Министерство образования Республики Беларусь  
Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Инженерно-экономический факультет  
Кафедра экономической информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовой работе  
по дисциплине   
“Основы конструирования программ”  
на тему  
  
“УЧЁТ КЛИЕНТОВ ПЛАТНОЙ КЛИНИКИ”

Выполнил: студент группы 972302

Пятницкий Сергей Вячеславович

Руководитель: Голда Ольга Алексеевна

Минск-2020

Содержание

[введение 3](#_Toc39084943)

[1.краткие теоретические сведения об используемых алгоритмах 5](#_Toc39084944)

[1.1. Сортировка методом Хоара 5](#_Toc39084945)

[1.2 Линейный поиск 6](#_Toc39084946)

[1.3 XOR-шифрование 7](#_Toc39084947)

[2.ОПИСАНИЕ ОРГАНИЗАЦИИ СТРУКТУР ХРАНИМЫХ ДАННЫХ 8](#_Toc39084948)

[3. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ ПРИЛОЖЕНИЯ 10](#_Toc39084949)

[4. Функциональная схема задачи, схемы алгоритмов работы приложения 12](#_Toc39084950)

введение

Поликлиника – место, куда мы приходим если у нас появились жалобы к своему здоровью, чтобы нам помогли. Ведь если вовремя не обращаться к врачу, то это может привести к необратимым последствиям. Правильное определение диагноза возможно лишь в том случае, когда врач изучит вашу историю болезни и обращений, чтобы определить уязвимые места в организме и проанализировать изменение ваших показателей. Для удобства работы работников поликлиники и повышения их эффективности, этот процесс требует систематизации, а, следовательно, в соответствующем программном обеспечении.

В связи с большим количеством информации об работниках поликлиники, её клиентах и обращениях, необходимостью хранить одновременно огромное количество данных, значительно упрощающих работу. Популярность данных систем заключается в удобстве и практичности.

В поликлинике хранится информация об истории болезней более тысячи пациентов: перенесённые болезни, противопоказания, рекомендации по правильному образу жизни и т.д. Врачу, определяя диагноз, необходимо изучить огромное количество информации. Упрощение этого процесса невозможно без использования специальных приложений. Они весомо облегчают работу, ведь сотрудникам не нужно просматривать множество данных и самостоятельно выбирать те, которые подходят под описание, нужно всего лишь отсортировать список по требуемому критерию.

Поликлиники нуждаются в особом контроле, т.к. от них зависят жизни людей. Для этого администрация поликлиник тщательно следят за заполнением информации. В настоящее время вся информация о работниках, пациенте, его история болезни, противопоказаниях и лечении может храниться в компьютере. Подобные приложения позволяют легко и быстро вносить изменения, вести учет и многое другое.

Администрация поликлиники должна иметь возможность просматривать, редактировать, сравнивать записи об врачах, чтобы увеличивать эффективность работников, а в результате и поликлиники, контролировать добросовестное выполнение своих обязанностей работниками. Также это требуется для того, чтобы контролировать квалифицированность врачей, т.к. от их опыта зависят жизни тысяч людей.

Основная цель работы заключается в снижении количества ошибок при обработке информации об клиентах платной клиники.

Поставленная цель потребовала решение следующих задач:

* Ознакомиться с (предметной областью)
* Ознакомиться со структурой хранения данных
* Разработать пользовательские функции приложения
* Разработать функциональную тему задачи
* Описать работу программы (разработать руководство пользователя)

Объектом исследования курсового проекта является процесс учета клиентов платной клиники.

Ключевые слова: функция, структура, алгоритм.

# 1.краткие теоретические сведения об используемых алгоритмах

## 1.1. Сортировка методом Хоара

Быстрая сортировка, сортировка Хоара (с английского «quicksort») – наиболее известный и эффективный алгоритм сортировки. Алгоритм, по принципу функционирования, входит в класс обменных сортировок (сортировка перемешиванием, пузырьковая сортировка и др.), выделяясь при этом высокой скоростью работы. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. (Таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов.)

Общая идея алгоритма состоит в следующем:

* выбрать из массива элемент, называемый опорным;
* разбиение массива на несколько меньших подмассивов;
* сортировка подмассивов;

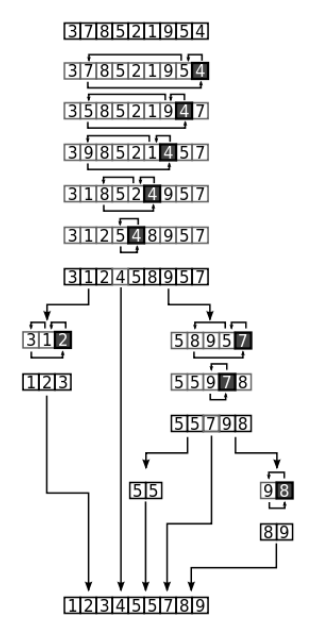
****

Рисунок 1.1 - Пример быстрой сортировки

Первым этапом является выбор опорного элемента. Опорным может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов.

Вторым этапом сортировки происходит сравнивание всех остальных элементов с опорным и разбиение массива на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие». На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

В третьем этапе сортировки для отрезков «меньших» и «больших» значений выполняется рекурсивно та же последовательность операций, если длина отрезка больше единицы. После завершение сортировки отрезков происходит их объединение в отсортированный массив (Рис. 1.1).

Быструю сортировку обязательно стоит рассматривать первой при выборе метода внутренней сортировки достаточно большого объёма данных. Алгоритм этой сортировки содержит сложную фазу разбиения и простую фазу слияния. В худшем случае выполненная работа эквивалентна работе при сортировке выбором.

## 1.2 Линейный поиск

Линейный, последовательный поиск — самый простой алгоритм поиска в программировании. Данный алгоритм является простейшим алгоритмом поиска и, в отличие, например, от двоичного поиска, не накладывает никаких ограничений и имеет простейшую реализацию (Рис. 1.2). Поиск значения функции осуществляется простым сравнением очередного рассматриваемого значения (как правило, поиск происходит слева направо, то есть от меньших значений аргумента к большим) и, если значения совпадают, то поиск считается завершённым. В связи с малой эффективностью по сравнению с другими алгоритмами линейный поиск обычно используют, только если отрезок поиска содержит очень мало элементов, тем не менее, линейный поиск не требует дополнительной памяти или обработки/анализа функции, так что может работать в потоковом режиме при непосредственном получении данных из любого источника.

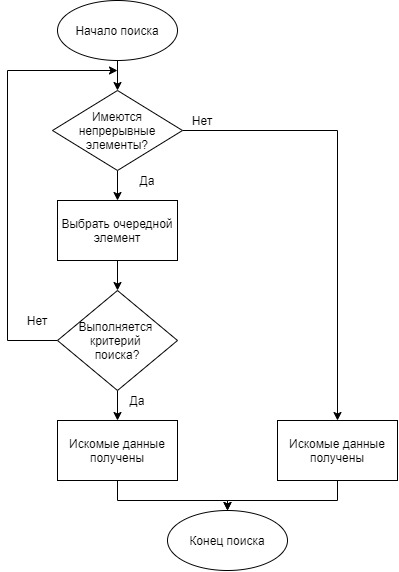


Рисунок 1.2 - Принцип работы линейного поиска

## 1.3 XOR-шифрование

Самым простым и одним из самых эффективных (при правильном использовании) алгоритмов шифрования является так называемое XOR-шифрование (Рис. 1.3). Простейший шифр на основе бинарной логики, который обладает высокой криптографической стойкостью. Без знания ключа, расшифровать его невозможно. Идея алгоритма заключается в том, что к каждому символу исходного применяется побитовая логическая операция XOR (в языке C операция XOR обозначается специальным знаком «^»). Операция XOR обладает симметричностью. Это значит, что если зашифровать один и тот же символ 2 раза с одним и тем же ключом, то на выходе получим сам этот файл без изменений. Из этого факта становится ясно, что для шифрования и расшифровывания будет использоваться одна и та же функция, что существенно упрощает реализацию алгоритма. Если же ключи при шифровке и дешифровке различаются, то на выходе будет получен некорректный символ.

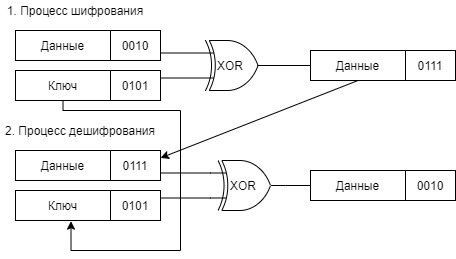


Рисунок 1.3 - Принцип работы XOR-шифрования

# 2.ОПИСАНИЕ ОРГАНИЗАЦИИ СТРУКТУР ХРАНИМЫХ ДАННЫХ

Структура – это сгруппированные под одним именем одна или несколько переменных (возможно, различных типов), объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации. Структуры упрощают написание и понимание принципов работы программ, а также помогают сгруппировать данные, объединяемые каким-либо общим понятием.

Программа, созданная в этом проекте, имеет 4 основные структуры:

Doctors;

Customers;

Appeals;

Date.

Структура Customers и Appeals имеют в себе вложенную структуру Data.

Структура Doctors.

typedef struct Doctors{

int ID;

char\* surname;

char\* name;

char\* middleName;

char\* specialty;

char\* category;

struct Doctors\* prev;

};

Структура Doctors содержит поле ID типа int, которое хранит личный номер врача, поле surname типа char\* хранит фамилию, поле name типа char\* хранит имя, поле middleName типа char\* хранит отчество, поле specialty типа char\* хранит специальность, поле category типа char\* хранит категорию доктора. Эта структура предназначена для хранения ФИО, специальности и категории врачей.

Структура Customers.

typedef struct Customers{

int ID;

char\* surname;

char\* name;

char\* middleName;

struct Date dateBirthday;

struct Customers\* prev;

};

Структура Customers содержит поле ID типа int, которое хранит личный номер пациента, поле surname типа char\* хранит фамилию, поле name типа char\* хранит имя, поле middleName типа char\* хранит отчество, поле dateBirthday типа Date хранит дату рождения пациента. Эта структура предназначена для хранения ФИО и даты рождения пациента.

Структура Appeals.

typedef struct Appeals{

int IDdoc;

int IDcust;

Date dateAppeal;

char\* diagnosis;

int costOfTreatment;

struct Appeals\* prev;

};

Структура Appeals содержит поле IDdoc типа int,которое хранит ID доктора, поле IDcust типа int хранит ID пациента, поле DateAppeals типа Date хранит дату обращения, поле diagnosis типа char\* хранит диагноз пациента, поле costOfTreatment типа int хранит стоимость лечения. Эта структура предназначена для хранения личных номеров докторов и пациентов, дату приёма, диагноз и стоимости лечения пациента.

Структура Date.

typedef struct Date{

int day;

int month;

int year;

};

Структура Date содержит поле day типа int, которое хранит день, поле

month типа int хранит месяц, поле year типа int хранит год. Эта структура предназначена для хранение даты.

# 3. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ ПРИЛОЖЕНИЯ

char\* readFromFile(FILE\* File); // считывание одной строки из файла

char\* strconv(char\* str); //привидение строки в нужную форму

void readingDataDoc(Doctors\*\* head, FILE\* Doc); //считывание информации в структуры с файла

void readingDataCust(Customers\*\* head, FILE\* Doc);//считывание информации в структуры с файла void readingDataApp(Appeals\*\* head, FILE\* Doc); //считывание информации в структуры с файла

void fprintfDoc(Doctors\*\* p, FILE\* File); //вывод в файл данных врача

void rewriteDoc(Doctors\*\* head, FILE\* File); //перезапись структуры в файл

void fprintfCust(Customers\*\* p, FILE\* File); // вывод в файл данных пациента

void rewriteCust(Customers\*\* head, FILE\* File); // перезапись структуры в файл

void fprintfApp(Appeals\*\* p, FILE\* File); // вывод в файл данных о приёме

void rewriteApp(Appeals\*\* head, FILE\* File); // перезапись структуры в файл

bool compare(char\*\* str, FILE\* File); //поиск введённых данных в файле

bool ReadAuthCheck(FILE\* File, char\* login, char\* pass, int mode); //проверка логина и пароля

int AuthFunction(int mode); //ввод логина и пароля

void registration(); //регистрация нового пользователя

void dataTableDoc(Doctors\*\* head); //все данные врачей в табличной форме

void dataTableCust(Customers\*\* head); //все данные пациентов в табличной форме

void dataTableApp(Appeals\*\* head); //все данные о приёмах в табличной форме

void pushDoc(Doctors\*\* head, FILE\* File); //добавление нового врача

void pushCust(Customers\*\* head, FILE\* File); //добавление нового пациента

void PushAppealCust(Appeals\*\* headApp, Customers\*\* headCust, FILE\* FileApp, FILE\* FileCust, int ID); //добавление нового приёма

void popDoc(Doctors\*\* head); //удаление врача

void popCust(Customers\*\* head); //удаление пациента

void editDataDoc(Doctors\*\* head); //изменение данных врача

void editDataCust(Customers\*\* head); //изменение данных пациента

void editDataApp(Appeals\*\* head); //изменение данных о приёме

char main\_menu(); //главное меню

void user\_menu(int ID); //меню врача

void admin\_menu(int ID); //меню главврача

void regist\_menu(); //меню медрегистратора

//дополняется..

# 4. Функциональная схема задачи, схемы алгоритмов работы приложения