



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:

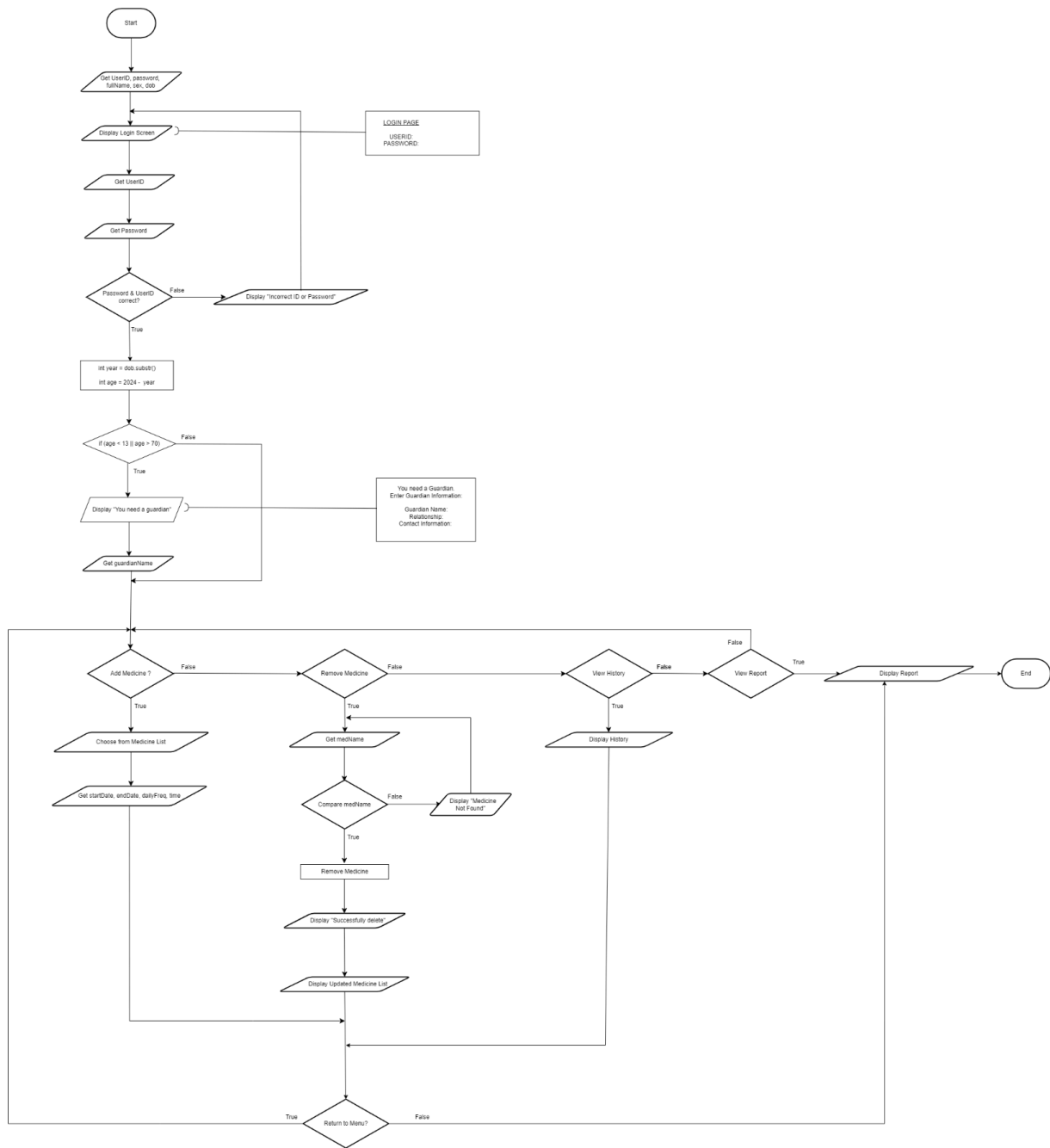
1. Medication scheduling: record prescribed medication details (medication name, shapes, color), shapes and colors are easier way to identify different medication
2. Keep track: dosage, timing, routine (before meal/after meal, daily) and progress (e.g. antibiotics take up to 2 weeks only)
3. Reminder: 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.

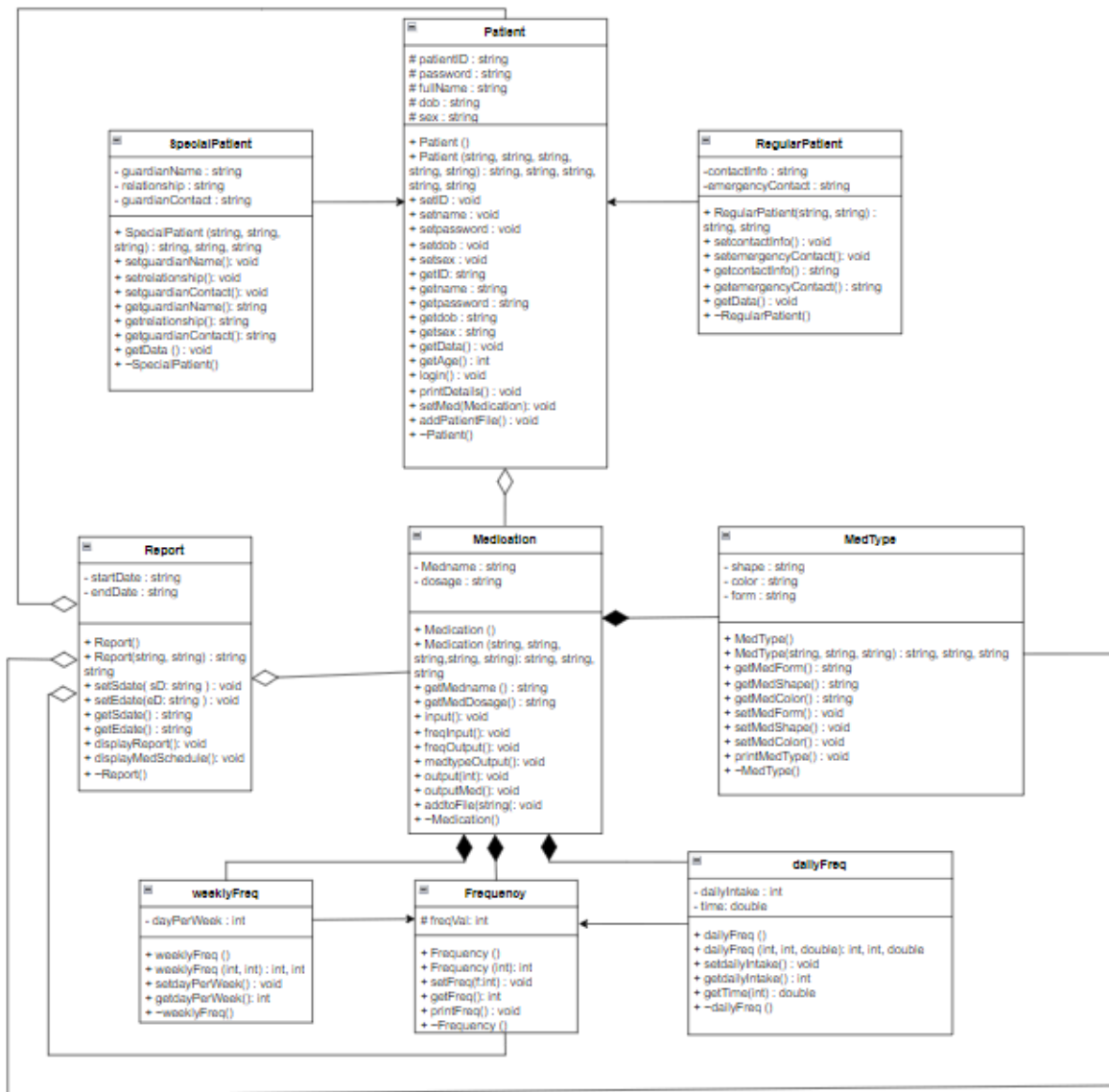
On June 23, 2024, we introduced the Medication Scheduler system to Madam Lizawati, who provided encouraging input and constructive feedback. In response, we have included a number of modifications and improvements to our analytical and design process. The modifications are:

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().

```
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);
    }
};
```

```
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);
}
```

```
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

```
1  #include <iostream>
2  #include <iomanip>
3  #include <string>
4  #include <ctime>
5  #include <exception>
6  #include <fstream>
7  #include <vector>
8  #include <conio.h>
9
10 using namespace std;
11
12 class Frequency
13 {
14     // so that child class have access
15     protected:
16         int freqVal;
17
18     public:
19         Frequency() : freqVal(1){}
20         Frequency(int freqVal):freqVal(freqVal){}
21
22         // MUTATOR
23         void setFreq()
24         {
25             cout << "\nNumber of DOSE(S) you need to take at one time : ";
26             cin >> freqVal;
27         }
28
29         // ACCESSOR
30         int getFreq() const { return freqVal; }
31
32         //POLYMORPHISM
33         // default print from parent class
34         virtual void printFreq()
35         {
36             cout << "Frequency : " << freqVal << " each time\n";
37         }
38
39         // Destructor
40         ~Frequency(){}
41
42     };
```

```

43
44 //-----
45
46 class dailyFreq : public Frequency
47 {
48
49     int dailyIntake;
50     double time[10];
51
52 public:
53     dailyFreq(): Frequency(1), dailyIntake(1), time() {}
54     dailyFreq(int f, int d, double t): Frequency(f), dailyIntake(d)
55     {
56         if(d > 1)
57         {
58             for(int i = 0; i < d; i++)
59             {
60                 time[i] = t;
61             }
62         }
63     }
64
65
66     //DAILY FREQUENCY DESTRUCTOR
67     ~dailyFreq(){}
68
69
70     //ACQUIRE DAILY INTAKE FROM USER
71     void setdailyIntake()
72     {
73
74         // setting daily intake
75         cout << "\nHow many TIMES do you need to take the the medicine in a day? ";
76         cin >> dailyIntake;
77
78         // setting time for user
79         for(int i = 0; i < dailyIntake; i++)
80         {
81             cout << "\nWhat's the time #" << i+1 << " you need to take the medication in a day?\n";
82             cout << "24hrs system (HH:MM) : ";
83             cin >> time[i];
84         }
85     }
86
87     //ACCESSORS
88     int getdailyIntake() const { return dailyIntake; }
89     double getTime(int i) const{ return time[i]; }
90
91
92     //PRINT DAILY FREQUENCY (POLYMORPHISM)
93     void printFreq() override
94     {
95         cout << fixed << setprecision(2);
96         cout << "\nYou need to take " << dailyIntake << " per day.\n";
97         cout << "Time: " ;
98         for(int i = 0; i < dailyIntake; i++)
99         {
100             cout << time[i] << "\n" << setw(11) << endl;
101         }
102         Frequency :: printFreq();
103     }
104
105 };

```

```

107 //-----
108 //Click to add a breakpoint
109 class weeklyFreq : public Frequency //inheritance
110 {
111     int dayPerWeek;
112
113     public:
114         weeklyFreq(): Frequency(1), dayPerWeek(1){}
115         weeklyFreq(int F, int dpw): Frequency(F), dayPerWeek(dpw){}
116
117         //WEEKLY FREQUENCY DESTRUCTOR
118         ~weeklyFreq(){}
119
120         //ACQUIRE DAYPERWEEK FROM USER
121         void setdayPerWeek()
122         {
123             cout << "\nHow many times do you need to take the medication per week? ";
124             cin >> dayPerWeek;
125         }
126
127         //ACCESSOR
128         int getdayPerWeek() const{ return dayPerWeek; }
129
130         //PRINT WEEKLY FREQUENCY (POLYMORPHISM)
131         void printFreq() override
132         {
133             cout << "\nThis medicine needs to be taken " << dayPerWeek << " day(s) per week, and\n";
134             //Frequency :: printFreq(); // print also the general frequency
135         }
136     };
137
138 //-----
139
140 class MedType {
141     string form, shape, color;
142
143     public:
144         //constructor
145         MedType(){}
146         MedType(string f, string s, string c): form(f), shape(s), color(c){}
147
148         //accessor
149         string getMedForm() const {return form;}
150         string getMedShape() const {return shape;}
151         string getMedColor() const {return color;}
152
153         //mutators
154         void setMedForm(const string &f) {form = f;}
155         void setMedShape(const string &s) {shape = s;}
156         void setMedColor(const string &c) {color = c;}
157
158         //functions
159         void read()
160         {
161             cout << "Enter form (tablet, capsule, powder, liquid): ";
162
163             getline(cin, form);
164             setMedForm(form);
165
166             if (form=="tablet" || form=="capsule")
167             {
168                 cout << "Enter shape (round, oval): ";
169                 getline(cin, shape);
170                 setMedShape(shape);
171             }
172
173             else if(form == "powder" || form == "liquid")
174             {
175                 shape = "None";
176             }
177
178             else shape = "-";
179
180             cout << "Enter color: ";
181
182             getline(cin, color);
183             setMedColor(color);
184         }
185
186         void printMedType()
187         {
188             cout << "Form" << setw(18) << ": " << form << "\n";
189             cout << "Shape" << setw(9) << ": " << shape << "\n";
190             cout << "Color" << setw(9) << ": " << color << "\n";
191         }
192
193         //destructor
194         ~MedType(){}
195     };
196
197
198

```

```

199 //*****
200
201 class Medication {
202     string medName, dosage;
203     MedType medType; //composition
204     Frequency frequency; //composition
205     dailyFreq dFreq;
206     weeklyFreq wFreq;
207
208     public:
209     //constructor
210     Medication(){}
211     //Medication(string n, string d): medName(n), dosage(d) {}
212     Medication(string n, string d, string s, string c, string f): medName(n), dosage(d), medType(s,c,f){}
213
214     //accessors
215     string getMedName() {return medName;}
216     string getMedDosage() {return dosage;}
217
218     //functions
219     void input()
220     {
221         cout << "Enter medication name: ";
222         cin.ignore();
223         getline(cin, medName);
224         cout << "Enter dosage(500mg, 5ml): ";
225         getline(cin, dosage);
226         medType.read();
227         frequency.setFreq();
228         dFreq.setdailyIntake();
229         wFreq.setdayPerWeek();
230     }
231
232     void freqOutput()
233     {
234         wFreq.printFreq();
235         dFreq.printFreq();
236         cout << "\n\n";
237     }
238
239     void medtypeOutput()
240     {
241         medType.printMedType();
242     }
243
244     void output(int num)
245     {
246         if(num==0){
247             cout << "No medication available.\n" << endl;
248         }else{
249             cout << left;
250             cout << setw(20) << "MEDICATION"<< setw(10) << "DOSAGE" << setw(10) << "FORM" << setw(10) << "SHAPE" << setw(10) << "COLOR" << endl;
251         }
252     }
253     void outputMed(){
254         cout << setw(20) << medName << setw(10) << dosage << setw(10) << medType.getMedForm() << setw(10) << medType.getMedShape() << setw(10) << medType.getMedColor() << "\n";
255     }
256
257     void addToFile(string filename) {
258         ofstream outfile(filename, ios::app);
259         if (outfile.is_open()) {
260             outfile << medName << " " << dosage << " " << medType.getMedForm() << " " << medType.getMedColor() << " " << medType.getMedShape() << endl;
261             outfile.close();
262         } else {
263             cout << "Error opening file for writing patient data." << endl;
264         }
265     }
266
267     //destructor
268     ~Medication(){}
269 };
270
271 //*****

```

```

273 class Patient {
274     protected:
275         string patientID,  fullname, password, dob, sex;
276         Medication *med = nullptr; //aggregation with Medication class
277
278     public:
279         class Wrong{};
280         Patient(string id=" ", string _name=" ", string pw=" ", string _dob=" ", string _sex=" "):
281             patientID(id), fullname(_name), password(pw), dob(_dob), sex(_sex) {} //argument constructor
282
283         //mutators
284         void setID(const string &id) {patientID = id;}
285         void setname(const string &n) {fullname = n;}
286         void setpassword(const string &pw) {password = pw;}
287         void setdob(const string &d) {dob = d;}
288         void setsex(const string &s) {sex = s;}
289
290         //accessors
291         string getID() const{return patientID;}
292         string getname() const{return fullname;}
293         string getpassword() const{return password;}
294         string getdob() const{return dob;}
295         string getsex() const{
296             if(sex=="f") return "Female";
297             else if(sex=="m") return "Male";
298             return "";} //M=Male, F=Female
299
300         virtual void getData() { //for first time
301             cout << "\t\t<< ENTER DETAILS >>" << endl
302                 << "\t\t<< TO REGISTER >>" << endl << endl;
303             cout << "\t\tPatient ID: ";
304             getline(cin, patientID);
305             setID(patientID);
306             cout << "\t\tFull Name: ";
307             getline(cin, fullname);
308             cout << "\t\tPassword (no space): ";
309             char ch = getch();
310             while (ch != 13) { // hide password
311                 password.push_back(ch);
312                 cout << '*';
313                 ch = getch();
314             }
315             setpassword(password);
316             cout << "\n\t\tDate of Birth (DD/MM/YYYY): ";
317             getline(cin, dob);
318             cout << "\t\tGender (M/F): ";
319             getline(cin, sex);
320             for(int i = 0; i < sex.length(); i++){
321                 sex = tolower(sex[i]);
322             }
323         }

```



```

324
325 //method to calculate age (assume DD/MM/YYYY format)
326 int getAge() const {
327     int year;
328     int age = 0;
329     try{
330         if(dob.length() > 7) {
331             size_t pos1 = dob.find('/');
332             size_t pos2 = dob.find('/', pos1 + 1);
333             year = stoi(dob.substr(pos2 + 1, 4));
334             age = 2024 - year;
335         } else {
336             throw (age);
337         }
338     } catch(...) {
339         cout << "\n\t\tSorry, cannot extract your age from DOB." << endl;
340     }
341
342     return age;
343 }
344
345
346
347 void login() {
348     string pt, pw;
349
350     cout << "\n\t\t<< LOGIN >>" << endl << endl;
351
352     cout << "\t\tPatient ID: ";
353     getline(cin, pt);
354     cout << "\t\tPassword (no space): ";
355     char ch = getch();
356     while (ch!=13){ // hide password
357         pw.push_back(ch);
358         cout << '*';
359         ch = getch();
360     }
361     //login credentials
362     if (pt == getID() && password == getpassword()) {
363         cout << "\n\t\tLOGIN SUCCESSFUL." << endl;
364     } else {
365         cout << "\n\t\tInvalid ID or Password!" << endl;
366         cout << "\t\tEnter again." << endl;
367         login();
368     }
369 }
370
371 virtual void printDetails() const{
372     cout << "---PATIENT DETAILS---" << endl;
373     cout << "NAME          : " << getname() << endl
374     << "DATE OF BIRTH : " << getdob() << endl
375     << "GENDER        : " << getsex() << endl
376     << "AGE           : " << getAge() << endl << endl;
377 }
378
379 //method to prescribe med (mutator)
380 void setMed(Medication *m) {
381     med = m;
382 }
383

```

```

384 void addPatientFile() {
385     ofstream outfile("patient_list.txt", ios::app);
386     if (outfile.is_open()) {
387         outfile << patientID << " " << fullname << " " << password << " " << dob << " " << sex << endl;
388         outfile.close();
389     } else {
390         cout << "Error opening file for writing patient data." << endl;
391     }
392
393     //for each patient (Example: Arisha_med_history.txt)
394     string medFilename = fullname + "_med_history.txt";
395     ofstream medFile(medFilename);
396     if (medFile.is_open()) {
397         med->addToFile(medFilename);
398     } else {
399         cout << "Error opening file for writing medications." << endl;
400     }
401
402     medFile.close();
403 }
404 ~Patient() {} //destructor
405 };
406
407 //-----
408
409 class RegularPatient : public Patient{
410     private:
411         string contactInfo, emergencyContact;
412
413     public:
414         RegularPatient(string contact=" ", string emergency=" "):
415             contactInfo(contact), emergencyContact(emergency) {}
416
417     //mutators
418     void setcontactInfo(const string &cont) {contactInfo = cont;}
419     void setemergencyContact(const string &emercon) {emergencyContact = emercon;}
420
421     //accessors
422     string getcontactInfo() const{return contactInfo;}
423     string getemergencyContact() const{return emergencyContact;}
424
425     //using polymorphism
426     void getData() {
427         Patient::getData();
428         cout << "\t\tContact Info (+60): ";
429         getline(cin, contactInfo);
430         cout << "\t\tEmergency Contact (+60): ";
431         getline(cin, emergencyContact);
432     }
433
434     ~RegularPatient() {} //destructor
435 };
436

```

```

437 //-----
438
439 class SpecialPatient: public Patient {
440     private:
441         string guardianName, relationship, guardianContact;
442
443     public:
444         SpecialPatient(string g = " ", string r = " ", string gc = " "):
445             guardianName(g), relationship(r), guardianContact(gc) {}
446
447         //mutators
448         void setguardianName(const string &g) {guardianName = g;}
449         void setrelationship(const string &r) {relationship = r;}
450         void setguardianContact(const string &gc) {guardianContact = gc;}
451
452         //accessors
453         string getguardianName() const{return guardianName;}
454         string getrelationship() const{return relationship;}
455         string getguardianContact() const{return guardianContact;}
456
457         void getData() {
458             cout << "\t\tGuardian Name: ";
459             getline(cin, guardianName);
460             cout << "\t\tRelationship with Patient: ";
461             getline(cin, relationship);
462             cout << "\t\tGuardian Contact Info (+60): ";
463             getline(cin, guardianContact);
464         }
465
466         ~SpecialPatient() {} //destructor
467     };
468

```

```

471 class Report
472 {
473     double startDate, endDate;
474     Medication *med = new Medication[50];
475     Patient *patient;
476     MedType *medtype = new MedType[50];
477     Frequency *freq = new Frequency[50];
478
479 public:
480     Report() : startDate(0), endDate(0){}
481     Report(double s, double e) : startDate(s), endDate(e) {}
482
483     // MUTATORS
484     int setSdate()
485     {
486         cout << "End Date and Time (YYMMDD.HHMM): ";
487         cin >> startDate;
488         cin.ignore();
489     }
490
491     void setEdate()
492     {cout << "End Date and Time (YYMMDD.HHMM): ";
493     cin >> endDate;
494     cin.ignore();}
495
496     // ACCESSORS
497     double getSdate(){return startDate;}
498     double getEdate(){return endDate;}
499
500     void displayReport(Patient *p)
501     {
502         cout << "\n\n" << setw(35) << 2024 << " MEDICATION REPORT SCHEDULE\n\n";
503         p->printDetails();
504     }
505
506     // Display medication (Aggregation)
507     void displayMedSchedule(Medication *m, MedType *mt, int medCount, double currentDateTime) {
508         cout << "\t\tBelow is your past medicine(s): \n";
509         for (int i = 0; i < medCount; ++i) {
510             if (startDate < currentDateTime) {
511                 cout << "Name" << setw(10) << ": " << m->getMedName() << "\n";
512                 cout << "Dosage" << setw(8) << ": " << m->getMedDosage() << "\n";
513
514                 if (mt!=NULL) m->medtypeOutput();
515
516                 m->freqOutput();
517                 cout << endl;
518             }
519         }
520     }
521
522     cout << "\n\t\tCurrent list of medicines:\n";
523     for (int i = 0; i < medCount; ++i) {
524         if (startDate >= currentDateTime) {
525             cout << "Name" << setw(10) << ": " << m->getMedName() << "\n";
526             cout << "Dosage" << setw(8) << ": " << m->getMedDosage() << "\n";
527
528             if (mt!=NULL) m->medtypeOutput();
529
530             m->freqOutput();
531             cout << endl;
532         }
533     }
534 }
535
536 ~Report(){}
537
538 };
539
540
541

```

```

544 void displayLine() {
545     cout << "\t\t";
546     for(int i = 0; i < 30; i++) {
547         cout << "-";
548     }
549     cout << endl;
550 }
551
552 int userOption() {
553     int useropt;
554     cout << "\n\t\tWelcome to MEDICATION SCHEDULER!" << endl
555         << "\t\tChoose your task for today." << endl;
556     cout << "\t\t[OPTION 1] => Add medication" << endl
557         << "\t\t[OPTION 2] => Remove medication" << endl
558         << "\t\t[OPTION 3] => View history" << endl
559         << "\t\t[OPTION 4] => View report and exit system." << endl << endl;
560     cout << "\t\t[OPTION] => [ ]\b\b";
561     cin >> useropt;
562     system("cls");
563     return useropt;
564 }
565
566 int returnorexit() {
567     int choose;
568     cout << "\n\t\tPress [1] to return to menu, [2] to exit system [ ]\b\b";
569     cin >> choose;
570     return choose;
571     system("cls");
572 }
573
574 void case4(int numMed, Medication med[], Report report[], Patient patient, MedType mt[], double currentDateTime) {
575     cout << "\t\tYou have chosen to VIEW REPORT and EXIT SYSTEM.\n\n";
576     displayLine();
577
578     if (numMed == 0) {
579         report[0].displayReport(&patient);
580         cout << "\n\n *You have no medication scheduled.\n\n";
581     } else {
582         for (int i = 0; i < numMed; i++) {
583             cout << "DATES FOR MEDICATION " << i + 1 << " : " << med[i].getMedName() << "\n";
584             cout << "When would you like to start your medication " << i + 1 << " ? ";
585             report[i].setSdate();
586
587             cout << "When does this medication " << i + 1 << " end? ";
588             report[i].setEdate();
589
590             system("cls");
591         }
592
593         report[0].displayReport(&patient); // Display report, display patient's information
594         for (int i = 0; i < numMed; ++i) {
595             report[0].displayMedSchedule(&med[i], &mt[i], numMed, currentDateTime);
596         }
597     }
598
599     system("pause");
600 }
601
602 int main() {

```

```

602 int main() {
603
604     int addMedNum=0, removeMedNum=0, numMed=0;
605     string addMed[20]; //store name of meds added
606     string removeMed[20]; //store name of meds removed
607
608     Patient* patient;
609     RegularPatient rPatient;
610     SpecialPatient sPatient;
611     Medication *med = new Medication[50];
612     MedType *mt = new MedType[50];
613     Report *report = new Report[50];
614     Frequency *freq = new Frequency[50];
615
616     //TIME-FOR MEDICATION INTAKE
617     time_t t = time(0);
618     struct tm* now = localtime(&t);
619     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);
620
621     displayLine();
622     cout << "\t\t|          HI!! WELCOME TO          |" << endl;
623     cout << "\t\t| 2024 MEDICATION SCHEDULER :) |" << endl;
624     displayLine();
625     // print the current time
626     cout << "\t\tCURRENT TIME: " << put_time(localtime(&t), "%Y-%m-%d %H:%M:%S") << endl << endl;
627
628     rPatient.getData(); //get patient data
629     patient = &rPatient;
630
631
632     system("cls");
633
634     patient->login(); //authenticate login process
635
636     int age = patient->getAge();
637
638     if(age < 13 || age > 70) {
639         cout << "\n\t\tYOU NEED A GUARDIAN." << endl;
640         sPatient.getData(); //for special patient
641         sPatient.printDetails();
642         system("cls");
643     }
644
645     patient->printDetails();
646
647     bool exit = 0;

```

```

648
649 while(!exit)
650 {
651     switch(userOption())
652     {
653         case 1:
654         {
655             cout << "\n\t\tYou have chosen to ADD MEDICATION" << endl;
656             displayLine();
657             cout << "\t\tHow many medications do you want to add? [    ]\b\b\b";
658             cin >> numMed;
659             system("cls");
660
661             for (int i = 0; i < numMed; ++i)
662             {
663                 cout << "\n\nMEDICATION " << i+1 << " : \n\n";
664                 med[i].input();
665                 patient->setMed(med); //point to med
666                 string medname = med[i].getMedName();
667                 patient->addPatientFile();
668                 addMed[addMedNum++] = medname;
669                 system("cls");
670             }
671             med->output(numMed);
672             for(int j = 0; j < numMed; j++) {
673                 med[j].outputMed();}
674
675             int c = returnnorexit();
676             if(c==2)
677                 case4(numMed, med, report, *patient, mt, currentDateTime);
678             break;
679         }

```

```

681     case 2:
682     {
683         if(numMed == 0){
684             cout << "\n\t\t! ERROR !" << endl
685             << "\t\tYou have no record of medication to remove" << endl
686             << "\t\tPress 1 to add medication" << endl << endl;
687         }
688         else
689         {
690             string mdname;
691
692             cout << "\t\tYou have chosen REMOVE MEDICATION" << endl;
693             displayLine();
694             cout << "\t\tEnter the medication name that you would like to delete from the list : ";
695             cin.ignore();
696             getline(cin, mdname);
697
698             bool found = false;
699             for(int i=0; i<numMed; i++)
700             {
701                 if(mdname == med[i].getMedName())
702                 {
703                     removeMed[removeMedNum++] = med[i].getMedName();
704                     patient->setMed(med);
705                     numMed--;
706                     found = true;
707                     break;
708                 }
709             }
710
711             if(!found) cout << "\n\t\tError! Medicine cannot be found.\n\n";
712
713         }
714         int c = returnnorexit();
715         if(c==2)
716             case4(numMed, med, report, *patient, mt, currentDate);
717         break;
718     }
719

```



```

720     case 3:
721     {system("cls");
722     cout << "\t\tYou have chosen VIEW HISTORY" << endl;
723     displayLine();
724
725     cout << "\t\tLIST OF MEDICINE(S) ADDED: " << endl;
726     for(int k = 0; k < addMedNum; k++) {
727     |     cout << k+1 << ". " << addMed[k] << endl << endl;
728     }
729
730     cout << "\t\tLIST OF MEDICINE(S) REMOVED: " << endl;
731     for(int j = 0; j < removeMedNum; j++) {
732     |     cout << j+1 << ". " << removeMed[j] << endl << endl;
733     }
734
735     int c = returnorexit();
736     system("cls");
737     |     if(c==2)
738     |     case4(numMed, med, report, *patient, mt, currentDateTime);
739     |     break;}
740
741     case 4:{case4(numMed, med, report, *patient, mt, currentDateTime);}
742
743     default:
744     {
745     |     cout << "\t\tInvalid option!" << endl
746     |     |     << "\t\tChoose between [1] to [4]" << endl << endl;
747     |     int c = returnorexit();
748     |     if(c==2)
749     |     exit = 1;
750     |     }
751     |     break;
752     }
753     }
754
755     delete[] freq;
756     delete[] report;
757     delete[] mt;
758     delete[] med;
759     system("pause");
760     return 0;
761 }

```