UTM
UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

Semester 2 2023/2024

Subject : Programming Technique (SECJ1023)
Section : 04
Task : Phase 4 – Final
Due : Week 14

Group 9 Member

	Name	**Matric Number**
1	ALYA QISTINA BINTI AWALUDDIN	A23CS0041
2	NUR ARISHA BINTI AMYRUL NAIM	A23CS0154
3	TANG JASMINE	A23CS0277

1.0 SECTION A

Our proposed system is the medication scheduler. The general idea of this system is to develop a system which can be used by individuals who have to take medication on a schedule with the accurate dosage, especially for those who have to take different medications with different dosage at a time. This is an upgraded version of the traditional system that uses labeled containers to alert patients on medicine intake. Instead, this system can be integrated into their device and can be accessed at any time.

Our system objectives are:

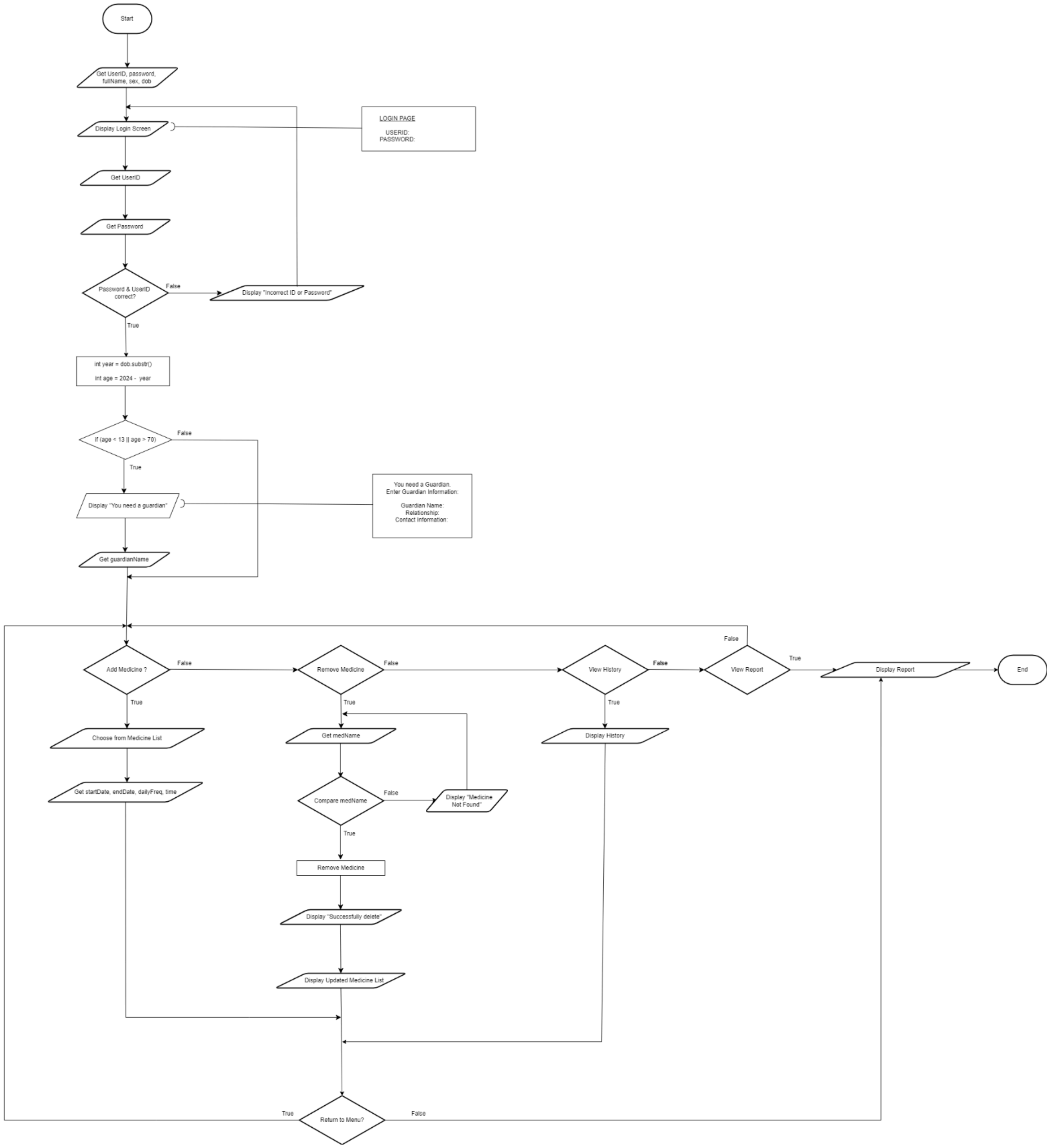
1. Medication scheduling: record prescribed medication details (medication name, shapes, color), shapes and colors are easier way to identify different medication
2. Keep track: dosage, timing, routine (before meal/after meal, daily) and progress (e.g. antibiotics take up to 2 weeks only)
3. Reminder: Alert user the time to take medicine by send notification to make sure user take the medication on time
4. Secondary assurance: supervision from guardian or personal doctor especially for the elderly, guardian/personal healthcare provider can monitor patient virtually
5. Portable (system accessible through any electronic device: smart watch/phone): offers the convenience of having all medication information in one place

Overall, the medication scheduler system manages their medications more effectively, helping individuals with better health outcomes and improved quality of life.

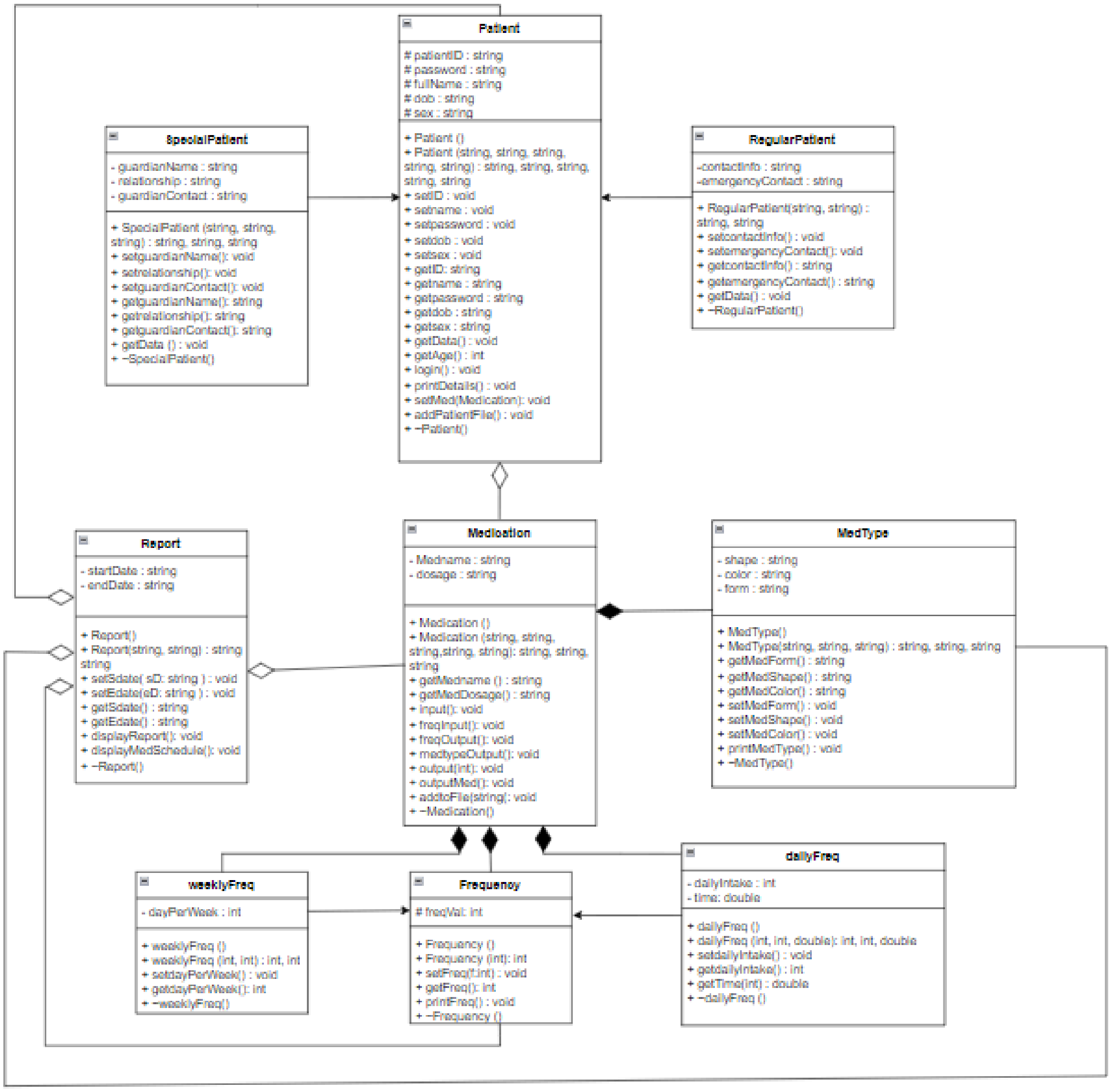
On June 23, 2024, we introduced the Medication Scheduler system to Madam Lizawati, who provided encouraging input and constructive feedback. In response, we have included a number of modifications and improvements to our analytical and design process. The modifications are:

1. Adding database by creating output file to store patient list, patientlist.txt and patient's medicine history, PatientName_med_history.txt
2. Hiding password by using getch() and include <conio.h>
3. Displaying the past and current medicine(s) that students have to take by using time function and include <ctime>

1.1 Flowchart



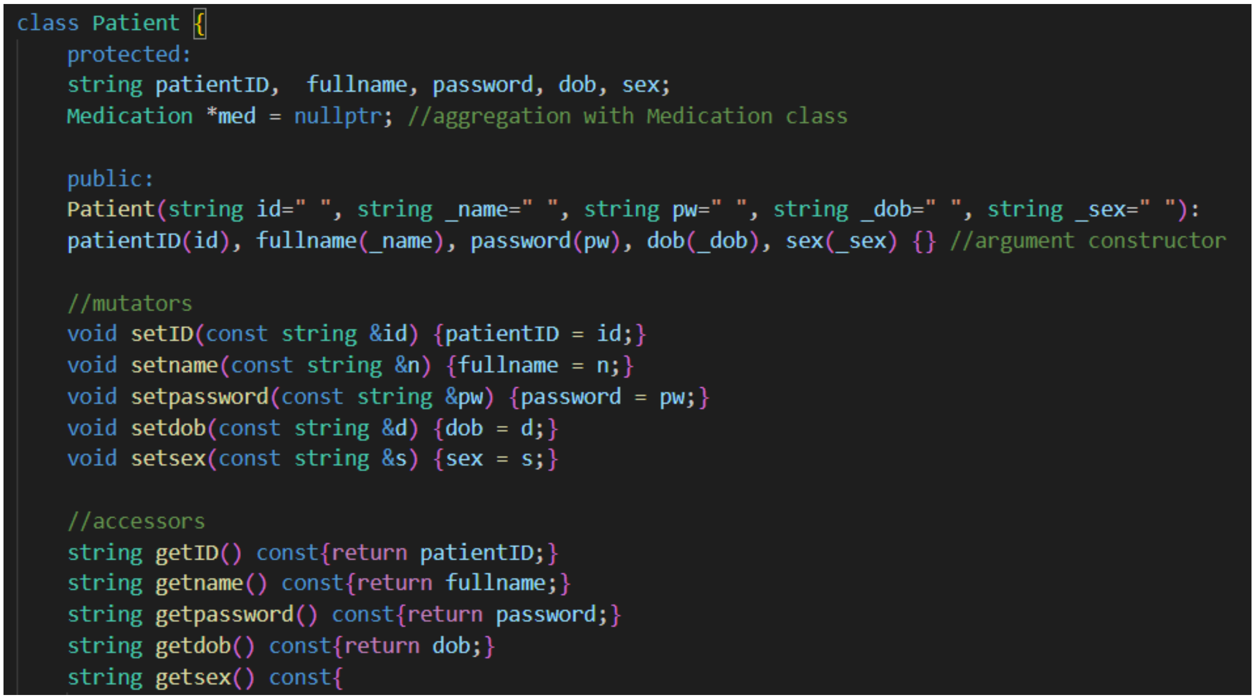
1.2 UML Class Diagram



2.0 SECTION B

2.1 Encapsulation

As proven in code provided in Section C, all data created are encapsulated within classes, with private, protected attributes and public accessor or mutator methods controlling access. It illustrates the use of bundling data and methods that operate on that data within a single class while still restricting access to some of the attributes. For instance, in the 'Patient' class, data members such as 'password' and 'fullname' are both marked as protected. This allows them to be accessed within the class itself and by derived class, if any, but more importantly not by any outside code. This concept guarantees that any sensitive information is guarded and can only be modified through the methods included in the public section such as 'setpassword' and 'setname'. Though, this makes it easy to access the attributes as few assessors have also been included like 'getpassword' resulting in more controlled and secure data retrieval. All classes are provided with constructors for smooth initialization and also destructors for clean-up objects. All in all, encapsulation applied in this code helps to maintain secure data by controlling how each data is accessed and modified.

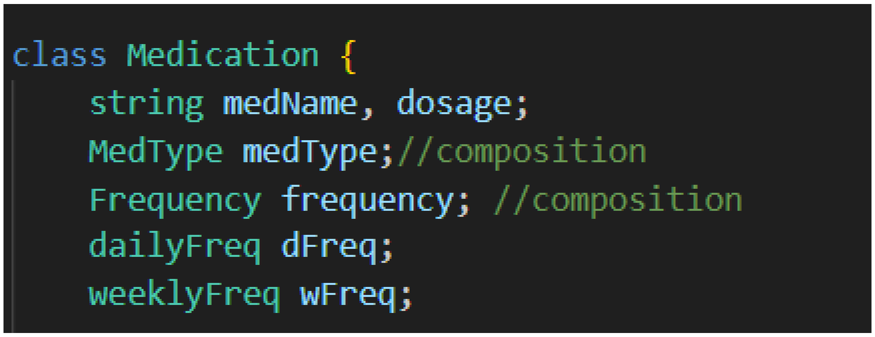


```
class Patient {
    protected:
    string patientID,  fullname, password, dob, sex;
    Medication *med = nullptr; //aggregation with Medication class

    public:
    Patient(string id=" ", string _name=" ", string pw=" ", string _dob=" ", string _sex=" "):
    patientID(id), fullname(_name), password(pw), dob(_dob), sex(_sex) {} //argument constructor

    //mutators
    void setID(const string &id) {patientID = id;}
    void setname(const string &n) {fullname = n;}
    void setpassword(const string &pw) {password = pw;}
    void setdob(const string &d) {dob = d;}
    void setsex(const string &s) {sex = s;}

    //accessors
    string getID() const{return patientID;}
    string getname() const{return fullname;}
    string getpassword() const{return password;}
    string getdob() const{return dob;}
    string getsex() const{
```

2.2 Composition

Composition refers to the enclosing object and enclosed object are highly dependent on each other. The existence of the enclosed objects are determined by the enclosing objects. In our system, we applied the concept of composition where class Medication is the enclosing object, objects of class Medtype, class Frequency, class dailyFreq and weeklyFreq are enclosed objects.

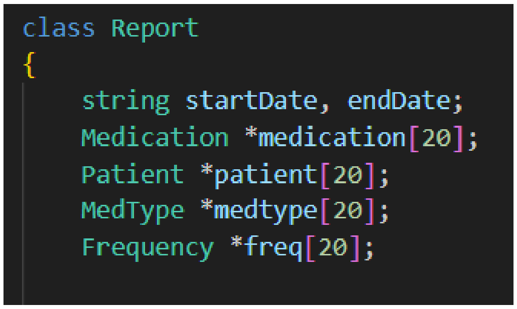
This means that Medication has Medtype, Frequency, dailyFreq and weeklyFreq. Hence, once the enclosing object in Medication class is destroyed, the enclosed object will be destroyed as well; if the enclosing object in Medication is created, the enclosed object will also be created.



```
class Medication {
    string medName, dosage;
    MedType medType;//composition
    Frequency frequency; //composition
    dailyFreq dFreq;
    weeklyFreq wFreq;
```

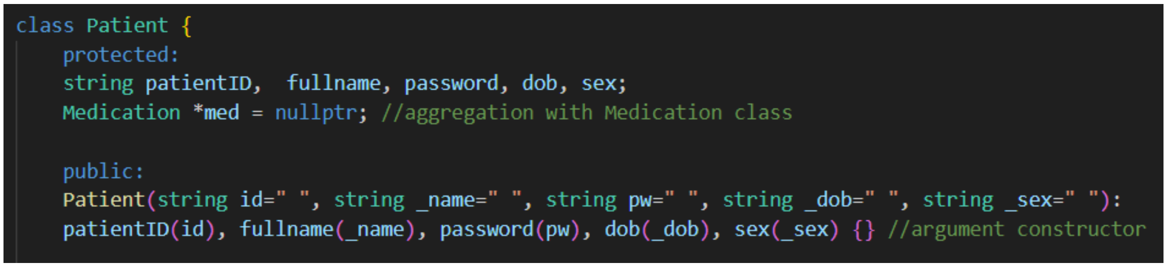
2.3 Aggregation

Aggregation is a one way relationship. The difference between aggregation and composition is that aggregation of both enclosed and enclosing objects exist independently. Based on the code attached below, the object of class Report has objects in class Medication, class Patient, Class MedType and class Frequency. As the relationship between classes is independent, the destroyed objects do not affect the other objects. The relationship between objects can be broken by only disconnecting the pointer.



```
class Report
{
    string startDate, endDate;
    Medication *medication[20];
    Patient *patient[20];
    MedType *medtype[20];
    Frequency *freq[20];
```

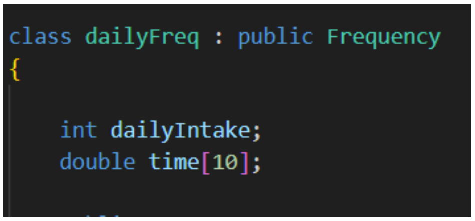
The code below shows that objects of class Patient have an aggregation relationship with the objects in class Medication.



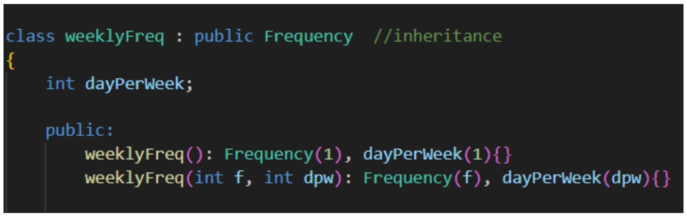
```
class Patient {
    protected:
    string patientID,  fullname, password, dob, sex;
    Medication *med = nullptr; //aggregation with Medication class

    public:
    Patient(string id=" ", string _name=" ", string pw=" ", string _dob=" ", string _sex=" "):
    patientID(id), fullname(_name), password(pw), dob(_dob), sex(_sex) {} //argument constructor
```

2.4 Inheritance

In the provided code included in Section C, one of the concepts of inheritance is demonstrated through ‘dailyFreq’ and ‘weeklyFreq’, both are derived from the parent class, ‘Frequency’ or also known as base class. Implementing this relationship and concept has allowed us to inherit and utilize the ‘freqVal’ function from the parent class. This not only makes the code reusable but also exhibits a clear format of hierarchical structure. Polymorphism is shown in ‘printFreq’ which is declared as virtual in the base class and overridden in both derived classes. Because of this, both derived classes can supply their own implementation of ‘printFreq’.

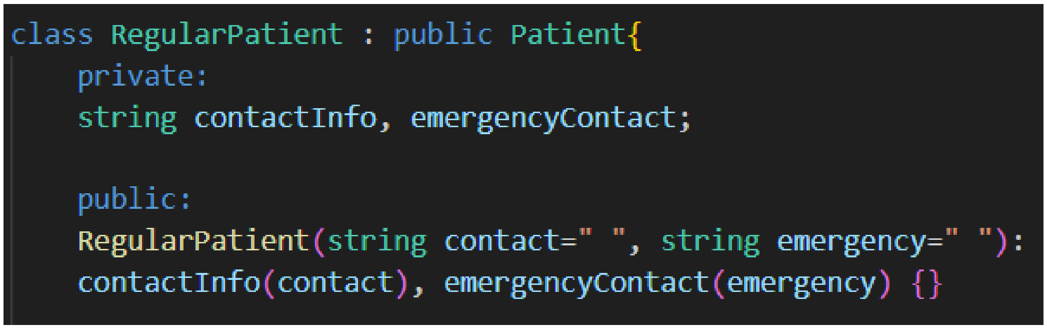


```
class dailyFreq : public Frequency
{

    int dailyIntake;
    double time[10];
```

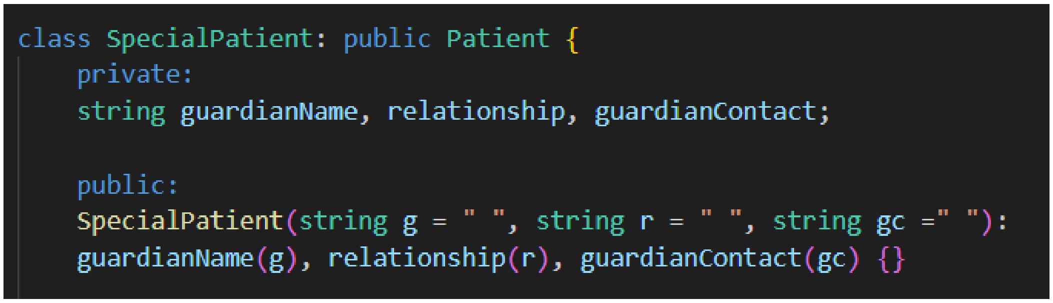


```
class weeklyFreq : public Frequency  //inheritance
{
    int dayPerWeek;

    public:
        weeklyFreq(): Frequency(1), dayPerWeek(1){}
        weeklyFreq(int f, int dpw): Frequency(f), dayPerWeek(dpw){}
```

Another inheritance relationship created in the code is in the context of a patient management system. The ‘Patient’ class serves as the parent class, encapsulating a few attributes such as ‘patientID’, ‘fullname’ and ‘password’.Derived from this is the ‘RegularPatient’ and ‘SpecialPatient’ that automatically inherits the properties and methods of the parent class while also adding its own specific items. This concept allows the extent of functionality of the ‘Patient’ class without the need to duplicate the code.



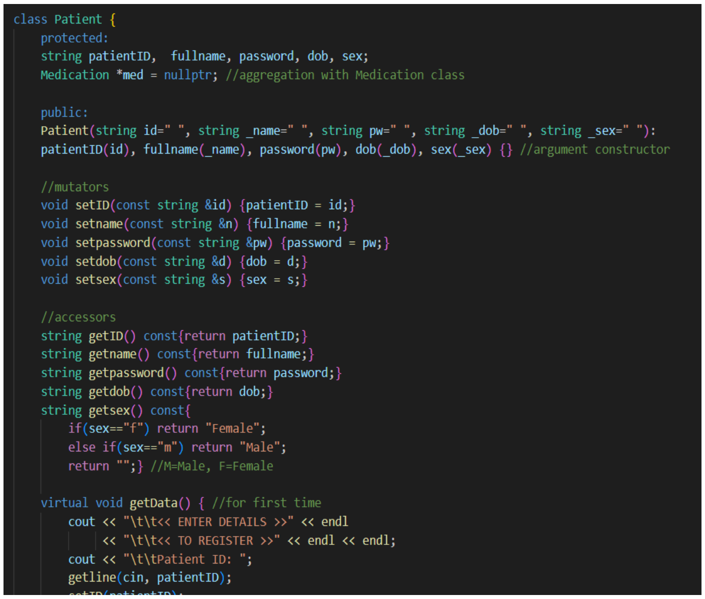
```
class RegularPatient : public Patient{
    private:
    string contactInfo, emergencyContact;

    public:
    RegularPatient(string contact=" ", string emergency=" "):
    contactInfo(contact), emergencyContact(emergency) {}
```



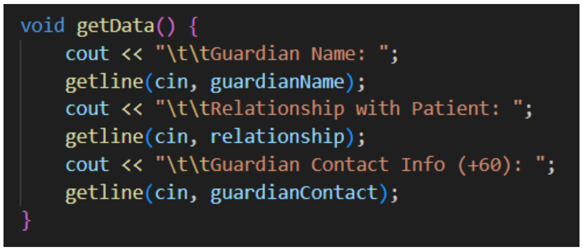
```
class SpecialPatient: public Patient {
    private:
    string guardianName, relationship, guardianContact;

    public:
    SpecialPatient(string g = " ", string r = " ", string gc =" "):
    guardianName(g), relationship(r), guardianContact(gc) {}
```

2.5 Polymorphism

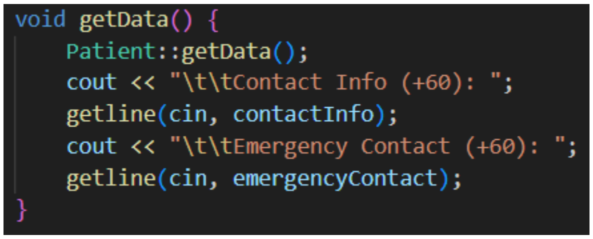
Polymorphism can be applied when there’s inheritance relationships between classes. As inheritance inherits attributes and methods from another class; Polymorphism uses those methods to perform different tasks. Polymorphism refers to a function that has the same action/same name but different behavior. As referring to our code: class Patient is a parent class, the function getData() is used in parent class and child classes, hence the functions are dynamically bound by specifying the methods as virtual in parent class. The child classes override the method getData().



```cpp
class Patient {
    protected:
    string patientID,  fullname, password, dob, sex;
    Medication *med = nullptr; //aggregation with Medication class

    public:
    Patient(string id=" ", string _name=" ", string pw=" ", string _dob=" ", string _sex=" "):
    patientID(id), fullname(_name), password(pw), dob(_dob), sex(_sex) {} //argument constructor

    //mutators
    void setID(const string &id) {patientID = id;}
    void setname(const string &n) {fullname = n;}
    void setpassword(const string &pw) {password = pw;}
    void setdob(const string &d) {dob = d;}
    void setsex(const string &s) {sex = s;}

    //accessors
    string getID() const{return patientID;}
    string getname() const{return fullname;}
    string getpassword() const{return password;}
    string getdob() const{return dob;}
    string getsex() const{
        if(sex=="f") return "Female";
        else if(sex=="m") return "Male";
        return "";} //M=Male, F=Female

    virtual void getData() { //for first time
        cout << "\t\t<< ENTER DETAILS >>" << endl
             << "\t\t<< TO REGISTER >>" << endl << endl;
        cout << "\t\tPatient ID: ";
        getline(cin, patientID);
        setID(patientID);
```

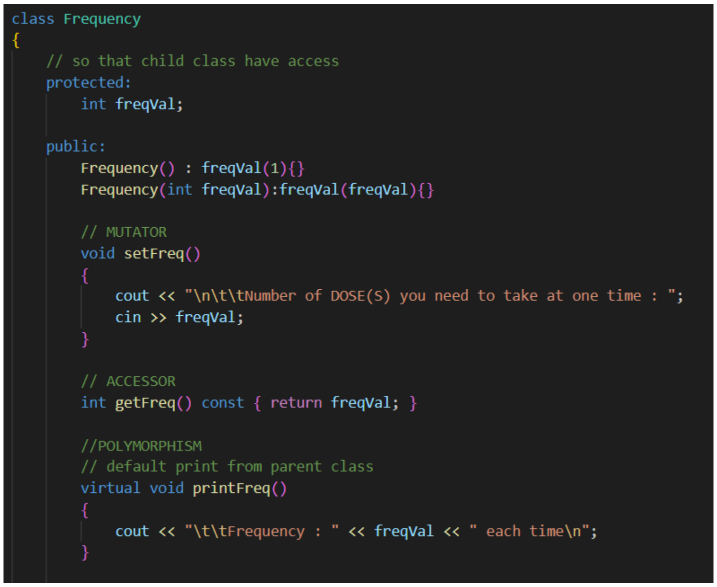


```cpp
void getData() {
    cout << "\t\tGuardian Name: ";
    getline(cin, guardianName);
    cout << "\t\tRelationship with Patient: ";
    getline(cin, relationship);
    cout << "\t\tGuardian Contact Info (+60): ";
    getline(cin, guardianContact);
}
```

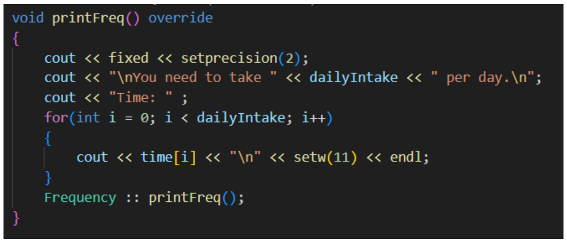


```cpp
void getData() {
    Patient::getData();
    cout << "\t\tContact Info (+60): ";
    getline(cin, contactInfo);
    cout << "\t\tEmergency Contact (+60): ";
    getline(cin, emergencyContact);
}
```

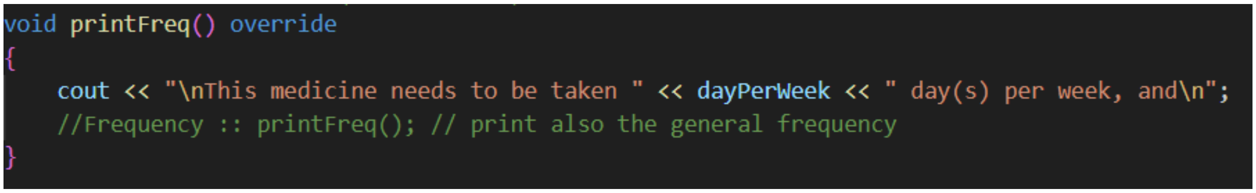
The concept of polymorphism is applied in class Frequency as the parent class. printFreq() function in parent class is a virtual function, allowing the function to be dynamically bound. The 2 other child classes overrides the method printFreq().



```
class Frequency
{
    // so that child class have access
    protected:
        int freqVal;

    public:
        Frequency() : freqVal(1){}
        Frequency(int freqVal):freqVal(freqVal){}

        // MUTATOR
        void setFreq()
        {
            cout << "\n\t\tNumber of DOSE(S) you need to take at one time : ";
            cin >> freqVal;
        }

        // ACCESSOR
        int getFreq() const { return freqVal; }

        //POLYMORPHISM
        // default print from parent class
        virtual void printFreq()
        {
            cout << "\t\tFrequency : " << freqVal << " each time\n";
        }
```



```
void printFreq() override
{
    cout << fixed << setprecision(2);
    cout << "\nYou need to take " << dailyIntake << " per day.\n";
    cout << "Time: " ;
    for(int i = 0; i < dailyIntake; i++)
    {
        cout << time[i] << "\n" << setw(11) << endl;
    }
    Frequency :: printFreq();
}
```

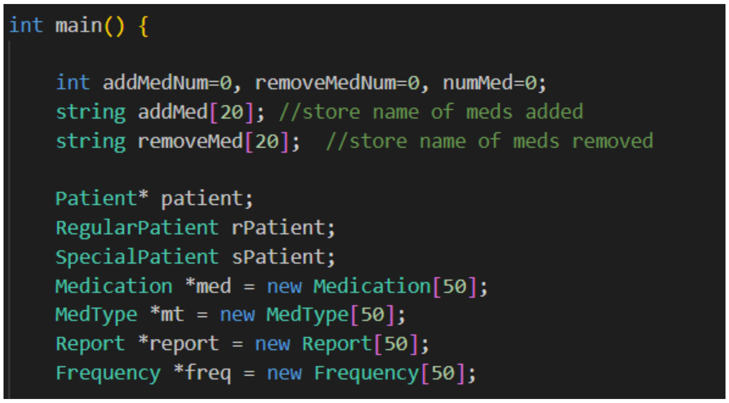


```
void printFreq() override
{
    cout << "\nThis medicine needs to be taken " << dayPerWeek << " day(s) per week, and\n";
    //Frequency :: printFreq(); // print also the general frequency
}
```

2.6 Array of Objects

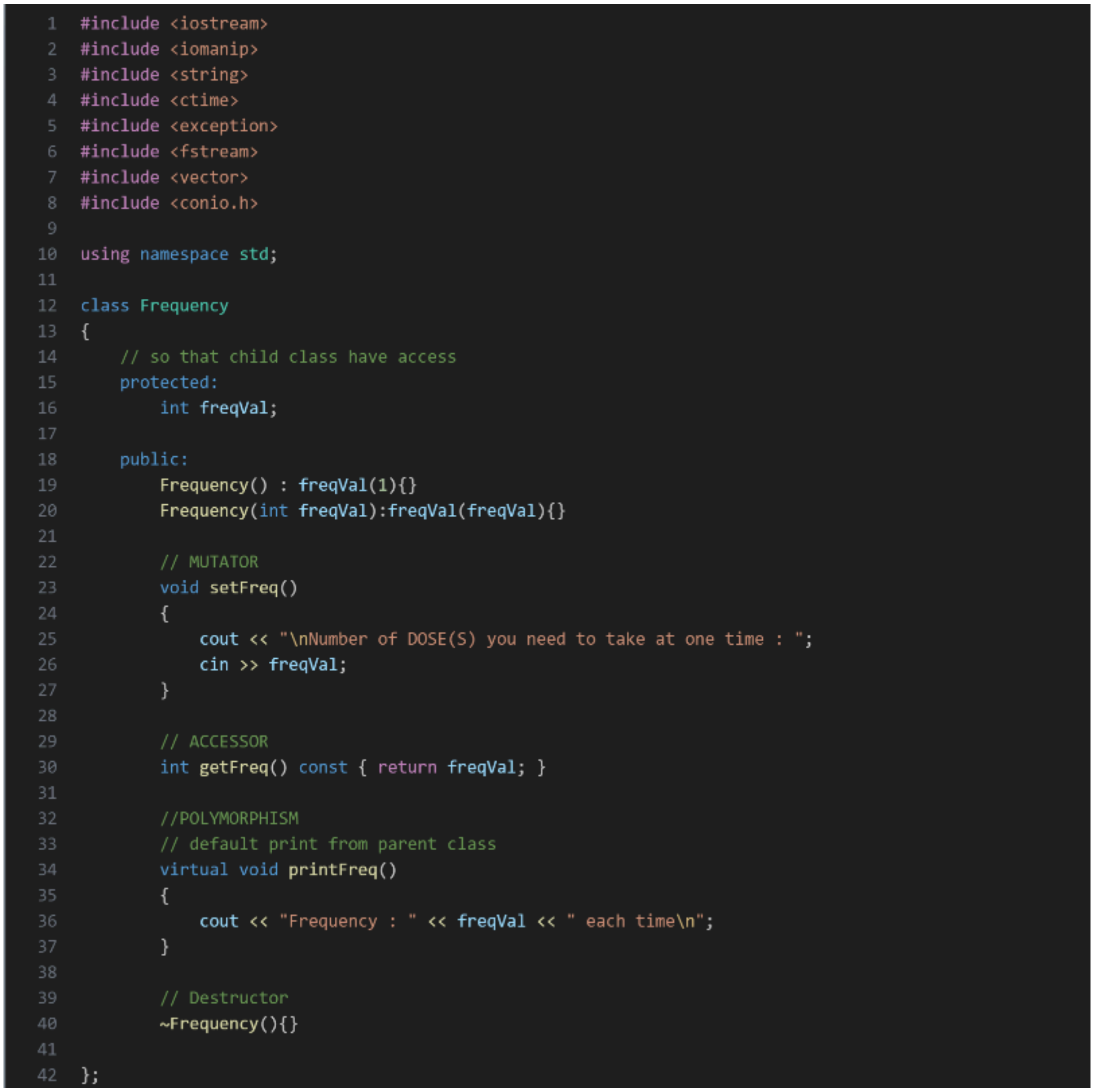
Based on the code below, we've created a static array where variables addMed and removeMed are able to hold 20 data.

We also implemented an array of objects where we use pointers to dynamically allocate the objects.

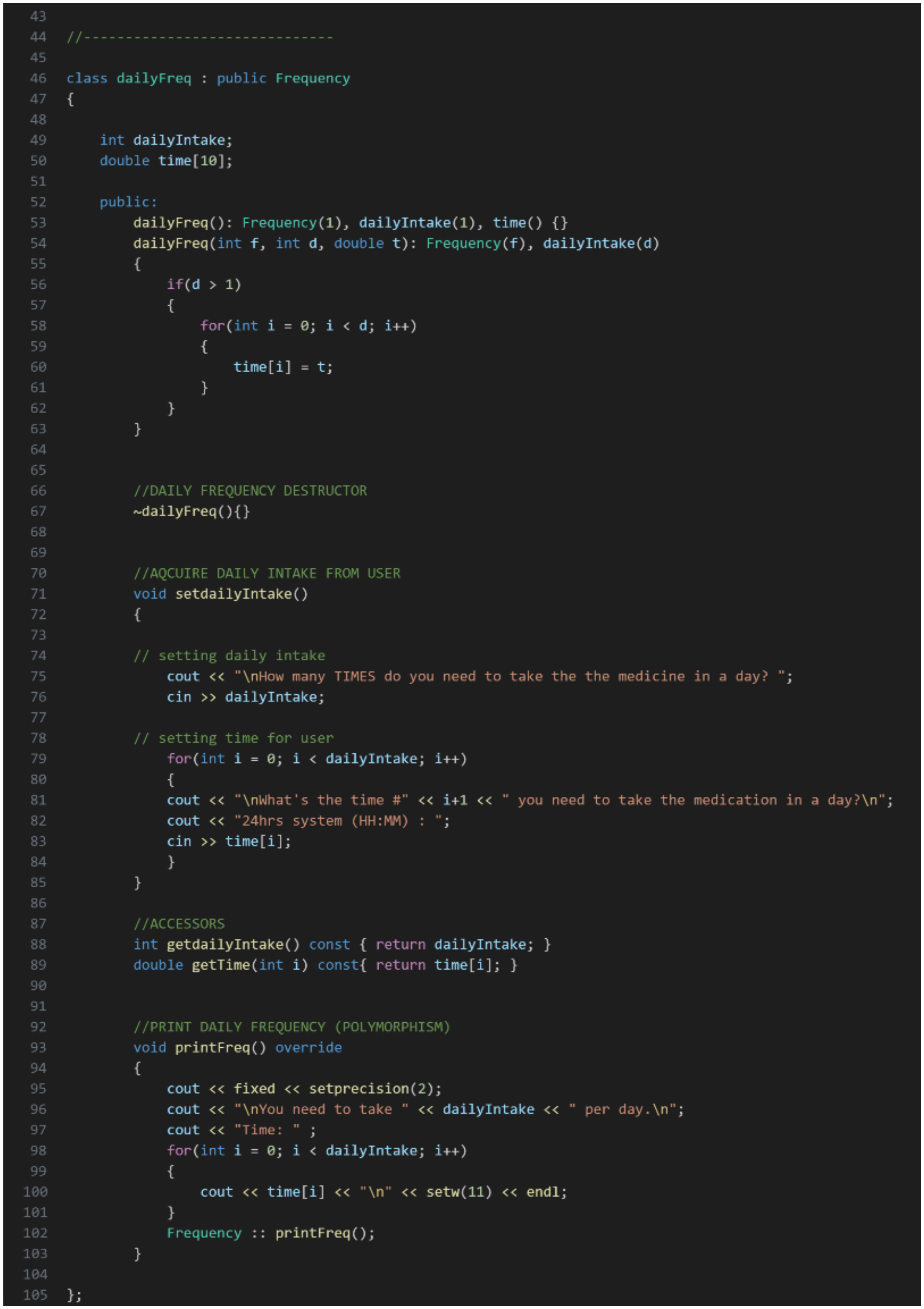


```
int main() {

    int addMedNum=0, removeMedNum=0, numMed=0;
    string addMed[20]; //store name of meds added
    string removeMed[20];  //store name of meds removed

    Patient* patient;
    RegularPatient rPatient;
    SpecialPatient sPatient;
    Medication *med = new Medication[50];
    MedType *mt = new MedType[50];
    Report *report = new Report[50];
    Frequency *freq = new Frequency[50];
```

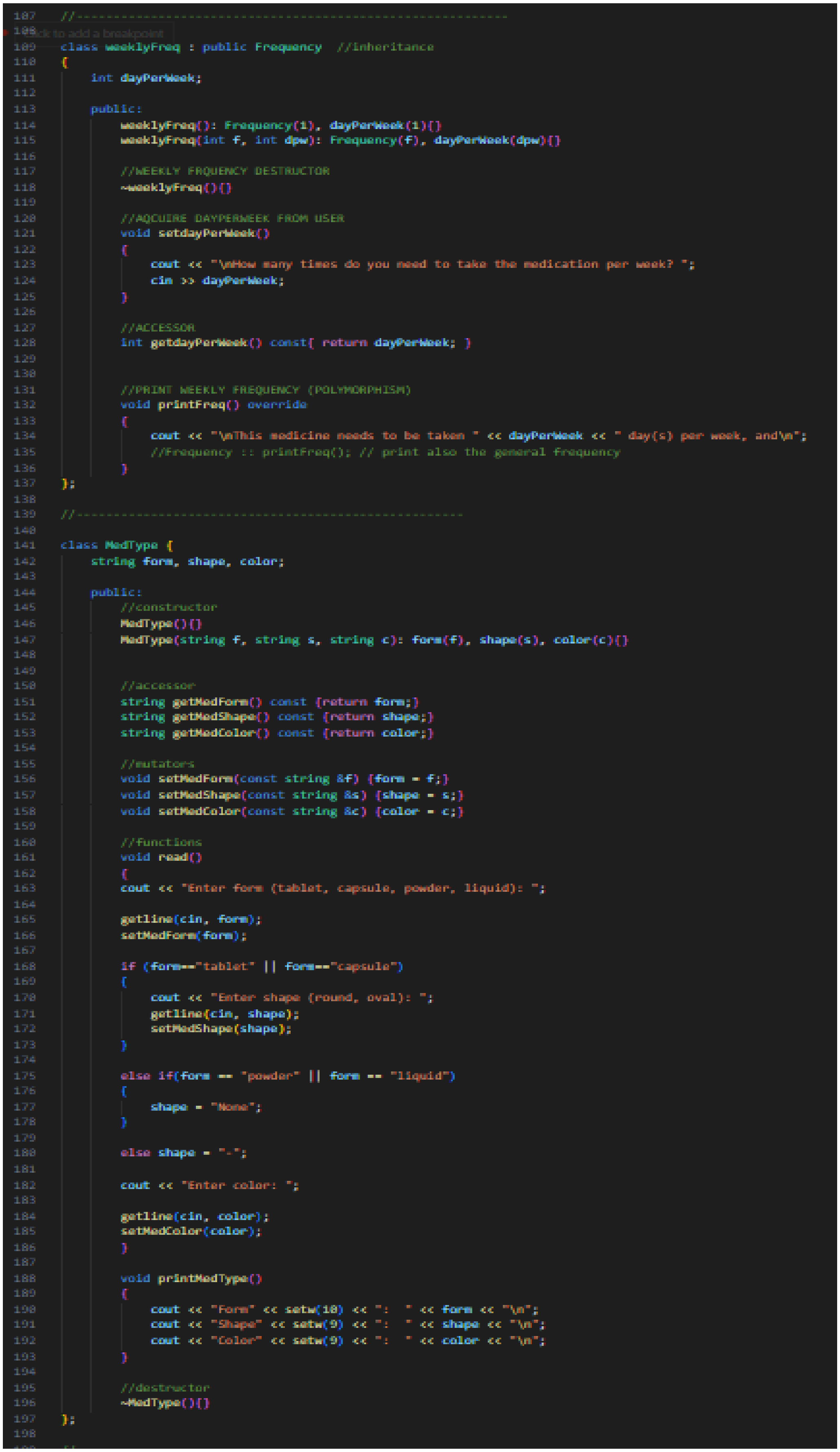
3.0 SECTION C : CODES



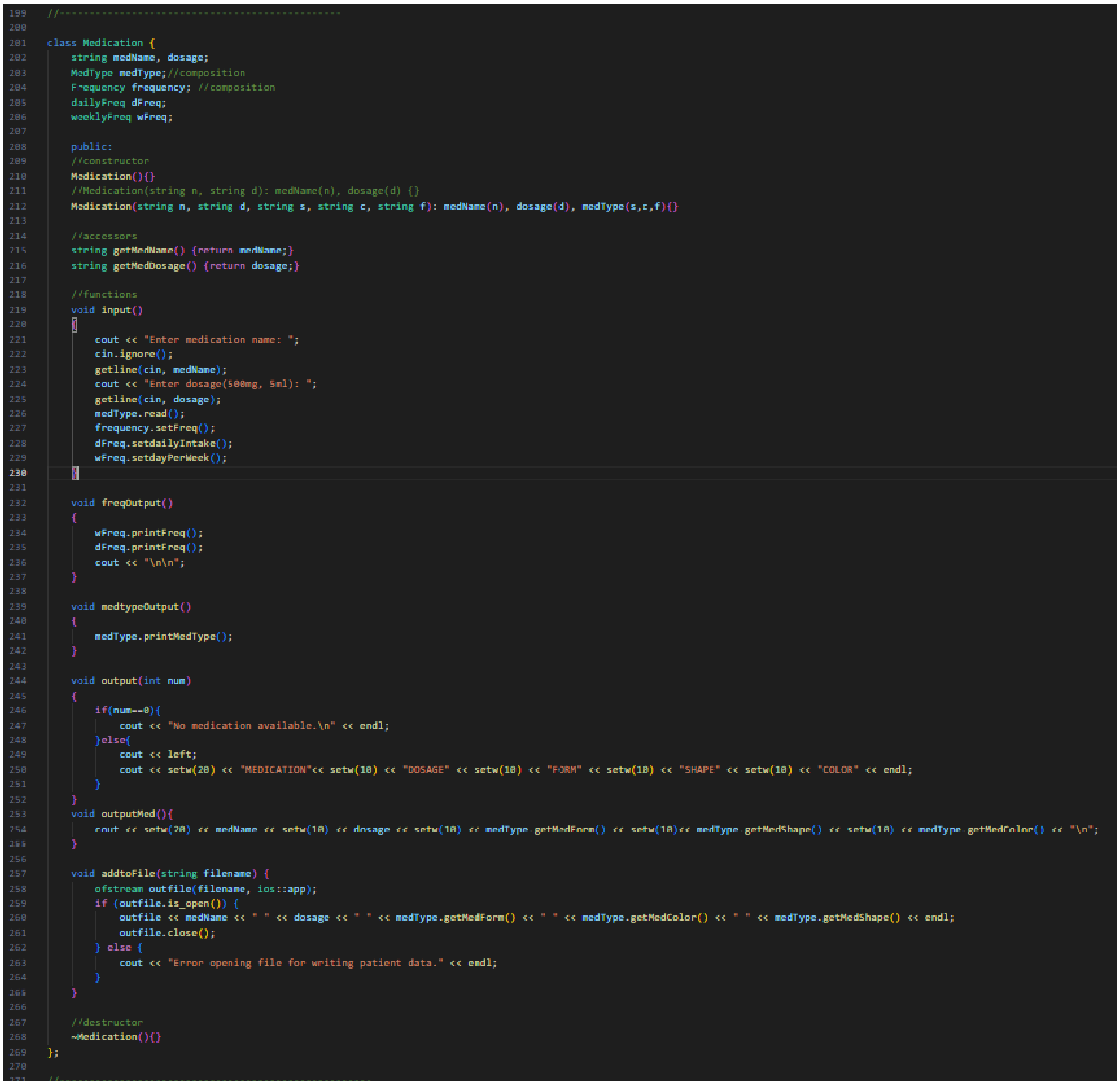
```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <ctime>
#include <exception>
#include <fstream>
#include <vector>
#include <conio.h>

using namespace std;

class Frequency
{
    // so that child class have access
    protected:
        int freqVal;

    public:
        Frequency() : freqVal(1){}
        Frequency(int freqVal):freqVal(freqVal){}

        // MUTATOR
        void setFreq()
        {
            cout << "\nNumber of DOSE(S) you need to take at one time : ";
            cin >> freqVal;
        }

        // ACCESSOR
        int getFreq() const { return freqVal; }

        //POLYMORPHISM
        // default print from parent class
        virtual void printFreq()
        {
            cout << "Frequency : " << freqVal << " each time\n";
        }

        // Destructor
        ~Frequency(){}

};
```



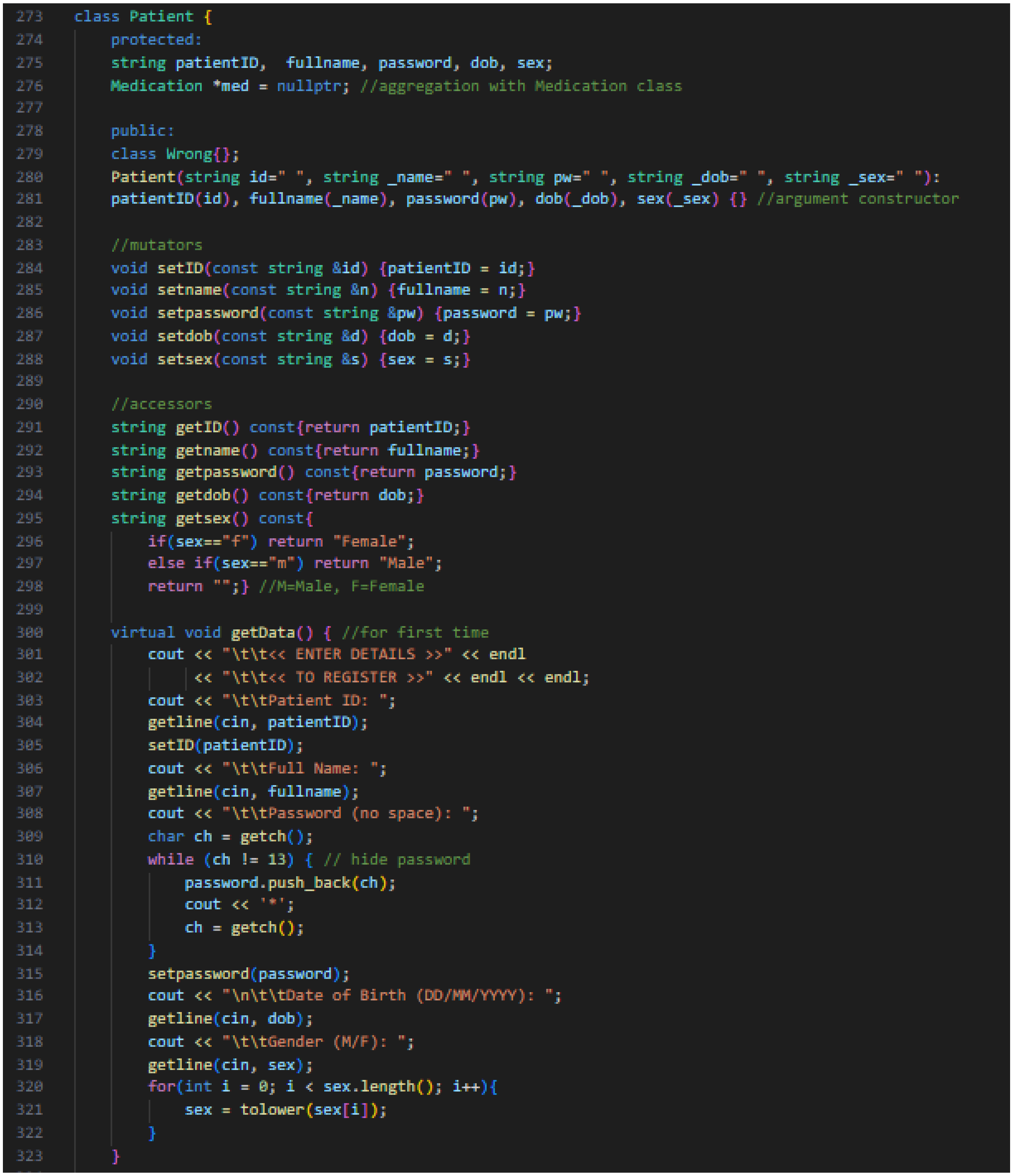
```cpp

//------------------------------

class dailyFreq : public Frequency
{

    int dailyIntake;
    double time[10];

    public:
        dailyFreq(): Frequency(1), dailyIntake(1), time() {}
        dailyFreq(int f, int d, double t): Frequency(f), dailyIntake(d)
        {
            if(d > 1)
            {
                for(int i = 0; i < d; i++)
                {
                    time[i] = t;
                }
            }
        }


        //DAILY FREQUENCY DESTRUCTOR
        ~dailyFreq(){}


        //AQCUIRE DAILY INTAKE FROM USER
        void setdailyIntake()
        {

        // setting daily intake
            cout << "\nHow many TIMES do you need to take the the medicine in a day? ";
            cin >> dailyIntake;

        // setting time for user
            for(int i = 0; i < dailyIntake; i++)
            {
            cout << "\nWhat's the time #" << i+1 << " you need to take the medication in a day?\n";
            cout << "24hrs system (HH:MM) : ";
            cin >> time[i];
            }
        }

        //ACCESSORS
        int getdailyIntake() const { return dailyIntake; }
        double getTime(int i) const{ return time[i]; }


        //PRINT DAILY FREQUENCY (POLYMORPHISM)
        void printFreq() override
        {
            cout << fixed << setprecision(2);
            cout << "\nYou need to take " << dailyIntake << " per day.\n";
            cout << "Time: " ;
            for(int i = 0; i < dailyIntake; i++)
            {
                cout << time[i] << "\n" << setw(11) << endl;
            }
            Frequency :: printFreq();
        }

};
```



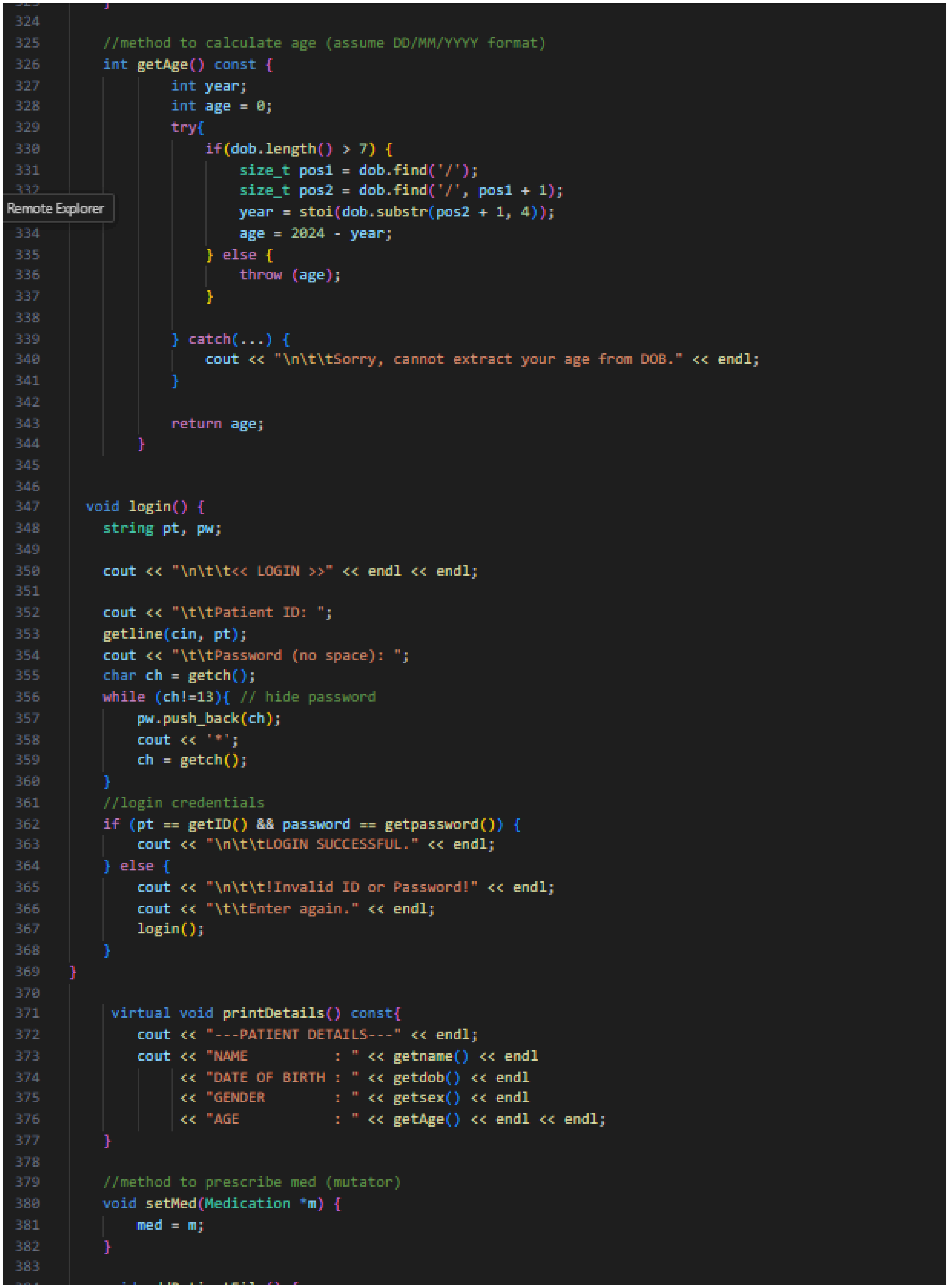
```cpp
//----------------------------------------------------------

class weeklyFreq : public Frequency  //inheritance
{
    int dayPerWeek;

    public:
        weeklyFreq(): Frequency(1), dayPerWeek(1){}
        weeklyFreq(int f, int dpw): Frequency(f), dayPerWeek(dpw){}

        //WEEKLY FRQUENCY DESTRUCTOR
        ~weeklyFreq(){}

        //AQCUIRE DAYPERWEEK FROM USER
        void setdayPerWeek()
        {
            cout << "\nHow many times do you need to take the medication per week? ";
            cin >> dayPerWeek;
        }

        //ACCESSOR
        int getdayPerWeek() const{ return dayPerWeek; }


        //PRINT WEEKLY FREQUENCY (POLYMORPHISM)
        void printFreq() override
        {
            cout << "\nThis medicine needs to be taken " << dayPerWeek << " day(s) per week, and\n";
            //Frequency :: printFreq(); // print also the general frequency
        }
};

//----------------------------------------------------

class MedType {
    string form, shape, color;

    public:
        //constructor
        MedType(){}
        MedType(string f, string s, string c): form(f), shape(s), color(c){}


        //accessor
        string getMedForm() const {return form;}
        string getMedShape() const {return shape;}
        string getMedColor() const {return color;}

        //mutators
        void setMedForm(const string &f) {form = f;}
        void setMedShape(const string &s) {shape = s;}
        void setMedColor(const string &c) {color = c;}

        //functions
        void read()
        {
        cout << "Enter form (tablet, capsule, powder, liquid): ";

        getline(cin, form);
        setMedForm(form);

        if (form=="tablet" || form=="capsule")
        {
            cout << "Enter shape (round, oval): ";
            getline(cin, shape);
            setMedShape(shape);
        }

        else if(form == "powder" || form == "liquid")
        {
            shape = "None";
        }

        else shape = "-";

        cout << "Enter color: ";

        getline(cin, color);
        setMedColor(color);
        }

        void printMedType()
        {
            cout << "Form" << setw(10) << ":  " << form << "\n";
            cout << "Shape" << setw(9) << ":  " << shape << "\n";
            cout << "Color" << setw(9) << ":  " << color << "\n";
        }

        //destructor
        ~MedType(){}
};

//
```



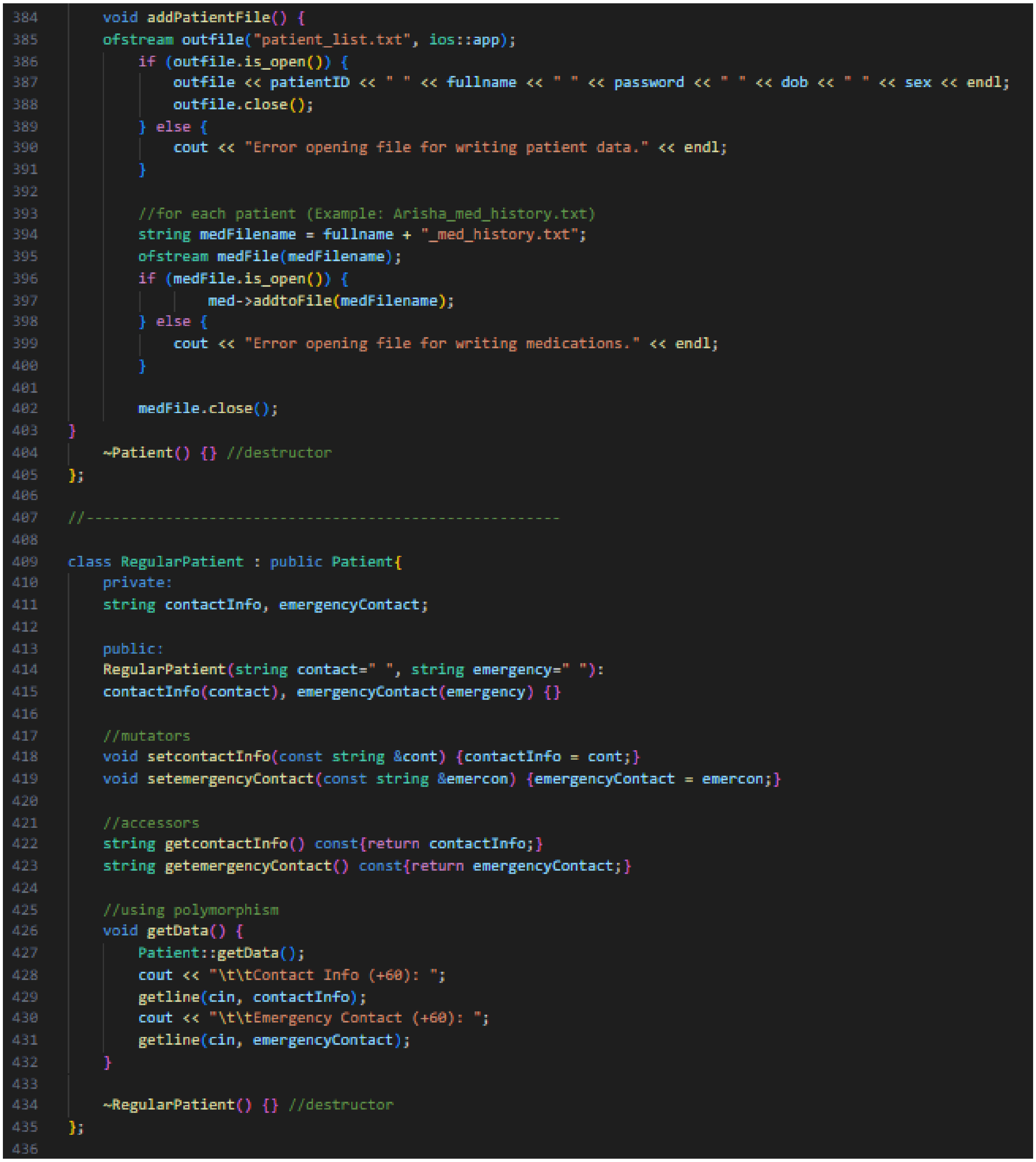
```cpp
//---------------------------------------------

class Medication {
    string medName, dosage;
    MedType medType;//composition
    Frequency frequency; //composition
    dailyFreq dFreq;
    weeklyFreq wFreq;

    public:
    //constructor
    Medication(){}
    //Medication(string n, string d): medName(n), dosage(d) {}
    Medication(string n, string d, string s, string c, string f): medName(n), dosage(d), medType(s,c,f){}

    //accessors
    string getMedName() {return medName;}
    string getMedDosage() {return dosage;}

    //functions
    void input()
    {
        cout << "Enter medication name: ";
        cin.ignore();
        getline(cin, medName);
        cout << "Enter dosage(500mg, 5ml): ";
        getline(cin, dosage);
        medType.read();
        frequency.setFreq();
        dFreq.setdailyIntake();
        wFreq.setdayPerWeek();
    }

    void freqOutput()
    {
        wFreq.printFreq();
        dFreq.printFreq();
        cout << "\n\n";
    }

    void medtypeOutput()
    {
        medType.printMedType();
    }

    void output(int num)
    {
        if(num==0){
            cout << "No medication available.\n" << endl;
        }else{
            cout << left;
            cout << setw(20) << "MEDICATION"<< setw(10) << "DOSAGE" << setw(10) << "FORM" << setw(10) << "SHAPE" << setw(10) << "COLOR" << endl;
        }
    }
    void outputMed(){
        cout << setw(20) << medName << setw(10) << dosage << setw(10) << medType.getMedForm() << setw(10)<< medType.getMedShape() << setw(10) << medType.getMedColor() << "\n";
    }

    void addtoFile(string filename) {
        ofstream outfile(filename, ios::app);
        if (outfile.is_open()) {
            outfile << medName << " " << dosage << " " << medType.getMedForm() << " " << medType.getMedColor() << " " << medType.getMedShape() << endl;
            outfile.close();
        } else {
            cout << "Error opening file for writing patient data." << endl;
        }
    }

    //destructor
    ~Medication(){}
};

//-----------------------------------------------------
```



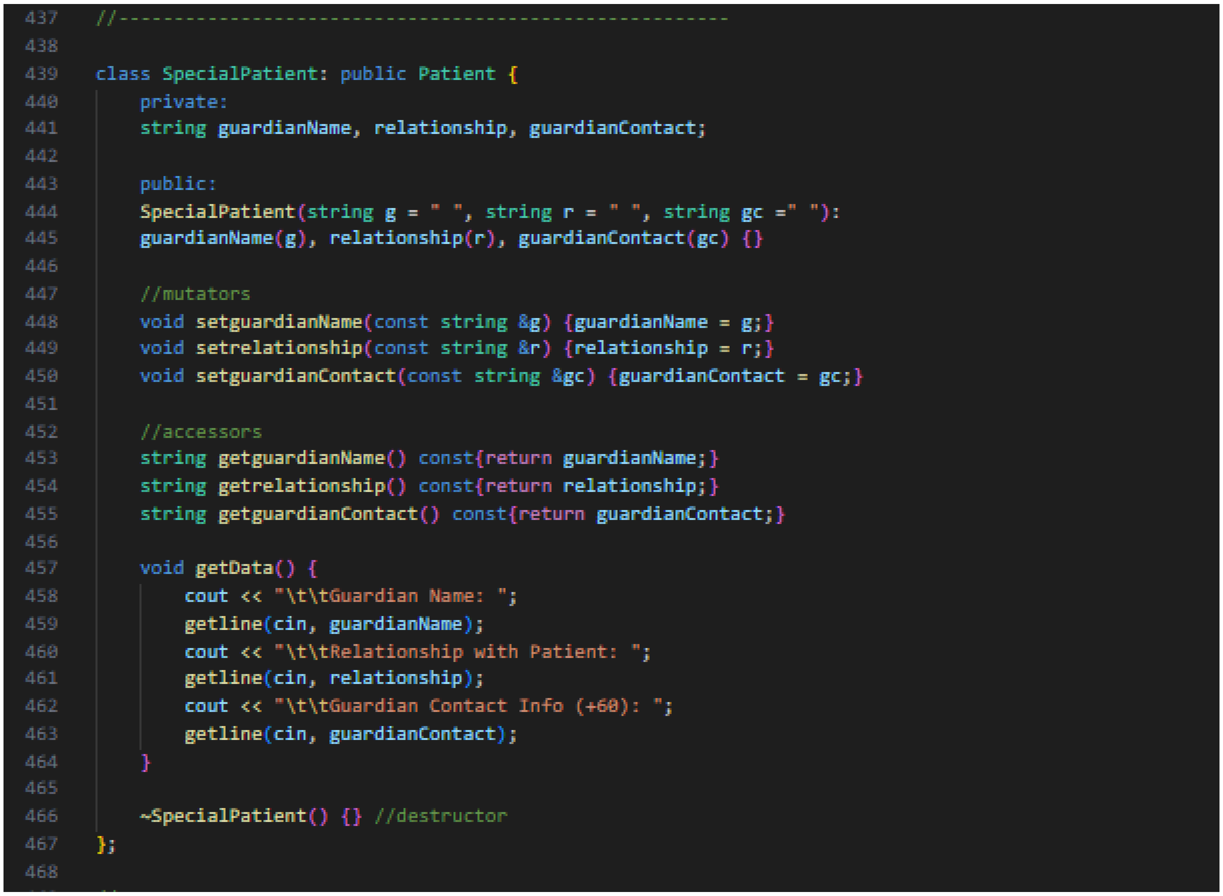
```cpp
class Patient {
    protected:
    string patientID,  fullname, password, dob, sex;
    Medication *med = nullptr; //aggregation with Medication class

    public:
    class Wrong{};
    Patient(string id=" ", string _name=" ", string pw=" ", string _dob=" ", string _sex=" "):
    patientID(id), fullname(_name), password(pw), dob(_dob), sex(_sex) {} //argument constructor

    //mutators
    void setID(const string &id) {patientID = id;}
    void setname(const string &n) {fullname = n;}
    void setpassword(const string &pw) {password = pw;}
    void setdob(const string &d) {dob = d;}
    void setsex(const string &s) {sex = s;}

    //accessors
    string getID() const{return patientID;}
    string getname() const{return fullname;}
    string getpassword() const{return password;}
    string getdob() const{return dob;}
    string getsex() const{
        if(sex=="f") return "Female";
        else if(sex=="m") return "Male";
        return "";} //M=Male, F=Female

    virtual void getData() { //for first time
        cout << "\t\t<< ENTER DETAILS >>" << endl
             << "\t\t<< TO REGISTER >>" << endl << endl;
        cout << "\t\tPatient ID: ";
        getline(cin, patientID);
        setID(patientID);
        cout << "\t\tFull Name: ";
        getline(cin, fullname);
        cout << "\t\tPassword (no space): ";
        char ch = getch();
        while (ch != 13) { // hide password
            password.push_back(ch);
            cout << '*';
            ch = getch();
        }
        setpassword(password);
        cout << "\n\t\tDate of Birth (DD/MM/YYYY): ";
        getline(cin, dob);
        cout << "\t\tGender (M/F): ";
        getline(cin, sex);
        for(int i = 0; i < sex.length(); i++){
            sex = tolower(sex[i]);
        }
    }
```



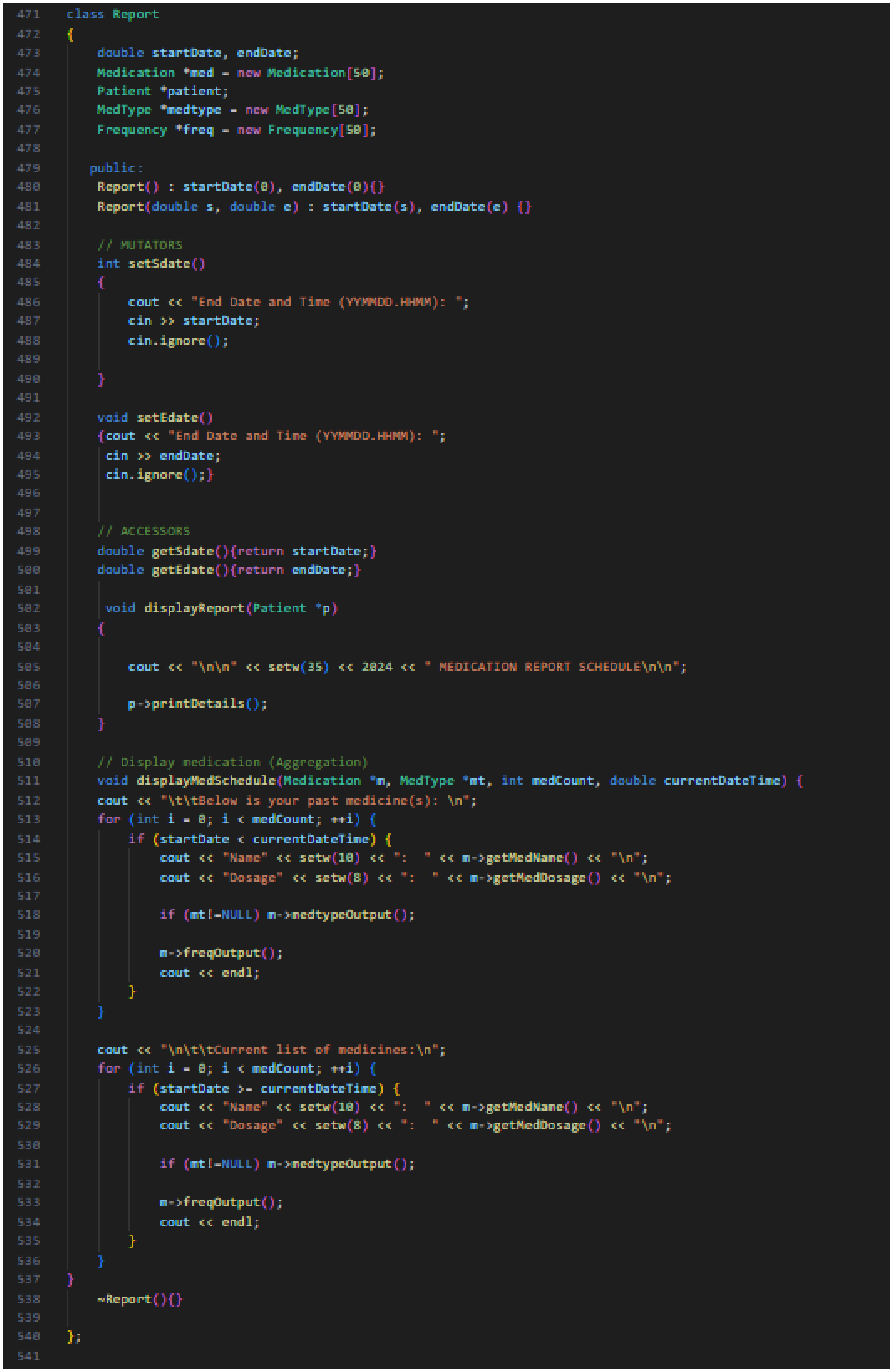
```cpp
    }

    //method to calculate age (assume DD/MM/YYYY format)
    int getAge() const {
            int year;
            int age = 0;
            try{
                if(dob.length() > 7) {
                    size_t pos1 = dob.find('/');
                    size_t pos2 = dob.find('/', pos1 + 1);
                    year = stoi(dob.substr(pos2 + 1, 4));
                    age = 2024 - year;
                } else {
                    throw (age);
                }

            } catch(...) {
                cout << "\n\t\tSorry, cannot extract your age from DOB." << endl;
            }

            return age;
        }


  void login() {
    string pt, pw;

    cout << "\n\t\t<< LOGIN >>" << endl << endl;

    cout << "\t\tPatient ID: ";
    getline(cin, pt);
    cout << "\t\tPassword (no space): ";
    char ch = getch();
    while (ch!=13){ // hide password
        pw.push_back(ch);
        cout << '*';
        ch = getch();
    }
    //login credentials
    if (pt == getID() && password == getpassword()) {
        cout << "\n\t\tLOGIN SUCCESSFUL." << endl;
    } else {
        cout << "\n\t\t!Invalid ID or Password!" << endl;
        cout << "\t\tEnter again." << endl;
        login();
    }
}

     virtual void printDetails() const{
        cout << "---PATIENT DETAILS---" << endl;
        cout << "NAME          : " << getname() << endl
             << "DATE OF BIRTH : " << getdob() << endl
             << "GENDER        : " << getsex() << endl
             << "AGE           : " << getAge() << endl << endl;
    }

    //method to prescribe med (mutator)
    void setMed(Medication *m) {
        med = m;
    }
```



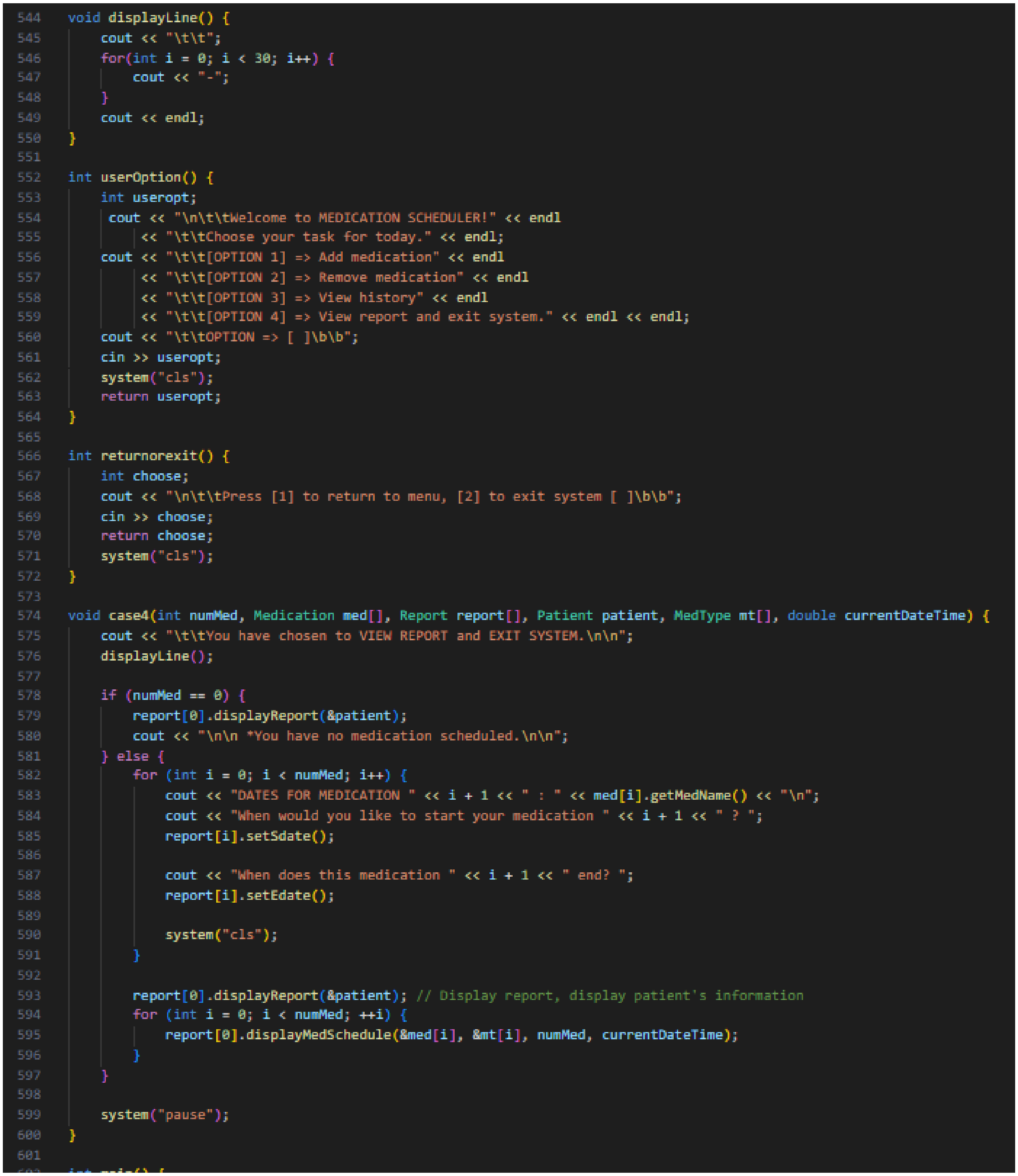
```
    void addPatientFile() {
    ofstream outfile("patient_list.txt", ios::app);
        if (outfile.is_open()) {
            outfile << patientID << " " << fullname << " " << password << " " << dob << " " << sex << endl;
            outfile.close();
        } else {
            cout << "Error opening file for writing patient data." << endl;
        }

        //for each patient (Example: Arisha_med_history.txt)
        string medFilename = fullname + "_med_history.txt";
        ofstream medFile(medFilename);
        if (medFile.is_open()) {
                med->addtoFile(medFilename);
        } else {
            cout << "Error opening file for writing medications." << endl;
        }

        medFile.close();
}
    ~Patient() {} //destructor
};

//-------------------------------------------------------

class RegularPatient : public Patient{
    private:
    string contactInfo, emergencyContact;

    public:
    RegularPatient(string contact=" ", string emergency=" "):
    contactInfo(contact), emergencyContact(emergency) {}

    //mutators
    void setcontactInfo(const string &cont) {contactInfo = cont;}
    void setemergencyContact(const string &emercon) {emergencyContact = emercon;}

    //accessors
    string getcontactInfo() const{return contactInfo;}
    string getemergencyContact() const{return emergencyContact;}

    //using polymorphism
    void getData() {
        Patient::getData();
        cout << "\t\tContact Info (+60): ";
        getline(cin, contactInfo);
        cout << "\t\tEmergency Contact (+60): ";
        getline(cin, emergencyContact);
    }

    ~RegularPatient() {} //destructor
};
```



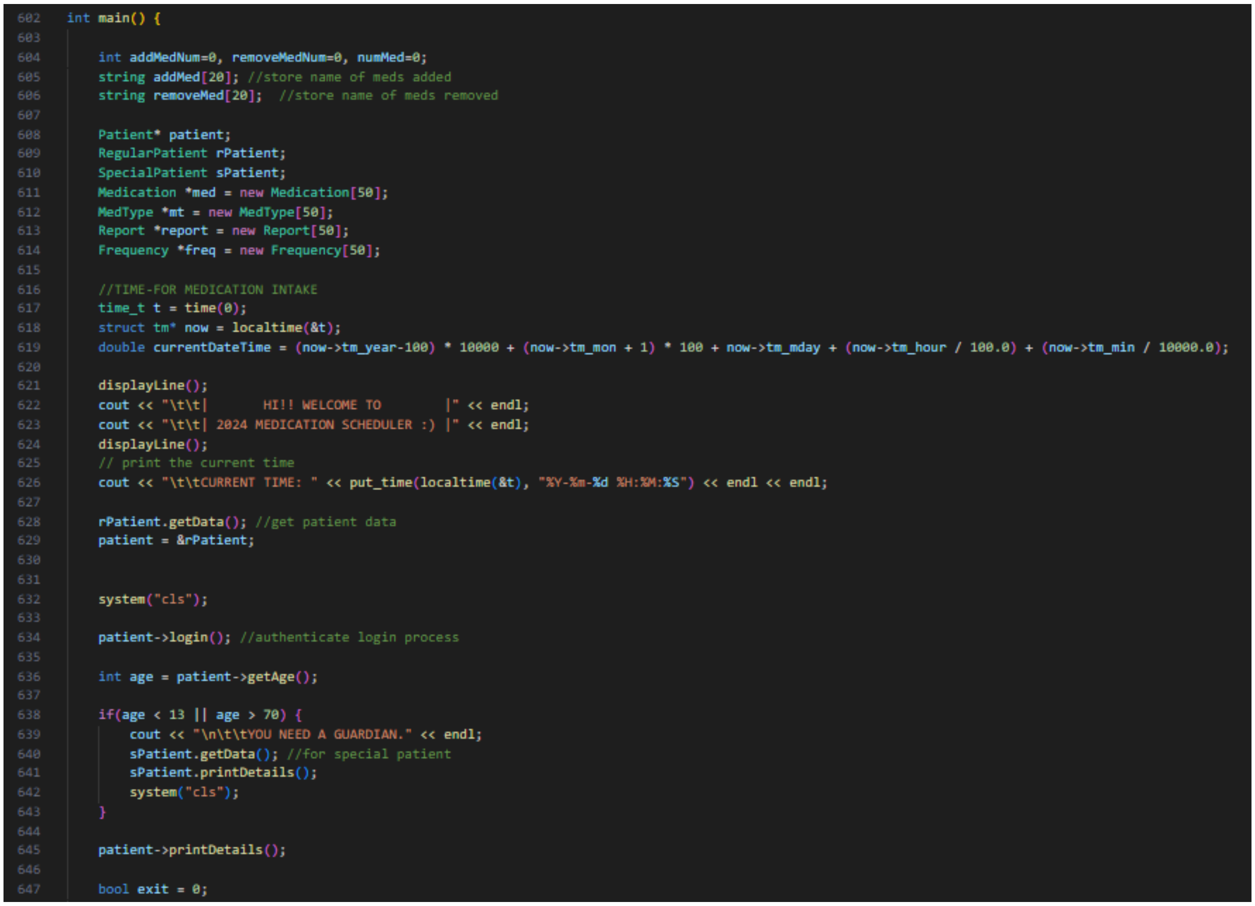
```cpp
//-------------------------------------------------------

class SpecialPatient: public Patient {
    private:
    string guardianName, relationship, guardianContact;

    public:
    SpecialPatient(string g = " ", string r = " ", string gc =" "):
    guardianName(g), relationship(r), guardianContact(gc) {}

    //mutators
    void setguardianName(const string &g) {guardianName = g;}
    void setrelationship(const string &r) {relationship = r;}
    void setguardianContact(const string &gc) {guardianContact = gc;}

    //accessors
    string getguardianName() const{return guardianName;}
    string getrelationship() const{return relationship;}
    string getguardianContact() const{return guardianContact;}

    void getData() {
        cout << "\t\tGuardian Name: ";
        getline(cin, guardianName);
        cout << "\t\tRelationship with Patient: ";
        getline(cin, relationship);
        cout << "\t\tGuardian Contact Info (+60): ";
        getline(cin, guardianContact);
    }

    ~SpecialPatient() {} //destructor
};
```



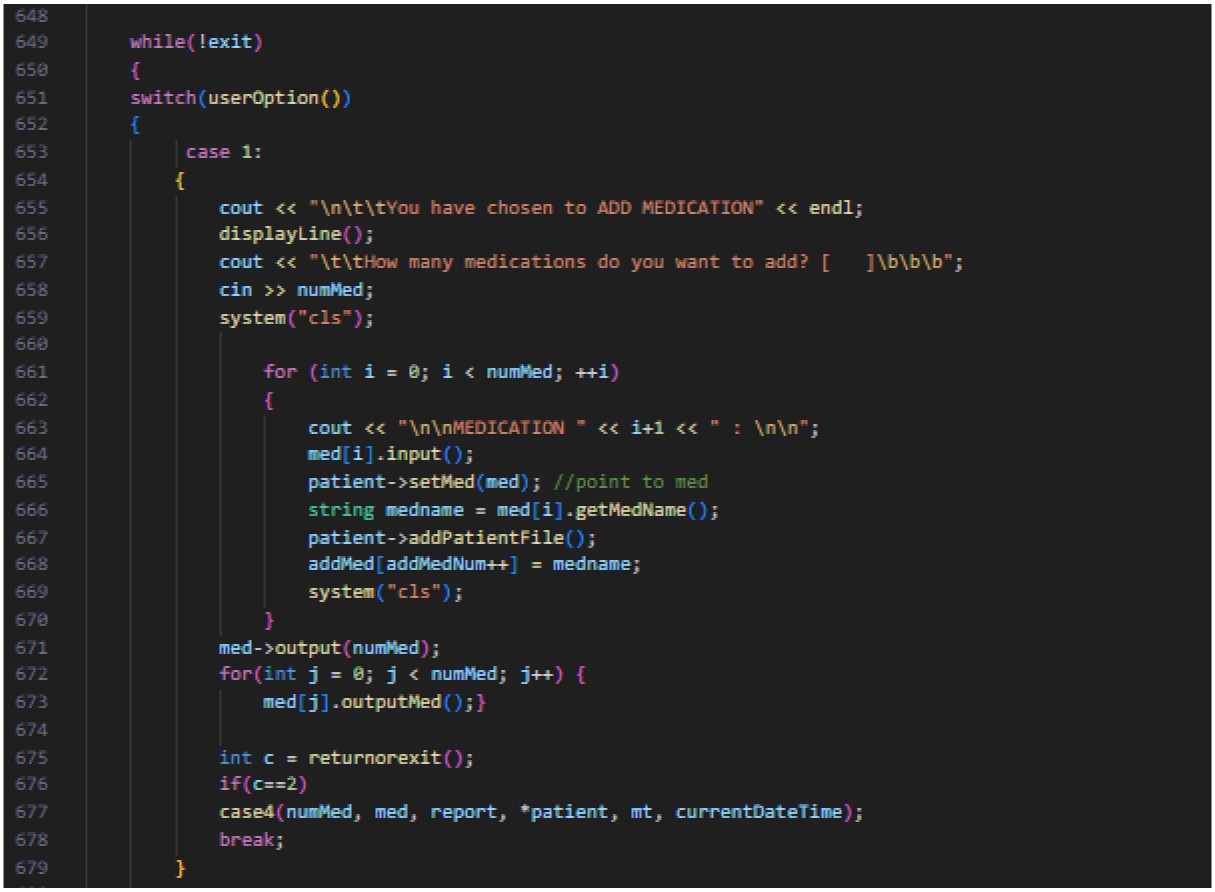
```cpp
class Report
{
    double startDate, endDate;
    Medication *med = new Medication[50];
    Patient *patient;
    MedType *medtype = new MedType[50];
    Frequency *freq = new Frequency[50];

   public:
    Report() : startDate(0), endDate(0){}
    Report(double s, double e) : startDate(s), endDate(e) {}

    // MUTATORS
    int setSdate()
    {
        cout << "End Date and Time (YYMMDD.HHMM): ";
        cin >> startDate;
        cin.ignore();

    }

    void setEdate()
    {cout << "End Date and Time (YYMMDD.HHMM): ";
     cin >> endDate;
     cin.ignore();}


    // ACCESSORS
    double getSdate(){return startDate;}
    double getEdate(){return endDate;}

     void displayReport(Patient *p)
    {

        cout << "\n\n" << setw(35) << 2024 << " MEDICATION REPORT SCHEDULE\n\n";

        p->printDetails();
    }

    // Display medication (Aggregation)
    void displayMedSchedule(Medication *m, MedType *mt, int medCount, double currentDateTime) {
    cout << "\t\tBelow is your past medicine(s): \n";
    for (int i = 0; i < medCount; ++i) {
        if (startDate < currentDateTime) {
            cout << "Name" << setw(10) << ":  " << m->getMedName() << "\n";
            cout << "Dosage" << setw(8) << ":  " << m->getMedDosage() << "\n";

            if (mt!=NULL) m->medtypeOutput();

            m->freqOutput();
            cout << endl;
        }
    }

    cout << "\n\t\tCurrent list of medicines:\n";
    for (int i = 0; i < medCount; ++i) {
        if (startDate >= currentDateTime) {
            cout << "Name" << setw(10) << ":  " << m->getMedName() << "\n";
            cout << "Dosage" << setw(8) << ":  " << m->getMedDosage() << "\n";

            if (mt!=NULL) m->medtypeOutput();

            m->freqOutput();
            cout << endl;
        }
    }
}
    ~Report(){}

};
```



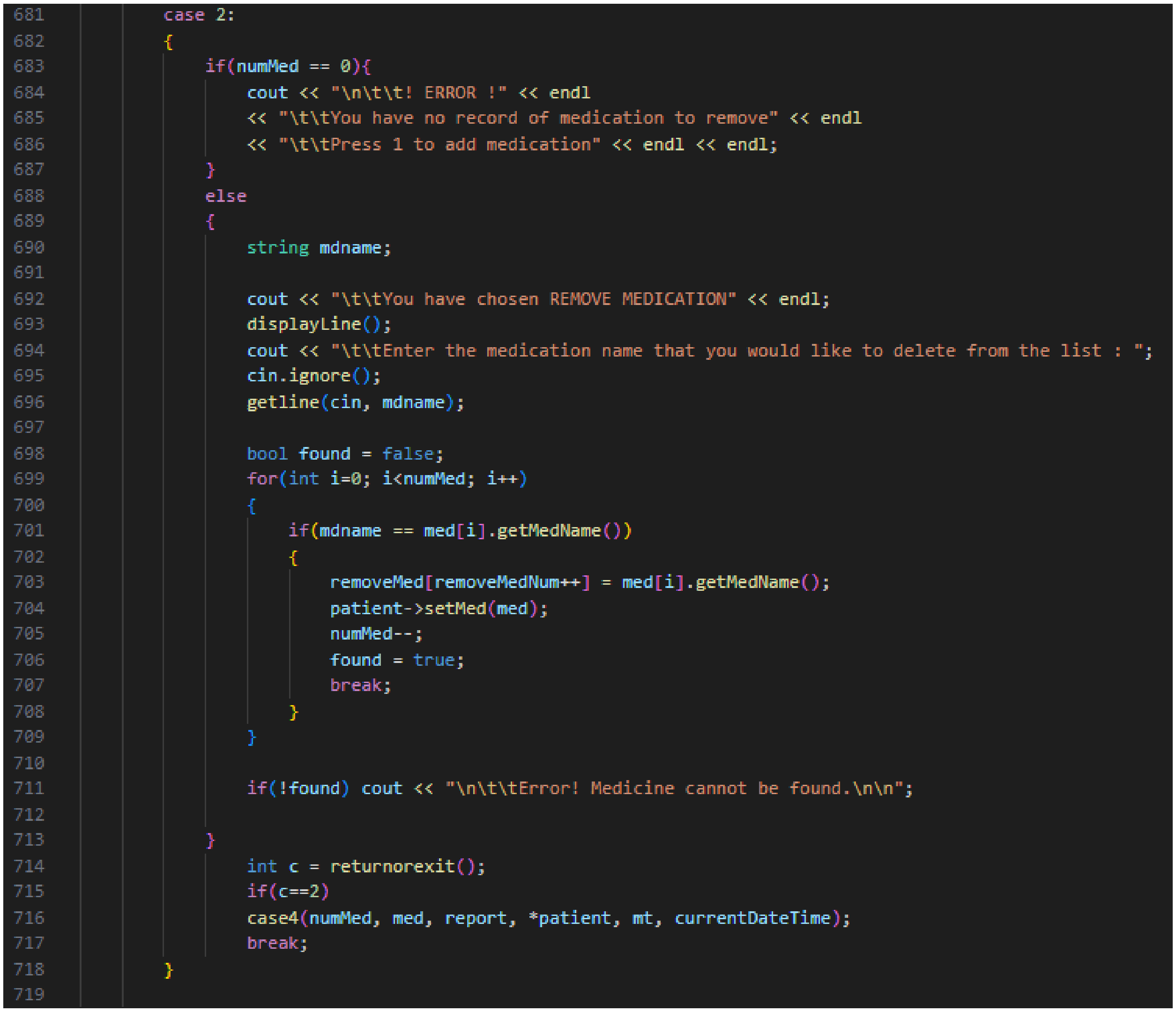
```cpp
void displayLine() {
    cout << "\t\t";
    for(int i = 0; i < 30; i++) {
        cout << "-";
    }
    cout << endl;
}

int userOption() {
    int useropt;
     cout << "\n\t\tWelcome to MEDICATION SCHEDULER!" << endl
         << "\t\tChoose your task for today." << endl;
    cout << "\t\t[OPTION 1] => Add medication" << endl
         << "\t\t[OPTION 2] => Remove medication" << endl
         << "\t\t[OPTION 3] => View history" << endl
         << "\t\t[OPTION 4] => View report and exit system." << endl << endl;
    cout << "\t\tOPTION => [ ]\b\b";
    cin >> useropt;
    system("cls");
    return useropt;
}

int returnorexit() {
    int choose;
    cout << "\n\t\tPress [1] to return to menu, [2] to exit system [ ]\b\b";
    cin >> choose;
    return choose;
    system("cls");
}

void case4(int numMed, Medication med[], Report report[], Patient patient, MedType mt[], double currentDateTime) {
    cout << "\t\tYou have chosen to VIEW REPORT and EXIT SYSTEM.\n\n";
    displayLine();

    if (numMed == 0) {
        report[0].displayReport(&patient);
        cout << "\n\n *You have no medication scheduled.\n\n";
    } else {
        for (int i = 0; i < numMed; i++) {
            cout << "DATES FOR MEDICATION " << i + 1 << " : " << med[i].getMedName() << "\n";
            cout << "When would you like to start your medication " << i + 1 << " ? ";
            report[i].setSdate();

            cout << "When does this medication " << i + 1 << " end? ";
            report[i].setEdate();

            system("cls");
        }

        report[0].displayReport(&patient); // Display report, display patient's information
        for (int i = 0; i < numMed; ++i) {
            report[0].displayMedSchedule(&med[i], &mt[i], numMed, currentDateTime);
        }
    }

    system("pause");
}

int main() {
```



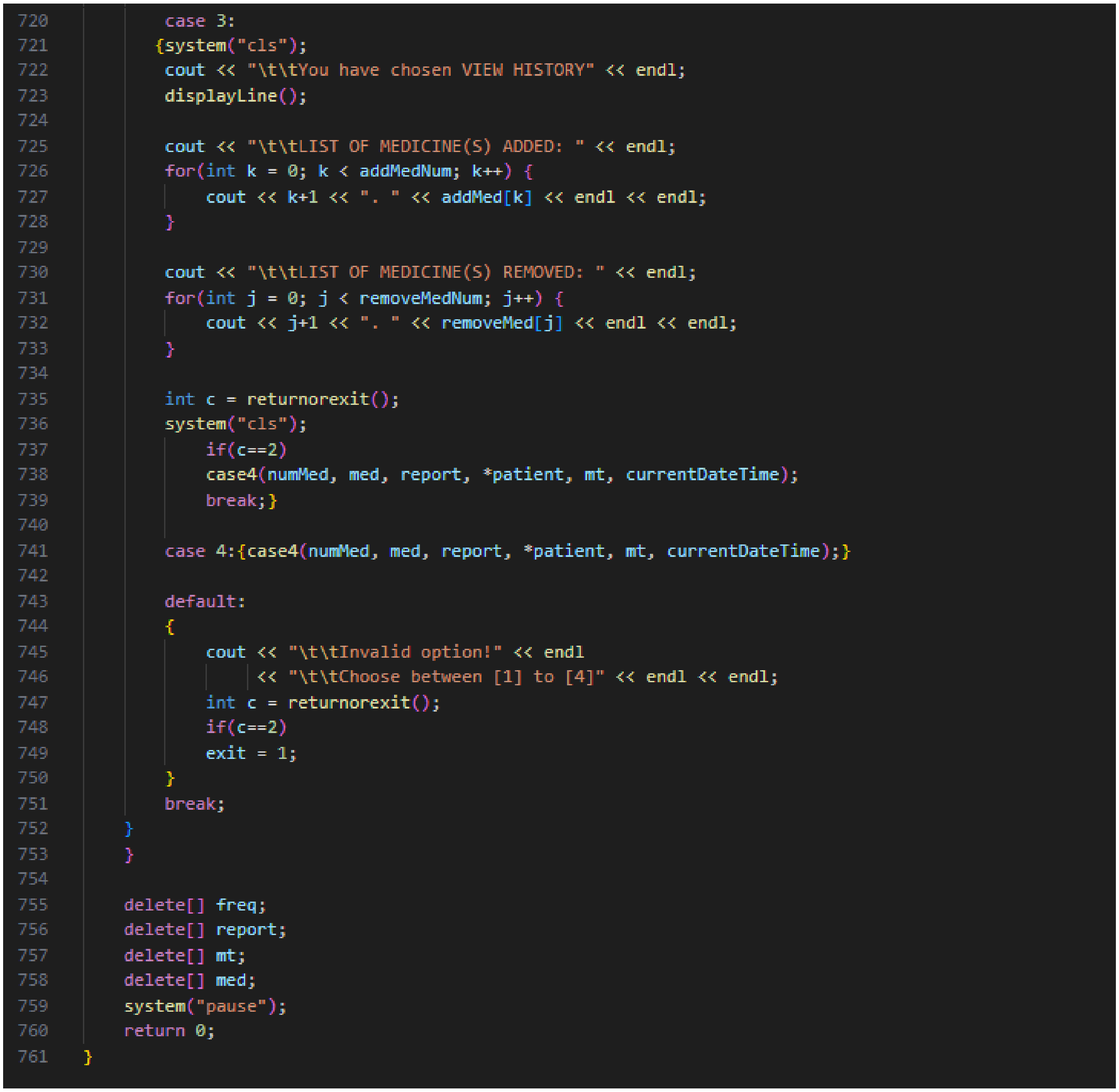
```cpp
int main() {

    int addMedNum=0, removeMedNum=0, numMed=0;
    string addMed[20]; //store name of meds added
    string removeMed[20];  //store name of meds removed

    Patient* patient;
    RegularPatient rPatient;
    SpecialPatient sPatient;
    Medication *med = new Medication[50];
    MedType *mt = new MedType[50];
    Report *report = new Report[50];
    Frequency *freq = new Frequency[50];

    //TIME-FOR MEDICATION INTAKE
    time_t t = time(0);
    struct tm* now = localtime(&t);
    double currentDateTime = (now->tm_year-100) * 10000 + (now->tm_mon + 1) * 100 + now->tm_mday + (now->tm_hour / 100.0) + (now->tm_min / 10000.0);

    displayLine();
    cout << "\t\t|       HI!! WELCOME TO        |" << endl;
    cout << "\t\t| 2024 MEDICATION SCHEDULER :) |" << endl;
    displayLine();
    // print the current time
    cout << "\t\tCURRENT TIME: " << put_time(localtime(&t), "%Y-%m-%d %H:%M:%S") << endl << endl;

    rPatient.getData(); //get patient data
    patient = &rPatient;


    system("cls");

    patient->login(); //authenticate login process

    int age = patient->getAge();

    if(age < 13 || age > 70) {
        cout << "\n\t\tYOU NEED A GUARDIAN." << endl;
        sPatient.getData(); //for special patient
        sPatient.printDetails();
        system("cls");
    }

    patient->printDetails();

    bool exit = 0;
```



```
    while(!exit)
    {
    switch(userOption())
    {
         case 1:
        {
            cout << "\n\t\tYou have chosen to ADD MEDICATION" << endl;
            displayLine();
            cout << "\t\tHow many medications do you want to add? [   ]\b\b\b";
            cin >> numMed;
            system("cls");

                for (int i = 0; i < numMed; ++i)
                {
                    cout << "\n\nMEDICATION " << i+1 << " : \n\n";
                    med[i].input();
                    patient->setMed(med); //point to med
                    string medname = med[i].getMedName();
                    patient->addPatientFile();
                    addMed[addMedNum++] = medname;
                    system("cls");
                }
            med->output(numMed);
            for(int j = 0; j < numMed; j++) {
                med[j].outputMed();}

            int c = returnorexit();
            if(c==2)
            case4(numMed, med, report, *patient, mt, currentDateTime);
            break;
        }
```



```
            case 2:
            {
                if(numMed == 0){
                    cout << "\n\t\t! ERROR !" << endl
                    << "\t\tYou have no record of medication to remove" << endl
                    << "\t\tPress 1 to add medication" << endl << endl;
                }
                else
                {
                    string mdname;

                    cout << "\t\tYou have chosen REMOVE MEDICATION" << endl;
                    displayLine();
                    cout << "\t\tEnter the medication name that you would like to delete from the list : ";
                    cin.ignore();
                    getline(cin, mdname);

                    bool found = false;
                    for(int i=0; i<numMed; i++)
                    {
                        if(mdname == med[i].getMedName())
                        {
                            removeMed[removeMedNum++] = med[i].getMedName();
                            patient->setMed(med);
                            numMed--;
                            found = true;
                            break;
                        }
                    }

                    if(!found) cout << "\n\t\tError! Medicine cannot be found.\n\n";

                }
                    int c = returnorexit();
                    if(c==2)
                    case4(numMed, med, report, *patient, mt, currentDateTime);
                    break;
            }

```



```
        case 3:
       {system("cls");
        cout << "\t\tYou have chosen VIEW HISTORY" << endl;
        displayLine();

        cout << "\t\tLIST OF MEDICINE(S) ADDED: " << endl;
        for(int k = 0; k < addMedNum; k++) {
            cout << k+1 << ". " << addMed[k] << endl << endl;
        }

        cout << "\t\tLIST OF MEDICINE(S) REMOVED: " << endl;
        for(int j = 0; j < removeMedNum; j++) {
            cout << j+1 << ". " << removeMed[j] << endl << endl;
        }

        int c = returnorexit();
        system("cls");
            if(c==2)
            case4(numMed, med, report, *patient, mt, currentDateTime);
            break;}

        case 4:{case4(numMed, med, report, *patient, mt, currentDateTime);}

        default:
        {
            cout << "\t\tInvalid option!" << endl
                 << "\t\tChoose between [1] to [4]" << endl << endl;
            int c = returnorexit();
            if(c==2)
            exit = 1;
        }
        break;
    }
    }

    delete[] freq;
    delete[] report;
    delete[] mt;
    delete[] med;
    system("pause");
    return 0;
}
```