



Beste Baydur

295597

OBJECT ORIENTED C++ PROJECT HOSPITAL SIMULATION SYSTEM

Table of Contents

1. Project Description.....	2
1.1 The Outline.....	2
1.2 Functionality	2
1.3 Flow Chart	2
2. Class Definitions	3
2.1 Hospital	3
2.1.1 Patient patientExists (int id).....	4
2.1.2 void addToFile(list<Patient*>, string)	4
2.1.3 Void showPatientPersonalInfo (int, int).....	5
2.1.4 Void showPatientMedicalHistory(int, int).....	5
2.2 People	5
2.3 Staff	5
2.4 Patient.....	5
2.4.1 Bool appendMedicalHistory(Patient *p, string b_group, bool terminal, int days, int risk_level, bool donor)	6
2.4.2 Bool appendPersonalInfo(Patient *p, int p_num, strng add, string dob, int height, int weight) 6	
2.5 Patient Personal Information.....	6
2.6 Patient Medical History.....	7
2.6.1 Bool donor.....	7
2.6.2 Bool terminal_illness.....	7
2.6.3 List <string> allergies.....	Error! Bookmark not defined.
3. Tests	8
3.1 Patient Tests.....	8
3.2 Hospital Tests	8
4. Usage.....	9

1. Project Description

The project has been divided mainly into 2 parts, the tests and the actual project file. The project was taken a test-driven development approach to develop the complete project. Unit testing with the libraries were done in the approach. Assertions were used for run-time testing to ensure the program.

1.1 The Outline

The motivation behind the project was to develop a record book that holds the patient's medical history in various hospitals. In an environment of a Hospital, it can have many patients, staff and each patient can belong to only one hospital. Each patient has an object of the class patientBio and patientMedicalHistory which include the patient's biodata or the personal information like address, phone number, date of birth etc., is stored, and patient's medical history will have information like blood group, diseases etc.,

1.2 Functionality

Each class in the project is a CRUD class, which means the objects can be Created, Read, Updated and Deleted with respective data members that are accessible from the other classes, but the sensitive data of the patient cannot be accessed since it is kept private. The Patient class and the hospital class are interlinked so that the information belonging to the patient can be looked up by hospital if the patient belongs to that hospital. There are built in functions in the class, that does this task. The program can generate a file using fileIO if required.

1.3 Flow Chart

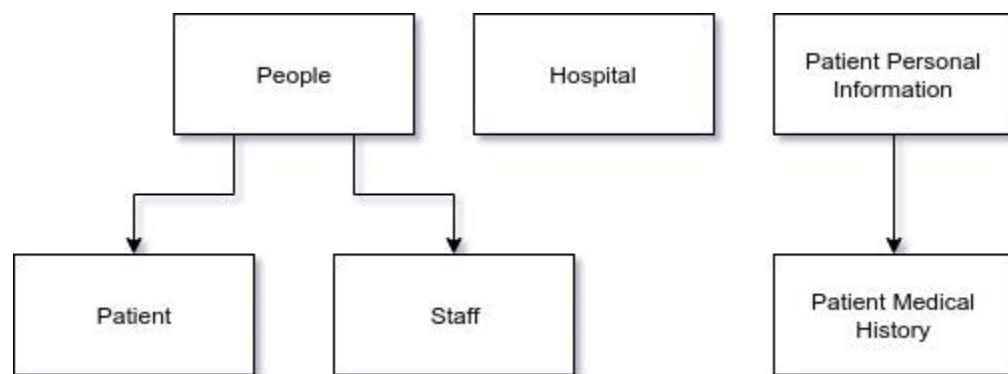


Fig 1. Flow chart of the Program.

2. Class Definitions

The following shows the structure of the project.

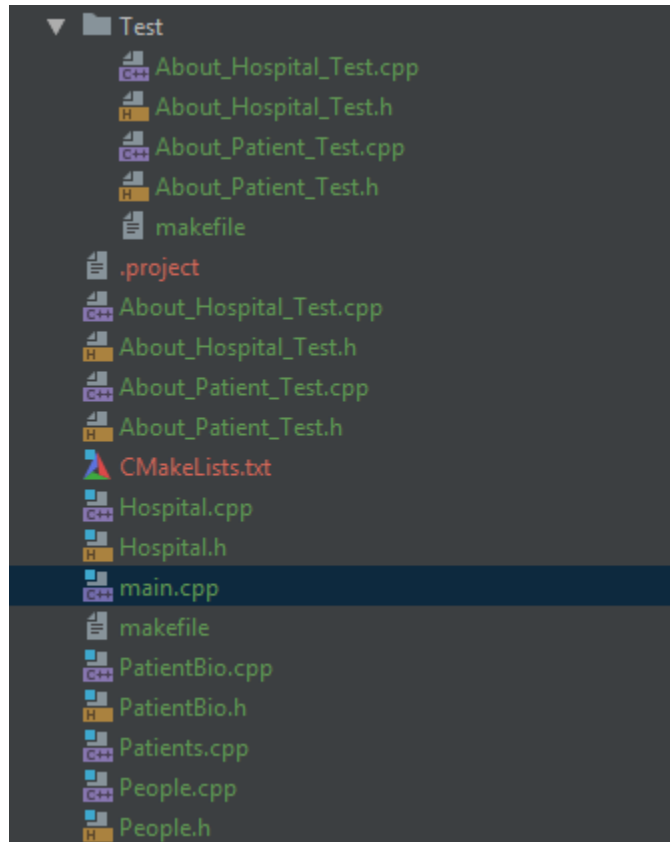


Fig 2. Structure of the project.

2.1 Hospital

The following shows the definition of class Hospital

```
class Hospital {
    int hospitalId;
    int patient_count, staff_count;
    string name;
    int key;
    std::list <Hospital*> hospitals;
    std::list <Patient*> patients;
    std::list <Staff*> staff;

public:
    //setters
    void hospitalId_(int);
    void staff_count_(int);
    void patient_count_(int);
    void name_(string);
    void key_(int);

    //getters
    int _hospitalID();
    int _staff_count();
    int _patient_count();
}
```

```

string _name();
int _key();

bool addHospital ( int id, int staff_count,
                  int patient_count, string name);
bool isEmpty();
Hospital& hospitalExists(int id);
bool updateInformation( int id, int staff_count, int patient_count, string name);
bool updateHospital( int, int, string);
void deleteHospital(int id);
//bool deletePatient(int patient_id);
Patient& patientExists(int id);
list <Patient*> getList();
list <Hospital*> getHospitalList();

void retrievePatients();
void showPatientPersonalInfo(int patient_id, int key);
void showPatientMedicalHistory(int patient_id, int key);
bool addToFile(list <Patient*>, string);

bool ifPatientsExists(int);
bool ifHospitalExists(int);

//Patient Methods
bool appendPatient( int patient_id, int hospital_id, string name,
                  char sex, int age, int room_num, bool hospitalized);
bool deletePatient(int patient_id);

bool patientUpdateInfo( int patient_id, int hospital_id, string name, char sex,
                      int age, int room_num, bool hospitalized);

//overloaded operators
Hospital &operator=(const Hospital &);
Hospital operator+(const Hospital &);
friend ostream& operator << (ostream& os, const Hospital &);

//default, overloaded and copy constructors
Hospital(const Hospital &);
Hospital(int, string);
Hospital() { };
~Hospital() {};
};

```

Most of the functions are self- explanatory, some of the functions are explained below.

2.1.1 Patient patientExists (int id)

This function takes in the integer type ID of the patient and retrieves the reference to the object of class patient by going through the list of all patients in the database.

2.1.2 void addToFile(list<Patient*>, string)

This function takes the list of the patients and a string type object which is necessarily the name of the file that is being created with the list of patients. A text file (.txt) is generated. The file is written using file IO methods built in C++.

2.1.3 Void showPatientPersonalInfo (int, int)

This function takes two integer values, namely patient ID and the authentication key. The function in turn calls the patientExists function with the patient ID and then reveals the patient's info after the authorization.

2.1.4 Void showPatientMedicalHistory(int, int)

This works similar to the previous function but in this case the patient's medical history is revealed instead of personal info.

2.2 People

Following shows the definition of the class People which is defined in the People.h file.

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

As seen there is no much details in the People class, but other classes are inherited from this class.

2.3 Staff

Following shows the definition of the class Staff, which is also defined in People.h file. This is an inherited class of people class.

```
class Staff : public People {  
    //in hours  
    int shift, salary;  
    bool isDoctor;  
public:  
    Staff() {};  
    ~Staff() {};  
};
```

2.4 Patient

This is also a class which is inherited from the people class, is also defined in People.h file.

```
class Patient : public People {  
    int patientId, hospitalId;  
    char sex;  
    int age, room_number;  
    bool hospitalized;  
  
    PatientBio *personal_info;  
    patientMedicalRecord *medical_history;  
public:  
    //setters  
    void patientId_(int);  
    void hospitalId_(int);  
    void name_(string);  
    void sex_(char);  
    void age_(int);  
    void room_number_(int);  
    void hospitalized_(bool);
```

```

//getters
int _patientId();
PatientBio* fetchPatientPersonalInfo();
patientMedicalRecord* fetchPatientMedicalHistory();
int _hospitalId();
string _name();
char _sex();
int _age();
int _room_number();
bool _hospitalized();
friend ostream& operator << (ostream& os, const Patient &);

//medical History methods
bool appendMedicalHistory(Patient *p, string b_group, bool terminal,
                          int days, int risk_level, bool donor/*,
                          list <string> allergies*/);
void retrieveMedicalHistory();
void retrievePersonalInfo();

//personal information methods
bool appendPersonalInfo(Patient *p, int p_num, string add, string dob,
                       int height, int weight);
Patient() { patientId = -1; };
Patient(int, int, string, char, int, int, bool);
~Patient() { };
};

```

Each patient has an object of patientMedicalHistory and patientBio classes which are defined later in this report.

2.4.1 Bool appendMedicalHistory(Patient *p, string b_group, bool terminal, int days, int risk_level, bool donor)

This function is to add the medical history of the patient, all the parameters in the function are self explanatory. The function takes in all these values and adds the information to the particular patient object.

2.4.2 Bool appendPersonalInfo(Patient *p, int p_num, strng add, string dob, int height, int weight)

This function is to add the personal information of the patient to the database, the function takes all the parameters as shown above and are assigned to the particular patient.

2.5 Patient Personal Information

The following shows the class PatientBio which is defined in the PatientBio.h file.

```

class PatientBio {
    string address;
    string dob;
    int height, weight;
    int tel_number;

public:
    void address_(string);
    void dob_(string);
    void height_(int);

```

```

void weight_(int);
void tel_number_(int);

int _height();
int _weight();
string _address();
string _dob();
int _tel_number();

PatientBio() {};
~PatientBio() {};
};

```

All the functions in this class are self-explanatory, so they are not further defined.

2.6 Patient Medical History

This Class is a derived class of PatientBio class, it is publicly inherited and this class is mentioned in PatientBio.h file.

```

class patientMedicalRecord : public PatientBio {

    bool terminal_illness;
    int days_hospitalized;
    bool donor;
    string blood_group;
    int risk_level;
    list <string> allergies;

public:
    //setters
    void blood_group_(string);
    void terminal_illness_(bool);
    void days_hospitalized_(int);
    void risk_level_(int);
    void donor_(bool);

    //getters
    string _blood_group();
    bool _terminal_illness();
    int _days_hospitalized();
    int _risk_level();
    bool _donor();

    patientMedicalRecord() {};
    ~patientMedicalRecord() {};
};

```

2.6.1 Bool donor

This variable will tell if the patient is willing to donate his organs to be harvested incase of his/her death.

2.6.2 Bool terminal_illness

This variable has the information about the patient if they have terminal illness or not.

3. Tests

All the tests that are done during the project are included in the "tests" folder within the project. There are 2 main test files, for hospital and patients' tests. All the tests are passing in the final build, which are tested using assert.

3.1 Patient Tests

The About_Patient_test.h/.cpp files include tests related to the About_Patient_test class, all the variables in the class are self explanatory and below shows the definition of this class.

```
class About_Patient_Test {
public:
    Patient *pat = new Patient(1, 1, "test", 'x', 25, 65, true);
    void appendedMedicalRecord();
    void appendedPersonalInfo();
    void staffCreated();
    void patientSuccessfullyAppended();
    About_Patient_Test() {}
    ~About_Patient_Test() {}
};
```

3.2 Hospital Tests

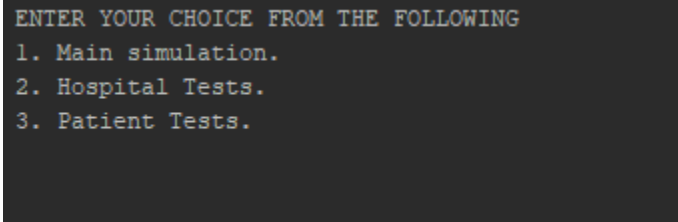
The About_Hospital_test.h/.cpp files include tests related to the About_Hospital_test class, all the variables in the class are self explanatory and below shows the definition of this class.

```
class hospitalTest{
public:
    Hospital *hosp = new Hospital (1, "My New Hospital");
    void afterHospitalAddIsNotEmpty();
    void afterPatientAddIsNotEmpty();
    void hospitalDoesntExistAfterDeleting();
    void patientDoesntExistAfterDeleting();
    void operatorCopiedCorrectly();
    void exportedPatientList();
    void updatedHospitalInfo();
    void updatedPatientInfo();
    void patientDeleted();
    hospitalTest() {}
    ~hospitalTest() {}
};
```

4. Usage

The main file has been created to run the project in 3 different instances, the debug screen will ask the user to input options from 1 to 3, and each of these inputs executes 3 different things, the input "1" executes the project file, "2" executes Hospital test and "3" executes the Patient test.

The output screen looks like below on running the project.



```
ENTER YOUR CHOICE FROM THE FOLLOWING
1. Main simulation.
2. Hospital Tests.
3. Patient Tests.
```

Fig 3. The output screen.

On selecting the option, the output is shown on the screen.

s