НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е.Алексеева

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА к курсовому проекту

Гореев Артем Дмитриевич

(фамилия, имя, отчество)

Факультет ИРИТ._____

Кафедра КТПП_____

Группа 19-ИСТ-2_____

Дата защиты «10» июня 2020г. _____ _____ _____

МИНОБРНАУКИ РОССИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е.АЛЕКСЕЕВА»
(НГТУ)

Институт радиоэлектроники и информационных технологий
Кафедра «Информатики и систем управления»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Программирование на языках высокого уровня»

РУКОВОДИТЕЛЬ

_____ Лупанова Е. А.
(подпись) (Ф.И.О.)

СТУДЕНТ

_____ Гореев А.Д._
(подпись) (Ф.И.О.)

____10.06.2020 г.__ _19-ИСТ-2__
(дата) (группа)

Работа защищена _____
(дата)

С оценкой _____

Нижний Новгород, 2020 г.

Содержание

Изм	Лист	№ докум.	Подп.	Дата			
Разраб.		Гореев А.Д.				Лит.	Лист	Листов
Пров.		Лупанова Е.А.					3	
Н.контр.							
Утв.							

Введение

Компьютеры давно стали частью нашей жизни. С помощью них можно решать разнообразные задачи как сложные, так и простые.

Создание медицинской карты больного задача очень актуальная и важная в наше время. С развитием компьютерной отрасли, активно развиваются и другие. Медкарта призвана помочь людям, которые не имеют специального технического образования легко в ней разобраться и назначить больному нужное лечение.

Компьютер позволяет решить поставленную задачу с помощью ООП. Медицинская карта может быть любой сложности, все зависит от задачи и технических средств.

В рамках данного курсового проекта решается задача создания медицинской карты больного. При разработке были реализованы те поля, которые необходимы при оказании помощи пациенту.

1. Разработка и анализ технического задания

1.1 Исследование предметной области

Первое, с чего начинается оформление медкарты – это выбор медсестрой или самим пациентом направления к тому врачу, который необходим. Врачи бывают разные – педиатры, хирурги, психолог и т.д. При выборе необходимо определиться с выбором и оформить талон, который необходим при посещении специалиста.

1.2 Техническое задание на курсовое проектирование

1.2.1 Назначение разработки

Программа предназначена для взаимодействия пациента и врачами. Она позволяет создавать медицинскую карту больного, выбирать разделы необходимые в данной ситуации, которые позволят быстро и качественно произвести лечение и осмотр больного.

1.2.2 Область применения

Данная программа может быть использована, чтобы облегчить работу медперсоналу. Кроме того, при дальнейшей доработке она может быть использована в больницах регионального значения.

1.2.3 Требования к функциональным характеристикам

1. Программа должна хранить данные о пациенте
2. Предоставлять возможность добавления информации о симптомах и прописанных лекарствах.
3. Программа должна предоставлять возможность редактирования данных о пациенте и лечащем враче.

1.2.4 Требования к количественным характеристикам

1. Время загрузки программы меньше 0,5 секунды
2. Время сохранения карты больного в файл не более 1 секунды
3. Количество рабочих мест 1

1.2.5 Требования к техническим средствам

Требования к техническим средствам определяются установленной операционной системой. Программа не является ресурсоемкой и не представляет никаких особых требований к техническим средствам.

1.2.6 Требования к информационной и программной совместимости

ПО должно работать на операционной системе семейства Windows. Программа должна работать независимо от наличия в системе средств разработки.

1.1.7 Требования к интерфейсу пользователя

Программа не предъявляет особых требований к интерфейсу.

2. Анализ технического задания

Программа состоит из нескольких частей. Данные пациента может быть реализовано с помощью текстового файла так и СУБД. Разработка с помощью СУБД очень ресурсоемко и будет тратой времени. Самый простой вариант – хранение данных пациента с возможностью его напечатать. Он и будет реализован в моей программе. Запись в файл будет реализована автоматически и не приведет к ошибкам из-за пользователя.

2 Разработка программного обеспечения

2.1 Разработка алгоритма работы программы

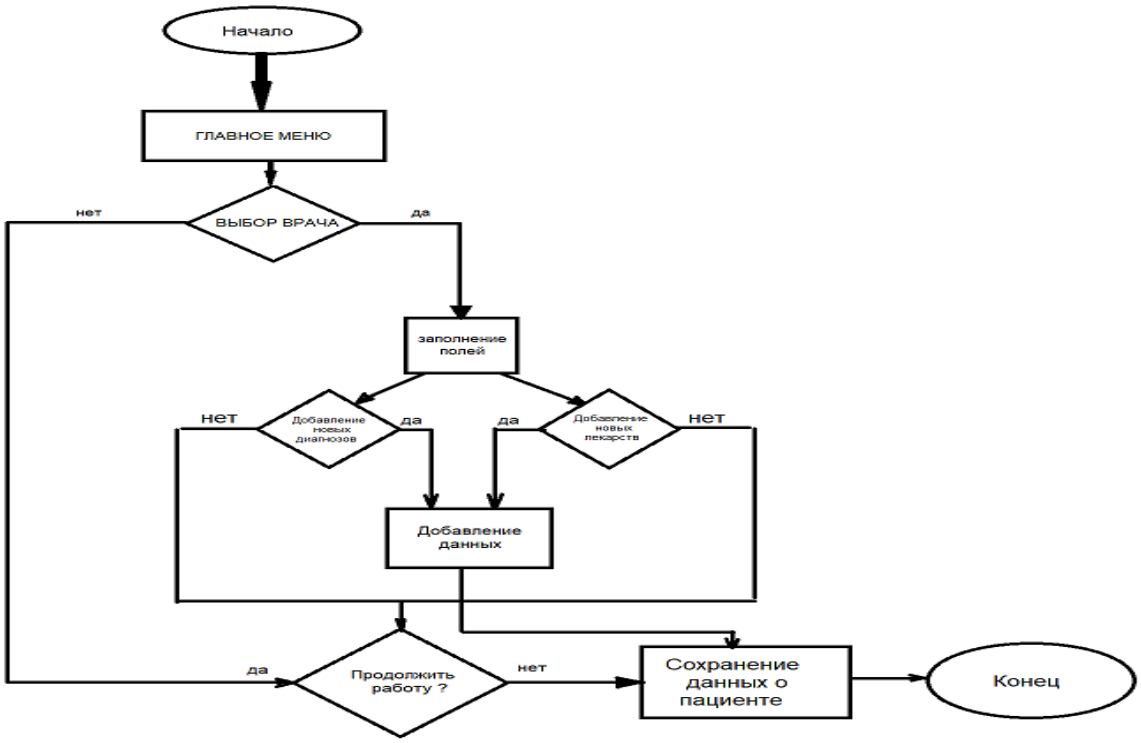

Рис. 1 Алгоритм работы программы

Алгоритм работы программы достаточно прост. Изначально пользователю предлагается совершить выбор: осуществить запись к нужному врачу, иначе выход из программы. Пользователь должен сам решить какие поля нужно заполнять, а какие нет.

После выполнения программы файл данных о пациенте перезаписывается. После чего программа прекращает работу.

1. Разработка классов

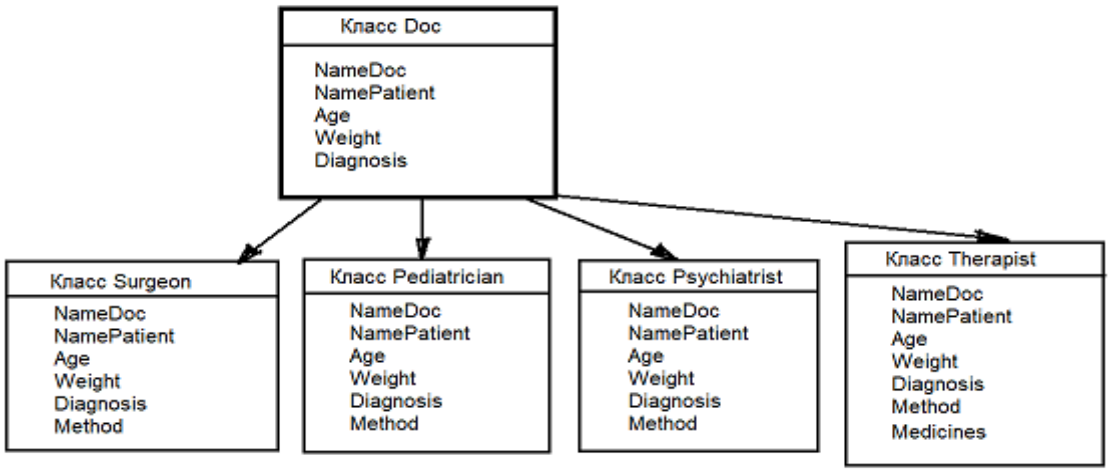

Рис. 2 Иерархия классов в программе

В программе было разработано 5 классов. Базовый класс программы – класс доктор, он хранит основную информацию о пациенте и докторе. Остальные классы очень похожи на основной, в них реализовано несколько собственных полей и переопределены виртуальные функции.

2.2.1 Класс Doc

Класс является базовым классом иерархии. Он содержит общие для всех классов свойства и методы. К методам класса относятся функции к атрибутам класса. Также переопределены несколько виртуальных функций.

Методы класса

```
#ifndef DOC_H
#define DOC_H

using namespace std;

class Doc {
private:
	string NameDoc;
	string NamePatient;

	int Age;
	string* Diagnosis;
	int Size;
	int MaxSize;
	double Weight;

	string Gender;
public:

	Doc();

	Doc(string namedoc, string namepatient, string gender, double weight, int age);

	string GetNameDoc() const { return NameDoc; }
	void SetNameDoc(string NameDoc) { this->NameDoc = NameDoc; }

	string GetNamePatient() const { return NamePatient; }
	void SetNamePatient(string NamePatient) { this->NamePatient = NamePatient; }

	double GetWeight() const { return Weight; }
	void SetWeight(double Weight) { this->Weight = Weight; }

	int GetAgePatient() const { return Age; }
	void SetAgePatient(int Age) { this->Age = Age; }

	string GetGender() { return Gender; }
	void SetGender(string Gender) { this->Gender = Gender; }

	Doc(string namedoc, string namepatient, string gender, double weight, int age,
string* NEW, int size);

	virtual ~Doc();

	Doc(const Doc& ref);

	void Add_Diagnosis(string diagnosis);
```

```
        virtual void Patient();

        virtual void Print(ostream& out) const;

        Doc& operator=(const Doc& right);
        Doc& operator +=(string NEW);
        Doc& operator++();
        Doc& operator++(int);
        bool operator==(const Doc& right) const;

        bool operator==(string namepatient);

        bool operator!=(const Doc& right) const;

        bool operator!=(string namepatient);

        string operator [](int index)const;
        string& operator[](int index);

};

ostream& operator<<(ostream& out, const Doc& H);

istream& operator>>(istream& in, Doc& H);
  #endif
```

2.2.2 Производные классы

В других классах определено собственное поле Method. Оно отвечает за выбор лечения пациента. В классе терапевт реализовано поле Medicines, оно отвечает за прописанные лекарства пациенту.

```
#ifndef PEDIATRICIAN_H
#define PEDIATRICIAN_H

using namespace std;

class Pediatrician :public Doc
{
private:
        string Method;
        string NameParent;

public:
        Pediatrician();

        Pediatrician(string namedoc, string namepatient, string nameparent, string gender,
double weight, int age);

        Pediatrician(string namedoc, string namepatient, string nameparent, string gender,
double weight, int age, string* NEW, int size);

        Pediatrician(const Pediatrician& ref);

        string GetNameParent() const { return NameParent; }
        void SetNameParent(string NameParent) { this->NameParent = NameParent; }

        void Method_Of_Treatment(int method);

        string Get_Method() const { return Method; }

        virtual ~Pediatrician();
```

```
        virtual void Patient() override;

        virtual void Print(ostream& out) const override;

        Pediatrician& operator=(const Pediatrician& right);
};

        #endif


#ifndef PSYCHIATRIST_H
#define PSYCHIATRIST_H

using namespace std;

class Psychiatrist :public Doc
{
private:

        string Method;

public:
        Psychiatrist();

        Psychiatrist(string namedoc, string namepatient, string gender, double weight, int
age, string met);

        Psychiatrist(string namedoc, string namepatient, string gender, double weight, int
age, string* NEW, int size, string met);

        Psychiatrist(const Psychiatrist& ref);

        void Method_Of_Treatment(int method);

        string Get_Method() const { return Method; }

        virtual ~Psychiatrist();

        virtual void Patient() override;

        virtual void Print(ostream& out) const override;

        Psychiatrist& operator =(const Psychiatrist& right);
};

        #endif


#ifndef SURGEON_H
#define SURGEON_H

using namespace std;

class Surgeon :public Doc
{
private:

        string Method;

public:
        Surgeon();
```

```
        Surgeon(string namedoc, string namepatient, string gender, double weight, int age,
string met);

        Surgeon(string namedoc, string namepatient, string gender, double weight, int age,
string* NEW, int size, string met);

        Surgeon(const Surgeon& ref);

        string Get_Method() const { return Method; }

        void Method_Of_Treatment(int method);

        virtual ~Surgeon();

        virtual void Patient() override;

        virtual void Print(ostream& out) const override;

        Surgeon& operator=(const Surgeon& right);
};

         #endif


#ifndef THERAPIST_H
#define THERAPIST_H

using namespace std;

class Therapist :public Doc
{
private:

        string* Medicines;
        int Count;
        int MaxCount;
        string Method;
public:
        Therapist();

        Therapist(string namedoc, string namepatient, string gender, double weight, int
age, string met);

        Therapist(string namedoc, string namepatient, string gender, double weight, int
age, string* NEW, int size, string* med, int count, string met);

        Therapist(const Therapist& ref);

        void Add_Medicines(string medicines);

        virtual ~Therapist();

        virtual void Patient() override;

        virtual void Print(ostream& out) const override;

        void Method_Of_Treatment(int method);

        string Get_Method() const { return Method; }

        Therapist& operator=(const Therapist& right);
};

         #endif
```

2. Программная реализация

2.3.1 Структура программы

Каждый класс реализован в своем модуле, функция main в отдельном файле.

Программа состоит из модулей

1. Doc-описание и определение функций базового класса
2. Surgeon -- описание и определение класса хирурга
3. Pediatrician -- описание и определение класса педиатра
4. Psychiatrist -- описание и определение класса психолога
5. Therapist -- описание и определение класса терапевта

Текст основных файлов приведен в приложении

2.3.2 Соглашения по именованию переменных

В ходе работы над курсовым проектом были выработаны следующие соглашения по именованию переменных и написанию кода

1. Все переменные в программе имеют осмысленное название

2. Имена всех пользовательских классов начинаются с заглавной буквы, имена функций классов также начинаются с заглавной буквы

3. Имена объектов классов и переменных, связанных с пользовательскими классами, начинаются с буквы, совпадающей с начальной буквой в имени класса

2.3.3 Входные и выходные данные

Входными данными программы являются данные, которые пользователь вводит с консоли.

Выходные данные – это измененное состояние файла пациента.

3 Руководство пользователя

Для установки программы достаточно скопировать файл с ней. В эту же дирекцию добавить файл с медкартой больного.

Если файл не был найден, то выведется сообщение об ошибке и программа завершит работу.

После запуска программы выведется окно пользователя. Для удобства на русском языке.

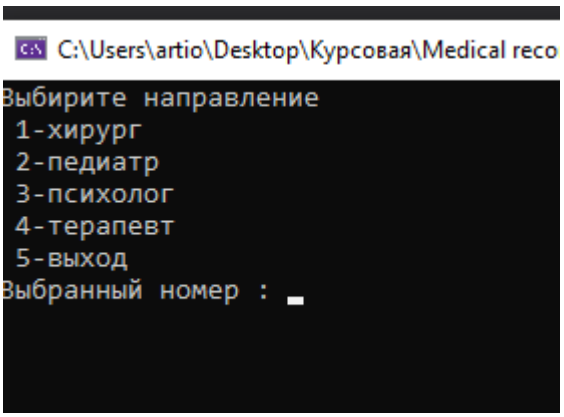

Рис. 3 Меню пользователя

Выбирая пункт меню, пользователь работает с программой. Когда пользователь завершит работу данные будут выведены на экран и файл содержащий данные о предыдущем пользователе будет перезаписан.

Заключение

В ходе курсового проекта была разработана программа для работы в медицинском учреждении. Программа позволяет хранить данные о пациенте, редактировать их и добавлять новые симптомы.

Плюсом этой программы является то, что она очень проста в обращении.

Любой пользователь легко разберется как в ней работать, что несомненно облегчит работу врачей, которые вынуждены заполнять карты больного от руки.

К недостаткам-непроработанный интерфейс.

Данный проект может и дальше развиваться, можно добавить большое количество врачей и методы лечения.

В целом задание, поставленное на курсовое проектирование выполнено.

Проект полностью соответствует техническому заданию.

Приложение

Листинг программы

```
#ifndef DOC_H
#define DOC_H

using namespace std;

class Doc {
private:
	string NameDoc;
	string NamePatient;

	int Age;
	string* Diagnosis;
	int Size;
	int MaxSize;
	double Weight;

	string Gender;
public:

	Doc();

	Doc(string namedoc, string namepatient, string gender, double weight, int age);

	string GetNameDoc() const { return NameDoc; }
	void SetNameDoc(string NameDoc) { this->NameDoc = NameDoc; }

	string GetNamePatient() const { return NamePatient; }
	void SetNamePatient(string NamePatient) { this->NamePatient = NamePatient; }

	double GetWeight() const { return Weight; }
	void SetWeight(double Weight) { this->Weight = Weight; }

	int GetAgePatient() const { return Age; }
	void SetAgePatient(int Age) { this->Age = Age; }

	string GetGender() { return Gender; }
	void SetGender(string Gender) { this->Gender = Gender; }

	Doc(string namedoc, string namepatient, string gender, double weight, int age,
string* NEW, int size);

	virtual ~Doc();

	Doc(const Doc& ref);

	void Add_Diagnosis(string diagnosis);


	virtual void Patient();

	virtual void Print(ostream& out) const;

	Doc& operator=(const Doc& right);
	Doc& operator +=(string NEW);
	Doc& operator++();
	Doc& operator++(int);
	bool operator==(const Doc& right) const;

	bool operator==(string namepatient);
```

```
        bool operator!=(const Doc& right) const;

        bool operator!=(string namepatient);

        string operator [](int index)const;
        string& operator[](int index);

};

ostream& operator<<(ostream& out, const Doc& H);

istream& operator>>(istream& in, Doc& H);
#endif

//////////////////////////////////////////////////////////////////////////////////////////

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

//////////////////////////////////////////////////////////////////////////////////////////

#ifndef PEDIATRICIAN_H
#define PEDIATRICIAN_H

using namespace std;

class Pediatrician :public Doc
{
private:
        string Method;
        string NameParent;

public:
        Pediatrician();

        Pediatrician(string namedoc, string namepatient, string nameparent, string gender,
double weight, int age);

        Pediatrician(string namedoc, string namepatient, string nameparent, string gender,
double weight, int age, string* NEW, int size);

        Pediatrician(const Pediatrician& ref);

        string GetNameParent() const { return NameParent; }
        void SetNameParent(string NameParent) { this->NameParent = NameParent; }

        void Method_Of_Treatment(int method);

        string Get_Method() const { return Method; }

        virtual ~Pediatrician();

        virtual void Patient() override;

        virtual void Print(ostream& out) const override;

        Pediatrician& operator=(const Pediatrician& right);
};

#endif

//////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////
```

```
#ifndef PSYCHIATRIST_H
#define PSYCHIATRIST_H

using namespace std;

class Psychiatrist :public Doc
{
private:

	string Method;

public:
	Psychiatrist();

	Psychiatrist(string namedoc, string namepatient, string gender, double weight, int
age, string met);

	Psychiatrist(string namedoc, string namepatient, string gender, double weight, int
age, string* NEW, int size, string met);

	Psychiatrist(const Psychiatrist& ref);

	void Method_Of_Treatment(int method);

	string Get_Method() const { return Method; }

	virtual ~Psychiatrist();

	virtual void Patient() override;

	virtual void Print(ostream& out) const override;

	Psychiatrist& operator =(const Psychiatrist& right);
};
//////////////////////////////////////////////////////////////////////////////////////////

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

//////////////////////////////////////////////////////////////////////////////////////////

#ifndef SURGEON_H
#define SURGEON_H

using namespace std;

class Surgeon :public Doc
{
private:

	string Method;

public:
	Surgeon();

	Surgeon(string namedoc, string namepatient, string gender, double weight, int age,
string met);

	Surgeon(string namedoc, string namepatient, string gender, double weight, int age,
string* NEW, int size, string met);

	Surgeon(const Surgeon& ref);

	string Get_Method() const { return Method; }

	void Method_Of_Treatment(int method);
```

```
        virtual ~Surgeon();

        virtual void Patient() override;

        virtual void Print(ostream& out) const override;

        Surgeon& operator=(const Surgeon& right);
};

#endif

//////////////////////////////////////////////////////////////////////////////////////////

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

//////////////////////////////////////////////////////////////////////////////////////////

#ifndef THERAPIST_H
#define THERAPIST_H

using namespace std;

class Therapist :public Doc
{
private:

        string* Medicines;
        int Count;
        int MaxCount;
        string Method;
public:
        Therapist();

        Therapist(string namedoc, string namepatient, string gender, double weight, int
age, string met);

        Therapist(string namedoc, string namepatient, string gender, double weight, int
age, string* NEW, int size, string* med, int count, string met);

        Therapist(const Therapist& ref);

        void Add_Medicines(string medicines);

        virtual ~Therapist();

        virtual void Patient() override;

        virtual void Print(ostream& out) const override;

        void Method_Of_Treatment(int method);

        string Get_Method() const { return Method; }

        Therapist& operator=(const Therapist& right);
};
//////////////////////////////////////////////////////////////////////////////////////////

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

//////////////////////////////////////////////////////////////////////////////////////////

#include <string>
#include<iostream>
#include "Doc.h"

using namespace std;
```

```
Doc::Doc(string namedoc, string nemepatient, string gender, double weight, int age) :
	NameDoc(namedoc), NamePatient(nemepatient), Gender(gender), Weight(weight),
Age(age), MaxSize(0),
	Size(0), Diagnosis(0)
{
}


Doc::Doc() :Weight(0), Age(0), MaxSize(0), Size(0), Diagnosis(0)
{}

Doc::~Doc()
{
	if (Size - 1 > 0) {
		delete[] Diagnosis;
		cout << "Memory cleaned" << endl;
	}
}

Doc::Doc(string namedoc, string namepatient, string gender, double weight, int age,
string* NEW, int size) :
	NameDoc(namedoc), NamePatient(namepatient), Gender(gender), Weight(weight),
Age(age), Size(size)
{
	MaxSize = Size * 2;
	Diagnosis = new string[MaxSize];
	for (int i = 0; i < Size; i++) {
		Diagnosis[i] = NEW[i];
	}
}

Doc::Doc(const Doc& ref) :
	NameDoc(ref.NameDoc), NamePatient(ref.NamePatient), Gender(ref.Gender),
Weight(ref.Weight), Age(ref.Age), MaxSize(ref.MaxSize),
	Size(ref.Size)
{
	Diagnosis = new string[MaxSize];
	for (int i = 0; i < Size; i++) {
		Diagnosis[i] = ref.Diagnosis[i];
	}
}

void Doc::Add_Diagnosis(string diagnosis) {
	if (Size == MaxSize) {
		MaxSize = Size * 2 + 1;
		string* NEW = new string[MaxSize];
		for (int i = 0; i < Size; i++)
			NEW[i] = Diagnosis[i];
		delete[]Diagnosis;
		Diagnosis = NEW;
	}

	Diagnosis[Size++] = diagnosis;
}



void Doc::Print(ostream& out) const
{
```

```
	out <<
"\n\n*****************************************************************************************
**" << endl;
	out << "*
*" << endl;
	out << "*                                МЕДИЦИНСКАЯ КАРТА БОЛЬНОГО
*" << endl;
	out << "*
*" << endl;
	out <<
"*****************************************************************************************"
<< endl;

	out << "Дата : " __DATE__ << "\nВремя : " << __TIME__ << "\n" << endl;

	out << " ФИО врача : " << NameDoc << "\n ФИО пациента : " << NamePatient << "\n
Пол : " << Gender << "\n Возраст : " << Age << "\n Вес " << (int)Weight << " килограмм "
<< (int)((Weight - (int)Weight) * 100) << " грамм" << endl;
	if (Size - 1 > 0) {
		out << "\n Жалобы на здоровье : " << endl;
		for (int i = 0; i < Size - 1; i++)
		{
			out << i + 1 << ") " << Diagnosis[i] << endl;
		}
	}
}

Doc& Doc::operator=(const Doc& right)
{
	if (this != &right) {
		NameDoc = right.NameDoc;
		NamePatient = right.NamePatient;
		Size = right.Size;
		MaxSize = right.MaxSize;
		delete[]Diagnosis;
		Diagnosis = new string[MaxSize];
		for (int i = 0; i < Size; i++)
		{
			Diagnosis[i] = right.Diagnosis[i];
		}

	}
	return *this;
}

Doc& Doc::operator+=(string NEW)
{
	Add_Diagnosis(NEW);
	return *this;
}

Doc& Doc::operator++()
{
	Age++;
	return *this;
}

Doc& Doc::operator++(int)
{
	Doc tmp(*this);
	++* this;
	return tmp;
}
```

```
bool Doc::operator==(const Doc& right) const
{
	return NamePatient == right.NamePatient;
}

bool Doc::operator==(string namepatient)
{
	return NamePatient == namepatient;
}

bool Doc::operator!=(const Doc& right) const
{
	return !(*this == right);
}

bool Doc::operator!=(string namepatient)
{
	return !(*this == namepatient);
}

string Doc::operator[](int index) const
{
	if (index >= 0 && index < Size)
		return Diagnosis[index];
	else "";
}

string& Doc::operator[](int index)
{
	if (index >= 0 && index < Size)
		return Diagnosis[index];
}

ostream& operator<<(ostream& out, const Doc& H)
{
	H.Print(out);
	return out;
}

istream& operator>>(istream& in, Doc& H)
{
	H.Patient();
	return in;
}

void Doc::Patient() {

	cout << "ФИО лечащего врача  ";
	string nd;
	cin.ignore();
	getline(cin, nd);
	SetNameDoc(nd);

	cout << "-------------------------------------------------------------------------
-------------" << endl;

	cout << "ФИО больного  ";
	string np;
	getline(cin, np);
	SetNamePatient(np);

	cout << "-------------------------------------------------------------------------
-------------" << endl;

	cout << "Пол : " << endl;
```

```
        cout << " 1-мужчина\n 0-женщина" << endl;
        int p;

        while ((!(cin >> p) || (cin.peek() != '\n')) || (p != 1 && p != 0))
        {

                cin >> p;
                cin.clear();
                while (cin.get() != '\n');
                cout << "Неправильный ввод данных, попробуйте снова" << endl;
        }

        if (p == 1) { SetGender("Мужчина"); }
        else { SetGender("Женщина"); }

        cout << "--------------------------------------------------------------------------
-------------" << endl;

        cin.ignore();
        cout << "Вес ";
        double W;
        while (!(cin >> W) || (cin.peek() != '\n'))
        {
                cin >> W;
                cin.clear();
                while (cin.get() != '\n');
                cout << "Вы неправильно указали вес пациента, попробуйте ввести данные
снова" << endl;
        }
        SetWeight(W);

        cout << "--------------------------------------------------------------------------
-------------" << endl;

        cout << "Возраст ";
        int A;
        cin.ignore();

        while (!(cin >> A) || (cin.peek() != '\n'))
        {
                cin >> A;
                cin.clear();
                while (cin.get() != '\n');
                cout << "Вы неправильно указали возраст пациента, попробуйте ввести данные
снова" << endl;
        }
        SetAgePatient(A);

        cout << "--------------------------------------------------------------------------
-------------" << endl;

        cout << "СИМПТОМЫ : " << endl;
        cout << "Чтобы перейти к следующему разделу нажмите 0" << endl;
        string a;
        int i = 0;
        cin.ignore();
        do {

                cout << i + 1 << ") ";

                getline(cin, a);
                i++;
                Add_Diagnosis(a);

        } while (a != "0");
```

```
}

////////////////////////////////////////////////////////////////////////////////////////

#include <string>
#include<iostream>

#include"Doc.h"

#include "Pediatrician.h"

using namespace std;

Pediatrician::Pediatrician() :Doc()
{
}

Pediatrician::Pediatrician(string namedoc, string namepatient, string nameparent, string
gender, double weight, int age) : Doc(namedoc, namepatient, gender, weight, age),
NameParent(nameparent)
{
}

Pediatrician::Pediatrician(string namedoc, string namepatient, string nameparent, string
gender, double weight, int age, string* NEW, int size) : Doc(namedoc, namepatient,
gender, weight, age, NEW, size), NameParent(nameparent)
{
}

Pediatrician::Pediatrician(const Pediatrician& ref) : Doc(ref),
NameParent(ref.NameParent)
{
}

void Pediatrician::Method_Of_Treatment(int method)
{
	switch (method) {
	case(1):
		cout << "Консультация" << endl;
		cout << "Назначить дату : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Дата консультации : " + Method;
		break;
	case(2):
		cout << "Осмотр" << endl;
		cout << "Назначить дату : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Дата проведения осмотра : " + Method;
		break;
	case(3):
		cout << "Профилактика" << endl;
		cout << "Назначить дату : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Дата проведения профилактики : " + Method;
		break;
	case(4):
		cout << "Вакцинация" << endl;
		cout << "Назначить дату : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Дата проведения вакцинации : " + Method;
		break;
```

```
	default:
		cout << "Неправильно выбран метод лечения" << endl;
		Method = "Вы неправильно заполнили это поле";
	}
}

Pediatrician::~Pediatrician()
{
}

void Pediatrician::Patient()
{
	cout <<
"\n\n*****************************************************************************************
**" << endl;
	cout << "
" << endl;
	cout << "                                          ПЕДИАТР
" << endl;
	cout << "
" << endl;
	cout <<
"*****************************************************************************************"
<< endl;

	cout << "ФИО родителя ";
	cin.ignore();
	string nr;
	getline(cin, nr);
	SetNameParent(nr);

	cout << "-------------------------------------------------------------------------
-------------" << endl;

	Doc::Patient();

	cout << "-------------------------------------------------------------------------
-------------" << endl;
	int k;
	cout << "Выбирите метод лечения" << endl;
	cout << "  1-консультация\n  2-осмотр\n  3-профилактика\n  4-вакцинаия" << endl;
	cout << "Номер : ";
	cin >> k;
	cout << "-------------------------------------------------------------------------
-------------" << endl;
	Method_Of_Treatment(k);
}

void Pediatrician::Print(ostream& out) const
{
	Doc::Print(out);
	out << "-------------------------------------------------------------------------
------------" << endl;
	out << "РОДИТЕЛЬ (ФИО) : " << NameParent << endl;
	out << "-------------------------------------------------------------------------
------------" << endl;
	out << "ОКАЗАНИЕ ПОМОЩИ\n" << Method << endl;
	out << "-------------------------------------------------------------------------
------------" << endl;
}
Pediatrician& Pediatrician::operator=(const Pediatrician& right)
{
	Doc::operator=(right);
	Method = right.Method;
```

```
	NameParent = right.NameParent;
	return *this;
}
////////////////////////////////////////////////////////////////////////////////////////
#include <string>
#include<iostream>

#include"Doc.h"

#include"Psychiatrist.h"

using namespace std;

Psychiatrist::Psychiatrist() :Doc()
{
}

Psychiatrist::Psychiatrist(string namedoc, string namepatient, string gender, double
weight, int age, string met) : Doc(namedoc, namepatient, gender, weight, age),
Method(met)
{
}

Psychiatrist::Psychiatrist(string namedoc, string namepatient, string gender, double
weight, int age, string* NEW, int size, string met) : Doc(namedoc, namepatient, gender,
weight, age, NEW, size), Method(met)
{
}

Psychiatrist::Psychiatrist(const Psychiatrist& ref) : Doc(ref)
{
}

void Psychiatrist::Method_Of_Treatment(int method)
{
	switch (method) {
	case(1):
		cout << "Неотложная психиатрическая помощь" << endl;
		cout << "Выдать направление : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Направление на проведение неотложной психиатрической помощи : " +
Method;
		break;
	case(2):
		cout << "Проведение психиатрической экспертизы" << endl;
		cout << "Выдать направление : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Направление на проведение психиатрической экспертизы : " +
Method;
		break;
	case(3):
		cout << "Участие в вопросах опеки указанных лиц" << endl;
		cout << "Выдать направление : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Участие в вопросах опеки указанных лиц. Направление : " + Method;
		break;
	case(4):
		cout << "Консультации по правовым вопросам" << endl;
		cout << "Выдать направление : ";
		cin.ignore();
		getline(cin, Method);
```

```
                Method = "Направление на проведение консультации по правовым вопросам : " +
Method;
                break;
        case(5):
                cout << "Оказание помощи в стационарных условаиях" << endl;
                cout << "Выдать направление : ";
                cin.ignore();
                getline(cin, Method);
                Method = "Направление на оказание помощи в стационарных условаиях : " +
Method;
                break;
        case(6):
                cout << "Социально бытовое устройство инвалидов и престарелых" << endl;
                cout << "Выдать направление : ";
                cin.ignore();
                getline(cin, Method);
                Method = "Направление на социально бытовое устройство инвалидов и
престарелых : " + Method;
                break;

        default:
                cout << "Неправильно выбран метод лечения" << endl;
                Method = "Неправильно выбран метод лечения";
        }
}

Psychiatrist::~Psychiatrist()
{
}

void Psychiatrist::Patient()
{
        cout <<
"\n\n*********************************************************************************
**" << endl;
        cout << "*
*" << endl;
        cout << "*                                        ПСИХИАТР
*" << endl;
        cout << "*
*" << endl;
        cout <<
"*********************************************************************************"
<< endl;
        Doc::Patient();

        cout << "-----------------------------------------------------------------------
-------------" << endl;
        int k;
        cout << "Выбирите метод лечения" << endl;
        cout << "  1-Неотложная психиатрическая помощь\n  2-Направление на проведение
неотложной психиатрической помощи\n  3-Участие в вопросах опеки указанных лиц\n  4-
Консультации по правовым вопросам\n  5-Оказание помощи в стационарных условаиях\n  6-
Социально бытовое устройство инвалидов и престарелых" << endl;
        cout << "Номер : ";
        cin >> k;
        cout << "-----------------------------------------------------------------------
-------------" << endl;
        Method_Of_Treatment(k);
}

void Psychiatrist::Print(ostream& out) const
{
        Doc::Print(out);
```

```
	out << "-------------------------------------------------------------------------
------------" << endl;
	out << "ОКАЗАНИЕ ПОМОЩИ\n" << Method << endl;
	out << "-------------------------------------------------------------------------
------------" << endl;
}

Psychiatrist& Psychiatrist::operator=(const Psychiatrist& right)
{
	Doc::operator=(right);
	Method = right.Method;
	return *this;
}
//////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////

#include <string>
#include<iostream>

#include"Doc.h"

#include "Surgeon.h"


using namespace std;

Surgeon::Surgeon() :Doc()
{
}

Surgeon::Surgeon(string namedoc, string namepatient, string gender, double weight, int
age, string met) : Doc(namedoc, namepatient, gender, weight, age), Method(met)
{
}

Surgeon::Surgeon(string namedoc, string namepatient, string gender, double weight, int
age, string* NEW, int size, string met) : Doc(namedoc, namepatient, gender, weight, age,
NEW, size), Method(met)
{
}

Surgeon::Surgeon(const Surgeon& ref) : Doc(ref)
{
}

void Surgeon::Method_Of_Treatment(int method)
{
	switch (method) {
	case(1):
		cout << "Консультация" << endl;
		cout << "Назначить дату : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Дата консультации : " + Method;
		break;
	case(2):
		cout << "Отсроченная операция" << endl;
		cout << "Назначить дату : ";
		cin.ignore();
		getline(cin, Method);
		Method = "Дата отсроченной операции : " + Method;
		break;
	case(3):
		cout << "Плановая операция" << endl;
```

```
                cout << "Назначить дату : ";
                cin.ignore();
                getline(cin, Method);
                Method = "Дата плановой операции : " + Method;
                break;
        case(4):
                cout << "Экстренная операция" << endl;
                cout << "Назначить дату : ";
                cin.ignore();
                getline(cin, Method);
                Method = "Дата экстренной операции : " + Method;
                break;

        default:
                cout << "Неправильно выбран метод лечения" << endl;
                Method = "Неправильно выбран метод лечения";
        }
}

Surgeon::~Surgeon()
{
}

void Surgeon::Patient()
{
        cout <<
"\n\n*****************************************************************************************
**" << endl;
        cout << "*
*" << endl;
        cout << "*                                          ХИРУРГ
*" << endl;
        cout << "*
*" << endl;
        cout <<
"*****************************************************************************************"
<< endl;
        Doc::Patient();

        cout << "-------------------------------------------------------------------------
-------------" << endl;
        int k;
        cout << "Выбирите метод лечения" << endl;
        cout << "  1-консультация\n  2-отсроченная операция\n  3-плановая операция\n  4-
экстренная операция" << endl;
        cout << "Номер : ";
        cin >> k;
        cout << "-------------------------------------------------------------------------
-------------" << endl;
        Method_Of_Treatment(k);
}

void Surgeon::Print(ostream& out) const
{
        Doc::Print(out);
        out << "-------------------------------------------------------------------------
------------" << endl;
        out << "ОКАЗАНИЕ ПОМОЩИ\n" << Method << endl;
        out << "-------------------------------------------------------------------------
------------" << endl;
}

Surgeon& Surgeon::operator=(const Surgeon& right)
{
        Doc::operator=(right);
```

```
	Method = right.Method;
	return *this;
}
////////////////////////////////////////////////////////////////////////////////////////////

#include <string>
#include<iostream>
#include"Doc.h"
#include "Therapist.h"


using namespace std;

Therapist::Therapist() :Doc(), Count(0), MaxCount(0), Medicines(0)
{
}

Therapist::Therapist(string namedoc, string namepatient, string gender, double weight,
int age, string met) : Doc(namedoc, namepatient, gender, weight, age), Count(0),
MaxCount(0), Medicines(0), Method(met)
{
}

Therapist::Therapist(string namedoc, string namepatient, string gender, double weight,
int age, string* NEW, int size, string* med, int count, string met) : Doc(namedoc,
namepatient, gender, weight, age, NEW, size), Count(count), Method(met)
{
	MaxCount = Count * 2;
	Medicines = new string[MaxCount];
	for (int i = 0; i < Count; i++) {
		Medicines[i] = med[i];
	}
}

Therapist::Therapist(const Therapist& ref) :Doc(ref), Count(ref.Count),
MaxCount(ref.MaxCount)
{
	Medicines = new string[MaxCount];
	for (int i = 0; i < Count; i++) {
		Medicines[i] = ref.Medicines[i];
	}
}

void Therapist::Add_Medicines(string medicines)
{
	if (Count == MaxCount) {
		MaxCount = Count * 2 + 1;
		string* NEW = new string[MaxCount];
		for (int i = 0; i < Count; i++)
			NEW[i] = Medicines[i];
		delete[]Medicines;
		Medicines = NEW;
	}

	Medicines[Count++] = medicines;
}

Therapist::~Therapist()
{
	if (Count - 1 > 0) {
		delete[] Medicines;
		cout << "Memory cleaned (med)" << endl;
	}
}
```

```
void Therapist::Patient()
{
	cout <<
"\n\n*****************************************************************************************
**" << endl;
	cout << "*
*" << endl;
	cout << "*                                          ТЕРАПЕВТ
*" << endl;
	cout << "*
*" << endl;
	cout <<
"*****************************************************************************************"
<< endl;
	Doc::Patient();

	cout << "-------------------------------------------------------------------------
-------------" << endl;
	int k;
	cout << "Выбирите метод лечения" << endl;
	cout << "  1-консультация\n  2-Запись на осмотр\n  3-запись другому врачу\n  4-
другое" << endl;
	cout << "Номер : ";
	cin >> k;
	cout << "-------------------------------------------------------------------------
-------------" << endl;
	Method_Of_Treatment(k);
	cout << "-------------------------------------------------------------------------
-------------" << endl;

	//cin.ignore();

	cout << "Выписать лекарственные препараты : " << endl;
	cout << "Чтобы перейти к следующему разделу нажмите 0" << endl;
	string M;

	int num = 0;
	do {

		cout << num + 1 << ") ";

		getline(cin, M);
		num++;
		Add_Medicines(M);

	} while (M != "0");

}

void Therapist::Print(ostream& out) const
{
	Doc::Print(out);
	out << "-------------------------------------------------------------------------
------------" << endl;
	out << "ОКАЗАНИЕ ПОМОЩИ\n" << Method << endl;
	if (Count - 1 > 0) {
		out << "-------------------------------------------------------------------
------------------" << endl;
		out << " Выписанные лекарства : " << endl;
		for (int i = 0; i < Count - 1; i++)
		{
			out << i + 1 << ") " << Medicines[i] << endl;
		}
```

```
                out << "-----------------------------------------------------------------------
-------------------" << endl;
        }
}

void Therapist::Method_Of_Treatment(int method)
{
        switch (method) {
        case(1):
                cout << "Консультация" << endl;
                cout << "Назначить дату : ";
                cin.ignore();
                getline(cin, Method);
                Method = "Дата консультации : " + Method;
                break;
        case(2):
                cout << "Записать на осмотр" << endl;
                cout << "Назначить дату : ";
                cin.ignore();
                getline(cin, Method);
                Method = "Дата осмотра : " + Method;
                break;
        case(3):
                cout << "Выдать направление другому врачу" << endl;
                cout << "Направление : ";
                cin.ignore();
                getline(cin, Method);
                Method = "Направление другому врачу : " + Method;
                break;
        case(4):
                cout << "Другое " << endl;
                cout << "Другое : ";
                cin.ignore();
                getline(cin, Method);
                Method = "Другое : " + Method;
                break;
        default:
                cout << "Неправильно выбран метод лечения" << endl;
                Method = "Неправильно выбран метод лечения";
        }
}

Therapist& Therapist::operator=(const Therapist& right)
{
        Doc::operator=(right);
        Method = right.Method;
        return *this;
}

//////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////

#include <string>
#include <iostream>
#include <Windows.h>
#include<fstream>
#include "Doc.h"
#include"Surgeon.h"
#include "Pediatrician.h"
#include"Psychiatrist.h"
#include "Therapist.h"
using namespace std;
template <typename T>
void WriteFile(ofstream& fout, T Human) {
```

```
	cin >> Human;
	cout << Human;
	Human.Print(fout);
	fout <<
"\n\n**********************************************************************************
**";
	fout << "\nЭТОТ ТАЛОН НУЖЕН ДЛЯ ПОСЕЩЕНИЯ ВРАЧА";
	fout << "\nЕсли возникли вопросы или нужна необходима медпомощь звоните по номеру
8800553535";
	fout <<
"\n**********************************************************************************
";
	cout << "\nФайл перезаписан или создан новый\n" << endl;
}

int main(int argc, char* argv[]) {

	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	if (argc < 2)
	{
		cout << "Wrong format" << endl;
		return -1;
	}

	ofstream fout(argv[1]);

	if (!fout) {
		cout << "Error writing file" << endl;
		return -3;
	}

	cout << "Выбирите направление\n 1-хирург\n 2-педиатр\n 3-психолог\n 4-терапевт\n
5-выход" << endl;

	int number;
	cout << "Выбранный номер : ";
	cin >> number;

	Surgeon S;
	Pediatrician P;
	Psychiatrist PS;
	Therapist T;
	switch (number) {

	case(1):
		WriteFile(fout, S);
		break;
	case(2):
		WriteFile(fout, P);
		break;
	case(3):
		WriteFile(fout, PS);
		break;
	case(4):
		WriteFile(fout, T);
		break;
	}

	fout.close();
	system("pause");
	return 0;
}
```