

Національний університет “Львівська політехніка”

Кафедра програмного забезпечення

КУРСОВА РОБОТА

з дисципліни «Об’єктно-орієнтоване програмування»

На тему:

«Медична база пацієнтів»

Стедента групи ПЗ-22

спеціальності 6.121

“Програмна інженерія”

Коваленко Д.М.

Керівник: доцент кафедри ПЗ,

к.т.н., доцент Коротєєва Т. О.

Національна шкала _____

Кількість _____ балів Оцінка ECTS _____

Члени комісії _____

Львів — 2022

Зміст

Завдання	4
1 Покроковий алгоритм розв'язку задачі	5
1.1 Задача сортування	5
1.2 Задача групування	6
1.3 Згрупувати людей та відсортувати кожен групу	6
1.4 Задача визначити групи людей зі списку за умовою	6
1.5 Задача визначити людей за вказаними даними	7
1.6 Задача виведення повідомлення	7
2 Діаграми	8
2.1 UML діаграма класів	8
2.2 Діаграма прецедентів	9
2.3 Діаграма послідовності виконання	10
3 Код розробленої програми	11
4 Протокол роботи програми	42
4.1 Сортування записів за показником артеріального тиску	42
4.2 Згрупувати людей за однаковими групами крові та однаковими резус факторами	43
4.3 Згрупувати людей за однаковими резус факторами та відсортувати кожен групу за показником пульсу	45
4.4 Визначити людей, які є універсальними донорами, а які є універсальними реципієнтами та сформувати загальну таблицю донорів та реципієнтів	46
4.5 Для вказаного показника вік визначити пацієнтів з підвищеними показниками артеріального тиску та пульсу	49
4.6 Всім пацієнтам з нормальним артеріальним тиском вивести повідомлення	50

5	Інструкція користувача та системні вимоги	51
5.1	Компоненти	51
5.2	Встановлення	51
5.3	Базові функції	52
5.4	Системні вимоги	53
6	Опис виняткових ситуацій	53
6.1	Неправильні дані у полі для вводу під час додавання нового пацієнта .	53
6.2	Спроба відкрити файл з неправильними даними	55
6.3	Обмежений діапазон даних для вводу	55
7	Структура файлу вхідних даних	56
	Висновки	56
	Список використаної літератури	57

Завдання

на курсову роботу з дисципліни «Об'єктно-орієнтоване програмування»
студента групи ПЗ-22 Коваленка Дмитра

Тема: «Медична база пацієнтів»

8. Створити таблицю у візуальному середовищі

№	Прізвище	Вік	Група крові	Резус-фактор	Артеріальний тиск	Пульс
---	----------	-----	-------------	--------------	-------------------	-------

1. Швидким алгоритмом відсортувати записи за показником артеріального тиску.
2. Згрупувати людей за однаковими групами крові та однаковими резус факторами.
3. Згрупувати людей за однаковими резус-факторами та відсортувати кожну групу за показником Пульсу.
4. Визначити людей, які є універсальними донорами, а які є універсальними реципієнтами та сформуванати загальну таблицю донорів та реципієнтів.
5. Для вказаного показника Вік визначити пацієнтів з підвищеними показниками артеріального тиску та пульсу.
6. Всім пацієнтам з нормальними артеріальним тиском вивести повідомлення «Прізвище — Здоровий!»

Для класу створити: 1) Конструктор за замовчуванням; 2) Конструктор з параметрами; 3) конструктор копій; 4) перевизначити операції $>>$, $<<$ для зчитування та запису у файл.

Зміст завдання та календарний план його виконання

№ з/п	Зміст завдання	Дата
1	Здійснити аналітичний огляд літератури за заданою темою та обґрунтувати вибір інструментальних засобів реалізації.	09.10
2	Побудова UML діаграм	10.10
3	Розробка алгоритмів реалізації	13.10
4	Реалізація завдання (кодування)	15.10
5	Формування інструкції користувача	17.10
6	Оформлення звіту до курсової роботи згідно з вимогами Міжнародних стандартів, дотримуючись такої структури: <ul style="list-style-type: none"> · зміст; · алгоритм розв'язку задачі у покроковому представленні; · діаграми UML класів, прецедентів, послідовності виконання; · код розробленої програми з коментарями; · протокол роботи програми для кожного пункту завдання · інструкція користувача та системні вимоги; · опис виняткових ситуацій; · структура файлу вхідних даних; · висновки; · список використаних джерел. 	18.10

Завдання прийнято до виконання: _____ Коваленко Д.М.

Керівник роботи: _____ Коротєєва Т. О.

1. Покроковий алгоритм розв'язку задачі

1.1. Задача сортування

Алгоритм А.

A1 Виклик функції сортування користувачем.

A2 Сорткування списку за заданим стовпцем.

A3 Виведення посортowanego списку на екран.

A4 Кінець.

1.2. Задача групування

Алгоритм В.

B1 Вибір користувачем стовпця, за яким необхідно згрупувати список.

B2 Сорткування списку за заданим стовпцем.

B3 Виведення згрупowanego списку на екран.

B4 Кінець.

1.3. Згрупувати людей та відсортувати кожну групу

Алгоритм С.

C1 Виклик функції сортування користувачем.

C2 Сорткування списку за двома умовами одночасно.

C3 Виведення посортowanego списку на екран.

C4 Кінець.

1.4. Задача визначити групи людей зі списку за умовою

Алгоритм D.

D1 Вибір користувачем групи яку необхідно визначити.

D2 Виведення на екран людей, дані яких задовільняють умову.

D3 Кінець.

1.5. Задача визначети людей за вказаними даними

Алгоритм Е.

Е1 Ввід користувачем необхідних даних.

Е2 Виведення на екран пацієнтів, дані яких задовільняють умову.

Е3 Кінець.

1.6. Задача виведення повідомлення

Алгоритм F.

F1 Визначення пацієнтів, дані яких задовільняють умову.

F2 Виведення на екран повідомлення для кожного пацієнта.

F3 Кінець.

2. Діаграми

2.1. UML діаграма класів

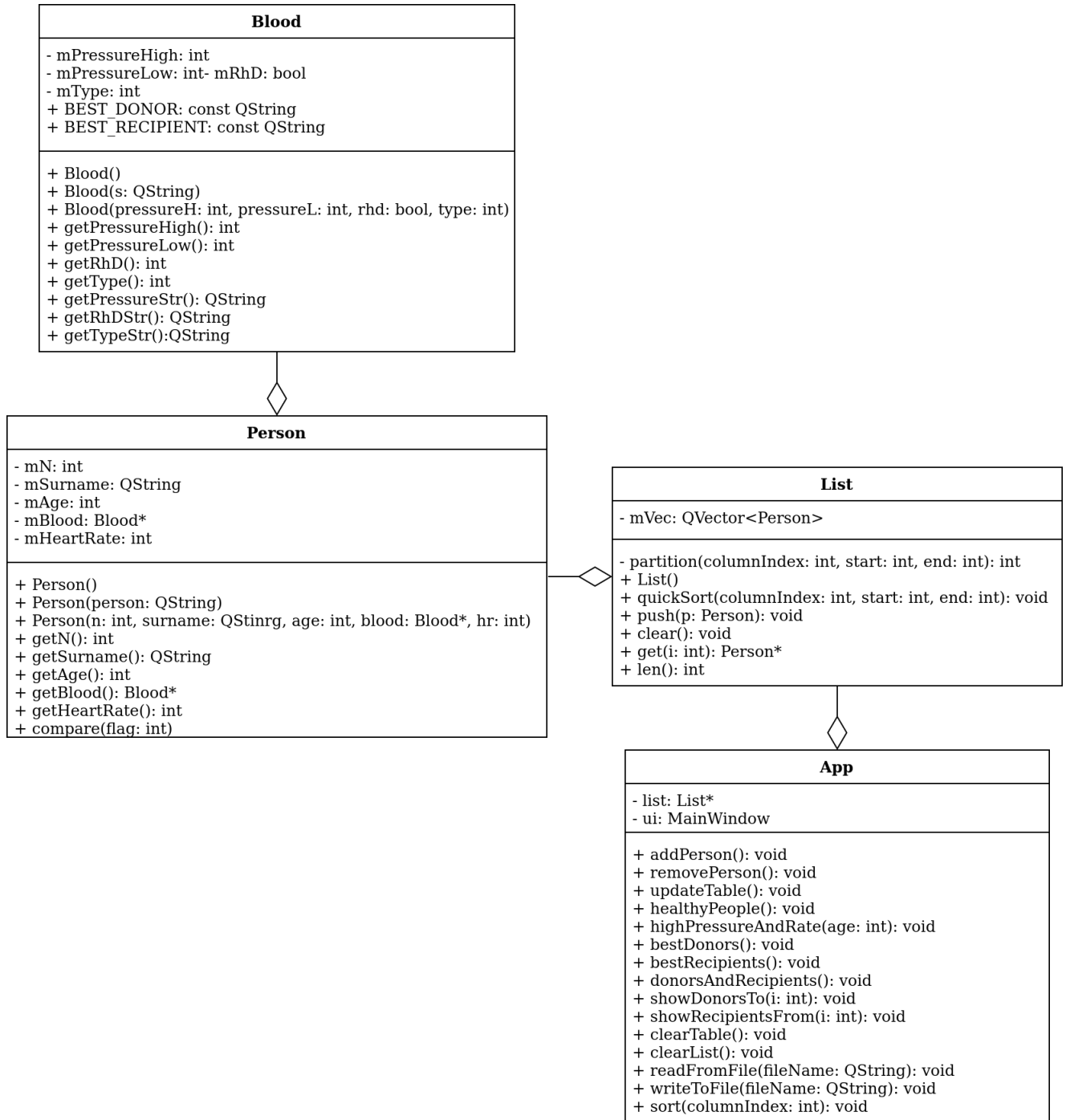


Рис. 2.1: UML діаграма класів

2.2. Діаграма прецедентів

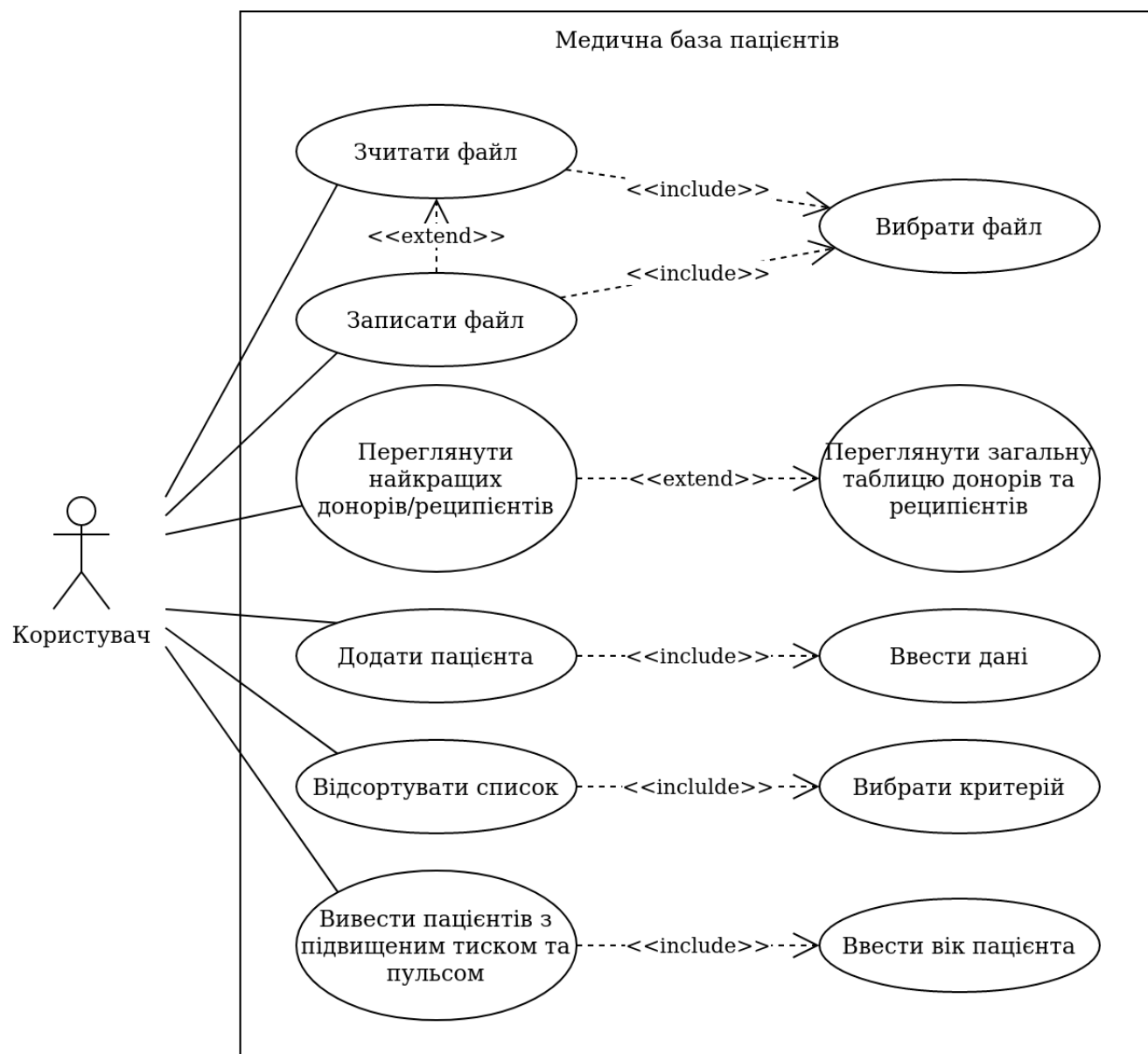


Рис. 2.2: Діаграма прецедентів

2.3. Діаграма послідовності виконання

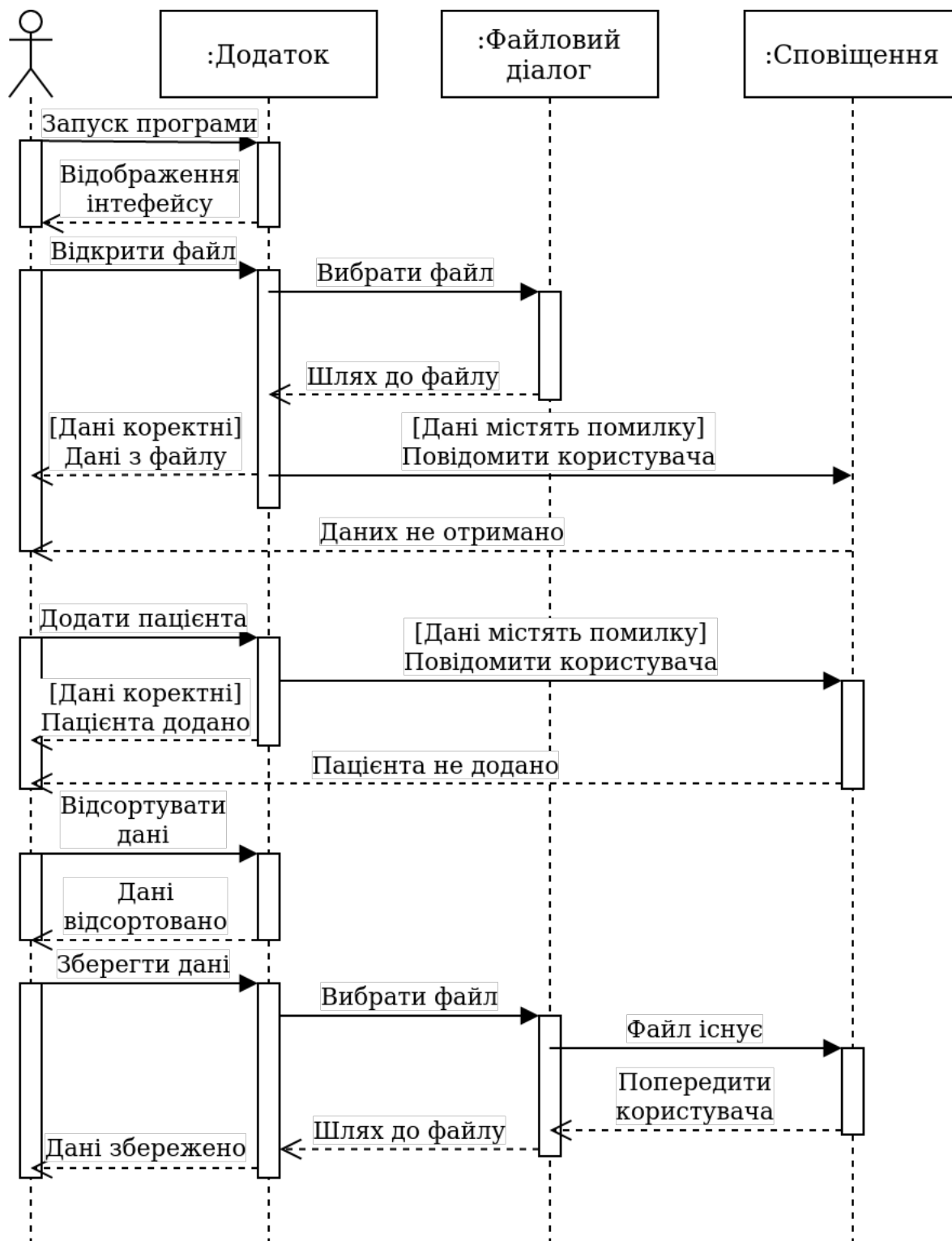


Рис. 2.3: Діаграма послідовності виконання

3. Код розробленої програми

файл *app.h*

```
#ifndef APP_H
#define APP_H

#include "list.h"
#include "mainwindow.h"
#include "ui_mainwindow.h"

class App
{
    private:
        List* list;
        Ui::MainWindow* ui;

    public:
        App() = default;
        App(Ui::MainWindow* ui);

        void addPerson();
        void removePerson();
        void updateTable();
        void healthyPeople();
        void highPressureAndRate(int age);
        void bestDonors();
        void bestRecipients();
        void donorsAndRecipients();
        void showDonorsTo(int i);
        void showRecipientsFrom(int i);
        void clearTable();
        void clearList();
        void readFromFile(QString fileName);
        void writeToFile(QString fileName);
        void sort(int columnIndex);
};
```

```

#endif // APP_H

    файл blood.h

#ifndef BLOOD_H
#define BLOOD_H

#include "QString"

class Blood
{
    private:
        int mPressureHigh;
        int mPressureLow;
        bool mRhD;
        int mType;

    public:
        const QString BEST_DONOR = "O";
        const QString BEST_RECIPIENT = "AB";

        Blood() = default;
        Blood(QString s);
        Blood(int pressureH, int pressureL, bool rhd, int type);

        int      getPressureHigh()    const { return this->mPressureHigh; }
        int      getPressureLow()     const { return this->mPressureLow; }
        bool     getRhD()             const { return this->mRhD; }
        int      getType()            const { return this->mType; }

        QString  getPressureStr();
        QString  getRhDStr();
        QString  getTypeStr();

        bool operator > (const Blood& other) const { return this->
            mPressureLow + this->mPressureHigh > other.mPressureHigh + other
            .mPressureLow; }
        bool operator < (const Blood& other) const { return this->

```

```

        mPressureLow + this→mPressureHigh < other.mPressureHigh + other
        .mPressureLow; }

};

#endif // BLOOD_H

    файл list.h

#ifndef LIST_H
#define LIST_H

#include "QVector"
#include "QFile"

#include "person.h"

class List
{
    private:
    QVector<Person> mVec;
    int      partition(int columnIndex, int start, int end);

    public:
    List() = default;

    void      quickSort(int columnIndex, int start, int end);
    void      push(Person p);
    void      clear();
    Person * get(int i);
    int      len() const;

    friend void operator << (QFile &output, const List* l);
    friend void operator >> (QFile &input, List* l);

};

#endif // LIST_H

    файл mainwindow.h

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

```

```

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_actionOpen_triggered();
    void on_actionSave_triggered();
    void on_addPersonBtn_clicked();
    void on_actionby_Blood_Pressure_triggered();
    void on_actionType_and_RhD_triggered();
    void on_actionRhD_and_Heart_Rate_triggered();
    void on_healthyPeople_triggered();
    void on_highPressureAndRate_triggered();
    void on_actionDefault_triggered();
    void on_bestDonors_triggered();
    void on_bestRecipients_triggered();
    void on_donorsRecepients_triggered();
    void on_tableWidget_cellDoubleClicked(int row, int column);
    void on_actionClose_triggered();
    void on_actionRhD_triggered();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H

файл person.h

```

```

#ifndef PERSON_H
#define PERSON_H

#include "QString"
#include "QTextStream"
#include "blood.h"
#include "QDebug"

class Person
{
    private:
        int      mN;
        QString mSurname;
        int      mAge;
        Blood * mBlood;
        int      mHeartRate;

    public:
        Person() = default;
        Person(QString person);
        Person(int n, QString surname, int age, Blood* blood, int hr);

        Person(const Person &other);

        int      getN()          const { return this->mN;          }
        QString getSurname()   const { return this->mSurname;    }
        int      getAge()        const { return this->mAge;        }
        Blood * getBlood()      const { return this->mBlood;      }
        int      getHeartRate()  const { return this->mHeartRate; }

        bool      compare(const Person& other, const int flag) const;

        friend void operator << (QTextStream &output, const Person* p);
        friend void operator >> (QTextStream &input, Person* p);
};

#endif // PERSON_H

```

файл *app.cpp*

```

#include "app.h"
#include "QDebug"

App::App(Ui::MainWindow* ui)
{
    this->list = new List();
    this->ui = ui;
}

void App::addPerson()
{
    bool ok;
    if (ui->nLE->text().toInt(&ok) == 0)
        if (!ok)
            throw 1;
    if (ui->ageLE->text().toInt(&ok) == 0)
        if (!ok)
            throw 3;
    if (ui->bloodtypeLE->text() != "O" && ui->bloodtypeLE->text() != "A"
        && ui->bloodtypeLE->text() != "B" && ui->bloodtypeLE->text()
            != "AB")
        throw 4;
    if (!ui->bloodpressureLE->text().contains("/"))
        throw 5;
    if (ui->rhLE->text() != "+" && ui->rhLE->text() != "-")
        throw 6;
    if (ui->heartrateLE->text().toInt(&ok) == 0)
        if (!ok)
            throw 7;
    QString s = ui->nLE->text() + "," +
        ui->surnameLE->text() + "," +
        ui->ageLE->text() + "," +
        ui->bloodpressureLE->text() + "_" +
        ui->bloodtypeLE->text() +
        ui->rhLE->text() + "," +
        ui->heartrateLE->text();
    Person p = Person(s);
    this->list->push(p);
}

```



```
}
```

```
void App::updateTable()
```

```
{
```

```
    ui->tableHealthy->setVisible(false);
```

```
    ui->tableWidget->setVisible(true);
```

```
    ui->tableDonorsAndRecipients->setVisible(false);
```

```
    this->clearTable();
```

```
    for (int i = 0; i < list->len(); i++)
```

```
    {
```

```
        ui->tableWidget->insertRow(i);
```

```
        Person * p = this->list->get(i);
```

```
        for (int j = 0; j < 7; j++)
```

```
        {
```

```
            QTableWidgetItem * item = new QTableWidgetItem();
```

```
            item->setTextAlignment(Qt::AlignCenter);
```

```
            switch (j)
```

```
            {
```

```
                case 0:
```

```
                item->setText(QString::number(p->getN()));
```

```
                break;
```

```
                case 1:
```

```
                item->setText(p->getSurname());
```

```
                break;
```

```
                case 2:
```

```
                item->setText(QString::number(p->getAge()));
```

```
                ;
```

```
                break;
```

```
                case 3:
```

```
                item->setText(p->getBlood()->getTypeStr());
```

```
                break;
```

```
                case 4:
```

```
                item->setText(p->getBlood()->getRhDStr());
```

```
                break;
```

```
                case 5:
```

```
                item->setText(p->getBlood()->getPressureStr()  
                    ());
```

```
                break;
```

```

        case 6:
            item->setText( QString::number(p->
                getHeartRate()));
            break;
        }
        ui->tableWidget->setItem(i, j, item);
    }
}

```

```

void App::healthyPeople()
{
    ui->tableHealthy->setVisible(true);
    ui->tableWidget->setVisible(false);
    ui->tableDonorsAndRecipients->setVisible(false);
    ui->tableHealthy->setColumnCount(2);
    ui->tableHealthy->setRowCount(0);
    ui->tableHealthy->setColumnWidth( 0, 360 );
    ui->tableHealthy->setColumnWidth( 1, 360 );
    QStringList labels;
    labels << "Surname" << "Message";
    ui->tableHealthy->setHorizontalHeaderLabels(labels);
    int r = 0;
    for (int i = 0; i < list->len(); i++)
    {
        Person * p = this->list->get(i);
        bool healthy = p->getBlood()->getPressureHigh() <= 140 &&
            p->getBlood()->getPressureLow() <= 100 &&
            p->getBlood()->getPressureHigh() >= 100 &&
            p->getBlood()->getPressureLow() >= 60;

        if (healthy)
            ui->tableHealthy->insertRow(r);
        else
            continue;
        for (int j = 0; j < 2; j++)
        {
            QTableWidgetItem * item = new QTableWidgetItem();

```

```

        item->setTextAlignment(Qt::AlignCenter);
        switch (j)
        {
            case 0:
                item->setText(p->getSurname());
                break;
            case 1:
                item->setText(QString::fromStdString("
                    Healthy"));
                break;
        }
        ui->tableHealthy->setItem(r, j, item);
    }
    r++;
}
}

```

```

void App::highPressureAndRate(int age)
{
    ui->tableHealthy->setVisible(false);
    ui->tableWidget->setVisible(true);
    ui->tableDonorsAndRecipients->setVisible(false);
    this->clearTable();
    int r = 0;
    for (int i = 0; i < list->len(); i++)
    {
        Person * p = this->list->get(i);
        bool highPressureAndRate = p->getHeartRate() >= 100 &&
            p->getBlood()->getPressureHigh() >= 140 &&
            p->getBlood()->getPressureLow() >= 100 &&
            p->getAge() == age;

        if (highPressureAndRate)
            ui->tableWidget->insertRow(r);
        else
            continue;
        for (int j = 0; j < 7; j++)
        {

```

```

        QTableWidgetItem * item = new QTableWidgetItem();
        item->setTextAlignment(Qt::AlignCenter);
        switch (j)
        {
            case 0:
                item->setText(QString::number(p->getN()));
                break;
            case 1:
                item->setText(p->getSurname());
                break;
            case 2:
                item->setText(QString::number(p->getAge()));
                ;
                break;
            case 3:
                item->setText(p->getBlood()->getTypeStr());
                break;
            case 4:
                item->setText(p->getBlood()->getRhDStr());
                break;
            case 5:
                item->setText(p->getBlood()->getPressureStr
                    ());
                break;
            case 6:
                item->setText(QString::number(p->
                    getHeartRate()));
                break;
        }
        ui->tableWidget->setItem(r, j, item);
    }
    r++;
}

void App::bestDonors()
{
    ui->tableHealthy->setVisible(false);

```

```

ui->tableWidget->setVisible(true);
ui->tableDonorsAndRecipients->setVisible(false);
this->clearTable();
int r = 0;
for (int i = 0; i < list->len(); i++)
{
    Person * p = this->list->get(i);
    bool bestDonor = p->getBlood()->getTypeStr() == p->getBlood
        ()->BEST_DONOR;

    if (bestDonor)
        ui->tableWidget->insertRow(r);
    else
        continue;
    for (int j = 0; j < 7; j++)
    {
        QTableWidgetItem * item = new QTableWidgetItem();
        item->setTextAlignment(Qt::AlignCenter);
        switch (j)
        {
            case 0:
                item->setText(QString::number(p->getN()));
                break;
            case 1:
                item->setText(p->getSurname());
                break;
            case 2:
                item->setText(QString::number(p->getAge()));
                ;
                break;
            case 3:
                item->setText(p->getBlood()->getTypeStr());
                break;
            case 4:
                item->setText(p->getBlood()->getRhDStr());
                break;
            case 5:
                item->setText(p->getBlood()->getPressureStr

```

```

        ());
        break;
        case 6:
            item->setText(QString::number(p->
                getHeartRate()));
            break;
    }
    ui->tableWidget->setItem(r, j, item);
}
r++;
}

}

void App::bestRecipients()
{
    ui->tableHealthy->setVisible(false);
    ui->tableWidget->setVisible(true);
    ui->tableDonorsAndRecipients->setVisible(false);
    this->clearTable();
    int r = 0;
    for (int i = 0; i < list->len(); i++)
    {
        Person * p = this->list->get(i);
        bool bestRecipient = p->getBlood()->getTypeStr() == p->
            getBlood()->BEST_RECIPIENT;

        if (bestRecipient)
            ui->tableWidget->insertRow(r);
        else
            continue;
        for (int j = 0; j < 7; j++)
        {
            QTableWidgetItem * item = new QTableWidgetItem();
            item->setTextAlignment(Qt::AlignCenter);
            switch (j)
            {
                case 0:
                    item->setText(QString::number(p->getN()));

```

```

        break;
        case 1:
            item->setText(p->getSurname());
            break;
        case 2:
            item->setText(QString::number(p->getAge()));
            ;
            break;
        case 3:
            item->setText(p->getBlood()->getTypeStr());
            break;
        case 4:
            item->setText(p->getBlood()->getRhDStr());
            break;
        case 5:
            item->setText(p->getBlood()->getPressureStr
                ());
            break;
        case 6:
            item->setText(QString::number(p->
                getHeartRate()));
            break;
    }
    ui->tableWidget->setItem(r, j, item);
}
r++;
}
}

```

```

void App::donorsAndRecipients()
{

```

```

    ui->tableHealthy->setVisible(false);
    ui->tableWidget->setVisible(true);
    ui->tableDonorsAndRecipients->setVisible(false);
    ui->tableWidget->setRowCount(0);
    ui->tableWidget->setColumnCount(5);
    QStringList labels;
    labels << "N" << "Surname" << "Age" << "Donor_to" << "Recipient_

```

```

        from";
    ui->tableWidget->setHorizontalHeaderLabels(labels);
    ui->tableWidget->setColumnWidth( 0, 40 );
    ui->tableWidget->setColumnWidth( 1, 400 );
    ui->tableWidget->setColumnWidth( 2, 40 );
    ui->tableWidget->setColumnWidth( 3, 100 );
    ui->tableWidget->setColumnWidth( 4, 100 );

    for (int i = 0; i < list->len(); i++)
    {
        ui->tableWidget->insertRow(i);
        Person * p = this->list->get(i);
        for (int j = 0; j < 5; j++)
        {
            QTableWidgetItem * item = new QTableWidgetItem();
            item->setTextAlignment(Qt::AlignCenter);
            switch (j)
            {
                case 0:
                    item->setText( QString::number(p->getN()) );
                    break;
                case 1:
                    item->setText(p->getSurname());
                    break;
                case 2:
                    item->setText( QString::number(p->getAge()) );
                    ;
                    break;
                case 3:
                    item->setText( QString::fromStdString("...") );
                    break;
                case 4:
                    item->setText( QString::fromStdString("...") );
                    break;
            }
            ui->tableWidget->setItem(i, j, item);
        }
    }

```



```

    }
}

void App::showDonorsTo(int i)
{
    this->clearTable();
    int personBloodType = list->get(i)->getBlood()->getType();
    int r = 0;
    for (int i = 0; i < list->len(); i++)
    {
        Person * p = this->list->get(i);
        bool donorTo = p->getBlood()->getType() >= personBloodType;
        if (personBloodType == 2 && p->getBlood()->getType() == 3)
            continue;
        if (donorTo)
            ui->tableWidget->insertRow(r);
        else
            continue;
        for (int j = 0; j < 7; j++)
        {
            QTableWidgetItem * item = new QTableWidgetItem();
            item->setTextAlignment(Qt::AlignCenter);
            switch (j)
            {
                case 0:
                    item->setText(QString::number(p->getN()));
                    break;
                case 1:
                    item->setText(p->getSurname());
                    break;
                case 2:
                    item->setText(QString::number(p->getAge()));
                    ;
                    break;
                case 3:
                    item->setText(p->getBlood()->getTypeStr());
                    break;
            }
        }
    }
}

```

```

        case 4:
            item->setText(p->getBlood()->getRhDStr());
            break;
        case 5:
            item->setText(p->getBlood()->getPressureStr
                ());
            break;
        case 6:
            item->setText(QString::number(p->
                getHeartRate()));
            break;
    }
    ui->tableWidget->setItem(r, j, item);
}
r++;
}
}

```

```

void App::showRecipientsFrom(int i)
{
    this->clearTable();
    int personBloodType = list->get(i)->getBlood()->getType();
    int r = 0;
    for (int i = 0; i < list->len(); i++)
    {
        Person * p = this->list->get(i);
        bool recipientFrom = p->getBlood()->getType() <=
            personBloodType;
        if (personBloodType == 3 && p->getBlood()->getType() == 2)
            continue;
        if (recipientFrom)
            ui->tableWidget->insertRow(r);
        else
            continue;
        for (int j = 0; j < 7; j++)
        {
            QTableWidgetItem * item = new QTableWidgetItem();
            item->setTextAlignment(Qt::AlignCenter);

```

```

        switch (j)
        {
            case 0:
                item->setText ( QString::number (p->getN ()) );
                break;
            case 1:
                item->setText (p->getSurname ()) ;
                break;
            case 2:
                item->setText ( QString::number (p->getAge ()) )
                    ;
                break;
            case 3:
                item->setText (p->getBlood ()->getTypeStr ()) ;
                break;
            case 4:
                item->setText (p->getBlood ()->getRhDStr ()) ;
                break;
            case 5:
                item->setText (p->getBlood ()->getPressureStr
                    ());
                break;
            case 6:
                item->setText ( QString::number (p->
                    getHeartRate ()) );
                break;
        }
        ui->tableWidget->setItem (r , j , item);
    }
    r++;
}

}

void App::clearTable ()
{
    ui->tableWidget->setColumnCount (7) ;
    ui->tableWidget->setRowCount (0) ;
    QStringList labels;

```

```

        labels << "N" << "Surname" << "Age" << "Type" << "RhD" << "Pressure"
            << "Rate";
        ui->tableWidget->setHorizontalHeaderLabels(labels);
        ui->tableWidget->setColumnWidth( 0, 40 );
        ui->tableWidget->setColumnWidth( 1, 400 );
        ui->tableWidget->setColumnWidth( 2, 40 );
        ui->tableWidget->setColumnWidth( 3, 40 );
        ui->tableWidget->setColumnWidth( 4, 20 );
        ui->tableWidget->setColumnWidth( 5, 80 );
        ui->tableWidget->setColumnWidth( 6, 40 );
    }

    void App::readFromFile( QString fileName )
    {
        QFile file( fileName );
        this->list->clear();
        file >> this->list;
        ui->actionClose->setEnabled(true);
    }

    void App::writeToFile( QString fileName )
    {
        if (this->list->len() == 0)
            throw 1;
        QFile file( fileName );
        file << this->list;
        ui->actionClose->setEnabled(true);
    }

    void App::sort( int columnIndex )
    {
        this->list->quickSort( columnIndex, 0, this->list->len() - 1 );
    }

    void App::clearList()
    {
        this->list->clear();
        ui->actionClose->setEnabled(false);
    }

```

```
}
```

файл *blood.cpp*

```
#include "blood.h"
```

```
#include "QStringList"
```

```
#include "QDebug"
```

```
Blood::Blood(int pressureH, int pressureL, bool rhd, int type)
```

```
{
```

```
    this->mPressureLow = pressureL;
```

```
    this->mPressureHigh = pressureH;
```

```
    this->mRhD = rhd;
```

```
    this->mType = type;
```

```
}
```

```
Blood::Blood(QString s)
```

```
{
```

```
    QStringList tokens = s.split("_");
```

```
    auto pressure_tokens = tokens[0].split("/");
```

```
    this->mPressureHigh = pressure_tokens[0].toInt();
```

```
    this->mPressureLow = pressure_tokens[1].toInt();
```

```
    if (tokens[1].right(1) == "+")
```

```
        this->mRhD = true;
```

```
    else if (tokens[1].right(1) == "-")
```

```
        this->mRhD = false;
```

```
    tokens[1].chop(1);
```

```
    if (tokens[1] == "O")
```

```
        this->mType = 1;
```

```
    else if (tokens[1] == "A")
```

```
        this->mType = 2;
```

```
    else if (tokens[1] == "B")
```

```
        this->mType = 3;
```

```
    else if (tokens[1] == "AB")
```

```
        this->mType = 4;
```

```
}
```

```
QString Blood::getPressureStr()
```

```

{
    return QString::number(this→mPressureHigh) + "/" + QString::number
        (this→mPressureLow);
}

QString Blood::getRhDStr()
{
    if (this→mRhD)
        return QString::fromStdString("+");
    else
        return QString::fromStdString("-");
}

QString Blood::getTypeStr()
{
    switch (this→mType)
    {
        case 1:
            return QString::fromStdString("O");
        case 2:
            return QString::fromStdString("A");
        case 3:
            return QString::fromStdString("B");
        case 4:
            return QString::fromStdString("AB");
        default:
            return QString::fromStdString("ERROR");
    }
}

```

файл *list.cpp*

```

#include "list.h"
#include "QTextStream"
#include "QDebug"

using namespace std;

void List::push(Person p)

```

```

{
    this→mVec.append(p);
}

Person * List::get(int i)
{
    return &this→mVec[i];
}

int List::len() const
{
    return this→mVec.length();
}

int List::partition(int columnIndex, int start, int end)
{
    int pivotIndex = end;

    int i = (start - 1);

    for (int j = start; j < end; j++) {
        if (this→mVec[pivotIndex].compare(this→mVec[j],
            columnIndex)) {
            //if (this→mVec[j] < this→mVec[pivotIndex]) {
                i++;
                this→mVec.swapItemsAt(i, j);
            }
        }

        this→mVec.swapItemsAt(i + 1, end);

        return (i + 1);
    }

    void List::quickSort(int columnIndex, int start, int end)
    {
        if (start < end)
        {

```

```

        int pivot = this->partition(columnIndex, start, end
        );

        this->quickSort(columnIndex, start, pivot-1);
        this->quickSort(columnIndex, pivot+1, end);
    }
}

void List::clear()
{
    this->mVec.clear();
}

// file << list
void operator << (QFile &output, const List* l)
{
    if (output.open(QIODevice::ReadWrite))
    {
        QTextStream stream(&output);
        for (Person p : l->mVec)
        {
            stream << &p;
        }
    }
}

// file >> list
void operator >> (QFile &input, List* l)
{
    if (input.open(QIODevice::ReadOnly))
    {
        QTextStream in(&input);
        while (!in.atEnd())
        {
            QString line = input.readLine();
            Person p = Person(line);
            l->push(p);
        }
    }
}

```



```

        input.close();
    }
}

```

файл *main.cpp*

```
#include "mainwindow.h"
```

```
#include <QApplication>
```

```

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}

```

файл *mainwindow.cpp*

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "person.h"
#include "list.h"
#include "app.h"
#include "QFileDialog"
#include "QInputDialog"
#include "QDebug"
#include "QMessageBox"

```

```

App* app;
bool fileChanged = false;

```

```

MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
, ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->tableHealthy->setVisible(false);
    ui->tableDonorsAndRecipients->setVisible(false);
    ui->tableWidget->setColumnWidth(0, 40);
}

```

```

        ui->tableWidget->setColumnWidth( 1, 400 );
        ui->tableWidget->setColumnWidth( 2, 40 );
        ui->tableWidget->setColumnWidth( 3, 40 );
        ui->tableWidget->setColumnWidth( 4, 20 );
        ui->tableWidget->setColumnWidth( 5, 80 );
        ui->tableWidget->setColumnWidth( 6, 40 );
        ui->actionClose->setEnabled( false );
        QStringList labels;
        labels << "N" << "Surname" << "Age" << "Type" << "RhD" << "Pressure"
            << "Rate";
        ui->tableWidget->setHorizontalHeaderLabels( labels );
        ui->tableWidget->horizontalHeader()->setSectionResizeMode (
            QHeaderView::Fixed);
        app = new App(ui);
        auto header = ui->tableWidget->horizontalHeader();
        connect( header, &QHeaderView::sectionClicked, [ this ]( int
            columnIndex ) {
            app->sort( columnIndex );
            app->updateTable();
        });
    }

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_actionOpen_triggered()
{
    QString fileName = QFileDialog::getOpenFileName( this,
        tr( "Open_ File" ), "/home/dmytro/", tr( "Data_file_( *.csv )" ));
    if ( !fileName.isEmpty() )
    {
        try
        {
            app->readFromFile( fileName );
            app->updateTable();
        }
    }
}

```

```

        catch (int err)
        {
            QMessageBox msgBox;
            msgBox.setIcon(QMessageBox::Critical);
            msgBox.setWindowTitle("Error");
            if (err == 1) msgBox.setText("This_file_is_
                corrupted!");
            msgBox.exec();
            return;
        }
    }

}

void MainWindow::on_actionSave_triggered()
{
    QString fileName = QFileDialog::getSaveFileName(this,
        tr("Save_File"), "/home/dmytro/", tr("Data_file (*.csv)"));
    if (!fileName.isEmpty())
    {
        try
        {
            app->writeToFile(fileName);
            fileChanged = false;
        }
        catch (int err)
        {
            QMessageBox msgBox;
            msgBox.setIcon(QMessageBox::Warning);
            msgBox.setWindowTitle("Warning");
            if (err == 1) msgBox.setText("Nothing_to_save!");
            msgBox.exec();
            return;
        }
    }
}

void MainWindow::on_addPersonBtn_clicked()
{

```

```

    try
    {
        app->addPerson();
        app->updateTable();
        fileChanged = true;
    }
    catch (int err)
    {
        QMessageBox msgBox;
        msgBox.setIcon(QMessageBox::Critical);
        msgBox.setWindowTitle("Error");
        if (err == 1) msgBox.setText("N_field_has_invalid_value!");
        else if (err == 2) msgBox.setText("Surname_field_has_
            invalid_value!");
        else if (err == 3) msgBox.setText("Age_field_has_invalid_
            value!");
        else if (err == 4) msgBox.setText("Blood_Type_field_has_
            invalid_value!");
        else if (err == 5) msgBox.setText("Blood_Pressure_field_has_
            _invalid_value!");
        else if (err == 6) msgBox.setText("RhD_field_has_invalid_
            value!");
        else if (err == 7) msgBox.setText("Heart_Rate_field_has_
            invalid_value!");
        msgBox.exec();
        return;
    }
}

void MainWindow::on_actionby_Blood_Pressure_triggered()
{
    app->sort(0);
    app->updateTable();
}

void MainWindow::on_actionType_and_RhD_triggered()
{
    app->sort(1);
}

```

```

        app->updateTable();
    }

    void MainWindow::on_actionRhD_triggered()
    {
        app->sort(3);
        app->updateTable();
    }

    void MainWindow::on_actionRhD_and_Heart_Rate_triggered()
    {
        app->sort(2);
        app->sort(2);
        app->sort(2);
        app->sort(2);
        app->updateTable();
    }

    void MainWindow::on_healthyPeople_triggered()
    {
        app->healthyPeople();
    }

    void MainWindow::on_highPressureAndRate_triggered()
    {
        int age = QDialog::getInt(this, "Enter", "Enter_Age:");
        if (age)
            app->highPressureAndRate(age);
    }

    void MainWindow::on_actionDefault_triggered()
    {
        app->updateTable();
    }

    void MainWindow::on_bestDonors_triggered()
    {
        app->bestDonors();
    }

```

```

}

void MainWindow::on_bestRecipients_triggered()
{
    app->bestRecipients();
}

void MainWindow::on_donorsReceipients_triggered()
{
    app->donorsAndRecipients();
}

void MainWindow::on_tableWidget_cellDoubleClicked(int row, int column)
{
    if (ui->tableWidget->columnCount() == 5) {
        if (column == 3) app->showDonorsTo(row);
        else if (column == 4) app->showRecipientsFrom(row);
    }
}

void MainWindow::on_actionClose_triggered()
{
    if (!fileChanged)
    {
        app->clearTable();
        app->clearList();
        return;
    }
    QMessageBox msgBox;
    msgBox.setText("The_file_has_been_modified.");
    msgBox.setInformativeText("Exit_without_saving?");
    msgBox.setStandardButtons(QMessageBox::Discard | QMessageBox::
        Cancel);
    msgBox.setDefaultButton(QMessageBox::Save);
    int ret = msgBox.exec();

    switch (ret) {
        case QMessageBox::Discard:

```

```

        app->clearTable();
        app->clearList();
        break;
        case QMessageBox::Cancel:
        break;
        default:
        break;
    }
}

```

файл *person.cpp*

```

#include "person.h"
#include "QDebug"

```

```

Person::Person(int n, QString surname, int age, Blood* blood, int heartRate
)
{
    this->mN = n;
    this->mSurname = surname;
    this->mAge = age;
    this->mBlood = blood;
    this->mHeartRate = heartRate;
}

```

```

Person::Person(QString person)
{
    QStringList tokens = person.split(",");
    if (tokens.length() != 5)
        throw 1;
    this->mN = tokens[0].toInt();
    this->mSurname = tokens[1];
    this->mAge = tokens[2].toInt();
    this->mBlood = new Blood(tokens[3]);
    this->mHeartRate = tokens[4].toInt();
}

```

```

Person::Person(const Person &other)
{

```

```

    this->mN = other.getN();
    this->mSurname = other.getSurname();
    this->mAge = other.getAge();
    this->mBlood = other.getBlood();
    this->mHeartRate = other.getHeartRate();
}

bool Person::compare(const Person& other, const int flag) const
{
    int thisPressure;
    int otherPressure;
    bool thisRhD;
    bool otherRhD;
    int thisType;
    int otherType;
    int thisHeartRate;
    int otherHeartRate;
    switch (flag)
    {
        case 0: // pressure
            thisPressure = this->getBlood()->getPressureHigh() + this->
                getBlood()->getPressureLow();
            otherPressure = other.getBlood()->getPressureHigh() + other
                .getBlood()->getPressureLow();
            return thisPressure > otherPressure;
        case 1: // rhD
            thisRhD = this->getBlood()->getRhD();
            otherRhD = other.getBlood()->getRhD();
            return thisRhD > otherRhD;
        case 2: // rhD + rate
            thisRhD = this->getBlood()->getRhD();
            otherRhD = other.getBlood()->getRhD();
            thisHeartRate = this->getHeartRate();
            otherHeartRate = other.getHeartRate();
            return thisRhD == otherRhD && thisHeartRate >
                otherHeartRate;
        case 3: // Type
            thisType = this->getBlood()->getType();

```



```

        otherType = other.getBlood()->getType();
        return thisType > otherType;
    }
    return false;
}

void operator << (QTextStream &output, const Person* p)
{
    output << p->getN()
    << " ,"
    << p->getSurname()
    << " ,"
    << p->getAge()
    << " ,"
    << p->getBlood()->getPressureStr()
    << " _"
    << p->getBlood()->getTypeStr()
    << p->getBlood()->getRhDStr()
    << " ,"
    << p->getHeartRate()
    << Qt::endl;
}

```

4. Протокол роботи програми

4.1. Сортування записів за показником артеріального тиску

File View

Add person

123 asd 12

A + 0

Add 123/123

Sort By

Blood Pressure

RhD

Blood Type

RhD and Rate

Show

Best Recipients

Best Donors

Donors and Recipients

Default

N	Surname	Age	Type	RhD	Pressure	Rate
7	Gonzalez	60	O	-	100/60	140
1	Hughes	30	B	-	100/60	120
5	Rogers	20	A	-	120/60	90
5	Williams	20	AB	-	100/80	80
2	Taylor	30	B	+	110/90	110
7	Perry	30	O	-	120/80	60
2	Roberts	50	B	+	140/60	100
3	Griffin	50	A	-	140/80	100
8	Parker	40	O	+	140/80	110
8	Hill	40	O	+	140/80	120
4	Sanders	40	A	+	140/100	80
4	Brooks	30	AB	+	140/100	70
1	Foster	60	B	-	150/100	130

Рис. 4.1: Сортування записів за показником артеріального тиску

4.2. Згрупувати людей за однаковими групами крові та однаковими резус факторами

FileView

Add person

123

asd

12

A

+

0

Add

123/123

Sort By

Blood Pressure

RhD

Blood Type

RhD and Rate

Show

Best Recipients

Best Donors

Donors and Recipients

Default

N	Surname	Age	Type	RhD	Pressure	Rate
8	Hill	40	O	+	140/80	120
7	Gonzalez	60	O	-	100/60	140
7	Perry	30	O	-	120/80	60
8	Parker	40	O	+	140/80	110
6	Long	60	A	+	200/120	150
4	Sanders	40	A	+	140/100	80
3	Griffin	50	A	-	140/80	100
5	Rogers	20	A	-	120/60	90
2	Taylor	30	B	+	110/90	110
2	Roberts	50	B	+	140/60	100
1	Foster	60	B	-	150/100	130
1	Hughes	30	B	-	100/60	120
5	Williams	20	AB	-	100/80	80

Рис. 4.2: Згрупувати людей за однаковими групами крові

File
View

Add person

123

asd

12

A

+

0

Add

123/123

Sort By

Show

Blood Pressure

RhD

Blood Type

RhD and Rate

Best Recipients

Best Donors

Donors and Recipients

Default

N	Surname	Age	Type	RhD	Pressure	Rate
3	Brooks	30	AB	-	160/120	110
7	Gonzalez	60	O	-	100/60	140
7	Perry	30	O	-	120/80	60
3	Griffin	50	A	-	140/80	100
5	Rogers	20	A	-	120/60	90
1	Foster	60	B	-	150/100	130
1	Hughes	30	B	-	100/60	120
5	Williams	20	AB	-	100/80	80
6	Allen	50	AB	+	190/120	60
2	Taylor	30	B	+	110/90	110
2	Roberts	50	B	+	140/60	100
6	Long	60	A	+	200/120	150
4	Sanders	40	A	+	140/100	80

Рис. 4.3: Згрупувати людей за однаковими резус факторами

4.3. Згрупувати людей за однаковими резус факторами та відсортувати кожну групу за показником пульсу

File
View

Add person

123

asd

12

A

+

0

Add

123/123

Sort By

Blood Pressure

RhD

Blood Type

RhD and Rate

Show

Best Recipients

Best Donors

Donors and Recipients

Default

N	Surname	Age	Type	RhD	Pressure	Rate
7	Perry	30	O	-	120/80	60
5	Williams	20	AB	-	100/80	80
5	Rogers	20	A	-	120/60	90
3	Griffin	50	A	-	140/80	100
3	Brooks	30	AB	-	160/120	110
1	Hughes	30	B	-	100/60	120
1	Foster	60	B	-	150/100	130
7	Gonzalez	60	O	-	100/60	140
6	Allen	50	AB	+	190/120	60
4	Brooks	30	AB	+	140/100	70
4	Sanders	40	A	+	140/100	80
2	Roberts	50	B	+	140/60	100
8	Parker	40	O	+	140/80	110

Рис. 4.4: Згрупувати людей за однаковими резус факторами та відсортувати кожну групу за показником пульсу

4.4. Визначити людей, які є універсальними донорами, а які є універсальними реципієнтами та сформувати загальну таблицю донорів та реципієнтів

FileView

Add person

123

asd

12

A

+

0

Add

123/123

Sort By

Blood Pressure

RhD

Blood Type

RhD and Rate

Show

Best Recipients

Best Donors

Donors and Recipients

Default

N	Surname	Age	Type	RhD	Pressure	Rate
7	Perry	30	O	-	120/80	60
7	Gonzalez	60	O	-	100/60	140
8	Parker	40	O	+	140/80	110
8	Hill	40	O	+	140/80	120

Рис. 4.5: Визначити людей, які є універсальними донорами

File
View

Add person

123

asd

12

A

+

0

Add

123/123

Sort By

Show

Blood Pressure

RhD

Blood Type

RhD and Rate

Best Recipients

Best Donors

Donors and Recipients

Default

N	Surname	Age	Type	RhD	Pressure	Rate
5	Williams	20	AB	-	100/80	80
3	Brooks	30	AB	-	160/120	110
6	Allen	50	AB	+	190/120	60
4	Brooks	30	AB	+	140/100	70

Рис. 4.6: Визначити людей, які є універсальними реципієнтами

File
View

Add person

123

asd

12

A

+

0

Add

123/123

Sort By

Blood Pressure
RhD
Blood Type
RhD and Rate

Show

Best Recipients
Best Donors
Donors and Recipients
Default

N	Surname	Age	Donor to	Recipient from
7	Perry	30
5	Williams	20
5	Rogers	20
3	Griffin	50
3	Brooks	30
1	Hughes	30
1	Foster	60
7	Gonzalez	60
6	Allen	50
4	Brooks	30
4	Sanders	40
2	Roberts	50
8	Parker	40

Рис. 4.7: Сформувати загальну таблицю донорів та реципієнтів

4.5. Для вказаного показника вік визначити пацієнтів з підвищеними показниками артеріального тиску та пульсу

FileView

Add person

123asd12

A+0

Add123/123

Sort By

Blood Pressure

RhD

Blood Type

RhD and Rate

Show

Best Recipients

Best Donors

Donors and Recipients

Default

N	Surname	Age	Type	RhD	Pressure	Rate
1	Foster	60	B	-	150/100	130
6	Long	60	A	+	200/120	150

Enter Age:

60

OK

Cancel

Рис. 4.8: Для вказаного показника вік визначити пацієнтів з підвищеними показниками артеріального тиску та пульсу

4.6. Всім пацієнтам з нормальним артеріальним тиском вивести повідомлення

FileView

Add person

123

asd

12

A

+

0

Add

123/123

Sort By

Blood Pressure

RhD

Blood Type

RhD and Rate

Show

Best Recipients

Best Donors

Donors and Recipients

Default

	Surname	Message
1	Perry	Healthy
2	Williams	Healthy
3	Rogers	Healthy
4	Griffin	Healthy
5	Hughes	Healthy
6	Gonzalez	Healthy
7	Brooks	Healthy
8	Sanders	Healthy
9	Roberts	Healthy
10	Parker	Healthy
11	Taylor	Healthy
12	Hill	Healthy

Рис. 4.9: Всім пацієнтам з нормальним артеріальним тиском вивести повідомлення

5. Інструкція користувача та системні вимоги

5.1. Компоненти

Програму розроблено на мові програмування C++ у середовищі розробки QtCreator і може експлуатуватися під управлінням сімейства операційних систем GNU/Linux. Під час проектування підсистем відбувалося поєднання об'єктно-орієнтованого підходу до програмування з процедурно орієнтованим.

5.2. Встановлення

Програма не потребує встановлення, достатньо надати виконуваному файлу програми права на виконання та запустити його.

5.3. Базові функції

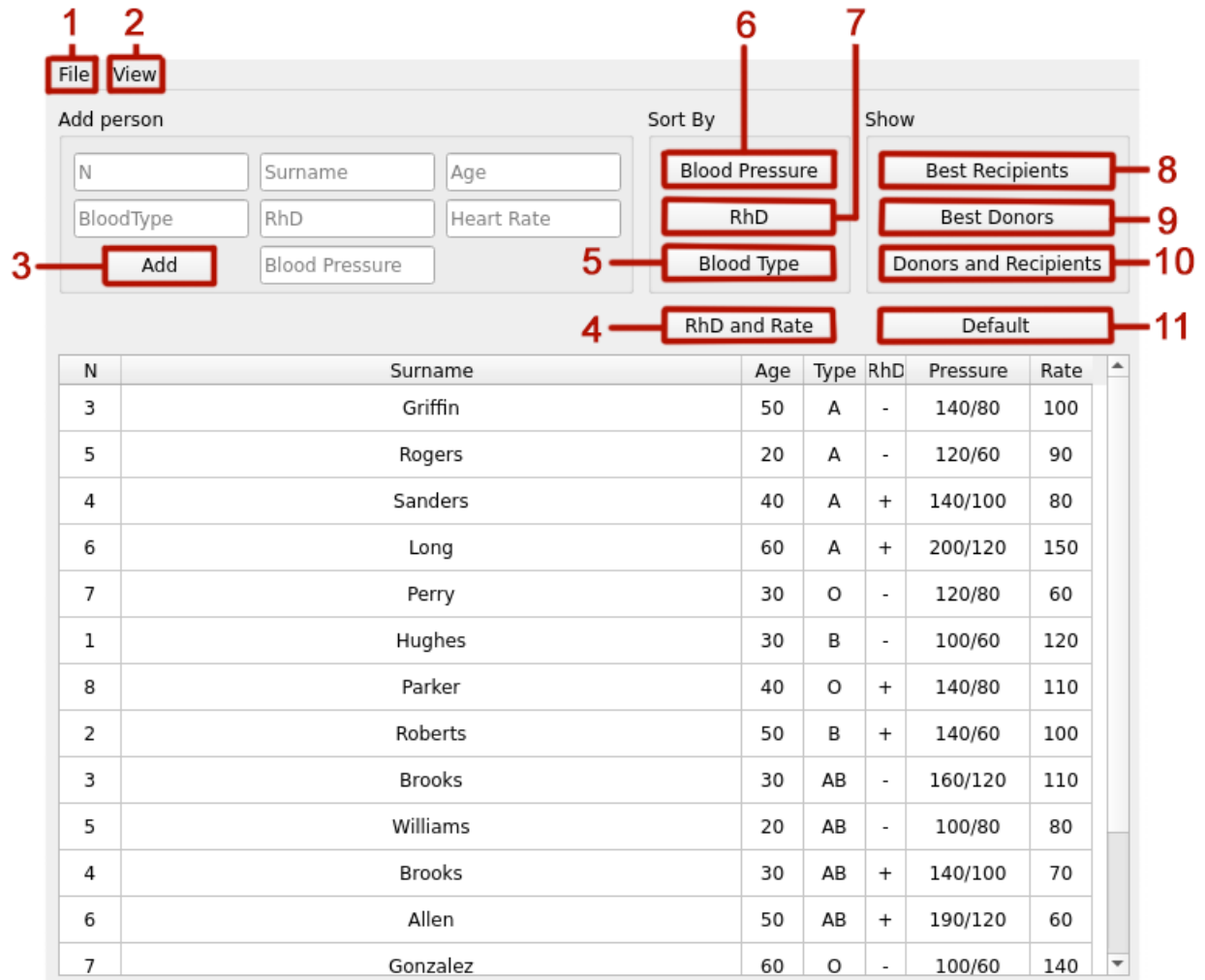


Рис. 5.1: Вигляд програми

1. Діалог меню для роботи з файлами.
2. Діалог меню для додаткових опцій сортування та перегляду.
3. Кнопка для додавання нового користувача.
4. Кнопка для групування за резус-фактором та сортування за пульсом.

5. Кнопка для сортування за групою крові.
6. Кнопка для сортування за артеріальним тиском.
7. Кнопка для сортування за резус-фактором.
8. Кнопка для перегляду таблиці універсальних реципієнтів.
9. Кнопка для перегляду таблиці універсальних донорів.
10. Кнопка для перегляду загальної таблиці донорів та реципієнтів.
11. Кнопка для повернення таблиці до початкового стану.

5.4. Системні вимоги

Для коректної роботи програми необхідна користувацька машина з процесором не менше 1 GHz, оперативною пам'яттю не менше 512 Mb. Для експлуатації пакету під управлінням сімейства операційних систем GNU/Linux необхідно встановити збірку класів Qt Framework 6.0.

6. Опис виняткових ситуацій

6.1. Неправильні дані у полі для вводу під час додавання нового пацієнта

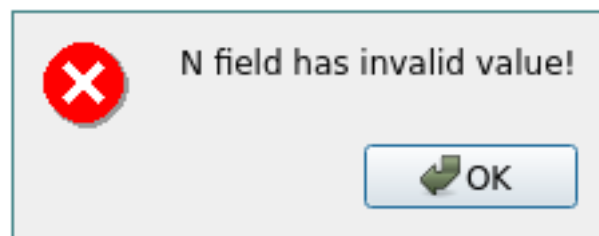


Рис. 6.1: Неправильні дані для поля N

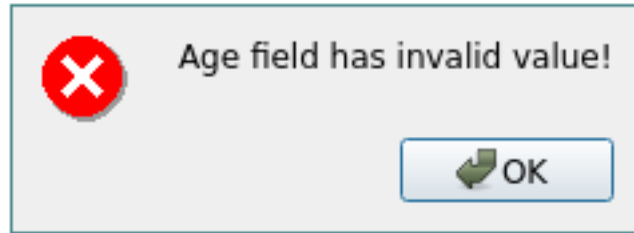


Рис. 6.2: Неправильні дані для поля Age

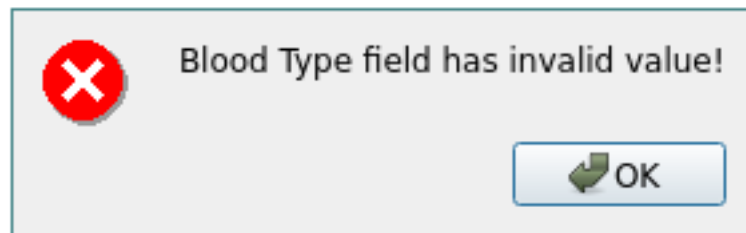


Рис. 6.3: Неправильні дані для поля Blood Type

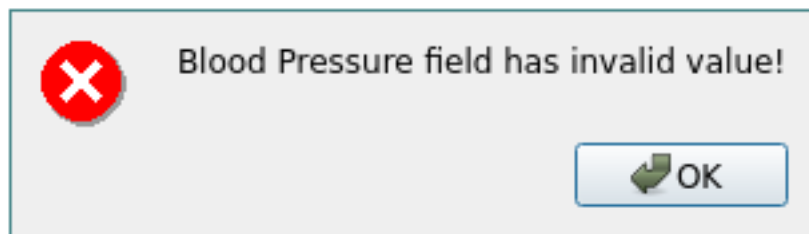


Рис. 6.4: Неправильні дані для поля Blood Pressure

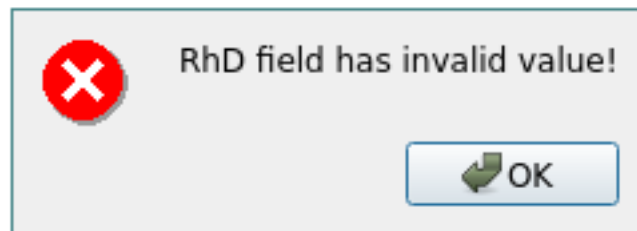


Рис. 6.5: Неправильні дані для поля RhD



Рис. 6.6: Неправильні дані для поля Heart Rate

6.2. Спроба відкрити файл з неправильними даними

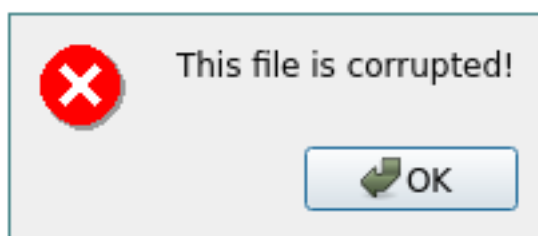


Рис. 6.7: Спроба відкрити файл з неправильними даними

6.3. Обмежений діапазон даних для вводу

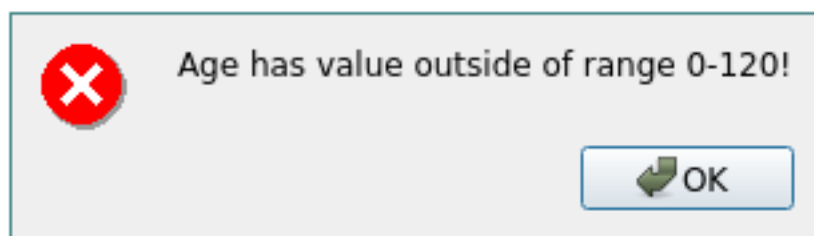


Рис. 6.8: Обмежений діапазон даних для поля Age

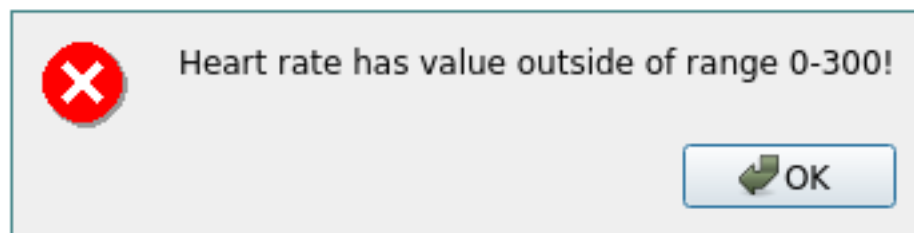
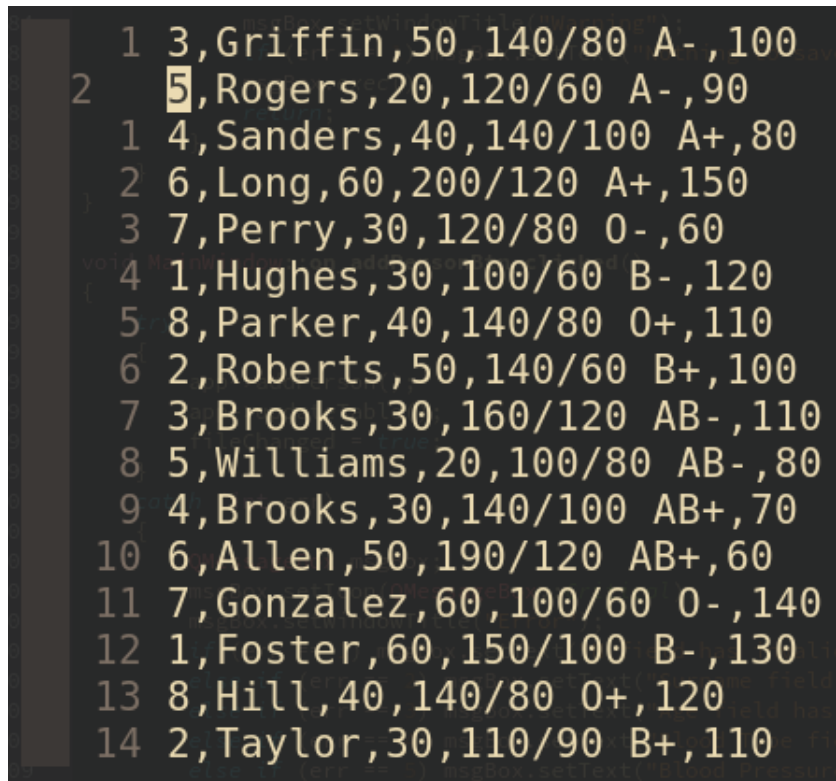


Рис. 6.9: Обмежений діапазон даних для поля Heart Rate

7. Структура файлу вхідних даних

Програма використовує класичний формат даних **.csv* (*comma separated values*), тобто запис даних, де колонки розділяються комами, і кожен рядок знаходиться на новому рядку.



```
1 3,Griffin,50,140/80 A-,100
2 5,Rogers,20,120/60 A-,90
1 4,Sanders,40,140/100 A+,80
2 6,Long,60,200/120 A+,150
3 7,Perry,30,120/80 O-,60
4 1,Hughes,30,100/60 B-,120
5 8,Parker,40,140/80 O+,110
6 2,Roberts,50,140/60 B+,100
7 3,Brooks,30,160/120 AB-,110
8 5,Williams,20,100/80 AB-,80
9 4,Brooks,30,140/100 AB+,70
10 6,Allen,50,190/120 AB+,60
11 7,Gonzalez,60,100/60 O-,140
12 1,Foster,60,150/100 B-,130
13 8,Hill,40,140/80 O+,120
14 2,Taylor,30,110/90 B+,110
```

Рис. 7.1: Приклад файлу даних для програми

Висновки

Під час виконання курсової роботи я, закріпив теоретичні знання та практичні навички, набуті при вивченні дисципліни «Об'єктно-орієнтоване програмування».

У ході виконання курсової роботи я навчився самостійно працювати з технічною літературою, документацією, сучасними засобами розробки додатків засобами об'єктно орієнтованого програмування.

Я самостійно побудував програму «Медична база пацієнтів» у середовищі розробки QtCreator за допомогою фреймворку для десктопних додатків Qt 6.0 на мові

C++ в об'єктно орієнтованому стилі.

Список використаної літератури

1. Курс Об'єктно Орієнтованого Програмування у віртуальному навчальному середовищі НУ «ЛП».
Посилання: <https://vns.lpnu.ua/course/view.php?id=711>
2. Методичні вказівки до виконання курсової роботи.
Посилання: <https://vns.lpnu.ua/mod/page/view.php?id=697909>
3. Документація до фреймворку для десктопних додатків Qt 6.0.
Посилання: <https://doc.qt.io/qt.html>
4. Кравець П.О. Об'єктно-орієнтоване програмування. - Навчальний посібник. Львів: Видавництво Львівської політехніки, 2012. 624 с.
5. Lee Zhi Eng, Qt5 C++ GUI Programming Cookbook - Second Edition, 2019. 428 р.