ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Проект: **«Областная больница»**

Разработала: **Ковалевская Алина**

1. Обзор программы

Программа предназначена для компьютеризации работы больницы. Она существенно облегчает труд врачей, позволяет легко и быстро найти нужного пациента в базе данных больницы, просмотреть его историю болезни и скорректировать курс лечения.

Программа рассчитана на пользователя, владеющего английским языком.

Дружественность интерфейса программы соответствует дружественности среднестатистического работника регистратуры.

2. Типы данных

В программе используются следующие типы данных:

1. Класс **Date**

   Предназначен для хранения дат (напр., даты рождения, госпитализации и выписки пациента) и их обработки (например, определения более ранней даты). Реализует вывод даты на экран.

```
class Date
{
public:

	static Date& chooseEarlierDate(Date& one, Date& another);

	Date() : day_(0), month_(0), year_(0) {}
	Date(int day, int month, int year): day_(day), month_(month), year_(year) {}

	void enterDate();
	void print();
	void addDaysSomehow(int offsetInDays); // условно

private:

	int day_;
	int month_;
	int year_;

	static Date& chooseEarlierYear(Date& one, Date& another);
	static Date& chooseEarlierMonth(Date& one, Date& another);
	static Date& chooseEarlierDay(Date& one, Date& another);
};
```

2. Класс **FullName**

Предназначен для хранения имени и фамилии, реализует их безопасный ввод и вывод на экран.

```
class FullName
{
public:

	FullName();
	FullName(const char* name, const char* surname);

	void enterFullName();
	void enterName();
	void enterSurname();

	void print();

private:

	char name_[MAX_WORD_LENGTH];
	char surname_[MAX_WORD_LENGTH];
	};
```

3. Класс **MedicalWorker**

Предназначен для хранения информации о медработнике. Эта информация доступна только отделению, в котором он числится.

```
class MedicalWorker
{
private:

	int hospitalUnitNo_;
	int medicalWorkerNo_;

	int currentPatientsAmount_;

	FullName fullName_;

	char post_[MAX_POST_LENGTH];
	char speciality_[MAX_SPECIALITY_LENGTH];

	MedicalWorker() : currentPatientsAmount_(0) {}
	MedicalWorker(int medicalWorkerNo) : medicalWorkerNo_(medicalWorkerNo) {}

	MedicalWorker(int hospitalUnitId, int id,
		      const char* name, const char* surname,
		      const char* post, const char* speciality);

	MedicalWorker(const char* name, const char* surname,
		      const char* post, const char* speciality);

	void setWorkingPosition(const char* post, const char* speciality);
	void setFullName(const char* name, const char* surname);

	void increaseCurrentPatientsAmount();
	void decreaseCurrentPatientsAmount();

	void printInformation();

	friend class HospitalUnit;
```

```
};
```

4. Класс **PresciptionsListItem**

   Хранит название медпрепарата как информационное поле элемента списка. Препараты не могут назначаться отдельно от пациента, поэтому доступ к полям имеет только класс Patient, внутри которого реализуется собственно сам список.

```
class PrescriptionsListItem
{
private:

	PrescriptionsListItem* next;
	PrescriptionsListItem* previous;
	char prescription[MAX_PRESCRIPTION_LENGTH];

	void enterPrescription();

	friend class Patient;
};
```

5. Класс **Diagnosis**

   Хранит наименование диагноза и уникальный ключ, по которому диагноз можно найти в перечне диагнозов отделения.

```
class Diagnosis
{
public:

	char diagnosis_[MAX_DIAGNOSIS_LENGTH];

	int hospitalUnitNo_;
	int key_;

	Diagnosis(const char* diagnosis, int hospitalUnitNo, int key);
	Diagnosis() {}
	};
```

6. Класс **ReceptionPatientForm**

   Предназначен для хранения персональных данных пациента. Подразумевается, что эта форма заполняется в регистратуре при поступлении пациента в больницу, и затем пациент уже с этой формой направляется в отделение. Имеет собственные функции вывода и чтения/записи в файл. Все поля доступны классу Patient, т. к. по сути это часть информации о пациенте.

```
class ReceptionPatientForm
{
public:

	void fillIn();
	void enterAddress();
	void computeDischargeDateSomehow();
```

```
        void print();

private:

        FullName fullName_;
        char address_[MAX_ADDRESS_LENGTH];
        Date dateOfBirth_;
        Date admissionDate_;
        Date dischargeDate_;

        void writeToFile(FILE* stream);
        void readFromFile(FILE* stream);

        friend class Patient;
};
```

7. Класс **Patient**

   Хранит информацию о пациенте. Каждый экземпляр имеет собственный список прописанных медпрепаратов, в который их можно добавлять/удалять. Содержит собственные функции чтения/записи в файл.

```
class Patient
{
public:

        static int idCounter;
        static void setIdCounter(int counter);
        static int getIdCounter();

private:

        static void increaseIdCounter();
        static void writeIdCounterToFile(FILE* stream);
        static void readIdCounterFromFile(FILE* stream);

        Patient();
        Patient(int hospitalUnitId, int key, int medicalWorkerId);
        Patient(int hospitalUnitId, ReceptionPatientForm& receptionPatientForm,
                int medicalWorkerNo);
        Patient(Patient& other);
        ~Patient();

        int patientNo_;
        ReceptionPatientForm receptionPatientForm_;
        int hospitalUnitId_;
        int diagnosisKey_;
        int medicalWorkerId_;

        int prescriptionsAmount_;
        PrescriptionsListItem* prescriptionsList_;
        PrescriptionsListItem* prescriptionsListTail_;
        void initializePrescriptionsList();

        void addPrescriptionToList(const char* prescription);
        void removePrescriptionFromList(PrescriptionsListItem* patientsListItem);
        void removePrescriptionFromList(const char* prescription);
        PrescriptionsListItem* findPrescriptionInList(const char* prescription);
        void removePrescriptionsList();
```

```
        void printPrescriptionsList();
        void writePrescriptionsListToFile(FILE* stream);
        void readPrescriptionsListFromFile(FILE* stream);
        void copyPrescriptionsList(Patient& other);

        void printReceptionForm();
        void writeToFile(FILE* stream);
        void readFromFile(FILE* stream);

        Date& getDischargeDate();
        void makeDiagnosis(int diagnosisKey);

        friend class HospitalUnit;
        friend class PatientsListItem;
        };
```

8. Класс **PatientsListItem**

```
class PatientsListItem
{
private:

        PatientsListItem* next;
        PatientsListItem* previous;
        Patient patient;

        PatientsListItem() {};
        PatientsListItem(Patient& other) : patient(other) {}

        friend class HospitalUnit;
};
```

9. Класс **HospitalUnit**

Представляет собой отделение больницы. Для каждого экземпляра создается своя директория, в которой находятся файлы с информацией об отделении, медработниках, пациентах и перечне диагнозов(пути фиксируются в соответствующих полях).
Медработники и диагнозы хранятся в виде динамических массивов, которые один раз инициализируются при первом запуске программы, а затем просто считываются из файла.
Пациенты хранятся в виде списка. При поступлении нового пациента выбирается наименее загруженный врач, и ему назначают пациента. При выписке, соответственно, пациент открепляется. Тут же реализуется подготовка к печати(просмотр) истории болезни пациента, назначение/отмена медпрепарата какому-либо пациенту, выписка пациентов. Имеются функции, осуществляющие чтение/запись всех структур из соответствующих файлов. Класс сам организует работу пользователя с меню на своем уровне. К полям имеет доступ лишь класс Hospital, т. к. отделение не может существовать отдельно от больницы.

```
class HospitalUnit
```

```
{
private:

	HospitalUnit();
	HospitalUnit(int i) : unitNo_(i) {}
	HospitalUnit(const char* name, int unitNo, int capacity);
	~HospitalUnit();

	char unitDirectoryPath_[_MAX_PATH];
	void formUnitDirectoryPath(const char* hospitalDirectory);
	void createUnitDirectory(const char* hospitalDirectory);

	char dataFilePath_[_MAX_PATH];
	void formDataFilePath();
	void writeToDataFile();
	void readFromDataFile();

	char medicalWorkersFilePath_[_MAX_PATH];
	void formMedicalWorkersFilePath();
	void writeMedicalWorkersToFile();
	void readMedicalWorkersFromFile();

	char diagnosisDatabaseFilePath_[_MAX_PATH];
	void formDiagnosisDatabaseFilePath();
	void writeDiagnosisDatabaseToFile();
	void readDiagnosisDatabaseFromFile();

	char patientsListFilePath_[_MAX_PATH];
	void formPatientsListFilePath();
	void readPatientsListFromFile();
	void readPatientsListFromFile(FILE* stream);
	void writePatientsListToFile();
	void writePatientsListToFile(FILE* stream);

	void formInnerFilesPath();

	char name_[MAX_HOSPITAL_UNIT_NAME_LENGTH];
	int unitNo_;
	int capacity_;

	int currentMedicalWorkersAmount_;
	int availableMedicalWorkersAmount_;
	MedicalWorker* medicalWorkers_;
	void initializeMedicalWorkersArray();

	void addMedicalWorker(MedicalWorker* medicalWorker);
	void addMedicalWorker(const char* name, const char* surname,
	      const char* post, const char* speciality);
	void increaseMedicalWorkersAvailableAmount();
	void printMedicalWorkers();
	void printMedicalWorker(int medicalWorkerNo);

	MedicalWorker* findFreeMedicalWorkerSomehow();
	void addPatientToMedicalWorker(int medicalWorkerNo);
	void removePatientFromMedicalWorker(int medicalWorkerNo);

	int currentDiagnosisDatabaseSize_;
	int availableDiagnosisDatabaseSize_;
	Diagnosis* diagnosisDatabase_;
	void initializeDiagnosisDatabase();

	void increaseDiagnosisDatabaseSize();
```

```
        void addDiagnosisToDatabase(const char* diagnosis);
        void printDiagnosisDatabase();

        int currentPatientsAmount_;
        PatientsListItem* patientsList_;
        PatientsListItem* patientsListTail_;
        void initializePatientsList();

        void addPatientToList(Patient* patient);
        void addPatientToList(ReceptionPatientForm& receptionPatientForm, int diagnosisKey);
        void removePatientsList();
        void removePatientFromList(PatientsListItem* patientsListItem);
        void removePatientFromList(int id);
        PatientsListItem* findPatientInList(int id);

        void printPatientsList();
        void givePrescriptionsToPatient(int id);
        void removePrescriptionsFromPatient(int id);
        void givePrescriptionsToPatient(PatientsListItem* patientsListItem);
        void removePrescriptionsFromPatient(PatientsListItem* patientsListItem);
        int getPatientAttendingMedicalWorkerNo(PatientsListItem* patientsListItem);
        void printPatientMedicalRecords(int patientNo);
        void printPatientMedicalRecords(PatientsListItem* patientsListItem);
        const char* getDiagnosis(PatientsListItem* patientsListItem);

        float getAverageWorkloadPerMedicalWorker();
        void printInformation();
        Date& findNearestDischargeDate();
        const char* getDiagnosis(int diagnosisKey);
        bool isFull();

        void workWithUnit();
        void printInvitation();
        void choosePatient();
        void workWithPatient(int patientNo);
        void showWorkWithPatientInvitation();

        friend class Hospital;
};
```

10. Класс **Hospital**

    Представляет собой больницу. Отделения хранятся в виде динамического массива. При первом запуске программы создается папка с именем, как название больницы, в которую будут помещаться папки отделений. Имеются функции для записи/чтения информации о больнице с устройства и функция workWith(), которая организует работу пользователя с меню.
    Предполагается, что пациент направлен из поликлиники с каким-то диагнозом (в программе этот диагноз выбирается случайным образом), и на основании этого диагноза его кладут в нужное отделение. Если требуемое отделение переполнено, находится ближайшая дата выписки пациентов отделения и выводится на экран. Если же места есть, то пациент заполняет форму в регистратуре (ReceptionPatientForm) и поступает в отделение.

```
class Hospital
{
```

```
public:

	Hospital();
	~Hospital();

	void createDefaultHospital();

	void restoreHospitalFromFile();
	void writeHospitalToFile();

	void workWith();

private:

	char name_[MAX_HOSPITAL_NAME_LENGTH];
	int hospitalNo_;
	Hospital(const char* name, int hospitalNo);

	int unitsAmount_;
	int availableUnitsAmount_;
	HospitalUnit* units_;

	void addUnit(const char* unitName, int unitCapacity);
	void increaseAvailableUnitsAmount();
	void initializeUnitsArray();

	int capacity_;
	int currentPatientsAmount_;

	int currentMedicalWorkersAmount_;

	void computeCapacity();
	void computeCurrentPatientsAmount();
	void computeMedicalWorkersAmount();

	char hospitalDirectoryPath_[_MAX_PATH];
	void createHospitalDirectory();
	void createHospitalDirectoryPath(const char* currentDirectory);

	void formInnerFilesNames();
	void formDataFileName();
	void formUnitDirectoriesFileName();

	char externalDataFilePath_[_MAX_PATH];
	void writeDataToExternalFile();
	void readDataFromExternalFile();

	char dataFilePath_[_MAX_PATH];
	void writeDataToFile();
	void readDataFromFile();

	char unitDirectoriesFilePath_[_MAX_PATH];
	void writeUnitDirectoriesToFile();
	void readUnitDirectoriesFromFile();

	void chooseRandomDiagnosis(int& unitNo, int& diagnosisKey);
	void admitPatientToUnit(int unitNo, int diagnosisKey);
	void admitPatient();
	void blowOffPatient(int unitNo);
	void referPatientToUnit(int unitNo, ReceptionPatientForm receptionPatientForm,
	                        int diagnosisKey);
	void signUpPatientAtReception(ReceptionPatientForm& receptionPatientForm);
```

```
        float getAverageWorkloadPerMedicalWorker();
        float getUnitAverageWorkloadPerMedicalWorker(int unitNo);
        bool isUnitNoValid(int unitNo);
        char* getUnitName(int unitNo);
        bool isUnitFull(int unitNo);
        const char* getDiagnosisFromDatabase(int unitNo, int diagnosisKey);

        void printInformation();
        void prepareAccount();

        void printInvitation();
        void chooseUnit();
};
```

3. Архитектура системы

Программа состоит из 13 модулей:

1. RunMode.h

   Определяет режим запуска программы. Если это первый запуск, то больница создается по умолчанию, также создаются новые директории для хранения информации о больнице и отделениях. Иначе программа восстановит информацию о больнице из существующих папок и файлов.

2. Constants.h

   Определения всех констант, используемых программой.

3. Hospital.h
4. HospitalUnit.h
5. MedicalWorker.h
6. Patient.h
7. PatientsListItem.h
8. ReceptionPatientForm.h
9. PrescriptionsListItem.h
10. Diagnosis.h
11. FullName.h
12. Date.h

    Модули 2–12 содержат определения одноименных классов.

13. Functions.h

    Содержит определения некоторых функций, используемых программой (в основном функции для работы с файлами и пользовательским вводом).

Работа с программой реализована в виде навигации по различным пунктам меню:

Карта меню:

1. **Режим работы со всей больницей**
    1. Просмотреть текущее состояние больницы
       Выводится номер, название, вместимость, текущее количество медработников и пациентов больницы, средняя нагрузка на врача в больнице, а также список всех отделений с информацией о них
    2. Принять нового пациента
       Диагноз выбирается случайным образом. На основании диагноза пациента направляют в нужное отделение. Если оно переполнено, то пользователь увидит сообщение о том, когда освободится место. Если же мест достаточно, то пользователь заполняет форму пациента в регистратуре.
    3. Перейти на следующий уровень
       Пользователь вводит номер требуемого отделения (можно узнать из п.1)
    4. Выйти из программы
2. **Режим работы с отделением больницы**
    1. Просмотреть текущее состояние отделения
       Выводится номер, название, вместимость, текущее количество медработников и пациентов отделения, средняя нагрузка на врача в отделении
    2. Просмотреть информацию о работниках отделения
       Выводится список медработников с их уникальными номерами, именами, должностью, специальностью и текущим количеством подопечных.
    3. Просмотреть перечень диагнозов, с которыми принимают в отделение
       Выводится перечень диагнозов и их ключи.
    4. Просмотреть информацию обо всех пациентах отделения
       Выведутся истории болезней всех пациентов отделения.
    5. Перейти на следующий уровень
       Пользователь вводит номер требуемого пациента (можно узнать из п. 4)
    6. Вернуться на предыдущий уровень
3. **Режим работы с пациентом отделения**
    1. Просмотреть историю болезни
       Выводится регистратурная форма, дата приема и предполагаемая дата выписки, диагноз, лечащий врач и список назначенных пациенту медпрепаратов.
    2. Выписать пациента
       Пользователь вводит номер пациента
    3. Прописать лекарства
       Пользователь вводит названия препаратов
    4. Отменить лекарства

       Пользователь вводит название препарата, который он хочет отменить.

    5. Вернуться на предыдущий уровень

Доступ к пунктам меню осуществляется путем нажатия клавиш, соответствующих позиции в меню. При нажатии клавиши «0» происходит выход из программы или возврат на предыдущий уровень меню. При выборе пункта меню, программа ведет пользователя далее, выводя на экран необходимые указания и по возможности обрабатывает некорректные действия пользователя.

4. Системные требования:

ОС Windows 10