



ENGINEERING FACULTY

Course Work

Course: Programming

Student: Jegors Guzovs

St. Code: 81948

Group: 4201BDA

Riga

2022

Contents

1. Requirements analysis for the subject area and task definition.....	3
2. Architecture and design	6
3. Detailed design	9
4. Implementation	12
5. Testing	16
6. User manual	23
7. Conclusion	27
8. Source code.....	29

1. Requirements analysis for the subject area and task definition

Subject Area: Patient Management System

The subject area of the application is the management of patient records.

The system will store and manage information about patients, including their code, surname, name, date of admission, illness, and doctor.

Objects and Properties:

Object: Patient

Properties:

code: Integer (unique identifier for each patient)

surname: String (patient's surname)

name: String (patient's name)

dateAdmissions: String (date of admission in the format dd/mm/yyyy)

illness: String (patient's illness)

doctor: String (name of the doctor treating the patient).

File Structure:

The patient data will be stored in a text file named "patients.txt".

Each line in the file represents a patient record and contains comma-separated values for each property of the patient object.

Table 1

Application Functionality

Function	Description
Display patient data	The application will display the details of all patients stored in the system.
Add new patient	Users can add a new patient to the system by providing the required information.
Edit patient data	Users can edit the details of a specific patient by searching for the patient using their code and updating the relevant information.
Delete patient	Users can delete a patient from the system based on their code, surname, or illness.
Search patients	Users can search for patients based on their code, surname, or illness and display the matching results.
Sort patients	Users can sort the patient records based on their code, surname, or illness and display the sorted list.
Calculate statistics	Users can calculate statistics related to the patient data, such as the average number of patients per doctor or the total number of patients for a specific illness.
Save data to file	Users can save the patient data to the "patients.txt" file.
Load data from file	Users can load previously saved patient data from the "patients.txt" file.

Invalid Input Handling:

When the user enters an invalid choice in the menu, the program displays the message "Invalid choice" and prompts the user again to enter a valid choice.

Information and Warning Messages:

When loading patient data from a file, the program displays a message indicating the number of patients loaded or an error message if the file cannot be opened.

When adding a new patient, the program displays a message confirming the successful addition of the patient or informs the user if the maximum capacity has been reached.

When editing a patient, the program displays a message confirming the successful update of the patient's information or informs the user if the specified patient code is not found.

When deleting a patient, the program displays a message confirming the deletion of the patient or informs the user if no patient matches the specified code, surname, or illness.

When searching for patients, the program displays a message indicating the number of matching patients found or informs the user if no matches are found.

Error Handling:

The program handles potential errors during file operations, such as opening, reading, or writing to the "patients.txt" file. It displays an error message if there is an issue with file operations.

Prompts and User Interaction:


The program displays a menu of options and prompts the user to enter their choice.

After executing each operation (e.g., displaying patients, adding, editing, deleting, etc.), the program prompts the user to press any key to continue, providing an opportunity to review the results before proceeding.

2. Architecture and design

It is going to be a single .cpp file program with all source code.

Here are screenshots of the User Interface structure:

A screenshot of a terminal window showing a main menu for a patient management program. The text is white on a black background. It starts with 'Data loaded from file.', followed by a numbered list of 9 options: 1. Display patient data, 2. Add new patient, 3. Edit patient data, 4. Delete patient, 5. Search patients, 6. Sort patients, 7. Calculate stats, 8. Save data to file, and 9. Exit. The prompt 'Enter choice:' is at the bottom.

```
Data loaded from file.  
1. Display patient data  
2. Add new patient  
3. Edit patient data  
4. Delete patient  
5. Search patients  
6. Sort patients  
7. Calculate stats  
8. Save data to file  
9. Exit  
Enter choice:
```

Pic.1. Screenshot of a main menu

```

8. Save data to file
9. Exit
Enter choice: 1

Code      Surname      Name      Date of Admission  Illness      Doctor
2345      Doe           John      23/05/2021         Covid        Dr.Smith
8953      Brown         Logan     02/02/2015         Asthma       Dr.Smith
2555      White         Henry     18/11/2018         Diabetes     Dr.Clark
9892      Lewis         Noah      09/07/2017         Arthritis    Dr.Thompson

Press any key to continue...

```

Pic.2. Screenshot of display patients functionality

```

Enter choice: 2

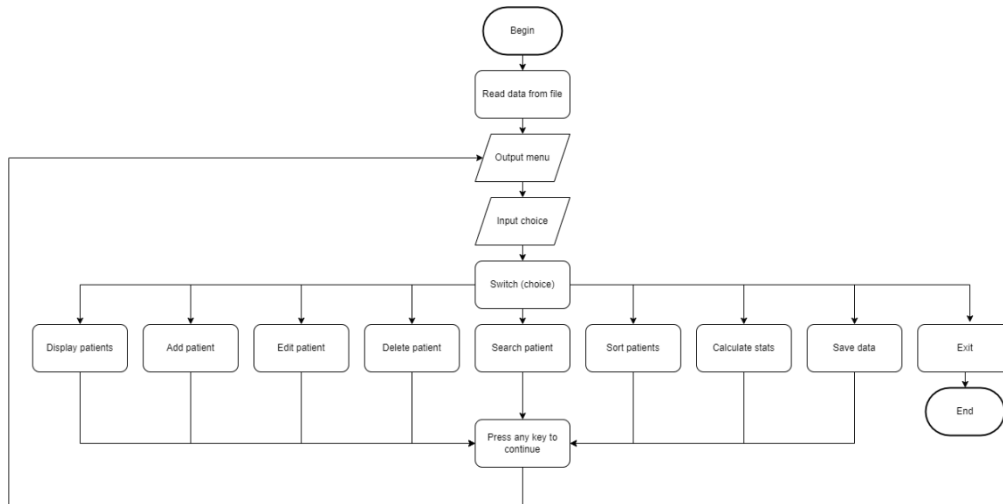
Enter patient code: 2152
Enter patient surname: Moore
Enter patient name: Jackson
Enter patient date of admissions (dd/mm/yyyy): 29/04/2023
Enter patient illness: Covid
Enter patient doctor: Dr.Smith
Added new patient with code 2152

Press any key to continue...

```

Pic.3. Screenshot of add a patient functionality

Structure of 'main' Function Body:



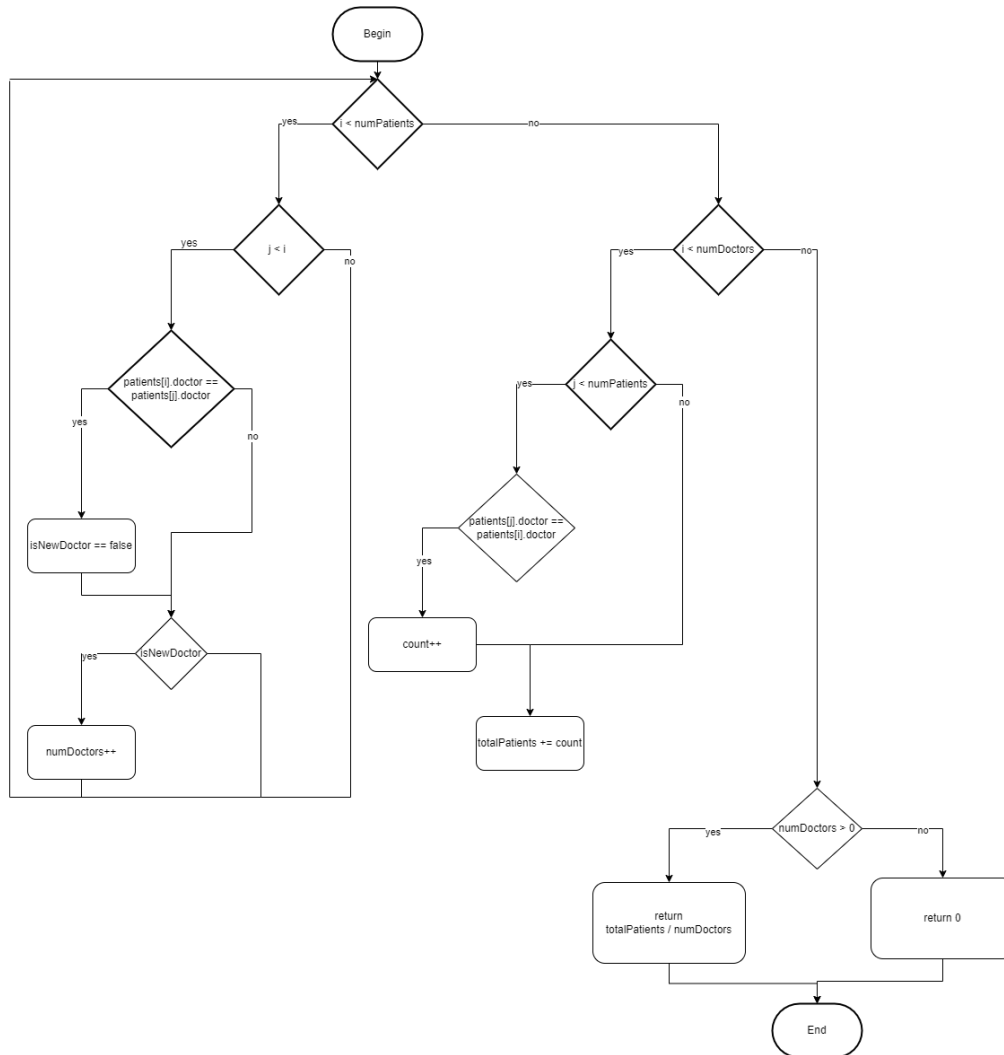
Pic.4. Structure of the 'main' function body

Here are the preliminary prototypes for the user-defined functions in the program:

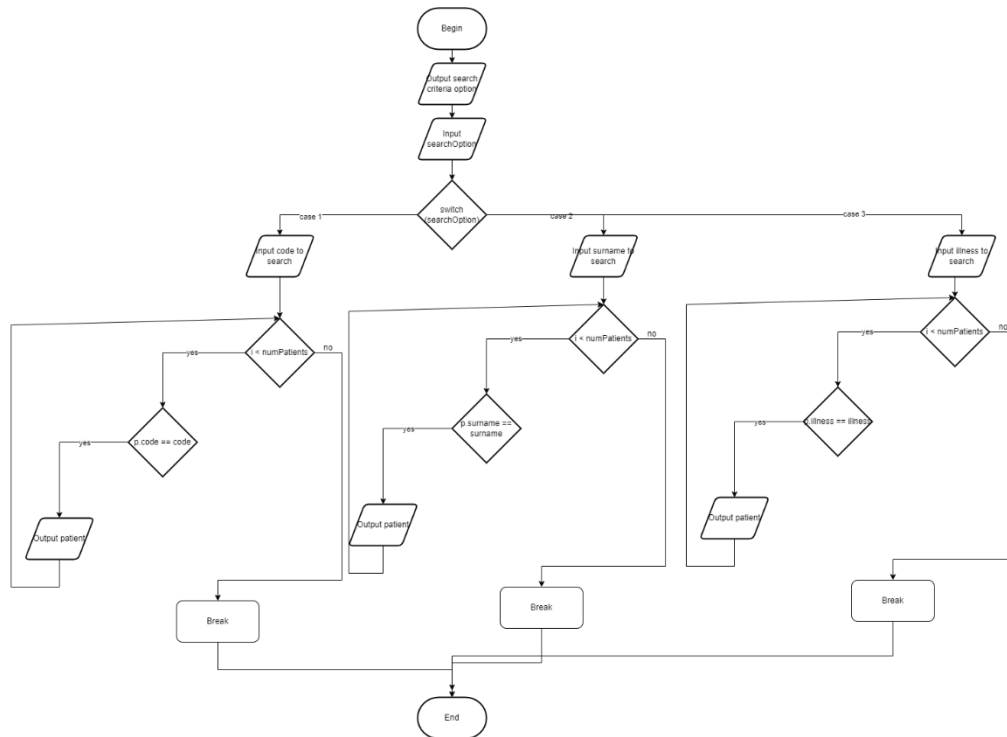
```

void loadPatientsFromFile(string filename);
void displayPatients();
void addPatient();
void editPatient();
void deletePatient();
void searchPatients();
void sortPatients();
float calculateAveragePatientsPerDoctor();
int calculateTotalPatientsForIllness();
void calculateStats();
void saveDataToFile();
void readDataFromFile();
  
```

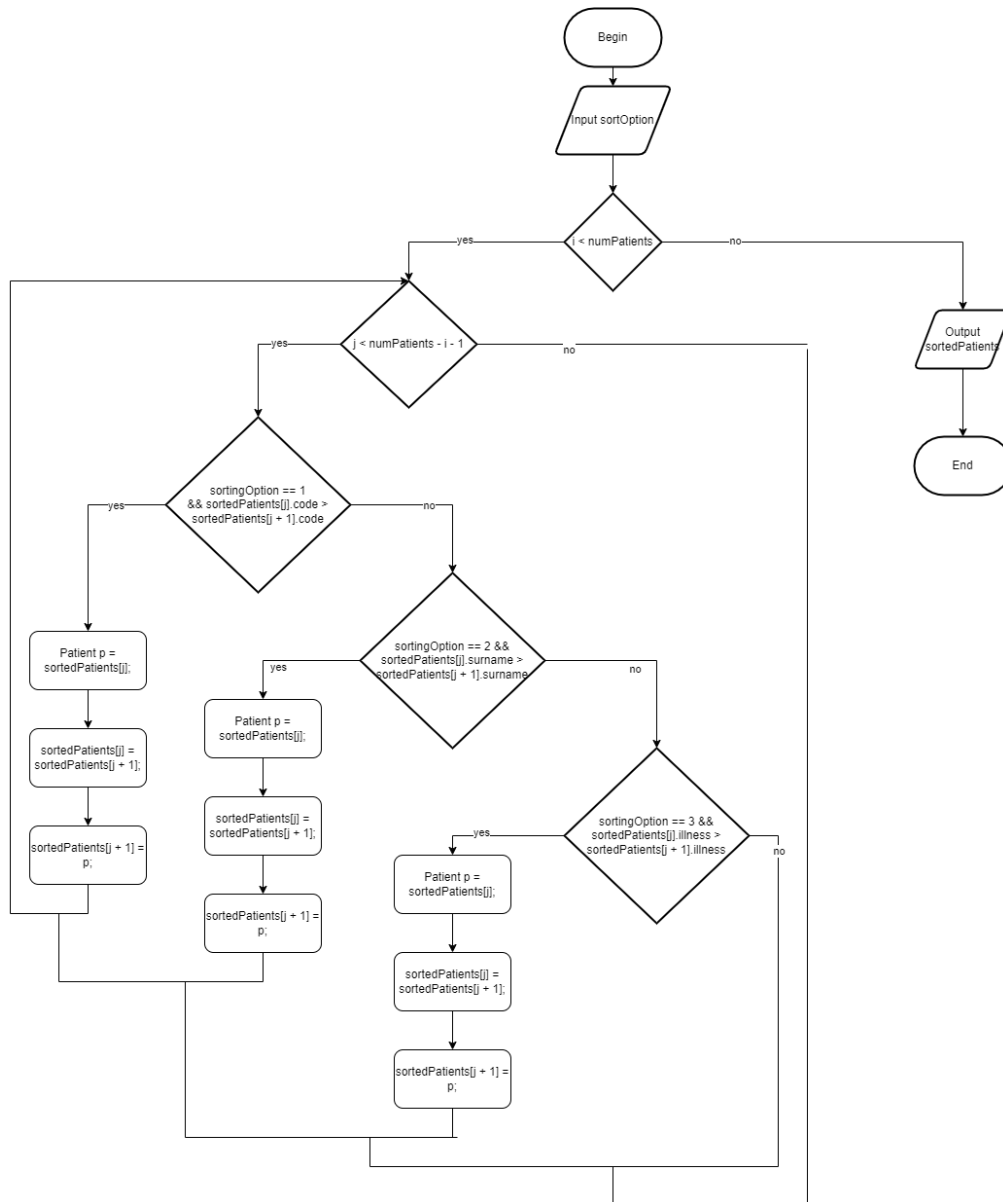

3. Detailed design



Pic.5. Algorithm for calculateAveragePatientsPerDoctor



Pic.6. Algorithm for searchPatients



Pic.7. Algorithm for sortPatients

4. Implementation

Description of Created Project File:

The project file contains Source Files and patients.txt file which is used to store data about patients.

Created .cpp Source Files:

Main.cpp.

Created Data Types:

Patient – a structure.

patients – array of patient objects.

Table 2

Created Functions

Function Name	Description
pressAnyKeyToContinue()	Displays a message and waits for any key press
separator(int length)	Prints a separator of given length
loadPatientsFromFile(string filename)	Loads patient data from a file into the program
displayPatients()	Displays the details of all patients
addPatient()	Adds a new patient to the system
editPatient()	Edits the details of a specific patient

Function Name	Description
<code>deletePatientAtIndex(int index)</code>	Deletes a patient at the specified index
<code>deletePatientsByCode(int code)</code>	Deletes patients based on the given code
<code>deletePatientsBySurname(const string& surname)</code>	Deletes patients based on the given surname
<code>deletePatientsByIllness(const string& illness)</code>	Deletes patients based on the given illness
<code>deletePatient()</code>	Provides options to delete a patient based on different criteria
<code>searchPatients()</code>	Provides options to search for patients based on different criteria
<code>sortPatients()</code>	Provides options to sort the patient records based on different criteria
<code>calculateAveragePatientsPerDoctor(const Patient* patients, int numPatients)</code>	Calculates the average number of patients per doctor
<code>calculateTotalPatientsForIllness(const Patient* patients, int numPatients, const string& illness)</code>	Calculates the total number of patients for a given illness
<code>calculateStats()</code>	Provides options to calculate various statistics on the patient data

Function Name	Description
saveDataToFile()	Saves the patient data to a file
readDataFromFile()	Reads patient data from a file

Table 3

Included .h Files

Header File	Description
iostream	Input/output stream operations
fstream	File stream operations
string	String operations
sstream	String stream operations
iomanip	Input/output manipulators
conio.h	Console input/output operations (non-standard)

Table 4

Used Existing Functions

Existing Function	Description
cout	Standard output stream object
cin	Standard input stream object
endl	Manipulator to insert a new line
_getch()	Reads a character from the console (non-standard)
ifstream	Input file stream object
ofstream	Output file stream object
setw	Manipulator to set field width
left	Manipulator to left-align output

5. Testing

Table 5

Test for adding a new patient

Test N	Sequence of actions	Results	
	Input 2 from main menu. Input code 3215, surname Hamilton, name Lewis, date 12/08/202 2, illness Covid, doctor Dr.Smith	Expected	Actual
1		Added new patient with code 3215	Added new patient with code 3215
		Actual result is correct/incorrect?	<i>Correct</i>

Passed.

Table 6

Test for editing a patient

Test N	Sequence of actions	Results	
		Expected	Actual
2	Input 3 from main menu. Input code 3215, surname Hamilton, name Lewis, date 13/08/202 2, illness Covid, doctor Dr.Clack	Patient with code 3215 has been updated.	Patient with code 3215 has been updated.
		Actual result is correct/incorrect?	<i>Correct</i>

Passed.

Table 7

Test for saving the data

Test N	Sequence of actions	Results	
		Expected	Actual
3	Input 8 in the main menu	Data saved to file.	Data saved to file.
		Actual result is correct/incorrect?	<i>Correct</i>

Passed.

Table 8

Test for saving the data

Initial data		2345,Doe,John,23/05/2021,Covid,Dr.Smith 8953,Brown,Logan,02/02/2015,Asthma,Dr.Smith 2555,White,Henry,18/11/2018,Diabetes,Dr.Clark 9092,Lewis,Noah,09/07/2017,Arthritis,Dr.Thompson 2152,Moore,Jackson,29/04/2023,Covid,Dr.Smith 3215,Hamilton,Lewis,13/08/2022,Covid,Dr.Clark	
Test N	Sequence of actions	Results	
	Input 6 in the main menu. Sort criteria choice – 1 (by code).	Expected	Actual
2152,Moore,Jackson,29/04/2023,Covid,Dr. Smith		2152,Moore,Jackson,29/04/2023,Covid,Dr. Smith	
2345,Doe,John,23/05/2021,Covid,Dr.Smith		2345,Doe,John,23/05/2021,Covid,Dr.Smith	
2555,White,Henry,18/11/2018,Diabetes,Dr. Clark		2555,White,Henry,18/11/2018,Diabetes,Dr.Clark	
3215,Hamilton,Lewis,13/08/2022,Covid,Dr. Clark		3215,Hamilton,Lewis,13/08/2022,Covid,Dr.Clark	
8953,Brown,Logan,02/02/2015,Asthma,Dr. Smith		8953,Brown,Logan,02/02/2015,Asthma,Dr.Smith	
9092,Lewis,Noah,09/07/2017,Arthritis,Dr. Thompson		9092,Lewis,Noah,09/07/2017,Arthritis,Dr. Thompson	
Actual result is correct/incorrect?		Correct	

Passed

Table 9

Test for deleting all patients with an illness

Initial data		2345,Doe,John,23/05/2021,Covid,Dr.Smith 8953,Brown,Logan,02/02/2015,Asthma,Dr.Smith 2555,White,Henry,18/11/2018,Diabetes,Dr.Clark 9092,Lewis,Noah,09/07/2017,Arthritis,Dr.Thompson 2152,Moore,Jackson,29/04/2023,Covid,Dr.Smith 3215,Hamilton,Lewis,13/08/2022,Covid,Dr.Clark	
Test N	Sequence of actions	Results	
	Input 4 in the main menu. Deleting criteria choice – 3 (by illness). Entering illness – Covid.	Expected	Actual
		Patient with code 2152, surname: Moore, and illness Covid has been deleted. Patient with code 2345, surname: Doe, and illness Covid has been deleted. Patient with code 3215, surname: Hamilton, and illness Covid has been deleted. 3 patient(s) with illness Covid have been deleted.	Patient with code 8953, surname: Brown, and illness Asthma has been deleted. Patient with code 3215, surname: Hamilton, and illness Covid has been deleted. Patient with code 3215, surname: Hamilton, and illness Covid has been deleted. 3 patient(s) with illness Covid have been deleted.
		Actual result is correct/incorrect?	<i>Incorrect</i>

Did not pass.

The problem I have encountered is due to how array is modified within a loop. When deleting an item from the array, the array size is reduced by one but the index i still gets incremented.

In the `deletePatientAtIndex` function, when removing a patient, it shifts all the patients after it one position back.

In the `deletePatientsByIllness` function, the program calls the `deletePatientAtIndex` function and then decrement the index `i` to compensate for the shift, but the `numPatients` variable is also being decremented inside `deletePatientAtIndex` function, which can result in `i` referring to an invalid position later on.

Additionally, the message printed in `deletePatientAtIndex` is incorrect because it is trying to access the information of the deleted patient after it has already been overwritten by the next patient in the array.

Here is the updated version of the 2 functions that is working right:

```
void deletePatientAtIndex(int index) {
    cout << "Patient with code " << patients[index].code << ",
surname: " << patients[index].surname << ", and illness " <<
patients[index].illness << " has been deleted." << endl;

    for (int j = index; j < numPatients - 1; j++) {
        patients[j] = patients[j + 1];
    }
    numPatients--;
}

void deletePatientsByIllness(const string& illness) {
    int deletedCount = 0;
    int i = 0;

    while (i < numPatients) {
        if (patients[i].illness == illness) {
            deletePatientAtIndex(i);
            deletedCount++;
        } else {
            i++;
        }
    }
}
```

```

    }
}

if (deletedCount > 0) {
    cout << deletedCount << " patient(s) with illness " <<
illness << " have been deleted." << endl;
}
else {
    cout << "No patients with illness " << illness << "
found." << endl;
}
}

```

6. User manual

1. General Description

The Patient Management System is an application designed to help hospitals and clinics manage their patient records.

This application allows the user to add, edit, delete, and display patient data. It also offers functionalities like searching, sorting, and calculating statistics based on the patient data.

This application is ideal for managing patient records efficiently.

2. Getting Started

To start using the Patient Management System, ensure that it is installed properly on your computer.

Once installed, launch the application by double-clicking the application icon or navigating to the application through the start menu.

3. Using the Application

No	Menu Options	Description
1	Add a new patient	Add new patient details to the system
2	Edit a patient	Modify the details of an existing patient
3	Delete a patient	Remove a patient's record from the system
4	Display all patients	View a list of all patients in the system
5	Search for patients	Search for a patient based on certain criteria
6	Sort patients	Sort the list of patients based on selected criteria
7	Calculate statistics	Compute statistics based on patient data
8	Load patients from file	Load patient data from an external file
9	Save patients to file	Save patient data to an external file
10	Exit	Close the application

3.1 Adding a Patient

Select "Add a new patient" from the main menu.

Enter the patient's code, surname, first name, date of admission, illness, and assigned doctor.

Confirm the information entered and save the patient record.

3.2 Editing a Patient

Select "Edit a patient" from the main menu.

Choose the patient you want to edit by entering their patient code.

Modify the necessary information and confirm to save changes.

3.3 Deleting a Patient

Select "Delete a patient" from the main menu.

Choose the criteria for deletion (patient code, surname, illness).

Enter the corresponding information and confirm the deletion.

3.4 Displaying Patients

Select "Display all patients" from the main menu.

The application will display a list of all the patients in the system.

3.5 Searching for Patients

Select "Search for patients" from the main menu.

Choose the criteria for searching (patient code, surname, illness).

Enter the corresponding information and the application will display matching records.

3.6 Sorting Patients

Select "Sort patients" from the main menu.

Choose the criteria for sorting (patient code, surname, date of admission, etc.).

The application will display a sorted list of patients based on the selected criteria.

3.7 Calculating Statistics

Select "Calculate statistics" from the main menu.

Choose the type of statistics you want to calculate (e.g., average patients per doctor, total number of patients for an illness).

The application will calculate and display the statistics.

3.8 Loading and Saving Patient Data

To load patient data from a file, select "Load patients from file" from the main menu and choose the file to load the data.

To save the patient data to a file, select "Save patients to file" from the main menu and choose the location and file name to save the data.

7. Conclusion

In conclusion, this Course Work shows simple development process of a small database management program for patients in a clinic.

In the requirements analysis phase, the subject area and the task definition were clearly outlined. The application was designed to manage patient records by allowing users to perform various tasks such as adding new patients, editing patient data, deleting patient data, searching for patients, sorting patient data, calculating statistics, and saving or loading data from a file. At this stage, also, prototypes for main function were already made.

In the architecture stage User Interface and main function body structure were implemented.

The program was developed as a single .cpp file and includes several functions for different operations, such as loading patient data from a file, displaying, adding, editing, deleting, searching, sorting patients, and saving data to a file.

In the detailed design stage algorithms for most complex functions were added and had made block-diagrams for them. The code was written and structured, data types and functions were created, and relevant header files were included. Attention was also paid to error handling and user interaction.

During implementation stage all the functions and data types were successfully written and described in the report. While in general program already worked because it could load, display, interact and save patients data, but still, like in the real-world testing stage was required, since a lot of bugs had come up.

The testing phase was crucial to ensure the functionality and reliability of the program. Different test cases were executed, and potential issues were documented and resolved. It is worth noting that during testing, a logical error was discovered and corrected.

User manual was also provided in the report. It describes main functionalities of the program. How to add a patient, edit, delete, search for a patient and etc.

It provides a way to quickly start using the program even with the general knowledge of Informational Technologies.

In the end, this coursework showcases the application of standard software development practices in creating a functional and reliable program. However, it is important to note that this program is relatively basic. There are

opportunities for future enhancements, such as improving the user interface, adding security features. Such improvements can make this system more robust and more professional.

8. Source code

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <iomanip>
#include <conio.h>

using namespace std;

struct Patient {
    int code;
    string surname;
    string name;
    string dateAdmissions;
    string illness;
    string doctor;
};

const int MAX_PATIENTS = 999;

Patient patients[MAX_PATIENTS];
int numPatients = 0;

void pressAnyKeyToContinue()
{
    cout << "Press any key to continue...";
    _getch();
    cout << endl;
}
```

```

void separator(int length)
{
    for (int i = 0; i <= length - 1; i++) {
        cout << endl;
    }
}

void loadPatientsFromFile(string filename) {
    ifstream infile(filename);
    if (infile.is_open()) {
        while (!infile.eof() && numPatients < MAX_PATIENTS) {
            Patient p;
            infile >> p.code >> p.surname >> p.name >>
p.dateAdmissions >> p.illness >> p.doctor;
            patients[numPatients++] = p;
        }
        infile.close();
        cout << "Loaded " << numPatients << " patients from file
" << filename << endl;
    }
    else {
        cout << "Error opening file " << filename << endl;
    }
}

void displayPatients() {
    cout << left << setw(10) << "Code"
        << left << setw(20) << "Surname"
        << left << setw(15) << "Name"
        << left << setw(20) << "Date of Admission"

```

```

        << left << setw(25) << "Illness"
        << left << setw(15) << "Doctor" << endl;

for (int i = 0; i < numPatients; i++) {
    Patient p = patients[i];
    cout << left << setw(10) << p.code
        << left << setw(20) << p.surname
        << left << setw(15) << p.name
        << left << setw(20) << p.dateAdmissions
        << left << setw(25) << p.illness
        << left << setw(15) << p.doctor << endl;
    }
}

void addPatient() {
    if (numPatients < MAX_PATIENTS) {
        Patient p;
        cout << "Enter patient code: ";
        cin >> p.code;
        cout << "Enter patient surname: ";
        cin >> p.surname;
        cout << "Enter patient name: ";
        cin >> p.name;
        cout << "Enter patient date of admissions (dd/mm/yyyy):
";
        cin >> p.dateAdmissions;
        cout << "Enter patient illness: ";
        cin >> p.illness;
        cout << "Enter patient doctor: ";
        cin >> p.doctor;
        patients[numPatients++] = p;
    }
}

```

```

        cout << "Added new patient with code " << p.code << endl;
    }
    else {
        cout << "Cannot add more patients, maximum capacity
reached." << endl;
    }
}

void editPatient() {
    int code;
    cout << "Enter patient code to edit: ";
    cin >> code;
    for (int i = 0; i < numPatients; i++) {
        if (patients[i].code == code) {
            Patient& p = patients[i];
            cout << "Enter new patient surname: ";
            cin >> p.surname;
            cout << "Enter new patient name: ";
            cin >> p.name;
            cout << "Enter new patient date of admissionsvalue:
";
            cin >> p.dateAdmissions;
            cout << "Enter new patient illness: ";
            cin >> p.illness;
            cout << "Enter new patient doctor :";
            cin >> p.doctor;
            cout << "Patient with code " << code << " has been
updated." << endl;
            return;
        }
    }
}

```



```

        cout << "Patient with code " << code << " not found." <<
endl;
    }

void deletePatientAtIndex(int index) {
    cout << "Patient with code " << patients[index].code << ",
surname: " << patients[index].surname << ", and illness " <<
patients[index].illness << " has been deleted." << endl;

    for (int j = index; j < numPatients - 1; j++) {
        patients[j] = patients[j + 1];
    }
    numPatients--;
}

void deletePatientsByCode(int code) {
    int deletedCount = 0;
    for (int i = 0; i < numPatients; i++) {
        if (patients[i].code == code) {
            deletePatientAtIndex(i);
            i--;
            deletedCount++;
        }
    }

    if (deletedCount > 0) {
        cout << deletedCount << " patient(s) with code " << code
<< " have been deleted." << endl;
    }
    else {
        cout << "No patients with code " << code << " found." <<
endl;
    }
}

```

```

    }
}

void deletePatientsBySurname(const string& surname) {
    int deletedCount = 0;
    for (int i = 0; i < numPatients; i++) {
        if (patients[i].surname == surname) {
            deletePatientAtIndex(i);
            i--;
            deletedCount++;
        }
    }

    if (deletedCount > 0) {
        cout << deletedCount << " patient(s) with surname " <<
surname << " have been deleted." << endl;
    }
    else {
        cout << "No patients with surname " << surname << "
found." << endl;
    }
}

void deletePatientsByIllness(const string& illness) {
    int deletedCount = 0;
    int i = 0;

    while (i < numPatients) {
        if (patients[i].illness == illness) {
            deletePatientAtIndex(i);
            deletedCount++;
        }
    }
}

```

```

        }
        else {
            i++;
        }
    }

    if (deletedCount > 0) {
        cout << deletedCount << " patient(s) with illness " <<
        illness << " have been deleted." << endl;
    }
    else {
        cout << "No patients with illness " << illness << "
        found." << endl;
    }
}

void deletePatient() {
    int deleteOption;
    cout << "Choose delete criteria:\n";
    cout << "1. Patient code\n";
    cout << "2. Patient surname\n";
    cout << "3. Patient illness\n";
    cout << "Enter your choice: ";
    cin >> deleteOption;

    switch (deleteOption) {
    case 1: {
        int code;
        cout << "Enter patient code to delete: ";
        cin >> code;
        deletePatientsByCode(code);
    }
    }
}

```

```

        break;
    }
    case 2: {
        string surname;
        cout << "Enter patient surname to delete: ";
        cin >> surname;
        deletePatientsBySurname(surname);
        break;
    }
    case 3: {
        string illness;
        cout << "Enter patient illness to delete: ";
        cin >> illness;
        deletePatientsByIllness(illness);
        break;
    }
    default:
        cout << "Invalid choice. Exiting delete." << endl;
        return;
    }
}

void searchPatients() {
    int searchOption;
    cout << "Choose search criteria:\n";
    cout << "1. Patient code\n";
    cout << "2. Patient surname\n";
    cout << "3. Patient illness\n";
    cout << "Enter your choice: ";
    cin >> searchOption;
}

```

```

int code;

string surname, illness, dateAdmissions;

int numMatches = 0;

switch (searchOption) {
case 1:
    cout << "Enter patient code to search: ";
    cin >> code;
    for (int i = 0; i < numPatients; i++) {
        Patient p = patients[i];
        if (p.code == code){
            cout << p.code << "\t" << p.surname << "\t" <<
p.name << "\t" << p.dateAdmissions << "\t" << p.illness << "\t"
<< p.doctor << endl;
            numMatches++;
        }
    }
    break;
case 2:
    cout << "Enter patient surname to search: ";
    cin >> surname;
    for (int i = 0; i < numPatients; i++) {
        Patient p = patients[i];
        if (p.surname == surname) {
            cout << p.code << "\t" << p.surname << "\t" <<
p.name << "\t" << p.dateAdmissions << "\t" << p.illness << "\t"
<< p.doctor << endl;
            numMatches++;
        }
    }
    break;

```

```

case 3:
    cout << "Enter patient illness to search: ";
    cin >> illness;
    for (int i = 0; i < numPatients; i++) {
        Patient p = patients[i];
        if (p.illness == illness) {
            cout << p.code << "\t" << p.surname << "\t" <<
p.name << "\t" << p.dateAdmissions << "\t" << p.illness << "\t"
<< p.doctor << endl;
            numMatches++;
        }
    }
    break;
default:
    cout << "Invalid choice. Exiting search." << endl;
    return;
}

if (numMatches == 0) {
    cout << "No matching patients found." << endl;
}
}

void sortPatients() {
    Patient sortedPatients[MAX_PATIENTS];

    for (int i = 0; i < numPatients; i++) {
        sortedPatients[i] = patients[i];
    }
}

```

```

int sortingOption;
cout << "Choose sort criteria:\n";
cout << "1. Patient code\n";
cout << "2. Patient surname\n";
cout << "3. Patient illness\n";
cout << "Enter your choice: ";
cin >> sortingOption;

for (int i = 0; i < numPatients - 1; i++) {
    for (int j = 0; j < numPatients - i - 1; j++) {
        if (sortingOption == 1 && sortedPatients[j].code >
sortedPatients[j + 1].code) {
            Patient p = sortedPatients[j];
            sortedPatients[j] = sortedPatients[j + 1];
            sortedPatients[j + 1] = p;
        }
        else if (sortingOption == 2 &&
sortedPatients[j].surname > sortedPatients[j + 1].surname) {
            Patient p = sortedPatients[j];
            sortedPatients[j] = sortedPatients[j + 1];
            sortedPatients[j + 1] = p;
        }
        else if (sortingOption == 3 &&
sortedPatients[j].illness > sortedPatients[j + 1].illness) {
            Patient p = sortedPatients[j];
            sortedPatients[j] = sortedPatients[j + 1];
            sortedPatients[j + 1] = p;
        }
    }
}

```

```

    cout << "Patients sorted by " << sortingOption << ":" <<
endl;
    cout << left << setw(10) << "Code"
        << left << setw(20) << "Surname"
        << left << setw(15) << "Name"
        << left << setw(20) << "Date of Admission"
        << left << setw(25) << "Illness"
        << left << setw(15) << "Doctor" << endl;
    for (int i = 0; i < numPatients; i++) {
        cout << left << setw(10) << sortedPatients[i].code
            << left << setw(20) << sortedPatients[i].surname
            << left << setw(15) << sortedPatients[i].name
            << left << setw(20) <<
sortedPatients[i].dateAdmissions
            << left << setw(25) << sortedPatients[i].illness
            << left << setw(15) << sortedPatients[i].doctor <<
endl;
    }
}

float calculateAveragePatientsPerDoctor(const Patient*
patients, int numPatients) {
    int numDoctors = 0;
    float totalPatients = 0;

    for (int i = 0; i < numPatients; i++) {
        bool isNewDoctor = true;

        for (int j = 0; j < i; j++) {
            if (patients[i].doctor == patients[j].doctor) {
                isNewDoctor = false;
                break;
            }
        }
    }
}

```



```

        }
    }

    if (isNewDoctor) {
        numDoctors++;
    }
}

for (int i = 0; i < numDoctors; i++) {
    int count = 0;

    for (int j = 0; j < numPatients; j++) {
        if (patients[j].doctor == patients[i].doctor) {
            count++;
        }
    }

    totalPatients += count;
}

if (numDoctors > 0) {
    return totalPatients / numDoctors;
}
else {
    return 0;
}
}

int calculateTotalPatientsForIllness(const Patient* patients,
int numPatients, const string& illness) {
    int totalPatients = 0;

```

```

        for (int i = 0; i < numPatients; i++) {
            if (patients[i].illness == illness) {
                totalPatients++;
            }
        }

        return totalPatients;
    }

void calculateStats() {
    int option;
    cout << "Choose calculating:\n";
    cout << "1. Calculate an average patients per doctor\n";
    cout << "2. Calculate the total number of patients for an
illness\n";
    cout << "Enter your choice: ";
    cin >> option;

    if (option == 1) {
        cout << calculateAveragePatientsPerDoctor(patients,
numPatients);
    }
    else if (option == 2) {
        string illnessToCalculate;
        cout << "Enter an illness to calculate patients for: "
<< endl;
        cin >> illnessToCalculate;
        cout << calculateTotalPatientsForIllness(patients,
numPatients, illnessToCalculate);
    }
    else {

```

```

        cout << "Invalid option!" << endl;
    }
}

void saveDataToFile() {
    ofstream outputFile("patients.txt");
    if (outputFile.fail()) {
        cout << "Error opening file for writing." << endl;
        return;
    }
    for (int i = 0; i < numPatients; i++) {
        Patient p = patients[i];
        outputFile << p.code << "," << p.surname << "," << p.name
        << "," << p.dateAdmissions << "," << p.illness << "," << p.doctor
        << endl;
    }
    outputFile.close();
    cout << "Data saved to file." << endl;
}

void readDataFromFile() {
    ifstream inputFile("patients.txt");
    if (inputFile.fail()) {
        cout << "Error opening file for reading." << endl;
        return;
    }
    string line;
    while (getline(inputFile, line)) {
        stringstream ss(line);
        string code, surname, name, dateAdmissions, illness,
        doctor;

```

```

        getline(ss, code, ',');
        getline(ss, surname, ',');
        getline(ss, name, ',');
        getline(ss, dateAdmissions, ',');
        getline(ss, illness, ',');
        getline(ss, doctor, ',');

        Patient p = { stoi(code), surname, name, dateAdmissions,
illness, doctor };
        patients[numPatients++] = p;
    }
    inputFile.close();
    cout << "Data loaded from file." << endl;
}

int main() {
    readDataFromFile();
    int choice;
    do {
        cout << "1. Display patient data" << endl;
        cout << "2. Add new patient" << endl;
        cout << "3. Edit patient data" << endl;
        cout << "4. Delete patient" << endl;
        cout << "5. Search patients" << endl;
        cout << "6. Sort patients" << endl;
        cout << "7. Calculate stats" << endl;
        cout << "8. Save data to file" << endl;
        cout << "9. Exit" << endl;
        cout << "Enter choice: ";
        cin >> choice;

        cout << endl;

        switch (choice) {

```

```
case 1:
    displayPatients();
separator(20);
pressAnyKeyToContinue();
cout << endl;
    break;
case 2:
    addPatient();
separator(20);
pressAnyKeyToContinue();
cout << endl;
    break;
case 3:
    editPatient();
separator(20);
pressAnyKeyToContinue();
cout << endl;
    break;
case 4:
    deletePatient();
separator(20);
pressAnyKeyToContinue();
cout << endl;
    break;
case 5:
    searchPatients();
separator(20);
pressAnyKeyToContinue();
cout << endl;
    break;
```

```

        case 6:
            sortPatients();
            separator(20);
            pressAnyKeyToContinue();
            cout << endl;
            break;
        case 7:
            calculateStats();
            separator(20);
            pressAnyKeyToContinue();
            cout << endl;
            break;
        case 8:
            saveDataToFile();
            separator(20);
            pressAnyKeyToContinue();
            cout << endl;
            break;
        case 9:
            cout << "Goodbye!" << endl;
            break;
        default:
            cout << "Invalid choice." << endl;
            pressAnyKeyToContinue();
            cout << endl;
            break;
    }
} while (choice != 9);

return 0;

```