Data Structures and Algorithms

Problem Statement:

Develop a complete working application in C++ as a part of your DSA semester project. This may be an individual effort or a group contribution as per your personal preferences. In case of a group project, the maximum group size is to be two students only. Please ensure that your application should be a modular programming based solution to a problem employing the maximum concepts of Data Structure and Algorithms (like Stacks, Queues, Link List, Trees and Graphs). Use as much as data structure in your project.
You can take 'help' (and not copy+paste) from the Internet and other resources as long as you clearly understand and refer them in your program.
Properly comment your program and use good programming practices (proper indenting, meaningful variable and function names, comments etc.)

Project Description:

Design and develop an application to automate a **Corona Virus Patient Management System**. The detail of the application is as below:

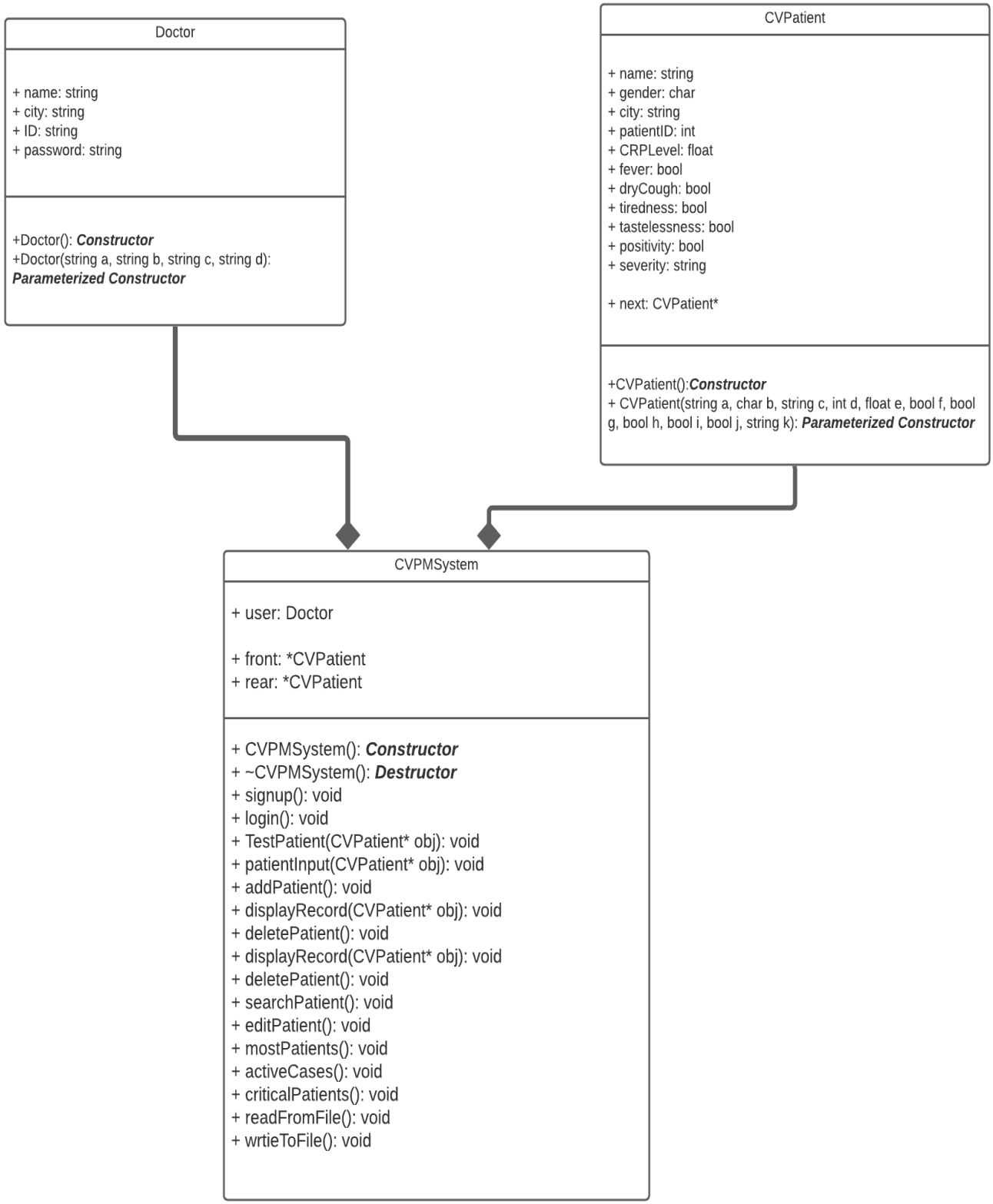
- Your application should have a proper login.
- Functionality to Add, Delete, Search and Modify records of the patient.
- A detail file (txt or binary) that includes the record of the patient along with all the attributes, some of them are symptoms, immune level, severity, city name etc.
- A function that computes which city of the Pakistan contains more patient then other cities.
- A person must be declared Corona Virus Patient if the immune level is below the minimum level and symptoms are fever, dry cough, tiredness etc.
- Application code must be divided as separate (.h, .cpp) files.

++++++++++++++++++++++++

Objectives:

- To show a strong understand of the concepts of Data Structure and Algorithms.
- To show how to break down a problem into small pieces and solve it using concepts of DSA.
- To show complete understanding of how to build a modular program in which all the pieces work together to build the complete program.
- To show how to use separate filing to build a program.
- To show how to write user-friendly code. Properly commenting each function and all the important parts of the program. Also using insightful variable names.
- To show good understand of OOP concepts and build a program that does not allow anyone except the authorized person to use the program and access records.
- To show a good understanding of filing so that the user entered by data can be stored for permanent storage and read again later on.
- To allow user to easily add, edit and delete data. And, updating related info as the record is modified by the user.
- To show how to read data from file and store it in a data structure to allow fast editing.
- To show how to store data contained in a data structure into a text file for permanent storage.

UML diagram:


Doctor
+ name: string
+ city: string
+ ID: string
+ password: string
+Doctor(): Constructor
+Doctor(string a, string b, string c, string d): Parameterized Constructor
CVPatient
+ name: string
+ gender: char
+ city: string
+ patientID: int
+ CRPLevel: float
+ fever: bool
+ dryCough: bool
+ tiredness: bool
+ tastelessness: bool
+ positivity: bool
+ severity: string
+ next: CVPatient*
+CVPatient():Constructor
+ CVPatient(string a, char b, string c, int d, float e, bool f, bool g, bool h, bool i, bool j, string k): Parameterized Constructor
CVPMSystem
+ user: Doctor
+ front: *CVPatient
+ rear: *CVPatient
+ CVPMSystem(): Constructor
+ ~CVPMSystem(): Destructor
+ signup(): void
+ login(): void
+ TestPatient(CVPatient* obj): void
+ patientInput(CVPatient* obj): void
+ addPatient(): void
+ displayRecord(CVPatient* obj): void
+ deletePatient(): void
+ displayRecord(CVPatient* obj): void
+ deletePatient(): void
+ searchPatient(): void
+ editPatient(): void
+ mostPatients(): void
+ activeCases(): void
+ criticalPatients(): void
+ readFromFile(): void
+ wrtieToFile(): void

Program Design:

The program includes three classes in total along with one implementation file and one file for the main program. In total, there are five files. Among the classes, the first one is the Doctor class. This class stores the data of person who is logging in and using the program. ID and password are two of its data members along with name and city.

The second class is CVPatient class (short for Corona Virus Patient). This class sotres all the information of a patient (name, city, symptoms, etc.). This class also has a pointer called “next”. The objects of this class are used as nodes in the program.

The third class is the CVPMSystem class (short for Corona Virus Patient Management System). This class has an object of Doctor class and two pointers of CVPatient class. Meaning, this program uses composition concepts. This class also has most of the functions used in the program.

(**Note:** Entire program code AND the data of text files is given in the end for easy copy/pasting and testing. In addition, the program will not work without the required text file of patient record. See end of report for more details.)

Acknowledgements:

1. https://www.tutorialspoint.com/what-is-the-use-of-cin-ignore-in-cplusplus
2. https://www.geeksforgeeks.org/substring-in-cpp/
3. https://www.geeksforgeeks.org/string-find-in-cpp/#:~:text=String%20find%20is%20used%20to,of%20starting%20position%20is%200.
4. http://www.cplusplus.com/reference/string/stoi/
5. https://stackoverflow.com/questions/1658386/sleep-function-in-c
6. https://stackoverflow.com/questions/25074624/why-is-stdgetline-skipped
7. http://lucid.app/lucidchart/

Functions:

Main() function:

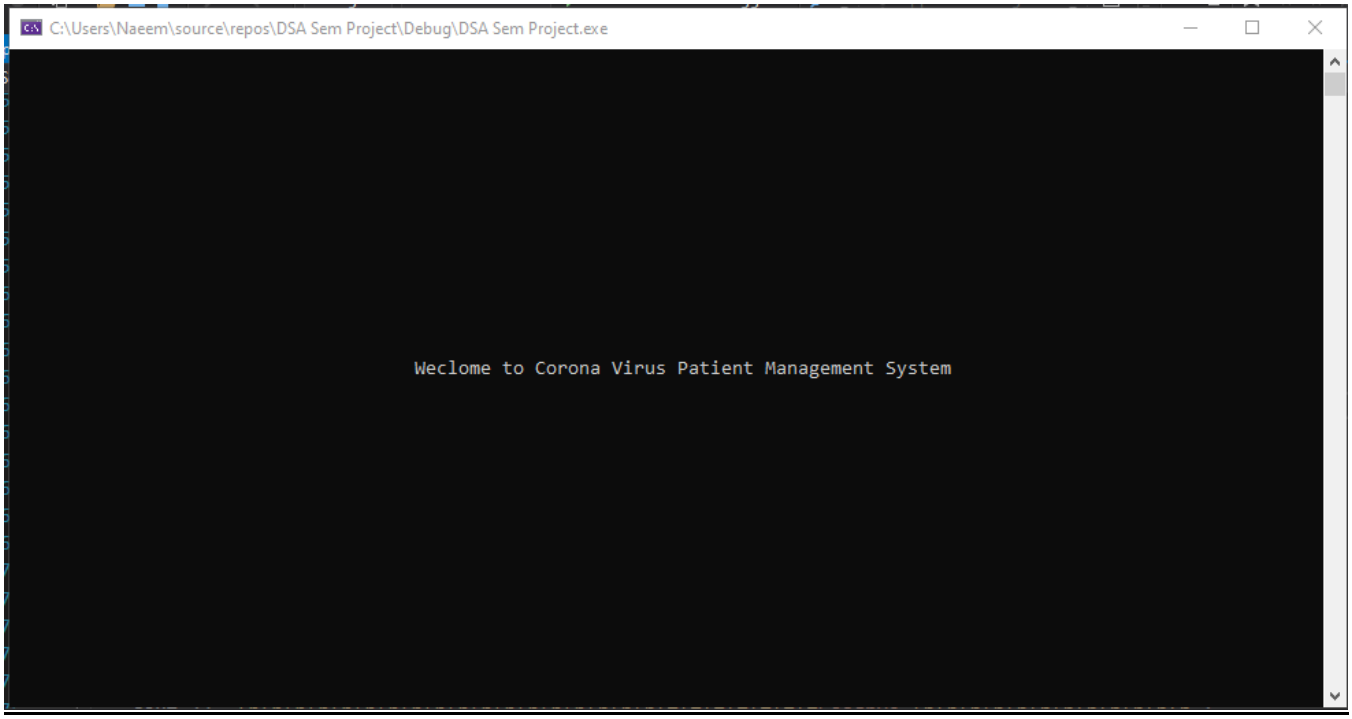
This is the first function that is called in the entire program. It initializes an object of the CVPMSystem class, calls the Login() and ReadFromFile() functions. After that, it also provides a main menu to the user and calls functions based on user input. When the user exits the program, the Main function also displays a goodbye message, calls the WriteToFile() function and then it calls the destructor of CVPMSystem class to delete the queue and free up the memory. Basically, the main function simply serves as the interface between the program and the user. It is designed in such a way that the user is able to invoke its functions whichever order the user desires.

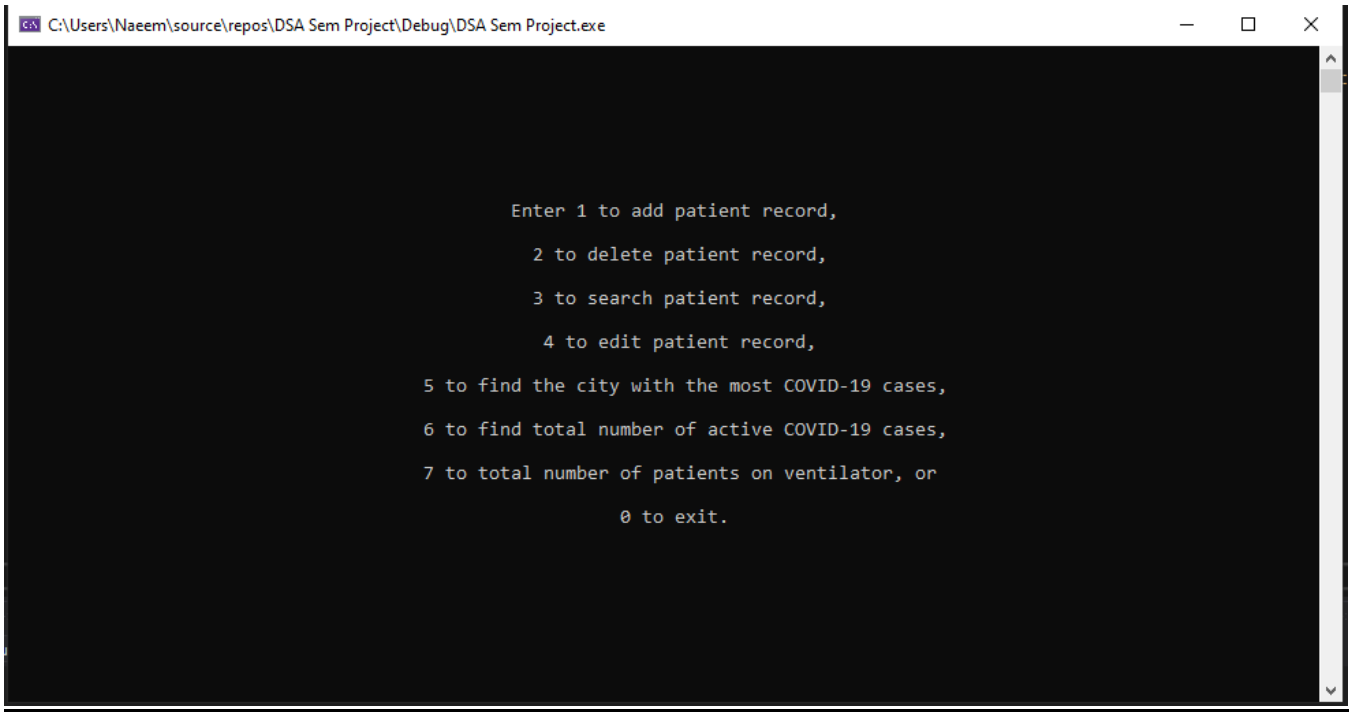
Code:

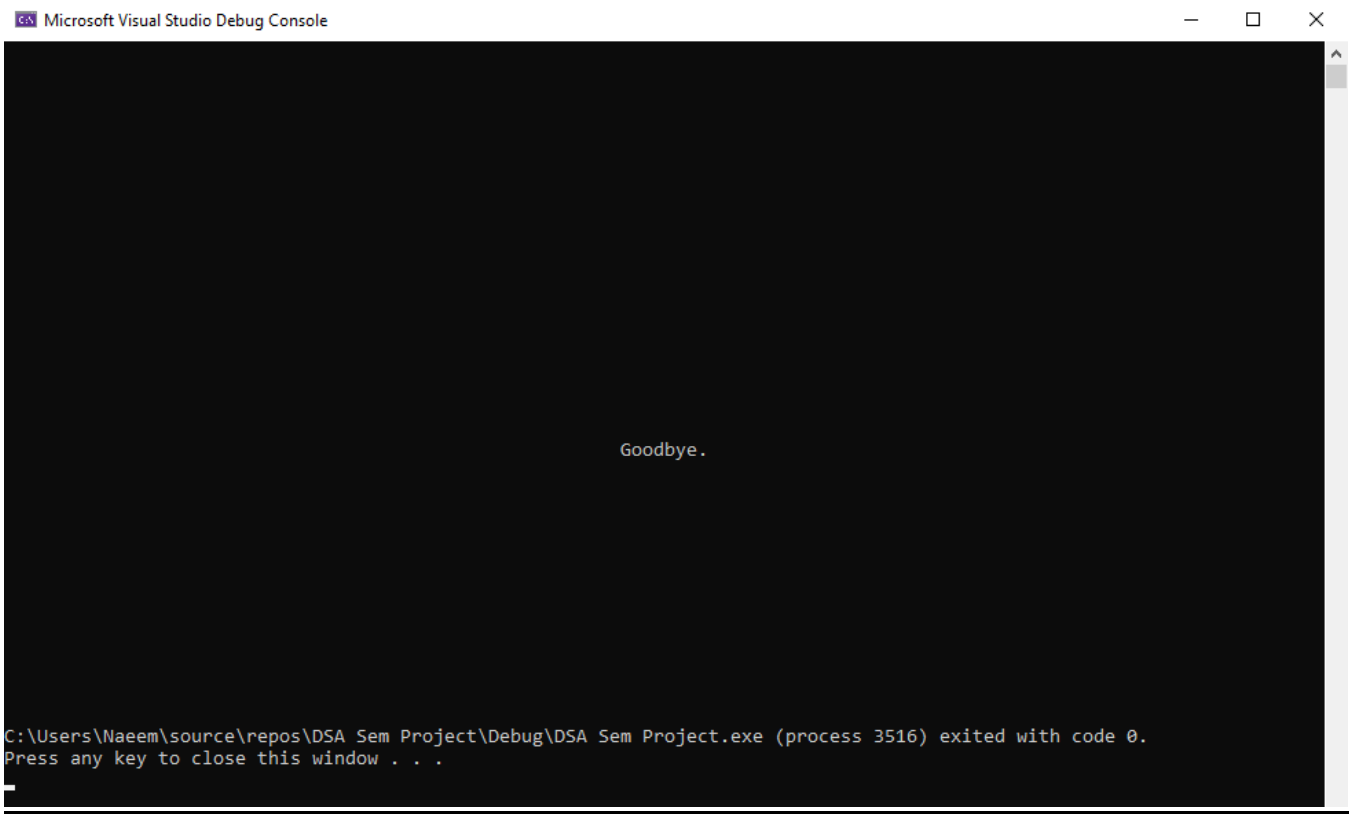
```
#include <iostream>
#include <windows.h>                              //to use Sleep function
#include "CVPMSystem.h"

using namespace std;


int main()
{
	CVPMSystem obj;
	int option = 0;

	cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t     Weclome to Corona Virus Patient
Management System";
	Sleep(2000);

	obj.login();                                          //start login function. User cannot
access program without logging in with correct ID nad password
	obj.readFromFile();                                   //reading patient data from file and
making queue aftewr the user successfully logged in

	while (1)
	{
		Sleep(500);
		system("CLS");

		cout << "\n\n\n\n\n\n\n\t\t\t\t\t      Enter 1 to add patient record, "
			//Main menu
			<< "\n\n\t\t\t\t\t\t2 to delete patient record, "
			<< "\n\n\t\t\t\t\t        3 to search patient record, "
			<< "\n\n\t\t\t\t\t\t 4 to edit patient record, "
```

```
                        << "\n\n\t\t\t\t\t      5 to find the city with the most COVID-19 cases, "
                        << "\n\n\t\t\t\t\t      6 to find total number of active COVID-19 cases, "
                        << "\n\n\t\t\t\t\t      7 to total number of patients on ventilator, or"
                        << "\n\n\t\t\t\t\t\t\t\t0 to exit.\n\n\n\t\t\t\t\t\t\t\t     ";
                cin >> option;

                Sleep(500);                                                     //adding a slight
delay and clearing screen
                system("CLS");

                if (option == 1)                                        //starting functions
according to user choice
                {
                        obj.addPatient();
                }
                else if (option == 2)
                {
                        obj.deletePatient();
                }
                else if (option == 3)
                {
                        obj.searchPatient();
                }
                else if (option == 4)
                {
                        obj.editPatient();
                }
                else if (option == 5)
                {
                        obj.mostPatients();
                }
                else if (option == 6)
                {
                        obj.activeCases();
                }
                else if (option == 7)
                {
                        obj.criticalPatients();
                }
                else
                {
                        break;
                }

        }

        Sleep(300);
        system("CLS");
        cout <<
"\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\tGoodbye.\n\n\n\n\n\n\n\n\n\n\n\n";

        obj.writeToFile();                              //After user exits, patient record file
is truncated and queue data is stored in file
        obj.~CVPMSystem();                              //After queue data is stored in file,
destructor is invoked to free memory.

}
```

Screenshots:






Microsoft Visual Studio Debug Console
Goodbye.
C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe (process 3516) exited with code 0.
Press any key to close this window . . .

Login function:

Apart from Main() and all the constructors, this is the first function that is called inside the program (after the welcome screen). This function asks the users whether they want to login, sign up (in case they are new to the program) or exit the program. As soon as the program runs, this function will be the first one to be invoked by the program. If the user passes the verification, they can access the functionalities of the interface.

When the user chooses login, the user has to enter ID and password and if they match with stored data the user is welcomed otherwise error is shown and user has to enter ID and Password again. In case the user chooses signup, the user has to enter name, city, ID and password which are then stored in file and the user is take to login page again.

In a real hospital, this system can be connected to hospital's database to confirm all new signups match with data already contained in the database. If the data is not already in database, the user should not be allowed to create a login.

If the user selects the third option, the program exits.

Code:

```
//Function for logging in, signing up or exiting the program. Program won't go ahead of this
function until user logs in
void CVPMSystem::login()
{
	int check = 0;

	while (check != 1)
	{
		Sleep(500);
		system("CLS");
		int option;
		Doctor temp;

		while (1)				//loop asking user to login, signup or exit.
User cannot go ahead if he does not login with correct ID and pass
		{
			cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t     Enter 1 to login, 2 to
sign up, or 0 to exit\n\n\t\t\t\t\t\t\t\t   ";
			cin >> option;

			if (option == 1)
			{
				break;
			}
			else if (option == 2)		//sign up and then restarting loop so
user can login
			{
				signup();
```

```
                        user.name = "";
                        Sleep(500);
                        system("CLS");
                        continue;
                }
                else if (option == 0)             //exitting
                {
                        Sleep(500);
                        system("CLS");
                        cout <<
"\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\t\tGoodbye.\n\n\n\n\n\n\n\n\n\n\n\n";
                        exit(0);
                }
                else                                                    //in case of
wrong input
                {
                        cout << "\n\n\t\t\t\t\t\t\tIncorrect input. Try again.\n";
                        Sleep(2000);
                        system("CLS");
                }
        }

        Sleep(500);
        system("CLS");

        cin.ignore();
        cout << "\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\t  Enter ID: ";
    //taking login input
        getline(cin, user.ID);
        cin.ignore();
        cout << "\n\n\t\t\t\t\t\tEnter password: ";
        getline(cin, user.password);

        ifstream in(::loginFile, ios::in);
        while (!in.eof())
        {
                getline(in, temp.name);                                 //taking
input from file
                getline(in, temp.city);
                getline(in, temp.ID);
                getline(in, temp.password);

                if (temp.ID == user.ID && temp.password == user.password)
    //if match found in file, program welcomes user
                {
                        in.close();
                        check = 1;
                        user.name = temp.name;
                        user.city = temp.city;
                        temp.name = temp.name.substr(0, temp.name.find(" "));
                        Sleep(500);
                        system("CLS");
                        cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\tWelcome, Dr. " <<
temp.name << endl;
                        Sleep(500);
                        break;
                }
        }
```

```
            if (in.eof() && user.name == "")                                    //if no login
match found, loop restarts
            {
                  Sleep(500);
                  system("CLS");
                  cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t   Entered username and
password don't match any record.\n";
                  Sleep(2000);
                  system("CLS");
            }
      }
}
```

Screenshots:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter 1 to login, 2 to sign up, or 0 to exit


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter ID: 123
Enter password: 123

Sign up function:

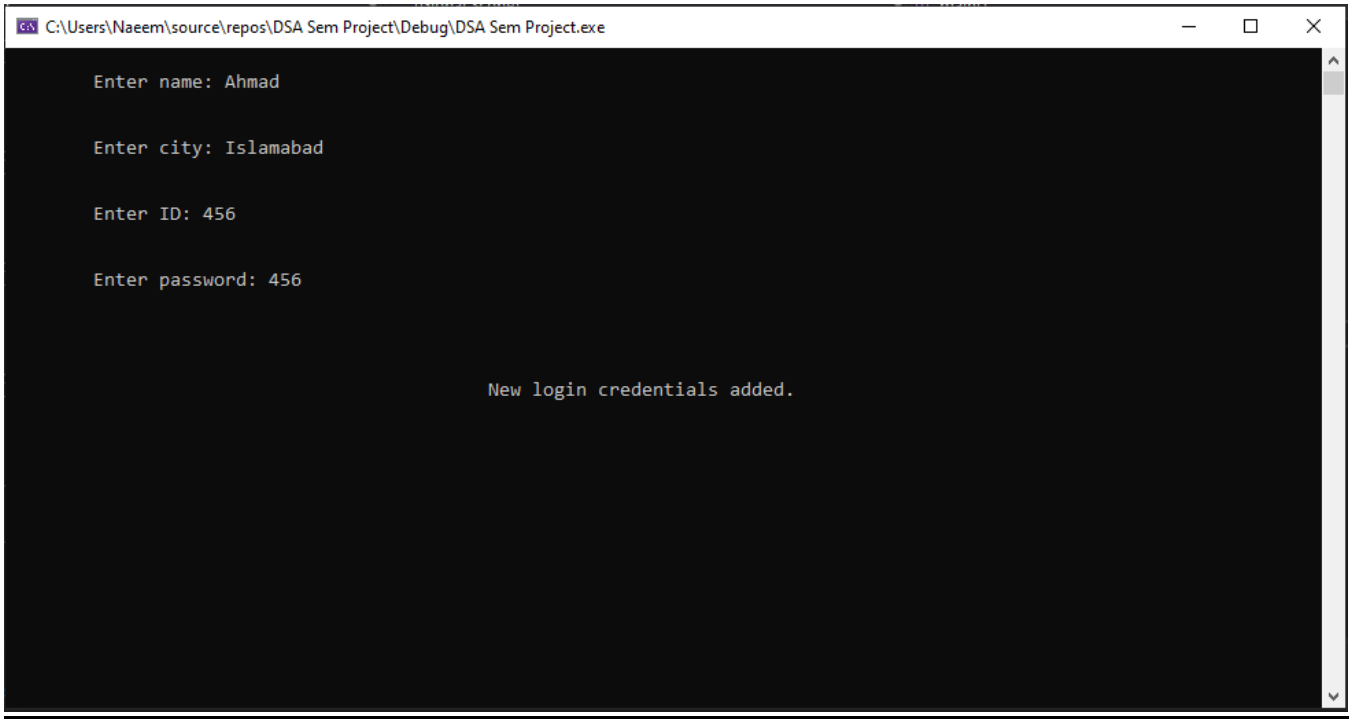
Inside the login function, the user can choose to sign up if they are using the program for the first time. In that case, this function is called. This function takes input from user and stores/updates it in the login file. Now this same login file will be read by the login function, and will grant access if matched correctly.

Keep in mind, that this function is designed in such a way that it **DOES NOT OVERWRITE**, and simply adds the information at the end of the txt file.

Code:

```
//Function to add new login info in permanent storage
void CVPMSystem::signup()                          //can be connected to the databse of hospitals
to confirm new signups match with records in database
{
	Sleep(500);
	system("CLS");                                          //waiting for 0.5 second and then
clearing the screen

	cin.ignore();
	cout << "\n\tEnter name: ";
	getline(cin, user.name);
	cin.ignore();
	cout << "\n\tEnter city: ";
	getline(cin, user.city);
	cin.ignore();
	cout << "\n\tEnter ID: ";
	getline(cin, user.ID);
	cin.ignore();
	cout << "\n\tEnter password: ";
	getline(cin, user.password);

	ofstream out(::loginFile, ios::app);
	if (!out)
	{
		cout << "\n\tError opening file.\n";
	}
	else
	//sotring new sign up information in file
	{
		out << user.name << endl
			<< user.city << endl
			<< user.ID << endl
			<< user.password << endl;
		out.close();

		cout << "\n\n\n\n\t\t\t\t\t    New login credentials added.\n";
		Sleep(1500);
	}
}
```

Screenshot:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter name: Ahmad
Enter city: Islamabad
Enter ID: 456
Enter password: 456
New login credentials added.

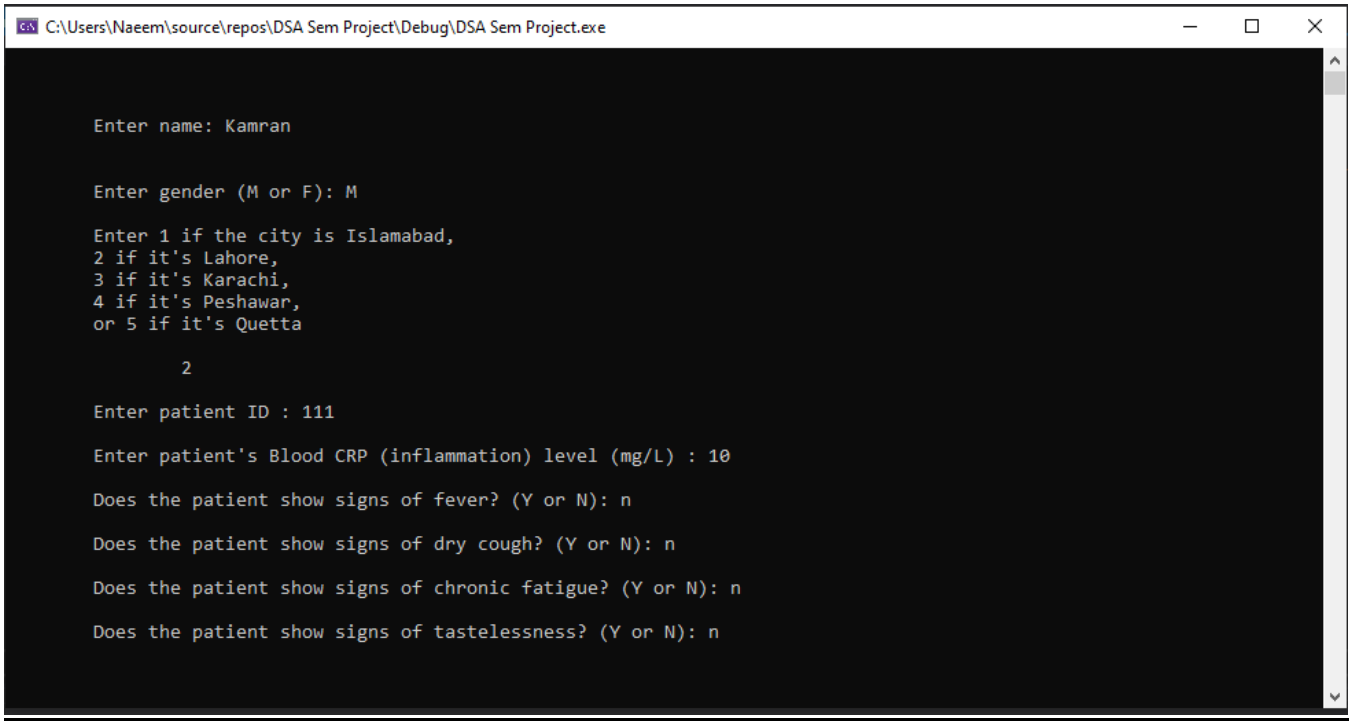
Add Patient function:

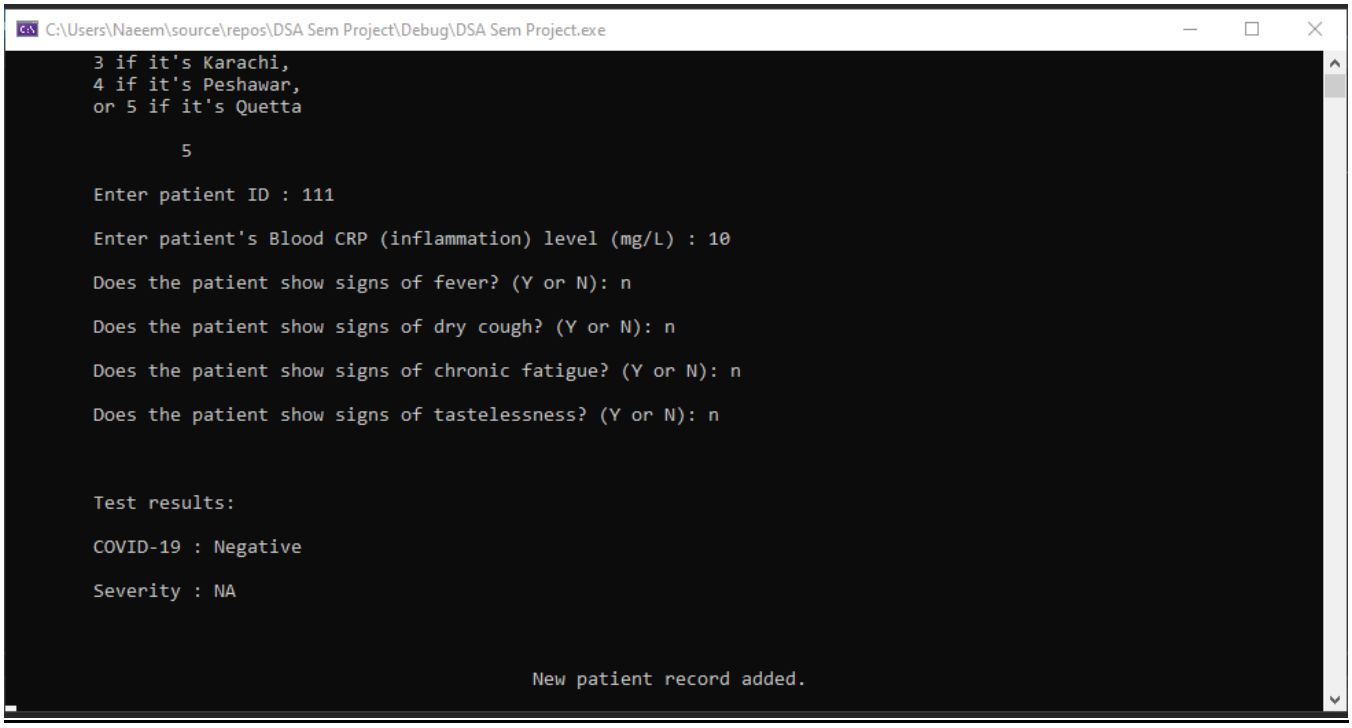
This function creates a new pointer of the CVPatient class and passes it to the PatientInput() function to take input from the user. After PatientInput() returns the pointer back to AddPatient, this function connects it to the queue in the end. Therefore, this function also doesn't overwrite other information stored, it simply adds after them.

Code:

```
//Function to add new patient record. It calls the Input function then connects new node in the
end
void CVPMSystem::addPatient()
{
	CVPatient* newPatient = new CVPatient;
	patientInput(newPatient);

	if (front == NULL && rear == NULL)
	{
		front = rear = newPatient;
	}
	else
	{
		rear->next = newPatient;
		rear = newPatient;
	}
	cout << "\n\n\t\t\t\t\t\tNew patient record added.\n";
	Sleep(1500);

}
```

Screenshots:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter name: Kamran
Enter gender (M or F): M
Enter 1 if the city is Islamabad,
2 if it's Lahore,
3 if it's Karachi,
4 if it's Peshawar,
or 5 if it's Quetta
2
Enter patient ID : 111
Enter patient's Blood CRP (inflammation) level (mg/L) : 10
Does the patient show signs of fever? (Y or N): n
Does the patient show signs of dry cough? (Y or N): n
Does the patient show signs of chronic fatigue? (Y or N): n
Does the patient show signs of tastelessness? (Y or N): n


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
3 if it's Karachi,
4 if it's Peshawar,
or 5 if it's Quetta
5
Enter patient ID : 111
Enter patient's Blood CRP (inflammation) level (mg/L) : 10
Does the patient show signs of fever? (Y or N): n
Does the patient show signs of dry cough? (Y or N): n
Does the patient show signs of chronic fatigue? (Y or N): n
Does the patient show signs of tastelessness? (Y or N): n
Test results:
COVID-19 : Negative
Severity : NA
New patient record added.

Patient Input function:

This function is called by the AddPatient() and EditPatient() functions. It fills an object of CVPatient class by taking input from the user. If the user enters X or x, the values are not changed.

This function was created with the main purpose of preventing repetition of code, and to make the code easier to read. If this function was not created, we would have to take separate inputs in AddPatient() and EditPatient. However, now that we have created the function, we can simply attach one line code in AddPatient() and EditPatient(). This is also the reason why this function accepts CVPatient* as an argument.

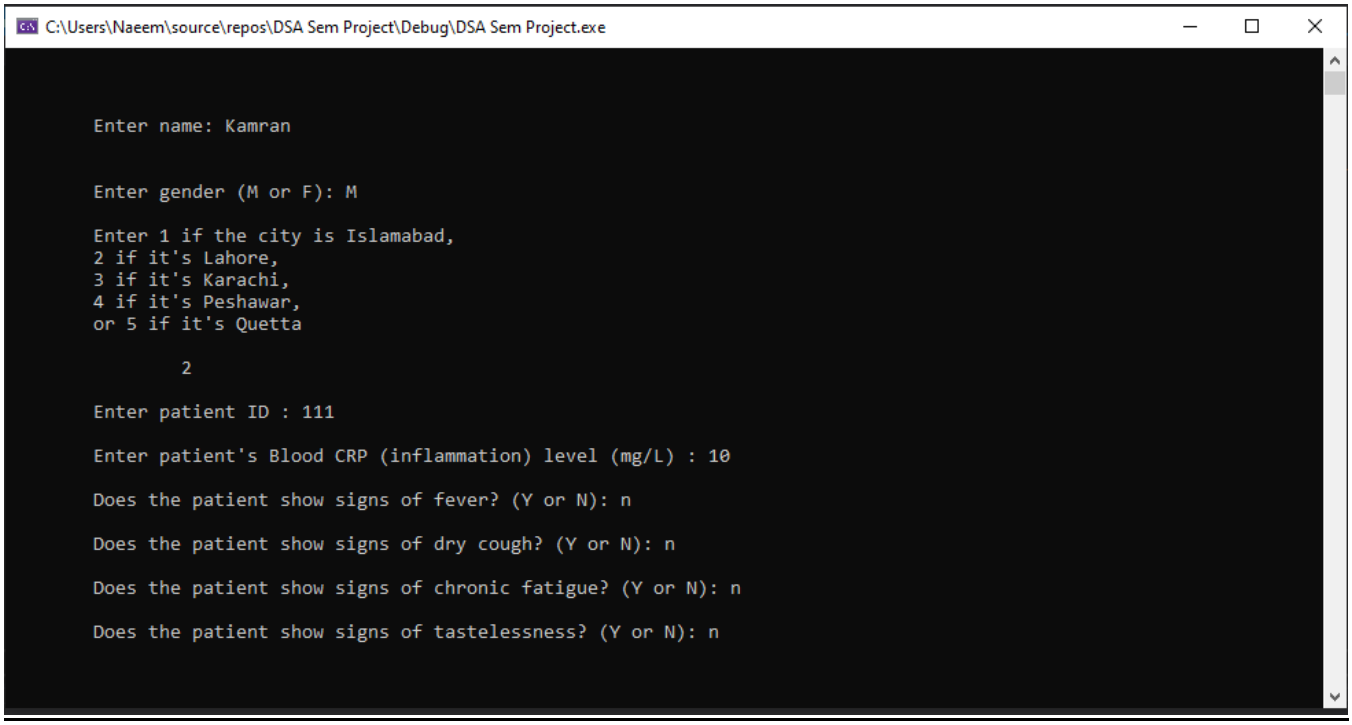
Code:

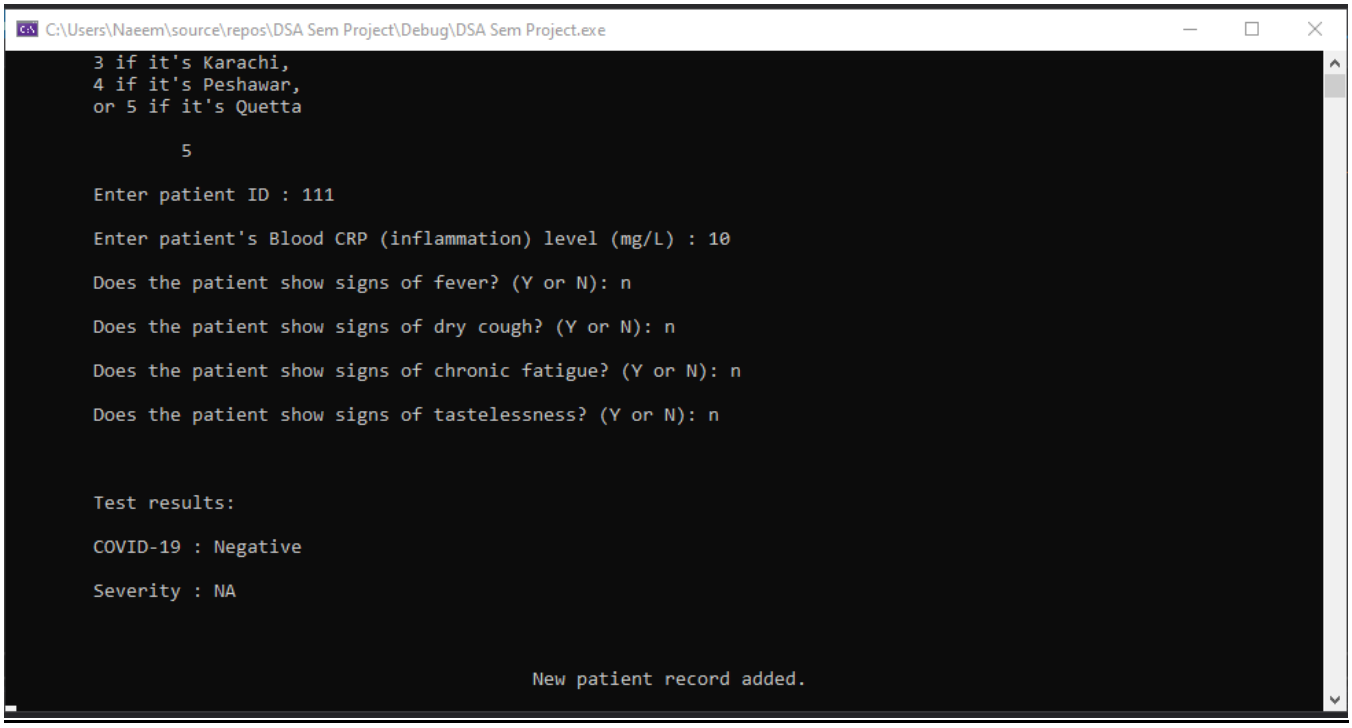
```
//Utility function to take input from user. Used in Add and Edit functions
void CVPMSystem::patientInput(CVPatient* obj)
{
	string temp;
	int option;
	char symptom;

	cin.ignore();
	cout << "\n\n\n\tEnter name: ";
	getline(cin, temp);
	if (temp != "X" && temp != "x")                         //if user enter X or x the value
is kept unchanged
	{
		obj->name = temp;
	}

	cin.ignore();
	cout << "\n\tEnter gender (M or F): ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		if (temp == "M" || temp == "m")
		{
			obj->gender = 'M';
		}
		else
		{
			obj->gender = 'F';
		}
	}

	cin.ignore();
	cout << "\n\tEnter 1 if the city is Islamabad, \n\t2 if it's Lahore, \n\t3 if it's
Karachi, \n\t4 if it's Peshawar, \n\tor 5 if it's Quetta \n\n\t        ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		option = stoi(temp);
```

```
		if (option == 1)
		{
			obj->city = "Islamabad";
		}
		else if (option == 2)
		{
			obj->city = "Lahore";
		}
		else if (option == 3)
		{
			obj->city = "Karachi";
		}
		else if (option == 4)
		{
			obj->city = "Peshawar";
		}
		else
		{
			obj->city = "Quetta";
		}
	}

	cin.ignore();
	cout << "\n\tEnter patient ID : ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		obj->patientID = stoi(temp);				//stoi converts string to
integer
	}

	cin.ignore();
	cout << "\n\tEnter patient's Blood CRP (inflammation) level (mg/L) : ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		obj->CRPLevel = stof(temp);			//stoi converts string to float
	}

	cin.ignore();
	cout << "\n\tDoes the patient show signs of fever? (Y or N): ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		if (temp == "Y" || temp == "y")				//setting bool
values
		{
			obj->fever = true;
		}
		else
		{
			obj->fever = false;
		}
	}

	cin.ignore();
	cout << "\n\tDoes the patient show signs of dry cough? (Y or N): ";
```

```
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		if (temp == "Y" || temp == "y")
		{
			obj->dryCough = true;
		}
		else
		{
			obj->dryCough = false;
		}
	}

	cin.ignore();
	cout << "\n\tDoes the patient show signs of chronic fatigue? (Y or N): ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		if (temp == "Y" || temp == "y")
		{
			obj->tiredness = true;
		}
		else
		{
			obj->tiredness = false;
		}
	}

	cin.ignore();
	cout << "\n\tDoes the patient show signs of tastelessness? (Y or N): ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		if (temp == "Y" || temp == "y")
		{
			obj->tastelessness = true;
		}
		else
		{
			obj->tastelessness = false;
		}
	}

	this->TestPatient(obj);                                  //setting positivty and severity
of CV

	Sleep(2000);
}
```

Screenshots:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter name: Kamran
Enter gender (M or F): M
Enter 1 if the city is Islamabad,
2 if it's Lahore,
3 if it's Karachi,
4 if it's Peshawar,
or 5 if it's Quetta
2
Enter patient ID : 111
Enter patient's Blood CRP (inflammation) level (mg/L) : 10
Does the patient show signs of fever? (Y or N): n
Does the patient show signs of dry cough? (Y or N): n
Does the patient show signs of chronic fatigue? (Y or N): n
Does the patient show signs of tastelessness? (Y or N): n


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
3 if it's Karachi,
4 if it's Peshawar,
or 5 if it's Quetta
5
Enter patient ID : 111
Enter patient's Blood CRP (inflammation) level (mg/L) : 10
Does the patient show signs of fever? (Y or N): n
Does the patient show signs of dry cough? (Y or N): n
Does the patient show signs of chronic fatigue? (Y or N): n
Does the patient show signs of tastelessness? (Y or N): n
Test results:
COVID-19 : Negative
Severity : NA
New patient record added.

Test Patient function:

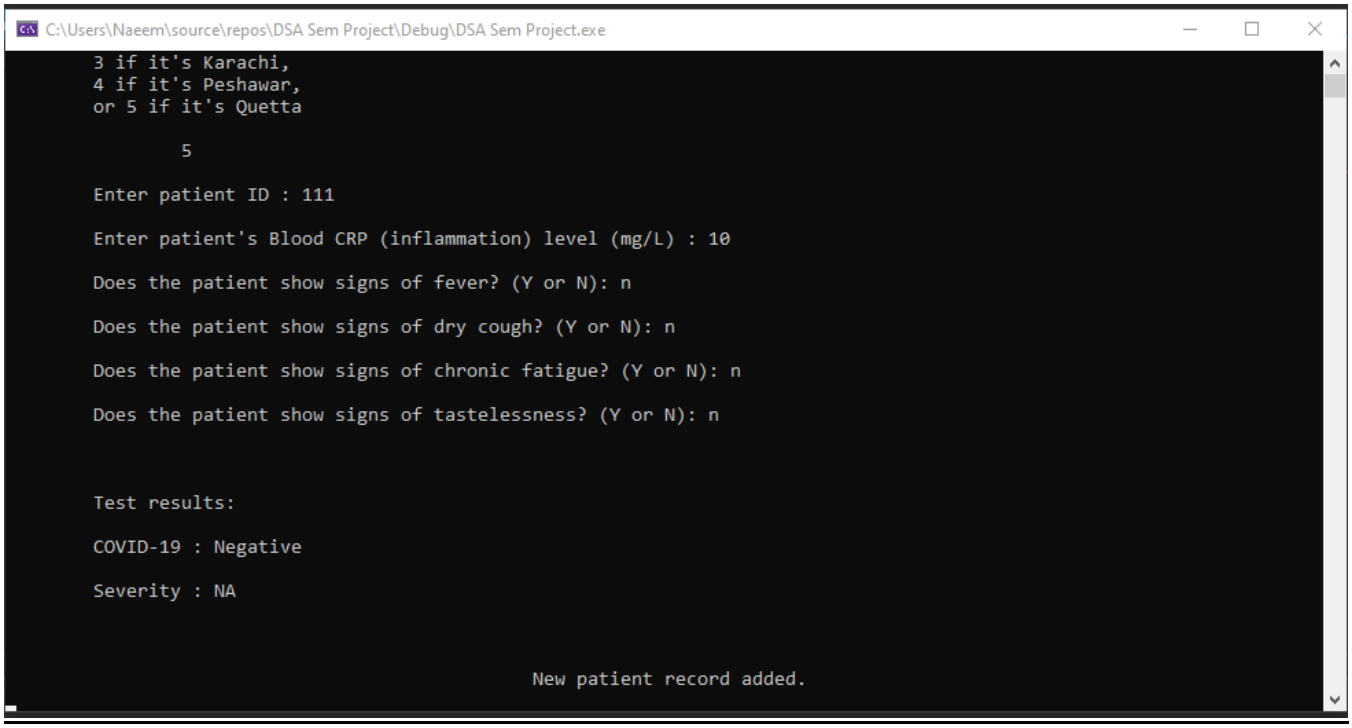
This function is called from inside the PatientInput function and tests whether the user has coronavirus or not based on his CRP levels and symptoms.

Its purpose is simply to detect if the patient is positive and to see how severe the condition of the patient is, and according to the tests, it either sets the positivity or keeps it at reset.

Code:

```
//Function tests patients for COVID19 when reocrd is added or updated
void CVPMSystem::TestPatient(CVPatient* obj)
{
	if ((obj->CRPLevel >= 11.0) && (obj->fever == true || obj->dryCough == true || obj->tiredness == true || obj->tastelessness == true))
	{
		obj->positivity = true;

		if (obj->CRPLevel >= 11.0 && obj->CRPLevel < 15.0)		//CRP tests are used to check if patient has COVID19 or not (10 is normal level, values abve 10 mean patient has virus)
		{
			obj->severity = "Mild";
		}
		else if (obj->CRPLevel >= 15.0 && obj->CRPLevel < 20.0)
		{
			obj->severity = "Moderate";
		}
		else
		{
			obj->severity = "Severe";
		}

		cout << "\n\n\n\tTest results: ";
		cout << "\n\n\tCOVID-19 : Positive";
		cout << "\n\n\tSeverity : " << obj->severity << "\n\n";
	}
	else
	{
		cout << "\n\n\n\tTest results: ";
		cout << "\n\n\tCOVID-19 : Negative";
		cout << "\n\n\tSeverity : NA\n\n";
	}
}
```

Screenshots:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
3 if it's Karachi,
4 if it's Peshawar,
or 5 if it's Quetta
5
Enter patient ID : 111
Enter patient's Blood CRP (inflammation) level (mg/L) : 10
Does the patient show signs of fever? (Y or N): n
Does the patient show signs of dry cough? (Y or N): n
Does the patient show signs of chronic fatigue? (Y or N): n
Does the patient show signs of tastelessness? (Y or N): n
Test results:
COVID-19 : Negative
Severity : NA
New patient record added.

Display Record function:

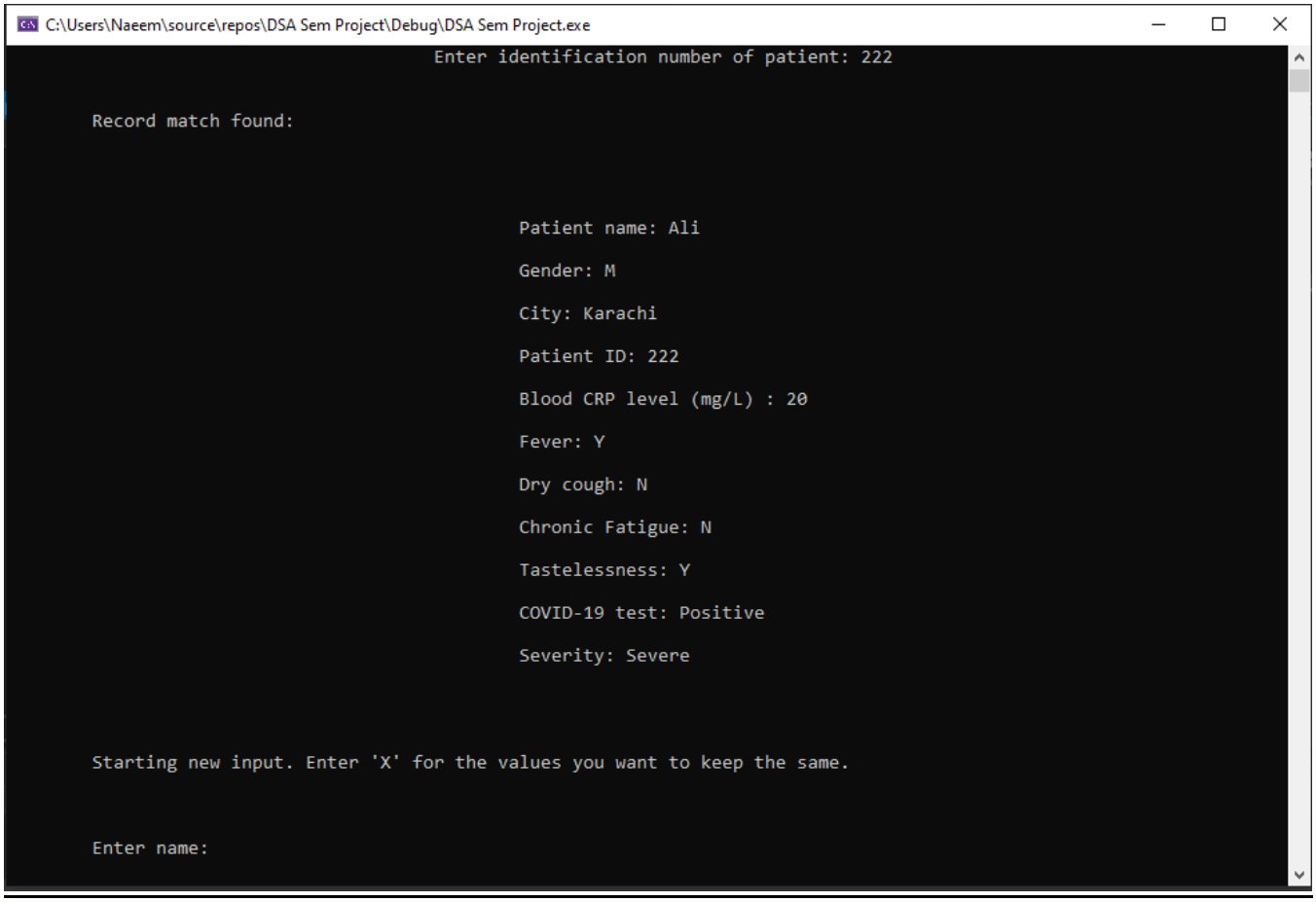
This is a utility function that is called from inside Edit, Delete and Search functions. It is passed a CVPatient class pointer and it displays the values of the pointer.

Code:

```
//Utility function used in Edit, Delete and Search
void CVPMSystem::displayRecord(CVPatient* obj)
{
	cout << "\n\n\n\t\t\t\t\t\tPatient name: " << obj->name
		<< "\n\n\t\t\t\t\t\tGender: " << obj->gender
		<< "\n\n\t\t\t\t\t\tCity: " << obj->city
		<< "\n\n\t\t\t\t\t\tPatient ID: " << obj->patientID
		<< "\n\n\t\t\t\t\t\tBlood CRP level (mg/L) : " << obj->CRPLevel;

	cout << "\n\n\t\t\t\t\t\tFever: ";
	if (obj->fever == true)
	{
		cout << "Y";
	}
	else
	{
		cout << "N";
	}

	cout << "\n\n\t\t\t\t\t\tDry cough: ";
	if (obj->dryCough == true)
	{
		cout << "Y";
	}
	else
	{
		cout << "N";
	}

	cout << "\n\n\t\t\t\t\t\tChronic Fatigue: ";
	if (obj->tiredness == true)
	{
		cout << "Y";
	}
	else
	{
		cout << "N";
	}

	cout << "\n\n\t\t\t\t\t\tTastelessness: ";
	if (obj->tastelessness == true)
	{
		cout << "Y";
	}
	else
	{
```

```
		cout << "N";
	}

	cout << "\n\n\t\t\t\t\t\tCOVID-19 test: ";
	if (obj->positivity == true)
	{
		cout << "Positive";
	}
	else
	{
		cout << "Negative";
	}

	cout << "\n\n\t\t\t\t\t\tSeverity: " << obj->severity << "\n\n";

}
```

Screenshots:



Delete Patient function:

This function asks user for ID number of the patient record he wants to delete. After taking input, it finds that record, displays it and after getting confirmation from the user, it deletes it from the queue.

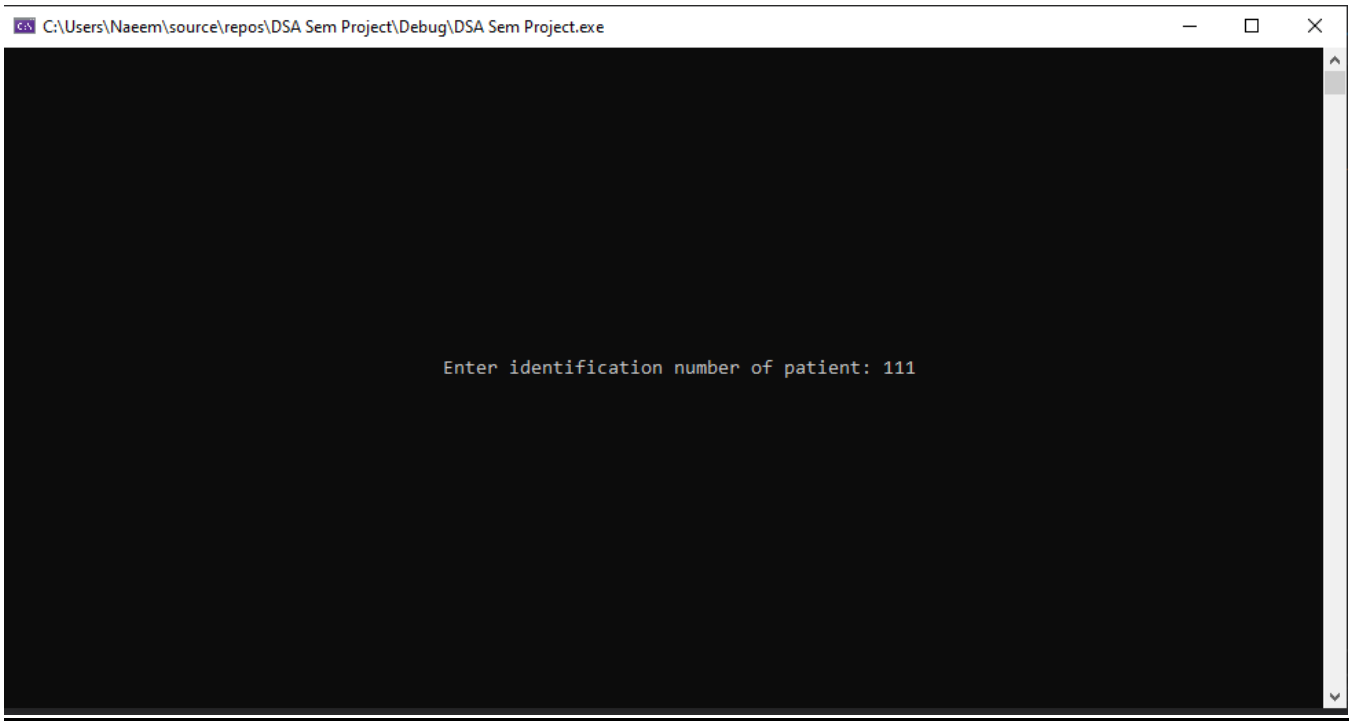
As indicated, this function has the capability of deleting information, which other functions don't quite have.

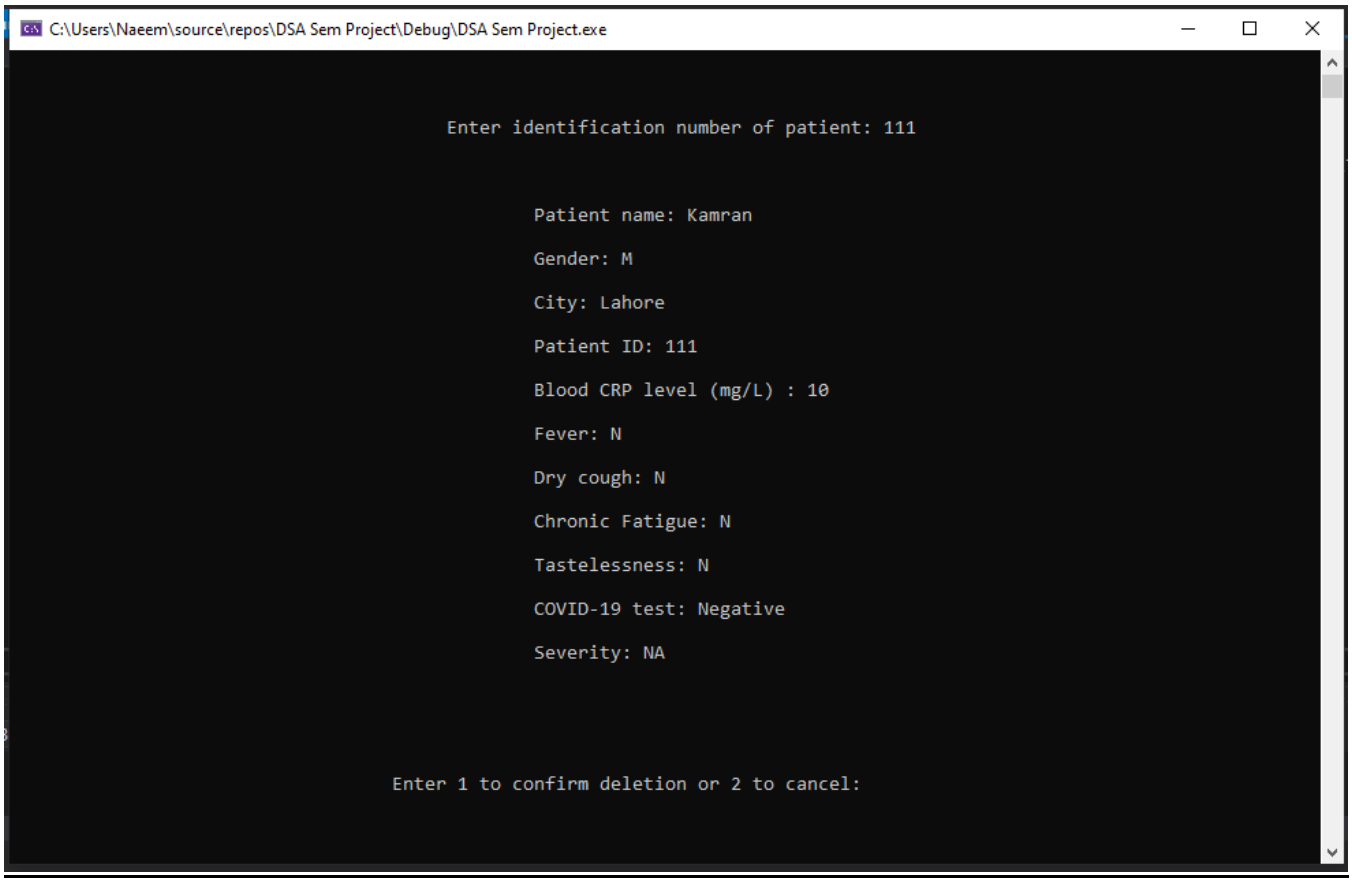
Code:

```
//Function for deleting node of a patient
void CVPMSystem::deletePatient()
{
	if (front == NULL)
	{
		cout << "\nNo data in queue.\n";
	}
	else
	{
		CVPatient* curr = front;			//used to find the node to delete
and the node before it
		CVPatient* prev = front;

		int IDnum;
		int check = 0;

		cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tEnter identification number of
patient: ";
		cin >> IDnum;

		while (curr != NULL)				//finding node to delete
		{
			if (curr->patientID == IDnum)		//match found
			{
				break;
			}
			else 							//match not found
yet
			{
				prev = curr;
				curr = curr->next;
			}
		}

		if (curr == NULL)					//no match found in all the
records
		{
			cout << "\n\n\t\t\t\t\t\tNo match found.\n";
		}
		else
	//match found
		{
```

```
                displayRecord(curr);
                cout << "\n\n\n\n\t\t\t\t   Enter 1 to confirm deletion or 2 to cancel:
\n\n\t\t\t\t\t\t\t";       //taking user confirmation after displaying record
                cin >> check;

                if (check == 1)
                {
                        if (front == rear)                 //only one node in queue
                        {
                                front = rear = NULL;
                        }
                        else if (curr == front && front != rear)        //first node to be
deleted
                        {
                                front = front->next;
                        }
                        else if (curr == rear && front != front)        //last node to be
deleted
                        {
                                prev->next = NULL;
                                rear = prev;
                        }
                        else
        //deleting from the middle
                        {
                                prev->next = curr->next;
                        }

                        delete curr;
                        cout << "\n\n\n\t\t\t\t\t\tRecord deleted.\n";
                }
                else
                {
                        cout << "\n\n\n\t\t\t\t\t\tOperation cancelled.\n";
                }
                Sleep(500);
            }
        }
}
```

Screenshots:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter identification number of patient: 111


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter identification number of patient: 111
Patient name: Kamran
Gender: M
City: Lahore
Patient ID: 111
Blood CRP level (mg/L) : 10
Fever: N
Dry cough: N
Chronic Fatigue: N
Tastelessness: N
COVID-19 test: Negative
Severity: NA
Enter 1 to confirm deletion or 2 to cancel:

Search Patient function:

This function asks user for ID number of the patient record he wants to search. After taking input, it finds that record and displays it.

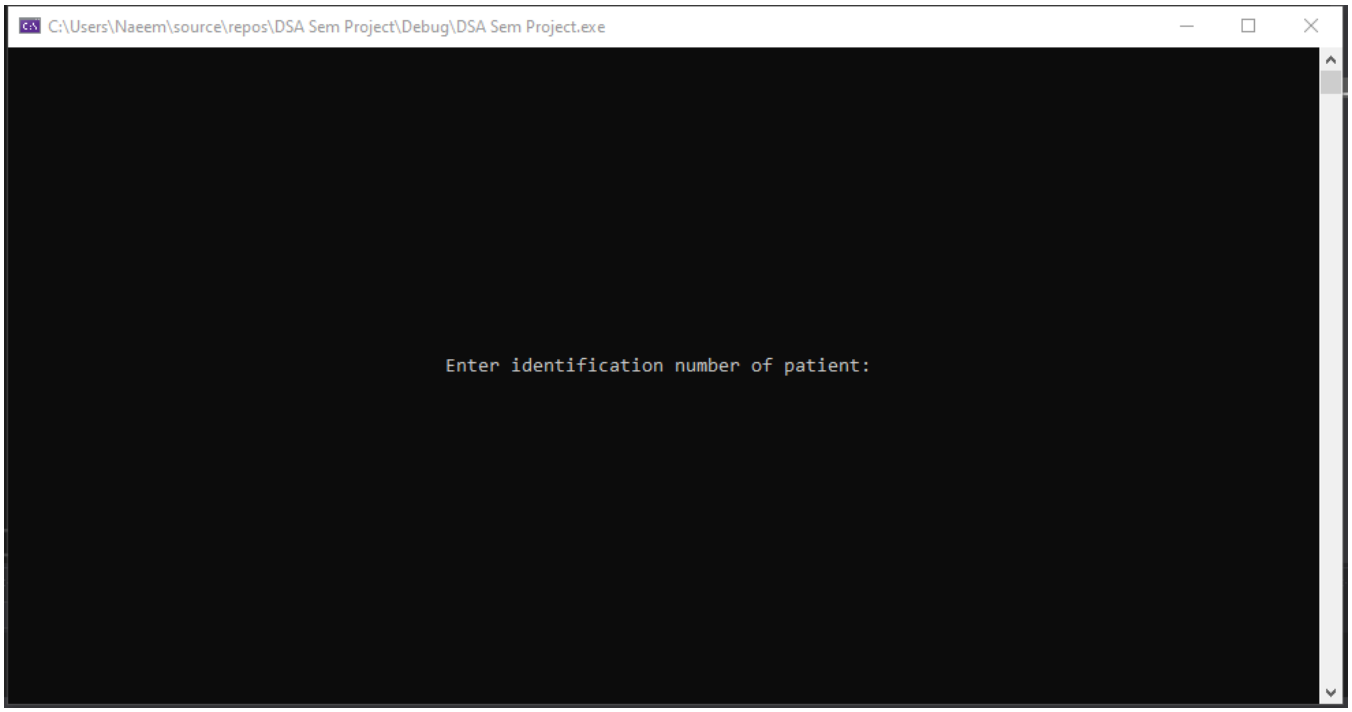
It serves as the arbitrary version of the Display Function, where the user has the freedom of checking which ever patient he wants. However, in real life, this type of function is kept private, as we don't want everyone to be able to access every patient's information and invade their privacy.

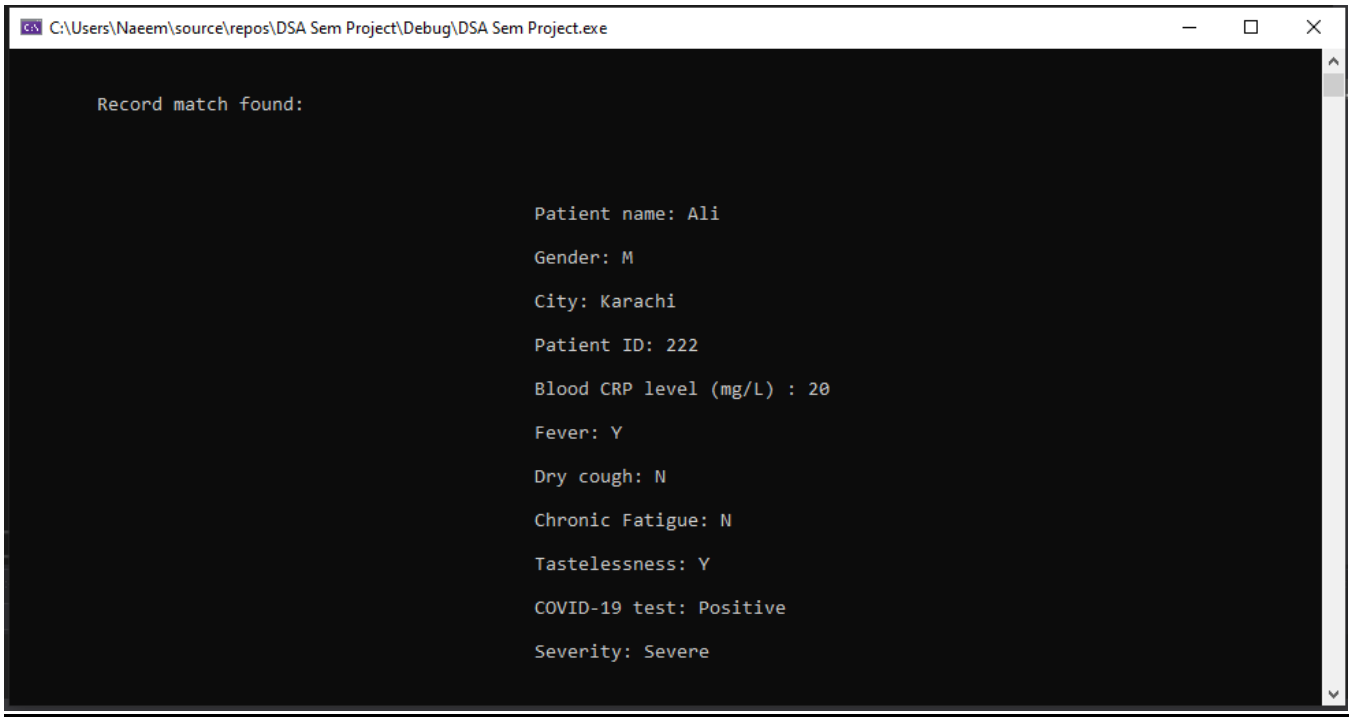
Code:

```
//Function to find and display record of a patient. It looks for the record in the queue
void CVPMSystem::searchPatient()
{
	if (front == NULL)
	{
		cout << "\nNo data in queue.\n";
	}
	else
	{

		CVPatient* curr = front;
		int IDnum;

		cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tEnter identification number of patient: ";
		cin >> IDnum;

		while (curr != NULL)                                    //loop to find node with matching ID number
		{
			if (curr->patientID == IDnum)
			{
				break;
			}
			else
			{
				curr = curr->next;
			}
		}

		if (curr == NULL)                                  //in case no match is found
		{
			cout << "\n\n\n\t\t\t\t\t\t     No match found.\n";
			Sleep(500);
		}
		else
		{
			cout << "\n\n\tRecord match found: \n\n";       //displaying record if match found
			displayRecord(curr);
			Sleep(4500);
```

```
            }
        }
}
```

Screenshots:





Edit Patient function:

This function asks user for ID number of the patient record he wants to edit. After taking input, it finds that record, displays it and calls the PatientInput function to take input again. The Input function runs the corona test again on the record after new values have been inserted.

But, before the PatientInput function is called, this user informs the user that they can enter 'X' or 'x' for those values that they don't want to change. After editing the record, it displays the modified record to the user.

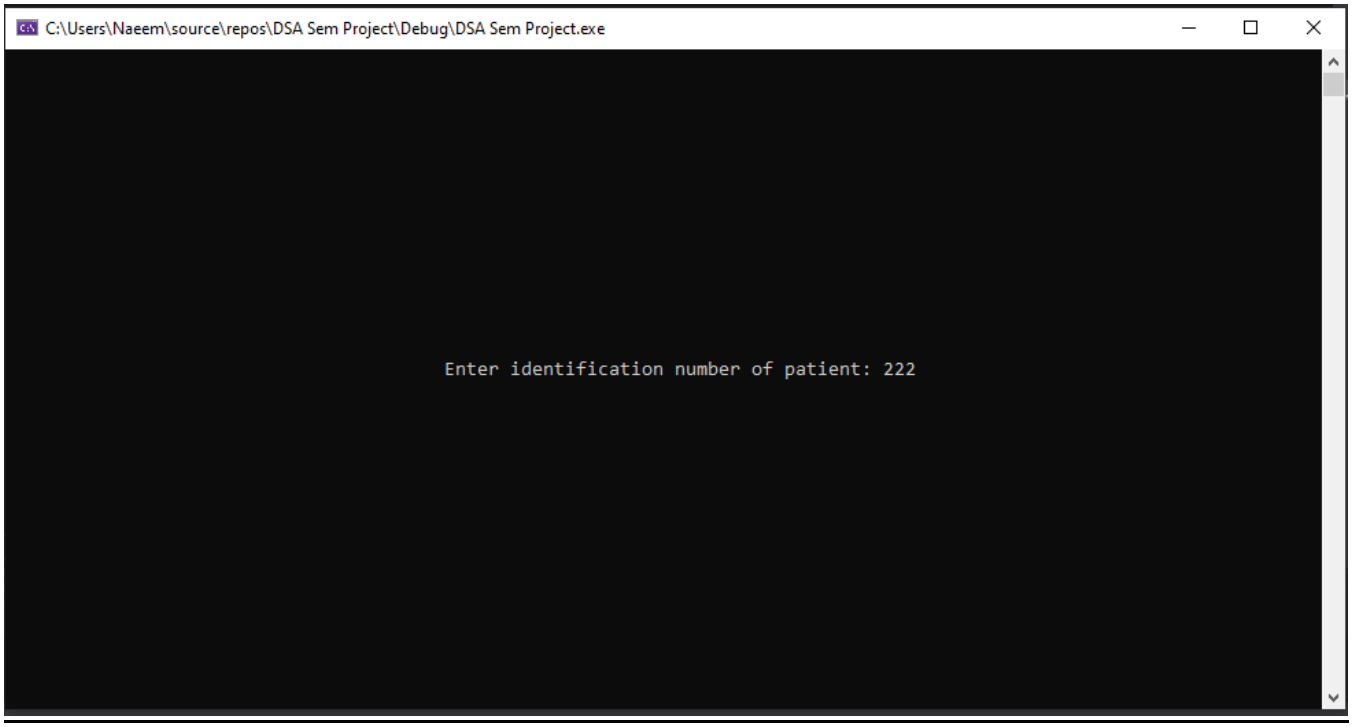
Unlike other functions, this function has the ability to overwrite (but only in the queue).

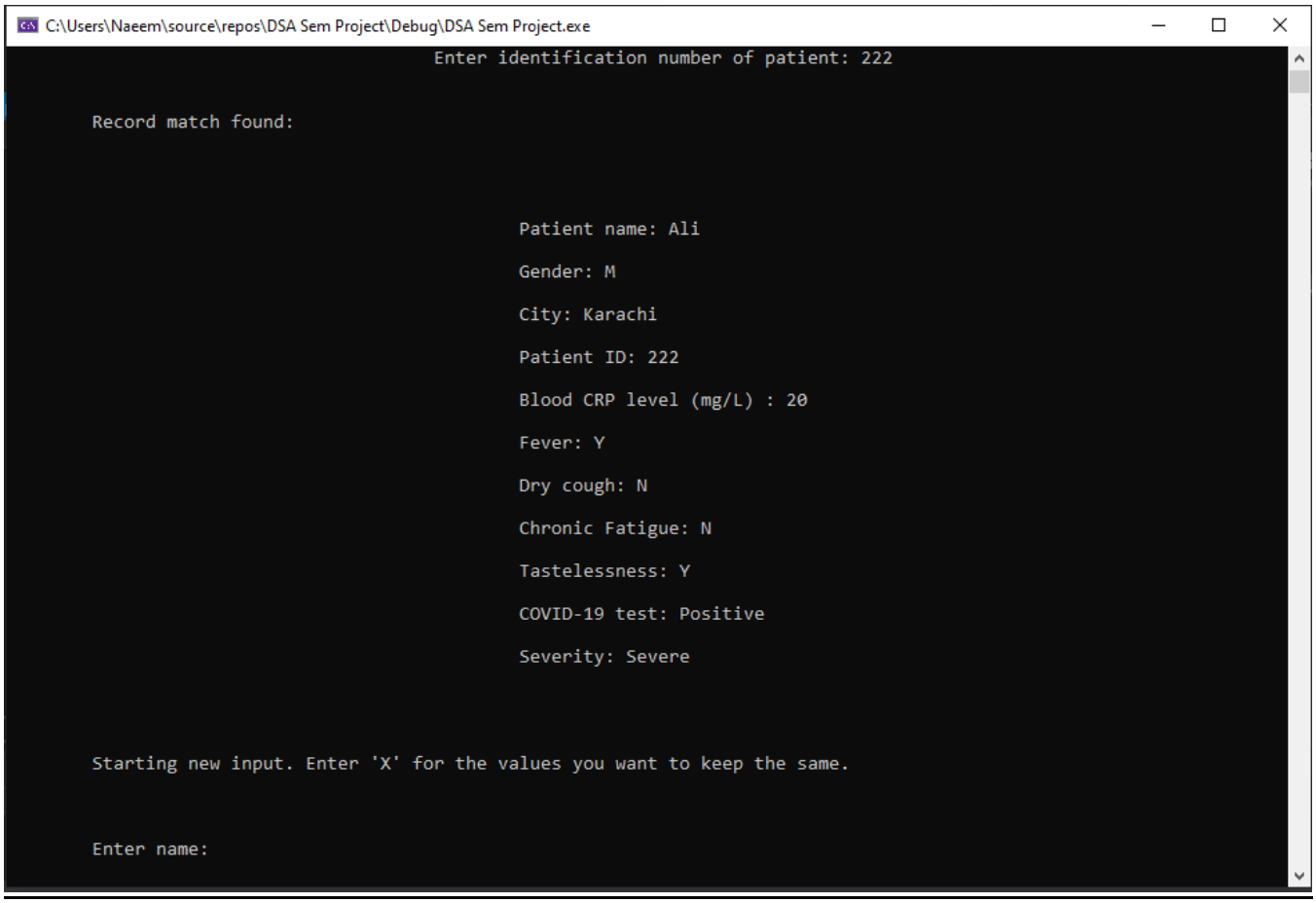
Code:

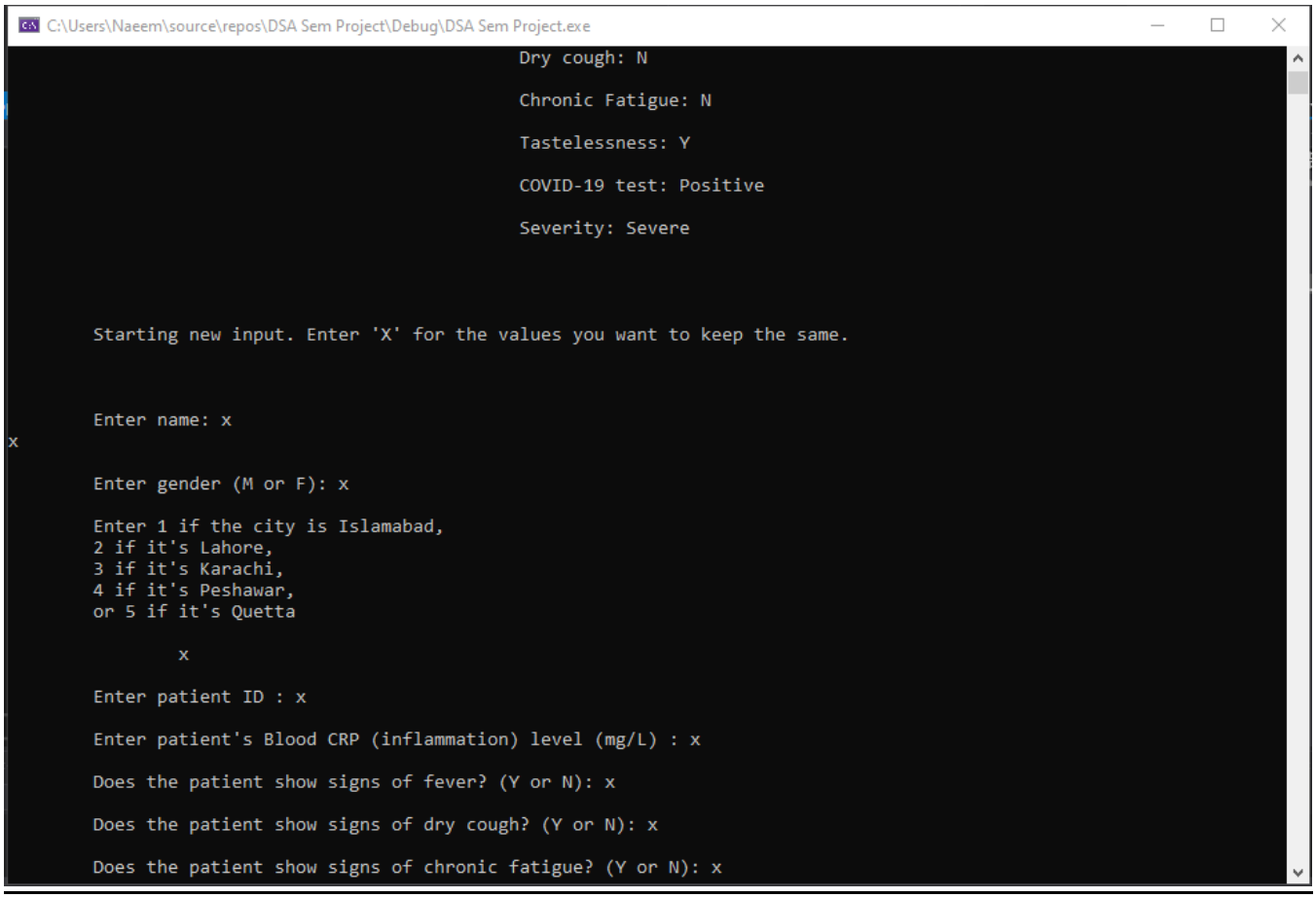
```
//Function to edit the record of a patient. It looks for the record in the queue.
void CVPMSystem::editPatient()
{
	if (front == NULL)
	{
		cout << "\nNo data in queue.\n";
	}
	else
	{

		CVPatient* curr = front;

		int IDnum;

		cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tEnter identification number of
patient: ";
		cin >> IDnum;

		while (curr != NULL)                              //loop to find the node to be
edited
		{
			if (curr->patientID == IDnum)
			{
				break;
			}
			else
			{
				curr = curr->next;
			}
		}

		if (curr == NULL)                                                 //if no match is
found in the queue
		{
			cout << "\n\n\n\t\t\t\t\t\t     No match found.\n";
```

```
			}
			else                                                                    //if match
found, show record, take new input and then show modified record
			{
				cout << "\n\n\tRecord match found: \n\n";
				displayRecord(curr);
				cout << "\n\n\n\tStarting new input. Enter 'X' for the values you want to
keep the same.\n";
				patientInput(curr);
				cout << "\n\n\tAfter modification: \n\n";
				displayRecord(curr);
				Sleep(4500);
			}
		}
}
```

Screenshots:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter identification number of patient: 222


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Enter identification number of patient: 222
Record match found:
Patient name: Ali
Gender: M
City: Karachi
Patient ID: 222
Blood CRP level (mg/L) : 20
Fever: Y
Dry cough: N
Chronic Fatigue: N
Tastelessness: Y
COVID-19 test: Positive
Severity: Severe
Starting new input. Enter 'X' for the values you want to keep the same.
Enter name:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Dry cough: N
Chronic Fatigue: N
Tastelessness: Y
COVID-19 test: Positive
Severity: Severe
Starting new input. Enter 'X' for the values you want to keep the same.
Enter name: x
x
Enter gender (M or F): x
Enter 1 if the city is Islamabad,
2 if it's Lahore,
3 if it's Karachi,
4 if it's Peshawar,
or 5 if it's Quetta
x
Enter patient ID : x
Enter patient's Blood CRP (inflammation) level (mg/L) : x
Does the patient show signs of fever? (Y or N): x
Does the patient show signs of dry cough? (Y or N): x
Does the patient show signs of chronic fatigue? (Y or N): x

```
After modification:


                                        Patient name: Ali
                                        Gender: M
                                        City: Karachi
                                        Patient ID: 222
                                        Blood CRP level (mg/L) : 20
                                        Fever: Y
                                        Dry cough: N
                                        Chronic Fatigue: N
                                        Tastelessness: Y
                                        COVID-19 test: Positive
                                        Severity: Severe
```

Most Patients function:

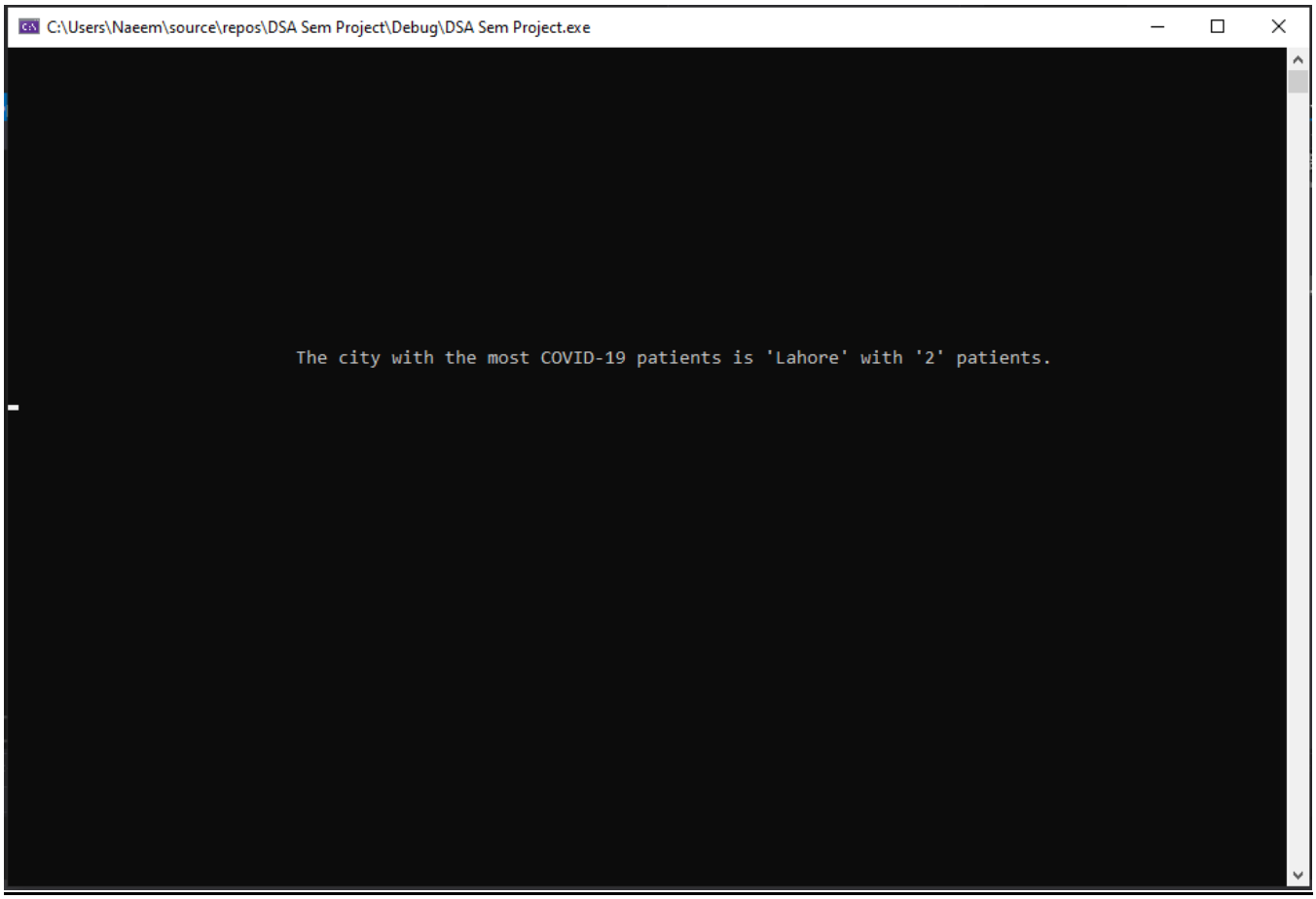
This function goes through the queue to find the city with the high number of COVID19 cases. After that, it displays the city name and patient count to the user. This function could further be used as a counter for each city as well, but it is called specifically if the user wants to know the most affected city.

Code:

```
//Function to find the city with the highest number of cases
void CVPMSystem::mostPatients()
{
	if (front == NULL)
	{
		cout << "\nNo data in queue.\n";
	}
	else
	{
		CVPatient* ptr = front;
		string cities[5] = { "Islamabad" , "Lahore" , "Karachi" , "Peshawar" , "Quetta" };
	//array with city names
		int patients[5];
                                                // array to store patients for each city
		int largestIndex = 0;
		int largest = 0;

		for (int i = 0; i < 5; ++i)                    //initializing patient array with
0
		{
			patients[i] = 0;
		}

		while (ptr != NULL)                                    //counting patients for
each city
		{
			if (ptr->city == "Islamabad")
			{
				++patients[0];
			}
			else if (ptr->city == "Lahore")
			{
				++patients[1];
			}
			else if (ptr->city == "Karachi")
			{
				++patients[2];
			}
			else if (ptr->city == "Peshawar")
			{
				++patients[3];
			}
			else
```

```
                {
                        ++patients[4];
                }
                ptr = ptr->next;
        }

        for (int i = 0; i < 5; ++i)                     //finding highest patient count
and the city
        {
                if (largest < patients[i])
                {
                        largest = patients[i];
                        largestIndex = i;
                }
        }

        cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t   The city with the most COVID-19
patients is '" << cities[largestIndex] << "' with '" << largest << "' patients.\n\n";
        Sleep(3000);
    }
}
```

Screenshots:



Active Cases function (Bonus function):

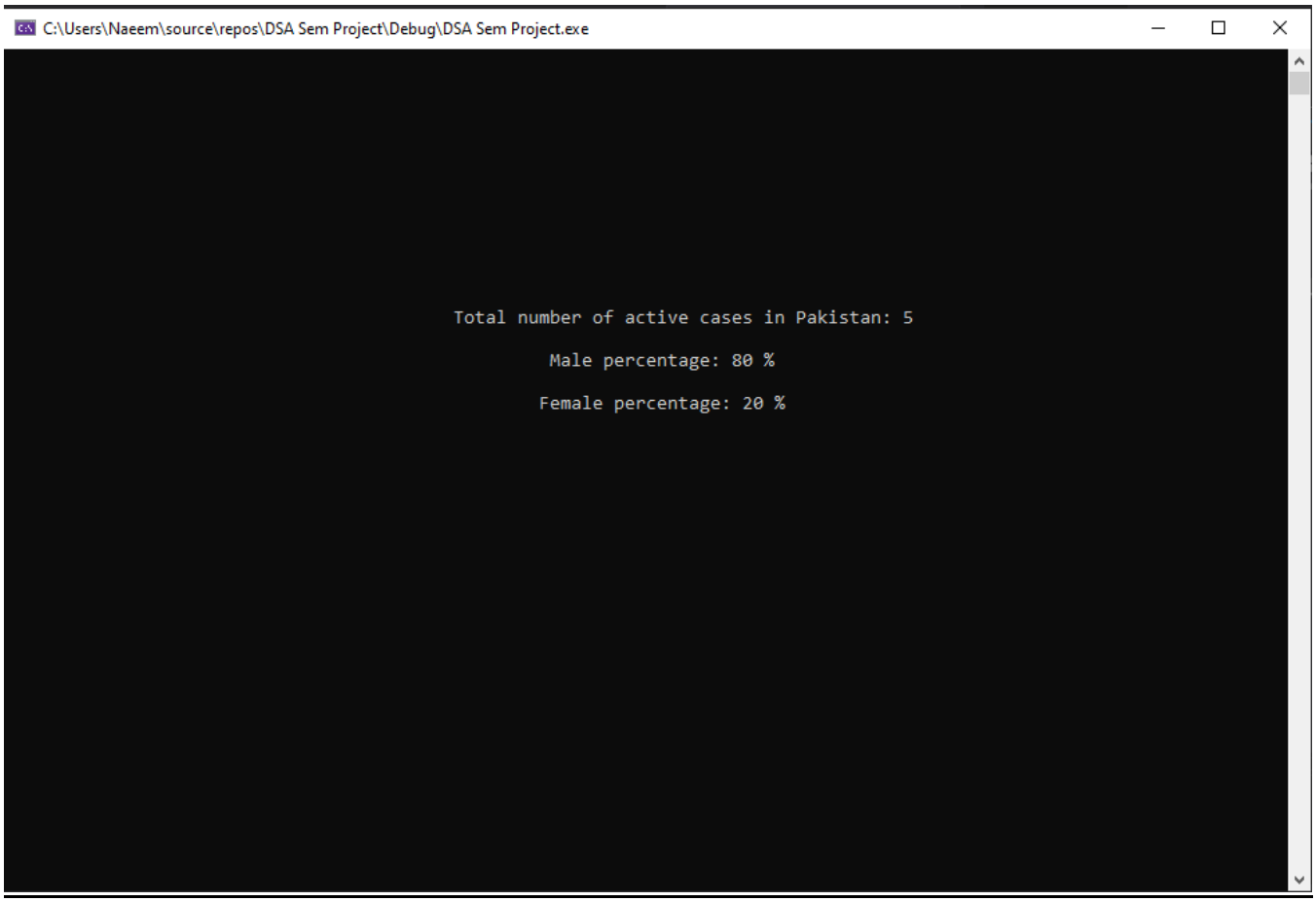
This function goes through the queue to count all the patients who test positive for COVID19. After that, it displays the total number of active cases along with what percentage of the active cases are and male and what percentage is female.

The equation used here is straightforward; we count the number of male patients, divide it by the total cases, and then multiply it by 100. This way we get the male percentage. For the female percentage, we simply minus the Male Patients from the Total, then divide it by total and then multiply it by 100, to achieve the total percentage for the female patients.

Code:

```
//BONUS function: displays total active cases in the country
void CVPMSystem::activeCases()
{
	float MPatients = 0.0;
	float active = 0.0;
	CVPatient* curr = front;

	while (curr != NULL)                                    //loop to find to total active
cases and how many of those are male
	{
		if (curr->positivity == true)
		{
			++active;
			if (curr->gender == 'M')
			{
				++MPatients;
			}
		}
		curr = curr->next;
	}

	cout << "\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t  Total number of active cases in Pakistan: "
<< active		//displaying total active cases, percentage of male patients and percentage
of female patients
		<< "\n\n\t\t\t\t\t\t   Male percentage: " << (float)((MPatients / active) * 100)
<< " %"
		<< "\n\n\t\t\t\t\t\t  Female percentage: " << (float)((( active – MPatients) /
active) * 100)) <<" %\n\n";
	Sleep(3500);
}
```

Screenshots:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Total number of active cases in Pakistan: 5
Male percentage: 80 %
Female percentage: 20 %

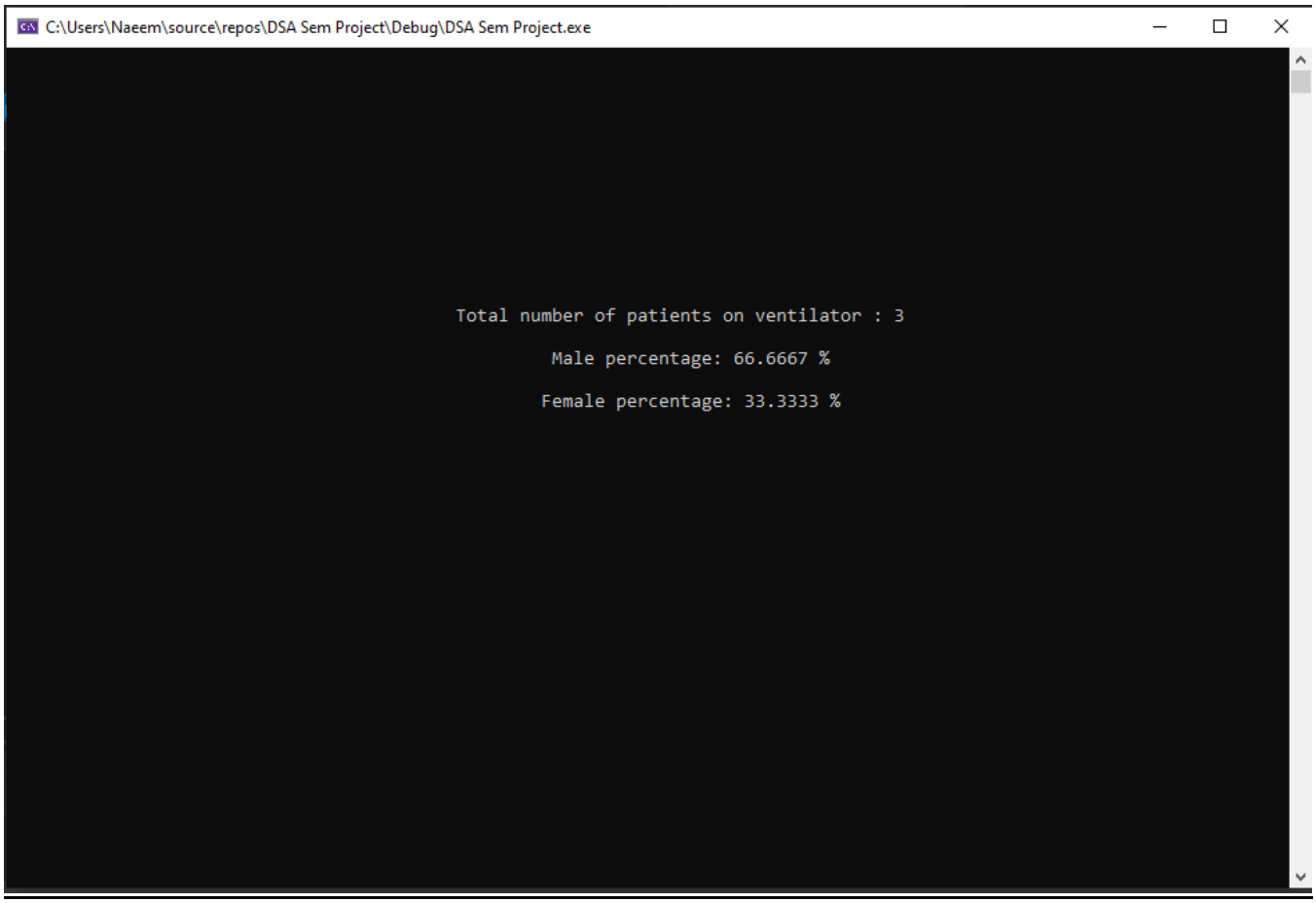
Critical Cases function (Bonus function):

This function goes through the queue to count all the patients who have severe form of COVID19. Patients with severe forms of the virus are the ones who are on ventilator. After that, it displays the total number of patients on ventilator along with what percentage of the them are and male and what percentage is female.

Code:

```
//BONUS function: displays total patients using ventilator (in the country)
void CVPMSystem::criticalPatients()
{
	float MPatients = 0.0;
	float critical = 0.0;
	CVPatient* curr = front;

	while (curr != NULL)                                    //loop to find to total patients
on ventilator and how many of those are male
	{
		if (curr->severity == "Severe")
		{
			++critical;
			if (curr->gender == 'M')
			{
				++MPatients;
			}
		}
		curr = curr->next;
	}

	cout << "\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t  Total number of patients on ventilator : "
<< (int)critical            //displaying total patients on ventilator, percentage of male
patients and percentage of female patients
		<< "\n\n\t\t\t\t\t\t   Male percentage: " << (float)((MPatients / critical) * 100)
<< " %"
		<< "\n\n\t\t\t\t\t\t  Female percentage: " << (float)(((critical - MPatients) /
critical) * 100)) << " %\n\n";
	Sleep(4000);
}
```

Screenshots:


C:\Users\Naeem\source\repos\DSA Sem Project\Debug\DSA Sem Project.exe
Total number of patients on ventilator : 3
Male percentage: 66.6667 %
Female percentage: 33.3333 %

Read From File function:

This is the first function called after the user successfully logs in. This functions reads patient records from the file and stores each record in a separate node. This function also assigns the first node in queue to the Front pointer and the last node in queue to the Rear pointer.

If the .txt file is tampered with, it will crash the program and its functionality. This function is one of the core functions that enables data storing.

Code:

```
//Utility function to read patient records from file and make a queue
void CVPMSystem::readFromFile()
{
	ifstream in(::recordFile, ios::in);
	string line;
	CVPatient* newPatient;

	if (!in)
	{
		cout << "\nError opening file.\n";
	}
	else
	{
		while (!in.eof())
		{
			newPatient = new CVPatient;			//taking input from file
and makling records

			getline(in, newPatient->name);
			getline(in, line);
			newPatient->gender = line[0];
			getline(in, newPatient->city);
			getline(in, line);
			newPatient->patientID = stoi(line);
			getline(in, line);
			newPatient->CRPLevel = stof(line);
			getline(in, line);
			if (line == "1")
			{
				newPatient->fever = true;
			}
			getline(in, line);
			if (line == "1")
			{
				newPatient->dryCough = true;
			}
```

```
                        getline(in, line);
                        if (line == "1")
                        {
                                newPatient->tiredness = true;
                        }
                        getline(in, line);
                        if (line == "1")
                        {
                                newPatient->tastelessness = true;
                        }
                        getline(in, line);
                        if (line == "1")
                        {
                                newPatient->positivity = true;
                        }
                        getline(in, newPatient->severity);


                        if (front == NULL && rear == NULL)                                      //if
queue is empty
                        {
                                front = rear = newPatient;
                        }
                        else
                //if queue is not empty
                        {
                                if (newPatient->name != "")
        //checking if record has data or not (due to empty line in end of file)
                                {
                                        rear->next = newPatient;
                                        rear = newPatient;
                                }
                                else
                //if record is empty delete it
                                {
                                        delete newPatient;
                                }
                        }
                }

                in.close();
        }
}
```

Screenshots:

(Doesn't display anything hence no screenshots).

Write To File function:

When the user exits the program, this function truncates the file that contains all the patient data and then it stores all the data in the queue in file. It writes every information on a separate line, so that it can be compatible with the ReadFromFunction().

Code:

```
//utility function to truncate data from file and write all data in the queue in the file
void CVPMSystem::writeToFile()
{
	ofstream out(::recordFile, ios::trunc);
	CVPatient* obj = front;

	if (!out)
	{
		cout << "\nError opening file.\n";
	}
	else
	{
		while (obj != NULL)
	//storign all queue data in file
		{
			out << obj->name << endl
				<< obj->gender << endl
				<< obj->city << endl
				<< obj->patientID << endl
				<< obj->CRPLevel << endl
				<< obj->fever << endl
				<< obj->dryCough << endl
				<< obj->tiredness << endl
				<< obj->tastelessness << endl
				<< obj->positivity << endl
				<< obj->severity << endl;

			obj = obj->next;
		}

		out.close();
	}
}
```

Screenshots:

(Doesn't display anything hence no screenshots).

Conclusion:

It can be seen that it is indeed possible to create such a patient management system using UML diagrams, classes, text files, data structures, loops and selection statements, composition, and pointers. The program we just created can be used for practical purposes, in hospitals or for Corona Virus Tracking Software. We kept everything in mind while creating this program, from privacy issues of the patients to the user-friendliness for our doctors or admins. This was truly a good project given by our esteemed lecturer – Sikander Hayat. This project enabled us to understand the practical uses of data structure and how to implement and solve real world problems with it.

Program code for testing:

Doctor.h header file:

```
#pragma once
#include <iostream>

using namespace std;

class Doctor                        //class to store login data of the user
{
public:
	string name;
	string city;
	string ID;
	string password;

	Doctor();
	//constructors
	Doctor(string a, string b, string c, string d);
};
```

CVPatient.h header file:

```
#pragma once
#include <iostream>

using namespace std;

class CVPatient                                 //class to store patient data. This class acts
like the node
{
public:
	string name;
	char gender;
	string city;
	int patientID;
	float CRPLevel;
	bool fever;
	bool dryCough;
	bool tiredness;
	bool tastelessness;
	bool positivity;
	string severity;
	CVPatient* next;                    //pointer to next node


	CVPatient();                                //Constructors
```

```
	CVPatient(string a, char b, string c, int d, float e, bool f, bool g, bool h, bool i,
bool j, string k);

};
```

CVPMSystem.h header file:

```
#pragma once
#include <iostream>
#include "Doctor.h"
#include "CVPatient.h"

using namespace std;


class CVPMSystem                        //class for entire system. This class acts like queue
class
{
public:
	Doctor user;                        //Composition. This object stores data of user who is
using the program
	CVPatient* front;                   //Composition. This pointer points to the first node of
the queue
	CVPatient* rear;                    //Composition. This pointer points to the last node of
the queue

	CVPMSystem();                       //constructor and destructor
	~CVPMSystem();
	void signup();                              //all function prototypes
	void login();
	void TestPatient(CVPatient* obj);
	void patientInput(CVPatient* obj);
	void addPatient();
	void displayRecord(CVPatient* obj);
	void deletePatient();
	void searchPatient();
	void editPatient();
	void mostPatients();
	void activeCases();
	void criticalPatients();
	void readFromFile();
	void writeToFile();

};
```

Source.cpp file:

```
#include <iostream>
```

```
#include <windows.h>                    //to use Sleep function
#include <string>                       //to use functions like getline
#include <fstream>                      //for filing
#include "Doctor.h"
#include "CVPatient.h"
#include "CVPMSystem.h"


using namespace std;


string loginFile = "Credentials.txt";                   //the file with all the ID and
Passwords used for login

string recordFile = "PatientRecord.txt";                //this file stores all patient
data



//Default constructor for Doctor class
Doctor::Doctor() : name(""), city(""), ID(""), password("")
{}

//Paremeterized constructor for Doctor class
Doctor::Doctor(string a, string b, string c, string d) : name(a), city(b), ID(c), password(d)
{}
```

```
//Default constructor for CVPatient class

CVPatient::CVPatient() : name(""), gender('M'), city(""), patientID(0), CRPLevel(10.0), fever(0),
dryCough(0), tiredness(0), tastelessness(0), positivity(0), severity("NA"), next(NULL)

{}


//Paremeterized constructor for CVPatient class

CVPatient::CVPatient(string a, char b, string c, int d, float e, bool f, bool g, bool h, bool i, bool j, string k)
: name(a), gender(b), city(c), patientID(d), CRPLevel(e), fever(f), dryCough(g), tiredness(h),
tastelessness(i), positivity(j), severity(k), next(NULL)

{}









//Default constructor for CVSystem class

CVPMSystem::CVPMSystem() : front(NULL), rear(NULL)

{}


//Destructor for CVSystem class

CVPMSystem::~CVPMSystem()

{

        if (front != NULL)
```

```
	{
		while (front != NULL)				//loop to delete all nodes
		{
			CVPatient* temp = front;
			front = front->next;
			delete temp;
		}
	}
}

//Function to add new login info in permanent storage
void CVPMSystem::signup()				//can be connected to the databse of hospitals to confirm new signups match with records in database
{
	Sleep(500);
	system("CLS");					//waiting for 0.5 second and then clearing the screen

	cin.ignore();
	cout << "\n\tEnter name: ";
	getline(cin, user.name);
	cin.ignore();
	cout << "\n\tEnter city: ";
	getline(cin, user.city);
	cin.ignore();
	cout << "\n\tEnter ID: ";
	getline(cin, user.ID);
	cin.ignore();
```

```
	cout << "\n\tEnter password: ";
	getline(cin, user.password);

	ofstream out(::loginFile, ios::app);
	if (!out)
	{
		cout << "\n\tError opening file.\n";
	}
	else
	//sotring new sign up information in file
	{
		out << user.name << endl
			<< user.city << endl
			<< user.ID << endl
			<< user.password << endl;
		out.close();

		cout << "\n\n\n\n\t\t\t\t\t    New login credentials added.\n";
		Sleep(1500);
	}
}

//Function for logging in, signing up or exiting the program. Program won't go ahead of this function until user logs in
void CVPMSystem::login()
{
	int check = 0;
```

```
	while (check != 1)
	{
		Sleep(500);
		system("CLS");
		int option;
		Doctor temp;

		while (1)				//loop asking user to login, signup or exit. User cannot go ahead if he does not login with correct ID and pass
		{
			cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t     Enter 1 to login, 2 to sign up, or 0 to exit\n\n\t\t\t\t\t\t\t\t   ";
			cin >> option;

			if (option == 1)
			{
				break;
			}
			else if (option == 2)		//sign up and then restarting loop so user can login
			{
				signup();
				user.name = "";
				Sleep(500);
				system("CLS");
				continue;
			}
			else if (option == 0)		//exitting
			{
```

```
                Sleep(500);
                system("CLS");
                cout <<
"\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\t\tGoodbye.\n\n\n\n\n\n\n\n\n\n\n\n";
                exit(0);
            }
            else                                                    //in case of
wrong input
            {
                cout << "\n\n\t\t\t\t\t\t\tIncorrect input. Try again.\n";
                Sleep(2000);
                system("CLS");
            }
        }

        Sleep(500);
        system("CLS");

        cin.ignore();
        cout << "\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\t  Enter ID: ";          //taking login
input
        getline(cin, user.ID);
        cin.ignore();
        cout << "\n\n\t\t\t\t\t\tEnter password: ";
        getline(cin, user.password);

        ifstream in(::loginFile, ios::in);
        while (!in.eof())
```

```
		{
			getline(in, temp.name);					//taking input from file
			getline(in, temp.city);
			getline(in, temp.ID);
			getline(in, temp.password);

			if (temp.ID == user.ID && temp.password == user.password)
	//if match found in file, program welcomes user
			{
				in.close();
				check = 1;
				user.name = temp.name;
				user.city = temp.city;
				temp.name = temp.name.substr(0, temp.name.find(" "));
				Sleep(500);
				system("CLS");
				cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tWelcome, Dr. " << temp.name << endl;
				Sleep(500);
				break;
			}
		}

		if (in.eof() && user.name == "")				//if no login match found, loop restarts
		{
			Sleep(500);
```

```
                system("CLS");
                cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t   Entered username and password don't match any record.\n";
                Sleep(2000);
                system("CLS");
            }
        }
}

//Function tests patients for COVID19 when reocrd is added or updated
void CVPMSystem::TestPatient(CVPatient* obj)
{
        if ((obj->CRPLevel >= 11.0) && (obj->fever == true || obj->dryCough == true || obj->tiredness == true || obj->tastelessness == true))
        {
            obj->positivity = true;

            if (obj->CRPLevel >= 11.0 && obj->CRPLevel < 15.0)                    //CRP tests are used to check if patient has COVID19 or not (10 is normal level, values abve 10 mean patient has virus)
            {
                obj->severity = "Mild";
            }
            else if (obj->CRPLevel >= 15.0 && obj->CRPLevel < 20.0)
            {
                obj->severity = "Moderate";
            }
            else
            {
```

```
                obj->severity = "Severe";
            }

            cout << "\n\n\n\tTest results: ";
            cout << "\n\n\tCOVID-19 : Positive";
            cout << "\n\n\tSeverity : " << obj->severity << "\n\n";
        }
        else
        {
            cout << "\n\n\n\tTest results: ";
            cout << "\n\n\tCOVID-19 : Negative";
            cout << "\n\n\tSeverity : NA\n\n";
        }
}

//Utility function to take input from user. Used in Add and Edit functions
void CVPMSystem::patientInput(CVPatient* obj)
{
        string temp;
        int option;
        char symptom;

        cin.ignore();
        cout << "\n\n\n\tEnter name: ";
        getline(cin, temp);
        if (temp != "X" && temp != "x")                    //if user enter X or x the value is
kept unchanged
```

```
	{
		obj->name = temp;
	}

	cin.ignore();
	cout << "\n\tEnter gender (M or F): ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		if (temp == "M" || temp == "m")
		{
			obj->gender = 'M';
		}
		else
		{
			obj->gender = 'F';
		}
	}

	cin.ignore();
	cout << "\n\tEnter 1 if the city is Islamabad, \n\t2 if it's Lahore, \n\t3 if it's Karachi, \n\t4 if it's Peshawar, \n\tor 5 if it's Quetta \n\n\t	";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		option = stoi(temp);
```

```
        if (option == 1)
        {
                obj->city = "Islamabad";
        }
        else if (option == 2)
        {
                obj->city = "Lahore";
        }
        else if (option == 3)
        {
                obj->city = "Karachi";
        }
        else if (option == 4)
        {
                obj->city = "Peshawar";
        }
        else
        {
                obj->city = "Quetta";
        }
}

cin.ignore();
cout << "\n\tEnter patient ID : ";
cin >> temp;
if (temp != "X" && temp != "x")
{
```

```
        obj->patientID = stoi(temp);                    //stoi converts string to integer
}

cin.ignore();
cout << "\n\tEnter patient's Blood CRP (inflammation) level (mg/L) : ";
cin >> temp;
if (temp != "X" && temp != "x")
{
        obj->CRPLevel = stof(temp);                     //stoi converts string to float
}

cin.ignore();
cout << "\n\tDoes the patient show signs of fever? (Y or N): ";
cin >> temp;
if (temp != "X" && temp != "x")
{
        if (temp == "Y" || temp == "y")                                 //setting bool values
        {
                obj->fever = true;
        }
        else
        {
                obj->fever = false;
        }
}

cin.ignore();
```

```
cout << "\n\tDoes the patient show signs of dry cough? (Y or N): ";
cin >> temp;
if (temp != "X" && temp != "x")
{
	if (temp == "Y" || temp == "y")
	{
		obj->dryCough = true;
	}
	else
	{
		obj->dryCough = false;
	}
}

cin.ignore();
cout << "\n\tDoes the patient show signs of chronic fatigue? (Y or N): ";
cin >> temp;
if (temp != "X" && temp != "x")
{
	if (temp == "Y" || temp == "y")
	{
		obj->tiredness = true;
	}
	else
	{
		obj->tiredness = false;
	}
```

```
	}

	cin.ignore();
	cout << "\n\tDoes the patient show signs of tastelessness? (Y or N): ";
	cin >> temp;
	if (temp != "X" && temp != "x")
	{
		if (temp == "Y" || temp == "y")
		{
			obj->tastelessness = true;
		}
		else
		{
			obj->tastelessness = false;
		}
	}

	this->TestPatient(obj);                                    //setting positivty and severity of CV

	Sleep(2000);
}

//Function to add new patient record. It calls the Input function then connects new node in the end
void CVPMSystem::addPatient()
{
	CVPatient* newPatient = new CVPatient;
	patientInput(newPatient);
```

```
	if (front == NULL && rear == NULL)
	{
		front = rear = newPatient;
	}
	else
	{
		rear->next = newPatient;
		rear = newPatient;
	}
	cout << "\n\n\t\t\t\t\t\tNew patient record added.\n";
	Sleep(1500);

}

//Utility function used in Edit, Delete and Search
void CVPMSystem::displayRecord(CVPatient* obj)
{
	cout << "\n\n\n\t\t\t\t\tPatient name: " << obj->name
		<< "\n\n\t\t\t\t\tGender: " << obj->gender
		<< "\n\n\t\t\t\t\tCity: " << obj->city
		<< "\n\n\t\t\t\t\tPatient ID: " << obj->patientID
		<< "\n\n\t\t\t\t\tBlood CRP level (mg/L) : " << obj->CRPLevel;

	cout << "\n\n\t\t\t\t\tFever: ";
	if (obj->fever == true)
	{
```

```
		cout << "Y";
	}
	else
	{
		cout << "N";
	}

	cout << "\n\n\t\t\t\t\t\tDry cough: ";
	if (obj->dryCough == true)
	{
		cout << "Y";
	}
	else
	{
		cout << "N";
	}

	cout << "\n\n\t\t\t\t\t\tChronic Fatigue: ";
	if (obj->tiredness == true)
	{
		cout << "Y";
	}
	else
	{
		cout << "N";
	}
```

```
	cout << "\n\n\t\t\t\t\t\tTastelessness: ";
	if (obj->tastelessness == true)
	{
		cout << "Y";
	}
	else
	{
		cout << "N";
	}

	cout << "\n\n\t\t\t\t\t\tCOVID-19 test: ";
	if (obj->positivity == true)
	{
		cout << "Positive";
	}
	else
	{
		cout << "Negative";
	}

	cout << "\n\n\t\t\t\t\t\tSeverity: " << obj->severity << "\n\n";

}

//Function for deleting node of a patient
void CVPMSystem::deletePatient()
{
```

```
	if (front == NULL)
	{
		cout << "\nNo data in queue.\n";
	}
	else
	{
		CVPatient* curr = front;				//used to find the node to delete and the node before it
		CVPatient* prev = front;

		int IDnum;
		int check = 0;

		cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tEnter identification number of patient: ";
		cin >> IDnum;

		while (curr != NULL)				//finding node to delete
		{
			if (curr->patientID == IDnum)		//match found
			{
				break;
			}
			else						//match not found yet
			{
				prev = curr;
				curr = curr->next;
```

```
                }
        }

        if (curr == NULL)                                   //no match found in all the
records
        {
                cout << "\n\n\t\t\t\t\t\tNo match found.\n";
        }
        else
    //match found
        {
                displayRecord(curr);
                cout << "\n\n\n\n\t\t\t\t   Enter 1 to confirm deletion or 2 to cancel:
\n\n\t\t\t\t\t\t\t";          //taking user confirmation after displaying record
                cin >> check;

                if (check == 1)
                {
                        if (front == rear)                  //only one node in queue
                        {
                                front = rear = NULL;
                        }
                        else if (curr == front && front != rear)          //first node to be
deleted
                        {
                                front = front->next;
                        }
                        else if (curr == rear && front != front)          //last node to be
deleted
```

```
                                {
                                        prev->next = NULL;
                                        rear = prev;
                                }
                                else
        //deleting from the middle
                                {
                                        prev->next = curr->next;
                                }

                                delete curr;
                                cout << "\n\n\n\t\t\t\t\t\tRecord deleted.\n";
                        }
                        else
                        {
                                cout << "\n\n\n\t\t\t\t\t\tOperation cancelled.\n";
                        }
                        Sleep(500);
                }
        }
}

//Function to find and display record of a patient. It looks for the record in the queue
void CVPMSystem::searchPatient()
{
        if (front == NULL)
        {
```

```
		cout << "\nNo data in queue.\n";
	}
	else
	{

		CVPatient* curr = front;
		int IDnum;

		cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tEnter identification number of patient:
";
		cin >> IDnum;

		while (curr != NULL)                                    //loop to find node with
matching ID number
		{
			if (curr->patientID == IDnum)
			{
				break;
			}
			else
			{
				curr = curr->next;
			}
		}

		if (curr == NULL)                                       //in case no match is found
		{
			cout << "\n\n\n\t\t\t\t\t\t     No match found.\n";
```

```
                Sleep(500);
            }
            else
            {
                cout << "\n\n\tRecord match found: \n\n";          //displaying record if match
found
                displayRecord(curr);
                Sleep(4500);
            }
        }
}

//Function to edit the record of a patient. It looks for the record in the queue.
void CVPMSystem::editPatient()
{
        if (front == NULL)
        {
            cout << "\nNo data in queue.\n";
        }
        else
        {

            CVPatient* curr = front;

            int IDnum;

            cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tEnter identification number of patient:
";
```

```
		cin >> IDnum;

		while (curr != NULL)					//loop to find the node to be edited
		{
			if (curr->patientID == IDnum)
			{
				break;
			}
			else
			{
				curr = curr->next;
			}
		}

		if (curr == NULL)					//if no match is found in the queue
		{
			cout << "\n\n\n\t\t\t\t\t\t     No match found.\n";
		}
		else							//if match found, show record, take new input and then show modified record
		{
			cout << "\n\n\tRecord match found: \n\n";
			displayRecord(curr);
			cout << "\n\n\n\tStarting new input. Enter 'X' for the values you want to keep the same.\n";
			patientInput(curr);
			cout << "\n\n\tAfter modification: \n\n";
```

```
                displayRecord(curr);
                Sleep(4500);
            }
        }
}

//Function to find the city with the highest number of cases
void CVPMSystem::mostPatients()
{
    if (front == NULL)
    {
        cout << "\nNo data in queue.\n";
    }
    else
    {
        CVPatient* ptr = front;
        string cities[5] = { "Islamabad" , "Lahore" , "Karachi" , "Peshawar" , "Quetta" };
    //array with city names
        int patients[5];
                                                    // array to store patients for each city
        int largestIndex = 0;
        int largest = 0;

        for (int i = 0; i < 5; ++i)                 //initializing patient array with 0
        {
            patients[i] = 0;
        }
```

```
        while (ptr != NULL)                                    //counting patients for each
city
        {
            if (ptr->city == "Islamabad")
            {
                ++patients[0];
            }
            else if (ptr->city == "Lahore")
            {
                ++patients[1];
            }
            else if (ptr->city == "Karachi")
            {
                ++patients[2];
            }
            else if (ptr->city == "Peshawar")
            {
                ++patients[3];
            }
            else
            {
                ++patients[4];
            }
            ptr = ptr->next;
        }

        for (int i = 0; i < 5; ++i)                    //finding highest patient count and the city
```

```
		{
			if (largest < patients[i])
			{
				largest = patients[i];
				largestIndex = i;
			}
		}

		cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t   The city with the most COVID-19 patients is '" << cities[largestIndex] << "' with '" << largest << "' patients.\n\n";
		Sleep(3000);
	}
}

//BONUS function: displays total active cases in the country
void CVPMSystem::activeCases()
{
	float MPatients = 0.0;
	float active = 0.0;
	CVPatient* curr = front;

	while (curr != NULL)                                    //loop to find to total active cases and how many of those are male
	{
		if (curr->positivity == true)
		{
			++active;
			if (curr->gender == 'M')
```

```
                {
                        ++MPatients;
                }
        }
        curr = curr->next;
    }

    cout << "\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t  Total number of active cases in Pakistan: " << active        //displaying total active cases, percentage of male patients and percentage of female patients
        << "\n\n\t\t\t\t\t\t\t   Male percentage: " << (float)((MPatients / active) * 100) << " %"
        << "\n\n\t\t\t\t\t\t\t  Female percentage: " << (float)(((active – MPatients) / active) * 100)) <<" %\n\n";
    Sleep(3500);
}

//BONUS function: displays total patients using ventilator (in the country)
void CVPMSystem::criticalPatients()
{
    float MPatients = 0.0;
    float critical = 0.0;
    CVPatient* curr = front;

    while (curr != NULL)                                    //loop to find to total patients on ventilator and how many of those are male
    {
        if (curr->severity == "Severe")
        {
```

```
                ++critical;
                if (curr->gender == 'M')
                {
                        ++MPatients;
                }
        }
        curr = curr->next;
    }

    cout << "\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t  Total number of patients on ventilator : " <<
(int)critical           //displaying total patients on ventilator, percentage of male patients and
percentage of female patients
        << "\n\n\t\t\t\t\t\t   Male percentage: " << (float)((MPatients / critical) * 100) << " %"
        << "\n\n\t\t\t\t\t\t  Female percentage: " << (float)(((critical – MPatients) / critical) *
100)) << " %\n\n";
    Sleep(4000);
}

//Utility function to read patient records from file and make a queue
void CVPMSystem::readFromFile()
{
    ifstream in(::recordFile, ios::in);
    string line;
    CVPatient* newPatient;

    if (!in)
    {
        cout << "\nError opening file.\n";
```

```
	}
	else
	{
		while (!in.eof())
		{
			newPatient = new CVPatient;				//taking input from file and
makling records

			getline(in, newPatient->name);
			getline(in, line);
			newPatient->gender = line[0];
			getline(in, newPatient->city);
			getline(in, line);
			newPatient->patientID = stoi(line);
			getline(in, line);
			newPatient->CRPLevel = stof(line);
			getline(in, line);
			if (line == "1")
			{
				newPatient->fever = true;
			}
			getline(in, line);
			if (line == "1")
			{
				newPatient->dryCough = true;
			}
			getline(in, line);
```

```
                    if (line == "1")
                    {
                            newPatient->tiredness = true;
                    }
                    getline(in, line);
                    if (line == "1")
                    {
                            newPatient->tastelessness = true;
                    }
                    getline(in, line);
                    if (line == "1")
                    {
                            newPatient->positivity = true;
                    }
                    getline(in, newPatient->severity);


                    if (front == NULL && rear == NULL)                                  //if
queue is empty
                    {
                            front = rear = newPatient;
                    }
                    else
            //if queue is not empty
                    {
                            if (newPatient->name != "")
        //checking if record has data or not (due to empty line in end of file)
                            {
```

```
                                        rear->next = newPatient;
                                        rear = newPatient;
                                }
                                else
                        //if record is empty delete it
                                {
                                        delete newPatient;
                                }
                        }
                }

                in.close();
        }
}

//utility function to truncate data from file and write all data in the queue in the file
void CVPMSystem::writeToFile()
{
        ofstream out(::recordFile, ios::trunc);
        CVPatient* obj = front;

        if (!out)
        {
                cout << "\nError opening file.\n";
        }
        else
        {
```

```
            while (obj != NULL)
        //storign all queue data in file
            {
                out << obj->name << endl
                    << obj->gender << endl
                    << obj->city << endl
                    << obj->patientID << endl
                    << obj->CRPLevel << endl
                    << obj->fever << endl
                    << obj->dryCough << endl
                    << obj->tiredness << endl
                    << obj->tastelessness << endl
                    << obj->positivity << endl
                    << obj->severity << endl;

                obj = obj->next;
            }

            out.close();
        }
}
```

Main.cpp file:

```
#include <iostream>
#include <windows.h>                        //to use Sleep function
#include "CVPMSystem.h"

using namespace std;
```

```
int main()
{
	CVPMSystem obj;
	int option = 0;

	cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t     Weclome to Corona Virus Patient
Management System";
	Sleep(2000);

	obj.login();                                    //start login function. User cannot
access program without logging in with correct ID nad password
	obj.readFromFile();                             //reading patient data from file and
making queue aftewr the user successfully logged in

	while (1)
	{
		Sleep(500);
		system("CLS");

		cout << "\n\n\n\n\n\n\n\t\t\t\t\t      Enter 1 to add patient record, "
			//Main menu
			<< "\n\n\t\t\t\t\t\t2 to delete patient record, "
			<< "\n\n\t\t\t\t\t        3 to search patient record, "
			<< "\n\n\t\t\t\t\t\t 4 to edit patient record, "
			<< "\n\n\t\t\t\t      5 to find the city with the most COVID-19 cases, "
			<< "\n\n\t\t\t\t      6 to find total number of active COVID-19 cases, "
			<< "\n\n\t\t\t\t      7 to total number of patients on ventilator, or"
			<< "\n\n\t\t\t\t\t\t\t0 to exit.\n\n\n\t\t\t\t\t\t\t     ";
		cin >> option;

		Sleep(500);                                                    //adding a slight
delay and clearing screen
		system("CLS");

		if (option == 1)                                        //starting functions
according to user choice
		{
			obj.addPatient();
		}
		else if (option == 2)
		{
			obj.deletePatient();
		}
		else if (option == 3)
		{
			obj.searchPatient();
		}
		else if (option == 4)
		{
			obj.editPatient();
		}
		else if (option == 5)
		{
			obj.mostPatients();
		}
		else if (option == 6)
```

```
			{
				obj.activeCases();
			}
			else if (option == 7)
			{
				obj.criticalPatients();
			}
			else
			{
				break;
			}

		}

		Sleep(300);
		system("CLS");
		cout <<
"\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\t\tGoodbye.\n\n\n\n\n\n\n\n\n\n\n\n";

		obj.writeToFile();				//After user exits, patient record file
is truncated and queue data is stored in file
		obj.~CVPMSystem();				//After queue data is stored in file,
destructor is invoked to free memory.

}
```

Required text files:

```
Credentials.txt		= for logging in

PatientRecord.txt	= for storing patient data
```

Just make two empty text files in project folder with these names. When logging in, **sign up first**. When using other functions, **use the addPatient() function before anything else** to add records.