EOOP Preliminary Project

Student Name: Beste Baydur

Index Number: 295597

Date: 07.12.2018

CONTENTS

1. Explanation

1.1 Introduction

The aim of the project is to maintain the records of patient's medical history in a hospital. A test-driven approach was adapted in the development of this project. So, the actual project is divided into two separate parts, the development and the tests.

The motivation behind this project is to make the personal information of patients easily available for the staff, and medical history of the patient easily available to the doctors whenever required. The personal information of the patient can contain information like, Name, Address, Phone Number, Etc., and the Patient's medical history contain information like the blood group, previous accidents, diseases, any more remarks etc.,

The project is developed in such way that the objects of the class, can be Created, Read, Updated and Deleted (CRUD), with the data members accessible from other classes apart from the confidential information of the patient. The confidential information of the patient like medical history, diseases are kept private. The search functions are built where one could access information of a patient, using this built in search function. All these data are stored in a file, so any data created, updated or deleted are on this file. This is done using the FileIO functionalities in C++.

1.2 Flow Chart.

The Flow chart below explains the basic structure of the program.

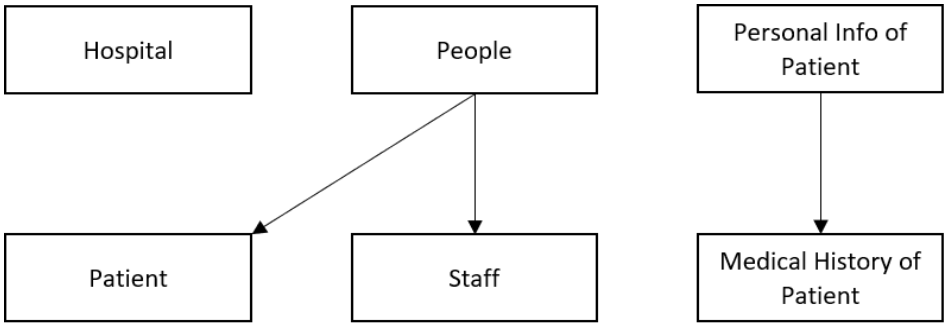

Fig 1. Flow Chart of the Program.

2 Classes

In this part of the report, each class is explained in brief. There are mainly 3 classes, that is a Hospital class, People class and a Class that contains the confidential information of the patient.

2.1 Hospital

The following shows the code of the class Hospital which is in the file Hospital.h file.

```
class Hospital {
int hospitalId, staffCount, patientCount, authKey;
string name;
std::list <Staff *> staff;
std::list <People*> People;
std::list <Hosiptal*> hospitals;
public:
//setters
        void hospitalId_(int);
        void patientCount_(int);
        void staffCount_(int);
        void name_(string);
        void authKey_(int);
```

```
        //getters
            int _hospitalId_();
            int _staffCount_();
            int _pateintCount_();
            string _name_();
            int _authKey_();
            bool _addHospital( int id, int sCount, int pCount, string name );
            bool enEmpty();
            patient& checkPatient(int id);
            Hospital& hospitalCheck(int id);
            bool update( int id, int sCount, int pCount, string name );
            bool updateParticularHosp( int sCount, int pcount, string name);
            void deleteHosp (int id);
            list<patient*> retrievePatientList();
            list<Hospital*> retrieveHospitalList();
            bool genPatFile( list<patient*>, string);
            bool patientCheck(int);
            bool hospitalCheck(int);
            void retrievePatients();
            void retrievePatientInfo( int patientID, int authenticationKey);
            void retrievePatientMHistory( int patientID, int authenticationKey);

        //patient functions
            bool addPatient( string name, int age, char sex, int room,
                                                    bool inHosp, int patientID, int hospId
);
            bool updatePatient( string name, int age, char sex, int room,
                                                    bool inHosp, int patientID, int hospId
);
            bool deletePatient( int patientID);

        //overload
            Hospital &operator = (cont Hospital &);
            Hospital operator+(cont Hospital &);
            friend ostream& operator << (ostream& os, const Hospital &);
            Hospital (cont Hospital &);
            Hospital(int, string);
            Hospital(){};
            ~Hospital(){};
};
```

The above shows the class of hospitals, some of the functions in the class are self-explanatory but some of the functions which are not, explained on the next page.

2.1.1 bool patientCheck(int)

This function takes in an integer value, which is the patients ID that was stored, and searches for this patient ID in the list of patients and returns a bool value True, if the patient exists in the list else, returns False.

2.1.2 bool genPatFile(list<patient>, string)*

This function takes in the list of patients and the string name of the output file, that is requested, the function creates a .txt file and stores in the requested directory. This file contains the list of the patients and their information in a particular hospital.

2.1.3 void retrievePatientInfo(int patientID, int authenticationKey)

The function lists the patient's personal info with the matching ID which is sent to this function but, only after if the input for function authenticationKey matches with the private data member of the class authKey.

2.1.4 void retrievePatientMHistory(int patientID, int authenticationKey)

The function lists the patient's medical history with the matching ID which is sent to this function but, only after if the input for function authenticationKey matches with the private data member of the class authKey.

2.2 People

The following shows the definition of the class People

```
class People {
        string name;
public:
        void addMember(string);
        People() {};
        ~People() {};
};
```

This class is used for adding members (patients/staff), also other classes are inherited from this class.

2.3 Staff

The class Staff which is a inherited class from the class People looks like below.

```
class Staff : public People {
        int workingHours, pay;
        bool isDoc;
public:
        void addStaff(int, int, bool)
        Staff() {};
        ~Staff() {};
};
```

The above class has a function addStaff, which adds the staff on the list with their working hours, salary and if they are doctor or other staff of the hospital.

2.4 Patient

This part of the document explains about the class patient which is also is inherited from People.

```
class Patient: public People {
        int patientID, hospitalID,age,roomNum;
        char sex;
        string name;
        bool isHospitalized;
        patientPersonalInfo *personal_details;
        patientMedHist *medical_history;
public:
        //setters
        void setName (string Name){ Name = name;};
        void setSex (char s) {sex = s;};
        void setAge(int a){age= a;};
        void setRoomNum(int rn){roomNum = rn;};
        void setHospitalized(bool val){isHospitalized = val;};
        void setPatientID(int id){ patientID = id;};
        void setHospitalID(int h_id) { hospitalID = h_id;};
```

```
//getters
        string getName() {return name;};
        char getSex() {return sex;};
        int getAge() {return age;};
        int getroomNum() {return roomNum;};
        bool ifHospitalized() {return isHospitalized;};
        int getPatientID() {return patientID;};
        int getHospitalID() {return hospitalID;};
        friend ostream& operator << (ostream& os, const Patient &);

        //medical History
        bool setMedicalHist( Patient *, string BGroup, int RiskLevel,
                bool donor, list <string> allergy, list <string> diseases);
        void getMedicalHist();

        //personal information
        bool setPersonalInfo( int patient_ID, string dateOfBirth, int height
                int weight, string addComments,phNum);
        Patient() {};
        ~Patient() {};
};
```

There are different functions in this class, that set and get personal information, medical history of the patient.

2.4.1 setMedicalHist():

This function takes in the data like blood group, risk levels of the patient, if the patient is donor, and list of allergies that patient has, and list of disease the patient is suffering from and stores it.

2.4.2 setPersonalInfo():

This exact function takes in the personal details like the patient ID, phone number, date of birth, height, weight, and comments if any and stores it.

All the other functions are self-explanatory and left unexplained in this short documentation.

2.5 Patient Personal Information

The following class has the information of personal information about the patient and is accessible to anone.

```
class patientPersonalInfo {
        string phNum,address,DOB;
        int height, weight;

public:
patientPersonalInfo(){};
~patientPersonalInfo(){};

};
```

2.6 Medical History of Patient

The medical history of a patient is a derived class from the personal information class, in this class, there are functions which can access data of the medical history of a patient over the duration. The information like the blood group, illness, allergies, diseases can be set and fetched from this class.

```
class medicalHistory: public patientPersonalInfo{
        string bGroup, diseases;
        bool isill, isDonor;
        int hospitilizedDays, risklevel;
        list <string> allergies;
public:
        //setters
        void _hospitilizedDays(int days){hospitilizedDays=days;};
        void _riskLevel( int risk){risklevel=risk;};
        void _donor(bool d){isDonor=d;};
        void _bGroup(string BGroup){bGroup=BGroup;};
        void _illness (bool ill){ isill = ill;};
        void _diseases( string dis){diseases = dis;};

        //getters
        string bGroup_(){return bGroup;};
        bool donor_(){return isDonor;};
        bool illness_(){return isill;};
        int hospitilizedDays_(){return hospitilizedDays;};
        int RiskLevel_(){return risklevel;};
        string diseases_(){return diseases;};
        medicalHistory(){};
        ~medicalHistory(){};     };
```

3. Tests

The above classes are tested creating a object of the class, this is done by passing the variables to create the patient with the necessary information to create a patient object. Same goes with the hospital too.

3.1 testPatient

This class is used to test the patient, the following shows the class and all the functions in this class are self-explanatory.

```
class testPatient{
public:
        Patient *pat = new Patient ( "Beste", 21, F , 1006, true, 25476, 631 );
        void addMedicalRecord();
        void addPersonalInfo();
        void staffcreate();
        void patientCreated();
        testPatient(){};
        ~testPatient(){};
};
```

The testing phase of the project is to test all the cases, activities and events of the class and see if the exceptions are thrown correctly and the code does not break.

The testing phase will use catch2 for some automated testing. The services offered.

- Make 1000's of patients and adding them to the hospitals.

  Catching of REQUIRE.
- Fast insertions and delete and to make sure the numbers match up.

  SECTION checking.
- Performance testing and timing events.
- SCENARIO based testing. Since this is a simulation, the highlight of testing cases would be with scenarios and GIVEN conditions.
- Make wrong insertions to groups.
- Make multiple combinations of instances.