and operator overloading from patient.h
  12. **ListOfDoctor.txt:** is used to save and load a list of doctor permanently
  13. **ListOfNurse.txt:** is used to save and load a list of nurse permanently
  14. **ListOfPatient.txt:** is used to save and load a list of patient permanently
  15. **HowToUse.txt:** contain some instruction how to used this program
  16. **Credit.txt:** contain some credits of implementor of this program

****

This program has a basic working principle as flow chart below:

Figure 1

The above diagram show a basic navigation of the program, which can be navigated by select a number that is corresponding to it.

For example: to show a number of Doctor, select 1.

to exit a program, select 0.

Etc.

In each choice or each block, there is their own implementation. But for some block can be applied to Doctor, Nurse, Patient with same implementation such as: Show a number, show a list, remove, find a person. A detail of each block is explained as below:

* **Show number of Doctor, Nurse, Patient**



Description:

Print number of people in a list to a screen.

Parameter:

* num: integer which is a number of people in a list.

Return:

Do not return anything.

* **Show list of Doctor, Nurse, Patient**



Description:

This function iterates through the list by an index of each person and print an information of each person respectively to a screen.

Parameter:

* DB: a container multimap which store list of Person.

Return:

Do not return anything.

* **Add a new Doctor, Nurse, Patient**







Description:

Create a new Doctor, Nurse, Patient and get an input information from user to a person that program just created, then compare if this new Person has already existed in a list, if not then add this new Person to a given list.

Parameter:

* DB: a container multimap that store a list of person with a key, a key is a string which will indicate a name of this person.
* num: a variable to store how many people there are in a list.

Return:

Don’t return anything.

* **Remove a Doctor, Nurse, Patient**



Description:

Check if a given list is empty or not, if not then ask a name that user wish to delete from user then show all information of this person for user, ask user again to confirm and if yes then program should remove this person from a list and also decrease a number of people in a list.

Parameter:

* DB: a container multimap that store a list of person with a key, a key is a string which will indicate a name of this person.
* num: a variable to store how many people there are in a list.

Return:

Do not return anything.

* **Find a Doctor, Nurse, patient**



Description:

Ask user for a name of a person that user wish to find, after get a name this function will iterate through the list and check if this name match with someone in a list. If a function could not find this person then tell user about not found message. If that person is found then print all information of this person to user.

Parameter:

* DB: a container multimap that store a list of person with a key, a key is a string which will indicate a name of this person.

Return:

Do not return anything.

* **How to use**



Description:

Open a corresponding file and print all instruction in this file to a screen. If a file cannot be opened or cannot be closed, a function throws an exception.

Parameter:

No parameter.

Return:

Don’t return anything.

* **Credits**



Description:

Open a corresponding file and print all information in this file to a screen. If a file cannot be opened or cannot be closed, a function throws an exception.

Parameter:

No parameter.

Return:

Don’t return anything.

But at first, a program should have a container that will store a lists of Doctor, Nurse, Patient which using a container call multimap. Each person will be store in a list with respect to their index. The program should have a variables that store how many people are there in a lists. The program will call function **loadDoctorList()**, **loadNurseList()**, **loadPatientList()**.

At the end before a program is terminated, a program will call a function **saveDoctorList()**, **saveNurseList()**, **savePatientList()** to save a list to a file then terminate a program.

* **Load List Function**

****

****

****

Description:

Open a file which store a list. First read a number of people in a list, set it to a variable in our program and then, read all information of each person from a list to our container.

Parameter:

* DB: a container multimap which can store a list of person with a key, a key is a string which will indicate a name of this person.
* num: a variable to store how many people there are in a list.

Return:

Do not return anything.

* **Print menu function**

A picture containing icon

Description automatically generated

Description:

Print all a menu to a user then get a choice from a user and return user’s choice.

Parameter:

* No parameter.

Return:

An integer which is a choice from user.

* **Save list to a file**

****

****



Description:

Open a file to be stored a list, first print a number of people in a list and iterate through the whole list, at each person that a function iterates to, print all information of a person into a file. If a file cannot be closed or opened, an exception will be throw.

Parameter:

* DB: a container multimap which can store a list of person with a key, a key is a string which will indicate a name of this person.
* num: a variable to store how many people there are in a list.

Return:

Don’t return anything.

**CHAPTER 3: SOLUTION**

To store a person into a list, I use a container called multimap, which can store Person with a key, the beneficial is a person can be searched, removed quickly by its key. In this list, a key will be a name of that person and multimap can grow dynamically which is what we need to create a list.

|  |  |
| --- | --- |
| KEY | VALUE |
| string 1 | Person\* 1 |
| string 2 | Person\* 2 |
| … | … |

A key of a map is type std::string and a value of a map is class Person\*

To have a person with necessary information, I will create them as a class with class member attributes and class member methods. I will create class Person to have a basic information that every person will have, then I will create class Doctor, Nurse and Patient which inherited from class Person

**Graphical user interface

Description automatically generated with medium confidence**

Base class

Derived class

Figure 2

* class Person
* this class has 8 private attributes:

1. **std::string name;** store a name of this person.
2. **int age**;store an age of this person.
3. **std::string ID**; store an ID of this person.
4. **std::string address**; store an address of this person.
5. **std::string gender**; store a gender of this person.
6. **std::string contactNumber**; store a phone number of this person.
7. **std::string dateOfBirth**; store a date of birth of this person.
8. **std::string dateOfMoveIn**; this attribute will be used in 2 different cases. If a person is Doctor or Nurse, this attribute will indicate a date of starting work. If a person is Patient, this attribute will indicate a date of move in to hospital.

* This class has a default constructor to set an instance of this class to initial state. For string, set to empty string. For int, set to 0.
  + This class has 8 methods to set a value to each attribute and this class has 8 methods to get a value of every attribute:
    1. **setName()**
    2. **setAge()**
    3. **setID()**
    4. **setAddress()**
    5. **setGender()**
    6. **setContact()**
    7. **setDateOfBirth()**
    8. **setDateOfMoveIn()**
    9. **getName()**
    10. **getAge()**
    11. **getID()**
    12. **getAddress()**
    13. **getGender()**
    14. **getContact()**
    15. **getDateOfBirth()**
    16. **getDateOfMoveIn**
  + this class also contain a virtual function of the derived class (Doctor, Nurse)
  + Operator overloading **operator<<** to print all information of a Person



Description:

Take all value of class Person’s attributes to output stream using getter member methods of class Person itself, after that call a function **print()** of derived class to print all value of attributes at derived class too. (a derived class must provide this function)

Parameter:

* os: output stream which we will give our value to.
* other: class Person which we want to print an information.

Return:

output stream that just take all information from class attributes.

* + Operator overloading **operator>>** which user can input data to Person



Description:

Ask user to give all information of a person respectively, then set these values to the private attribute using 8 methods to set a values to 8 attributes of this class, after that call a function **getInfo()** of a derived class to get more information to an attributes of derived class. (a derived class must provide this function)

Parameter:

* is: input stream which will provide an information to Person.
* other: class Person which we want set an information to.

Return:

Input stream that will provide all information for a class attributes.

* class Doctor
  + this class is inherited from class Person
  + this class has 4 more private attributes:
    1. **std::string salary;** store an amount of salary of this doctor.
    2. **std::string specialized**; store what specialized is this doctor.
    3. **std::string office**; store an office of this doctor.
    4. **std::string email**; store an email of this doctor.
* This class has a default constructor to set an instance of this class to initial state. First call a default constructor of base class Person and for other attributes in this class, set them to an empty string.
* This class has 4 methods to set a value to each attribute and this class has 4 methods to get a value of every attribute:
  1. **setSalary()**
  2. **setSpecialized()**
  3. **setOffice()**
  4. **setEmail()**
  5. **getSalary()**
  6. **getSpecialized()**
  7. **getOffice()**
  8. **getEmail()**
* **Print derived class’s attributes**



Description:

Print a value of a derived class’s attributes to output stream that is given using a getter methods of class Doctor.

Parameter:

* os: output stream which will take a values of derived class’s attributes.

Return:

Output stream that just take all information from class’s attributes.

* **Set derived class’s attributes**



Description:

Get value from input stream and set it to class Doctor’s attributes using setter methods.

Parameter:

* is: input stream which will provide a value for each additional attributes of class Doctor.

Return:

Do not return anything.

* **Save all information of this Doctor into a file**



Description:

Put all attribute’s value of class Doctor including all attributes from class Person into the given output stream which will eventually save into a file

Parameter:

* os: output stream which will take a values of this Doctor.

Return:

Do not have return value

* **Load all information of this Doctor from a file**



Description:

Get value from input stream and set it to class Doctor’s attributes using setter methods.

Parameter:

* is: input stream which will give all value to class Doctor from an input stream which is a file.

Return:

Do not return anything.

* Operator overloading **operator==**



Description:

Compare all attributes of this Doctor with all attributes of other Doctor, if there is any attributes that is not the same then they are different, else they are the same.

Parameter:

* other: a Person that will be compared with this Doctor using getter methods of class Doctor and Person.

Return:

bool type indicating whether both are the same or not.

* class Nurse
  + this class is inherited from class Person
  + this class has 2 more attributes:
    1. **std::string salary**; store an amount of salary of this nurse.
    2. **std::string email**; store an email of this nurse.
  + This class has a default constructor to set an instance of this class to initial state. First call a default constructor of base class Person and for other attributes in this class, set them to an empty string.
  + This class has 2 methods to set a value to each attribute and this class has 2 methods to get a value of every attribute:
    1. **setSalary**
    2. **setEmail**
    3. **getSalary**
    4. **getEmail**
* **Print derived class’s attributes**



Description:

Print a value of a derived class’s attributes to output stream that is given using a getter methods of class Nurse.

Parameter:

* os: output stream which will take a values of derived class’s attributes.

Return:

Output stream that just take all information from class’s attributes.

* **Set derived class’s attributes**



Description:

Get value from input stream and set it to class Nurse’s attributes using setter methods.

Parameter:

* is: input stream which will provide a value for each additional attributes of class Nurse.

Return:

Do not return anything.

* **Save all information of this Doctor into a file**



Description:

Put all attribute’s value of class Nurse including all attributes from class Person into the given output stream which will eventually save into a file

Parameter:

* os: output stream which will take a value of this Nurse.

Return:

Do not have return value

* **Load all information of this Doctor from a file**



Description:

Get value from input stream and set it to class Nurse’s attributes using setter methods.

Parameter:

* is: input stream which will give all value to class Nurse from an input stream which is a file.

Return:

Do not return anything.

* Operator overloading **operator==**



Description:

Compare all attributes of this Nurse with all attributes of other Nurse, if there is any attributes that is not the same then they are different, else they are the same.

Parameter:

* other: a Person that will be compared with this Nurse using a getter method of class Nurse and Person.

Return:

bool type indicating whether both are the same or not.

* class Patient
  + this class is inherited from class Person
  + this class has 2 more attributes:
    1. **std::string symptoms**; store a symptom of patient.
    2. **std::string room**; store a room of that patient.
  + This class has a default constructor to set an instance of this class to initial state. First call a default constructor of base class Person and for other attributes in this class, set them to an empty string.
  + This class has 2 methods to set a value to each attribute and this class has 2 methods to get a value of every attribute:
    1. **setSymtoms**
    2. **setRoom**
    3. **getSymtoms**
    4. **getRoom**
* **Print derived class’s attributes**



Description:

Print a value of a derived class’s attributes to output stream that is given using a getter methods of class Patient.

Parameter:

* os: output stream which will take a values of derived class’s attributes.

Return:

Output stream that just take all information from class’s attributes.

* **Set derived class’s attributes**



Description:

Get value from input stream and set it to class Patient’s attributes using setter methods.

Parameter:

* is: input stream which will provide a value for each additional attributes of class Patient.

Return:

Do not return anything.

* **Save all information of this Doctor into a file**



Description:

Put all attribute’s value of class Patient including all attributes from class Person into the given output stream which will eventually save into a file

Parameter:

* os: output stream which will take a value of this Patient.

Return:

Do not have return value

* **Load all information of this Doctor from a file**



Description:

Get value from input stream and set it to class Patient’s attributes using setter methods.

Parameter:

* is: input stream which will give all value to class Patient from an input stream which is a file.

Return:

Do not return anything.

* Operator overloading **operator==**



Description:

Compare all attributes of this Patient with all attributes of other Patient, if there is any attributes that is not the same then they are different, else they are the same.

Parameter:

* other: a Person that will be compared with this Patient using a getter method of class Patient and Person.

Return:

bool type indicating whether both are the same or not.

**CHAPTER 4: CHANGES OF “PROJECT APPLICATION FORM”**

At the beginning I planned to write a database of a village, but later on, I feel that this topic is kind of too general. So, I changed my topic from a database of village to a database of a hospital which has same complexity and same domain. But I can play with more things like doctor, nurse, patient in the hospital, etc. Overall, it is more interested for me as well. Other specification like how I will manipulate this database remains the same as a database for the village.

**CHAPTER 5: TESTING**

* At first test, a list is empty then program read a person’s information from user and add these persons into a list, then we can check a list

Text

Description automatically generated

At first a list of Nurse is empty as shown in a figure on the right

Text

Description automatically generated

Then we add a Nurse and we fill an information of this Nurse

Text

Description automatically generated

Now we check a list of Nurse again and we see James Lapis is there.

This test is successful!

* In second test, at first, there is a Nurse in a list, now we will try to search for this nurse from a list.

Graphical user interface, text

Description automatically generated

At first, we see that James Lapis is in the list of Nurse

Graphical user interface, text

Description automatically generated

Then we search for James Lapis and we got all her information

Graphical user interface, text

Description automatically generated

If we try to search for Makki, we got that Makki is not in a list of Nurse, which is correct because there is only James Lapis in a list of Nurse.

This test is successful!

* In third test, at first, there is James Lapis in a list of Nurse, now we will remove James Lapis from a list of Nurse and we will check a list of Nurse again.

Graphical user interface, text

Description automatically generated

At first, there is James Lapis inside list of Nurse

Graphical user interface, text

Description automatically generated

Then we remove James Lapis from list of Nurse

Graphical user interface, text

Description automatically generated

Now we check list of Nurse again, we see that James Lapis is no longer inside this list.

This test is successful!

* In forth test, at first, there is Makki in a file, then we will remove Makki from a file and we will add Lapis in this file instead.

Graphical user interface, text

Description automatically generated

At first, there is Makki in a file.

Graphical user interface, text

Description automatically generated

We can also check in program that Makki is in a list of Doctor

Graphical user interface, text

Description automatically generated

Then we add Lapis into a list of Doctor

Text

Description automatically generated

Now we see that both Lapis and Makki are in a list of Doctor

Text

Description automatically generated

Now we will remove Makki from list of Doctor

Graphical user interface, text

Description automatically generated

We can check in program that there is only Lapis left in list of Doctor

Graphical user interface, text

Description automatically generated

Now we see that in a file there is Lapis there instead of Makki

File handling test is successful!

**SUMMARY:**

Base on the test cases above and a lot of test case which is done on background, we can verify that all features of this program work properly as well as file handling is working as expected. This program is completed as it is specified.