МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | М. А. Мурашова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ «РЕГИСТРАЦИЯ БОЛЬНЫХ В ПОЛИКЛИНИКЕ» |
| по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | М011 |  |  |  | П.Н.Казакова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

Оглавление

[1. Задание на курсовой проект 3](#_Toc102394924)

[2. Введение 4](#_Toc102394925)

[3. Алгоритмы и структуры данных 5](#_Toc102394926)

[4. Описание программы 8](#_Toc102394927)

[5. Тестирование программы 22](#_Toc102394928)

[Заключение 30](#_Toc102394929)

[Список литературы 31](#_Toc102394930)

[Приложение 32](#_Toc102394931)

### 1. Задание на курсовой проект

- Предметная область: Регистрация больных в поликлинике

- Метод хеширования: Закрытое хеширование с двойным хешированием

- Метод сортировки: Слиянием

- Вид списка: Циклический двунаправленный

- Метод обхода дерева: Симметричный

- Алгоритм поиска слова в тексте: Прямой

### 2. Введение

Разрабатываемая информационная система, призвана облегчить для пользователя процесс записи на приём к врачу. Данные системы в ближайшем будущем будут иметь все частные медицинские компании, поэтому решение подобных задач, уже актуально и будет набирать популярность дальше. Разработанная система должна обеспечивать добавление пациента, добавление врача, выдачу направления и его возврат.

### 3. Алгоритмы и структуры данных

Структуры

Данные о пациенте хранятся в структуре «Patient»:

struct Patient

{

string regist\_number; // «MM-NNNNNN»,где MM – номер участка(цифры); NNNNNN – порядковый номер(цифры)

string all\_name; // ФИО

int year\_of\_birth; // Год рождения

string address; // Адрес

string place\_of\_work; // место работы

};

Данные о враче хранятся в структуре «Doctor»:

struct Doctor

{

string fio\_doctor; //ФИО врача

string position; // должность

int number\_cabinet; // номер кабинета

string time; // время работы

};

Данные о направлении хранятся в структуре «Ticket»:

struct Ticket

{

string registr;

string fio\_doctor;

string date;

string time;

};

Данные обо всех пациентах хранятся в хэш-таблице структуры «Hash»:

struct Hash

{

int collision = 0; // Кол-во коллизий

int position = 0; // Позиция

Patient\* patient=NULL; // больной

Hash\* next = NULL; // Следующий элемент двусвязного списка

Hash\* prev = NULL; // Предыдущий элемент двусвязного списка

string metka;

};

Структура «Three» - АВЛ дерево, хранящее в себе всех врачей:

struct Three

{

int height; // Высота поддерева

Doctor\* doctor = NULL; // Данные о враче

Three\* left = NULL; // Левый элемент поддерева

Three\* right = NULL; // Правый элемент поддерева

};

Данные о выданных направлениях хранятся в структуре «Data»:

struct Data

{

Ticket\* ticket;

Data\* next = NULL;

Data\* prev = NULL;};

Алгоритмы

**Двойное хеширование** — это метод разрешения коллизий в открытых адресных хеш-таблицах. Двойное хеширование использует идею применения второй хэш-функции к ключу при возникновении коллизий.

Преимущество двойного хеширования заключается в том, что это один из эффективных методов разрешения коллизий, обеспечивающий равномерное распределение записей по всей хеш-таблице.

Двойное хеширование использует хеш функцию вида

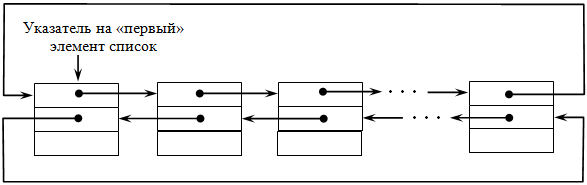
*h(к,i) = (h1(к) + i\*h2 (к)) mod m,*

где h1(к) и h2 — вспомогательные хеш функции

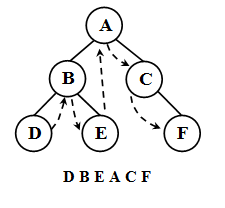
**Сортировка слиянием.** Объединение двух (или более) последовательностей в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент.

Процедура слияния предполагает объединение двух предварительно упорядоченных подпоследовательностей размерности n/2 в единую последовательность размерности n. Начальные элементы предварительно упорядоченных последовательностей сравниваются между собой, и из них выбирается наименьший. Соответствующий указатель перемещается на следующий элемент. Процедура повторяется до тех пор, пока не достигнут конец одной из подпоследовательностей. Оставшиеся элементы другой подпоследовательности при этом передаются в результирующую последовательность в неизменном виде.

**Циклический двунаправленный список** – имеет два указателя, один из  
которых указывает на следующий элемент в списке, а второй указывает на  
предыдущий элемент.



**Симметричный обход дерева** (Центричный) заключается в обходе сначала левого поддерева, а потом правого поддерева.



**Прямой поиск слова в тексте** - предполагается, что символы текста (массив T) и слова (массива W) будут сравниваться поэлементно.

На первом этапе работы алгоритма искомое слово сопоставляется с текстом так, что первый символ слова соотносится с первым символом текста. Далее происходит поэлементное сравнение первого символа слова с первым символом текста, второго символа слова со вторым символом текста и так далее. Если все символы совпадают, то результат работы алгоритма положительный (слово в тексте найдено). Если на каком-либо символе обнаружилось несоответствие, то происходит смещение слова на одну позицию так, что первый символ слова соответствует второму символу текста. Далее повторяется посимвольное сравнение. Смещение слова может повторяться до тех пор, пока не будет обнаружено полное совпадение символов либо пока не будет достигнут конец массива. Под достижением конца массива в данном случае понимается, что при попытке очередного смещения для последнего символа слова уже не будет соответствовать символ текста, так как текст закончился.

### 4. Описание программы

1) Данные о каждом больном должны содержать:

* Регистрационный номер – строка формата «MM-NNNNNN», где MM – номер участка (цифры), NNNNNN – порядковый номер (цифры);
* ФИО – строка;
* Год рождения – целое;
* Адрес – строка;
* Место работы (учебы) – строка.

Примечание – длина строк (кроме «Регистрационного номера») определяется студентом самостоятельно.

2) Данные о каждом враче должны содержать:

* ФИО врача – строка длиной до 25 символов, содержащая фамилию врача и его инициалы;
* Должность – строка;
* Номер кабинета – целое;
* График приема – строка.

Примечание – длина строк (кроме «ФИО врача») определяется студентом

самостоятельно.

3) Данные о выдаче или возврате направлений к врачу должны содержать:

* Регистрационный номер – строка, формат которой соответствует аналогичной строке в данных о больных;
* ФИО врача – строка, формат которой соответствует аналогичной строке в данных о врачах;
* Дата направления – строка;
* Время направления – строка.

Информационная система «Регистрация больных в поликлинике» должна осуществлять следующие операции:

– регистрацию нового больного;

– удаление данных о больном;

– просмотр всех зарегистрированных больных;

– очистку данных о больных;

– поиск больного по регистрационному номеру. Результаты поиска – все сведения о найденном больном и ФИО врача, к которому он имеет направление;

– поиск больного по его ФИО. Результаты поиска – список найденных больных с указанием регистрационного номера и ФИО;

– добавление нового врача;

– удаление сведений о враче;

– просмотр всех имеющихся врачей;

– очистку данных о врачах;

– поиск врача по «ФИО врача». Результаты поиска – все сведения о найденном враче, а также ФИО и регистрационные номера больных, которые имеют направление к этому врачу;

– поиск врача по фрагментам «Должность». Результаты поиска – список найденных врачей с указанием ФИО врача, должности, номера кабинета, графика приема;

– регистрацию выдачи больному направления к врачу;

– регистрацию возврата врачом или больным направления к врачу

Примечания:

1. Наличие в этих данных записи, содержащей в поле «Регистрационный номер» значение X и в поле «ФИО врача» значение Y означает выдача направления больному с регистрационным номером X к врачу с ФИО Y. Отсутствие такой записи означает, что больной с регистрационным номером X не имеет направления к врачу с ФИО Y.

2. К одному врачу могут направляться несколько больных в течение одного дня, но в разное время. Таким образом, могут быть данные, имеющие повторяющиеся значения в некоторых своих полях.

3. Поиск по фрагментам «Должности» осуществляется путем систематического обхода АВЛ-дерева поиска методом симметричного обхода. При поиске врача по фрагментам «Должности» могут быть заданы как полное наименование должности врача, так и его часть. Для обнаружения заданного фрагмента в должности врача должен применяться алгоритм поиска слова в тексте – прямой

4. Регистрация выдачи направлений к врачу на определенную дату и время должна осуществляться только при отсутствии уже выданного направления к этому же врачу на те же дату и время.

*Основные функции хеш-таблицы*

// Создание хэш-таблицы

void create\_hash\_table(Hash\*& start\_ht, Hash\*& end\_ht)

{

for (int i = 0; i < SIZE\_TABLE; i++)

{

if (i == 0)

{

end\_ht = start\_ht;

continue;

}

Hash\* append = new Hash;

append->position = i;

append->next = start\_ht;

append->prev = end\_ht;

end\_ht->next = append;

end\_ht = append;

append->metka = "";

append->patient = NULL;

}

}

Добавление больного

void add\_patient(Hash\*& start\_ht, Hash\*& end\_ht, int& k)

{

Patient\* bolnoy = new Patient;

int hash1;

string key;

cout << "Введите номер участка (ММ-цифры); ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, key);

checkM(key);

if (isfull(start\_ht) == 0)

{

cout << "Table is full" << endl;

system("pause");

return;

}

bolnoy->regist\_number = generate\_regist\_number(key,k);

cout << "Введите ФИО больного: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, bolnoy->all\_name);

bolnoy->all\_name=check\_fio(bolnoy->all\_name);

cout << "Введите год рождения больного: ";

check\_year(bolnoy->year\_of\_birth);

cout << "Введите адрес проживания: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, bolnoy->address);

bolnoy->address = check\_adress(bolnoy->address);

cout << "Введите место работы (учебы): ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, bolnoy->place\_of\_work);

bolnoy->place\_of\_work=check\_work(bolnoy->place\_of\_work);

system("cls");

hash1 = hash\_position(bolnoy->regist\_number); // Вычисление позиции в таблице

// если таблица не создана

if (start\_ht->next == NULL)

{

create\_hash\_table(start\_ht, end\_ht); // создаём таблицу

Hash\* help = start\_ht;

// помещаем в таблицу больного

for (int i = 0; i <= hash1; i++)

{

if (help->position == hash1)

{

// помещаем в таблицу и выходим из функции

help->patient = bolnoy;

return;

}

help = help->next;

}

}

// создаём вспомогательную переменную для итераций по таблице

Hash\* help = start\_ht;

// ищем место в таблице и обрабатываем коллизии

for (int i = 0; i <= (hash1+1); i++)

{

// если хэш совпал и по этому адресу не хранится элемент

if (help->position == hash1 && help->patient == NULL)

{

help->patient = bolnoy;

if (help->metka == "del")

help->metka = "";

return;

}

// иначе если хэш совпал и по этому адресу хранится элемент

else if (help->position == hash1 && help->patient != NULL)

{

help->collision++; // добавляем коллизию

int hash2 = hash\_position2(bolnoy->regist\_number);

for (int j = 0;; j++)

{

hash1 = (hash1 + j \* hash2) % SIZE\_TABLE; // двойное хеширование

help = start\_ht; // создаём переменную для поисковых итераций

for (int n = 0; n <= (hash1+1); n++)

{

// если хэш совпал и по этому адресу не хранится элемент

if (help->position == hash1 && help->patient == NULL)

{

help->patient = bolnoy;

if (help->metka == "del")

help->metka = "";

return;

}

// если хэш совпал и по этому адрусу хранится элемент

else if (help->position == hash1 && help->patient != NULL)

{

help->collision++;

break;

}

help = help->next;

}

}

return;

}

help = help->next;

}

}

Удаление больного

void del\_patient(Hash\*& start\_ht, Hash\*& end\_ht,Data\*& head, Data\*& tail)

{

if (start\_ht->next == NULL)

{

cout << "======================================" << endl;

cout << "Больных нет\n";

cout << "======================================" << endl;

system("pause");

return;

}

string registr; // тут храним регистрационнный номер

cout << "Введите регистрационный номер «MM-NNNNNN»,где MM – номер участка(цифры); NNNNNN – порядковый номер(цифры): ";

checkRegist(registr); // вводим регистрационный номер корректно

if (check\_registr\_in\_data(head, tail, registr))

{

cout << "===========================================================" << endl;

cout << "Удаление больного невозможно, у него есть направление к врачу\n";

cout << "===========================================================" << endl;

system("pause");

return;

}

Hash\* help = start\_ht;

int count = 0; // для проверки наличия больного в списке

int hash1 = hash\_position(registr);

int hash2 = hash\_position2(registr);

for (int i = 0; i <= hash1; i++)

{

if (help->position == hash1 && help->patient != NULL && help->patient->regist\_number == registr)

{

count++;

delete help->patient;

help->patient = nullptr;

help->patient = NULL;

help->collision = 0;

help->metka = "del";

cout << "======================================" << endl;

cout << "Больной удалён\n";

cout << "======================================" << endl;

system("pause");

return;

}

else if (help->position == hash1 && (help->patient == NULL || help->patient != NULL && help->patient->regist\_number != registr))

{

for (int j = 0; j < SIZE\_TABLE \* 11; j++)

{

hash1 = (hash1 + j \* hash2) % SIZE\_TABLE;

help = start\_ht;

for (int n = 0; n <= hash1; n++)

{

if (help->position == hash1 && help->patient != NULL && help->patient->regist\_number == registr)

{

count++;

delete help->patient;

help->patient = nullptr;

help->patient = NULL;

help->collision = 0;

help->metka = "del";

cout << "======================================" << endl;

cout << "Больной удалён\n";

cout << "======================================" << endl;

system("pause");

return;

}

help = help->next;

}

if (count != 0)

break;

}

break;

}

help = help->next;

}

if (count==0)

{

cout << "======================================" << endl;

cout << "Больного с таким регистрационным номером нет\n";

cout << "======================================" << endl;

}

system("pause");

}

Вывод больных

void show\_patient(Hash\*& start\_ht, Hash\*& end\_ht)

{

Hash\* help = start\_ht;

int count = 0; // для проверки наличия больных в списке

for (int i = 0; i < SIZE\_TABLE; i++)

{

if (start\_ht->next == NULL)

{

cout << "Больных нет\n";

cout << "================================" << endl;

system("pause");

return;

}

if (help->patient != NULL)

{

count++;

cout << "ФИО больного: " << help->patient->all\_name << endl;

cout << "Регистрационный номер: " << help->patient->regist\_number << endl;

cout << "Год рождения: " << help->patient->year\_of\_birth << endl;

cout << "Адрес: " << help->patient->address << endl;

cout << "Место работы (учебы): " << help->patient->place\_of\_work << endl;

cout << "position number: " << help->position << endl;

cout << "================================" << endl;

}

help = help->next;

}

if (!count)

{

cout << "Больных нет\n";

cout << "================================" << endl;

}

system("pause");

}

Получение хеша

int HashFunctionHorner(string s, int key)

{

int hash\_result = 0;

for (int i = 0; i < s.size(); i++)

hash\_result = (key \* hash\_result + s[i]) % SIZE\_TABLE;

hash\_result = (hash\_result \* 2 + 1) % SIZE\_TABLE;

return hash\_result;

}

// первая хеш функция

int hash\_position(string r)

{

return HashFunctionHorner(r, SIZE\_TABLE - 1);

}

//вторая хеш функция

int hash\_position2(string r)

{

return HashFunctionHorner(r, SIZE\_TABLE + 1);

}

*Основные функции АВЛ-дерева*

//для проверки высоты

int height(Three\*& element)

{

if (element == NULL)

{

return -1;

}

else

{

return element->height;

}

}

//малый левый поворот

Three\* single\_left\_rotate(Three\*& element)

{

Three\* help = element->right;

element->right = help->left;

help->left = element;

element->height = max(height(element->left), height(element->right)) + 1;

help->height = max(height(element->left), element->height) + 1;

return help;

}

//малый правый поворот

Three\* single\_right\_rotate(Three\*& element)

{

Three\* help = element->left;

element->left = help->right;

help->right = element;

element->height = max(height(element->left), height(element->right)) + 1;

help->height = max(height(help->left), element->height) + 1;

return help;

}

//большой левый поворот

Three\* big\_left\_rotate(Three\*& element)

{

element->right = single\_right\_rotate(element->right);

return single\_left\_rotate(element);

}

//большой правый поворот

Three\* big\_right\_rotate(Three\*& element)

{

element->left = single\_left\_rotate(element->left);

return single\_right\_rotate(element);

}

Добавление элементов в дерево

void add\_doctor(Three\*& element, string value)

{

if (element == NULL)

{

Doctor\* r = new Doctor;

r->fio\_doctor = value;

cout << "Введите должность: ";

r->position = check\_position();

cout << "Введите номер кабинета: ";

r->number\_cabinet= check\_cabinet();

cout << "Введите график работы:";

r->time = check\_time();

element = new Three;

element->height = 0;

element->doctor = r;

}

else

{

if (value < element->doctor->fio\_doctor)

{

add\_doctor(element->left, value);

//Если произошла разбалансировка

if (height(element->left) - height(element->right) == 2)

{

if (value < element->left->doctor->fio\_doctor)

{

element = single\_right\_rotate(element);

}

else

{

element = big\_right\_rotate(element);

}

}

}

else if (value > element->doctor->fio\_doctor)

{

add\_doctor(element->right, value);

if (height(element->right) - height(element->left) == 2)

{

if (value > element->right->doctor->fio\_doctor)

{

element = single\_left\_rotate(element);

}

else

{

element = big\_left\_rotate(element);

}

}

}

}

element->height = max(height(element->left), height(element->right)) + 1;

}

Вывод врачей

void show\_doctors(const Three\* element)

{

if (element == NULL)

{

return;

}

show\_doctors(element->left); // Обошли левое поддерево

cout << "ФИО врача: " << element->doctor->fio\_doctor << endl;

cout << "Должность: " << element->doctor->position << endl;

cout << "Номер кабинета: " << element->doctor->number\_cabinet << endl;

cout << "График работы: " << element->doctor->time << endl;

cout << "====================================\n";

show\_doctors(element->right); // Обошли правое поддерево

}

Удаление элемента из дерева

// Удаление минимального значения

Doctor\* deletemin(Three\*& element)

{

Doctor\* a;

if (element->left == NULL)

{

a = element->doctor;

element = element->right;

return a;

}

else

{

a = deletemin(element->left);

return a;

}

}

void del\_doctor(Three\*& element, string value, Data\*& head, Data\*& tail)

{

if (check\_fio\_in\_data(head, tail, value))

{

cout << "Удаление врача невозможно, к нему есть направление\n";

cout << "===========================================================" << endl;

return;

}

Three\* d;

if (element == NULL)

{

cout << "Такого врача нет" << endl;

cout << "======================================" << endl;

}

else if (value < element->doctor->fio\_doctor)

{

del\_doctor(element->left, value, head, tail);

}

else if (value > element->doctor->fio\_doctor)

{

del\_doctor(element->right, value, head, tail);

}

else if ((element->left == NULL) && (element->right == NULL))

{

d = element;

free(element->doctor);

element->doctor = nullptr;

free(d);

d = nullptr;

element = nullptr;

cout << "Врач удален" << endl;

cout << "======================================" << endl;

}

else if (element->left == NULL)

{

Three\* help = element->right;

d = element;

delete element->doctor;

element->doctor = nullptr;

delete d;

d = nullptr;

element = help;

cout << "Врач удален" << endl;

cout << "======================================" << endl;

}

else if (element->right == NULL)

{

d = element;

element = element->left;

free(element->doctor);

element->doctor = nullptr;

free(d);

d = nullptr;

cout << "Врач удален" << endl;

cout << "======================================" << endl;

}

else

{

free(element->doctor);

element->doctor = nullptr;

element->doctor = deletemin(element->right);

cout << "Врач удален" << endl;

cout << "======================================" << endl;

}

}

*Основные функции списка*

Добавление направления в список

void Insert(Data\*& head, Data\*& tail, Hash\*& start\_ht, Hash\*& end\_ht, Three\*& element)

{

Ticket\* data2 = new Ticket;

string registr;

cout << "Введите регистрационный номер\n";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, registr);

data2->registr = registr;

if (search\_data(start\_ht, end\_ht, registr))

{

s++;

cout << "Введите фио доктора\n";

string fio;

fio = check\_fio\_doctor();

while (!search\_doc(element, fio))

{

cout << "Сведения о враче отсутствуют." << endl;

cout << "Введите фио доктора повторно\n";

fio = check\_fio\_doctor();

}

data2->fio\_doctor = fio;

string date1;

cout << "Введите дату выдачи номерка(пример: 03.06.2022): ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, date1);

while (Date(date1))

{

cin.sync();

if (Date(date1))

{

cout << "Некорректный ввод, введите дату приема: ";

getline(cin, date1);

}

}

data2->date = date1;

cout << "Введите время приема\n";

string time;

time = time1();

Data\* st = head;

while (!searchtime(data2, st,time))

{

cout << "Запись на это время занята. Введите другое время" << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, time);

}

data2->time = time;

Data\* help = new Data;

help->ticket = data2;

if (head == NULL)

{

head = help;

tail = help;

tail->next = head;

tail->prev = help;

head->prev = tail

}

else

{

help->next = tail->next;

help->next->prev = help;

tail->next = help;

help->prev = tail;

tail = help;

if (s > 1)

{

Data\* st = head;

st->prev = NULL;

tail->next = NULL;

head = sort(st, tail, s);

Data\* help = head;

while (help->next != 0)

{

tail = help->next;

tail->prev = help;

help = help->next;

}

head->prev = tail;

tail->next = head;

}

}

}

else

{

cout << "Данного регистрационного номера нет в базе\n";

cout << "======================================================" << endl;

return;

}

}

Удаление направления

void Delete(Data\*& head, Data\*& tail, Hash\*& start\_ht, Hash\*& end\_ht, Three\*& element)

{

Ticket\* data2 = new Ticket;

string registr;

cout << "Введите регистрационный номер\n";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, registr);

data2->registr = registr;

if (search\_data(start\_ht, end\_ht, registr))

{

cout << "Введите фио доктора\n";

string fio;

fio = check\_fio\_doctor();

while (!search\_doc(element, fio))

{

cout << "Сведения о враче отсутствуют." << endl;

cout << "Введите фио доктора повторно\n";

fio = check\_fio\_doctor();

}

data2->fio\_doctor = fio;

cout << "Введите дату приема\n";

string date;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, date);

data2->date = date;

cout << "Введите время приема\n";

string time;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, time);

Data\* help = head;

if (searchtime(data2, help, time))

{

cout << "Записи нет в базе\n";

cout << "======================================================" << endl;

return;

}

else

{

s--;

data2->time = time;

Data\* help = head;

// если всего 1 элемент

if (head == tail)

{

delete head;

head = nullptr;

tail = nullptr;

cout << "Направление возвращено\n";

cout << "======================================================" << endl;

return;

}

// если удаляем элемент откуда-то из центра

else

{

// бежим, пока не встретим следующим элементом удаляемый

do

{

if (data2->registr == help->ticket->registr && data2->date == help->ticket->date && time == help->ticket->time && data2->fio\_doctor == help->ticket->fio\_doctor)

{

help->prev->next = help->next;

help->next->prev = help->prev;

delete help;

help = nullptr;

break;

}

help = help->next;

} while (help != head);

cout << "Направление возвращено\n";

cout << "======================================================" << endl;

return;

}

}

}

else

{

cout << "Данного регистрационного номера нет в базе\n";

cout << "======================================================" << endl;

return;

}

}

Просмотр всех направлений

void show(Data\*& head, Data\*& tail)

{

Data\* help = head;

if (head == NULL)

{

cout << "Записей нет.\n";

return;

}

do

{

cout << "Регистрационный номер: " << help->ticket->registr << endl;

cout << "ФИО врача: "<< help->ticket->fio\_doctor << endl;

cout <<"Дата приема: "<< help->ticket->date << endl;

cout <<"Время приема: "<< help->ticket->time << endl;

cout << "====================================\n";

help = help->next;

} while (help != head);

}

### 5. Тестирование программы

Для проверки корректности работы приложения были использованы тестовые данные, приведенные в таблице 4.1.

Таблица 4.1 – Тестовые данные

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название этапа тестирования | Тестовые данные | Ожидаемый результат |
| 1 | **Главное меню** | - | Вывод на консоль окна меню |
|  | Некорректный ввод пункта меню | Sdf | Ошибка! Повторите ввод! |
| Ыва | Ошибка! Повторите ввод! |
| 4.5 | Ошибка! Повторите ввод! |
| 16 | Ошибка! Повторите ввод! |
| Корректный ввод пункта меню | 0 | Завершение программы |
| 2 | **Регистрация больного (меню больного)** | Пункт 1 |  |
|  | Некорректный ввод ФИО | Sdf | S - Недопустимый символ! |
| 4.5 | 4 - Недопустимый символ! |
| Казакова пн | [б] - Недопустимый символ! |
| .Казакова ПН | [.] - Недопустимый символ! |
| Некорректный ввод года рождения | 1800 | Ошибка! Повторите ввод! |
| 2222 | Ошибка! Повторите ввод! |
| Более 50 символов | Ошибка! Повторите ввод |
| 22 | Ошибка! Повторите ввод! |
| Некорректный ввод регистрационного номера | Sdf  0  -55 | Ошибка! Повторите ввод! |
| Некорректный адрес  Некорректное место работы | fjhgh | Ошибка! Повторите ввод! |
| 45546 | Ошибка! Повторите ввод! |
|  |  |
| Корректная регистрация пациента | 01  Казакова Полина Николаевна  2000  Варшавская 8  ГУАП | Регистрационный номер добавлен.  Данные о новых больных добавлены |
| 3 | **Удаление больного по регистрационному номеру** | Пункт 2 |  |
| Ввод несуществующего номера | 02-000001 | Пациента отсутствует |
| Ввод номера, у которого есть направление | 01-000001 | У данного пациента есть направление к врачу |
| Ввод номера, который существует и без направлений | 02-000003 | Больной удален |
| 4 | **Просмотр списка зарегистрированных больных** | Пункт 3 | Вывод таблицы с информацией о зарегистрированных больных |
| 5 | **Поиск больного по регистрационному номеру** | Пункт 4 | Вывод таблицы с информацией о зарегистрированных больных |
| Ввод регистрационного номера, которого нет в БД  Некорректный ввод регистрационного номера | 04-000009 | Пациент с веденным регистрационным номером отсутствует. |
| 14-00009 | Ошибка! Повторите ввод |
| Ввод регистрационного номера, который есть в БД | 01-000001 | Вывод информации о найденном больном и докторе(если есть направление) |
| 6 | **Поиск больного по ФИО** | Пункт 5 |  |
| Ввод ФИО, которого нет в БД | Соколов Иван Иванович | Данного пациента нет. |
| Ввод ФИО, которое есть в БД  Если есть хотя бы одно выданное направление | Казакова Полина Николаевна | Вывод информации о найденном больном |
| 7 | **Удаление всех пациентов** | Пункт 6 |  |
| Если есть хоть одно направление |  | Удаление невозможно. У пациентов есть направления |
| Если направлений нет |  | Пациенты удалены |
| 8 | **Добавление нового врача (меню врача)** | Пункт 1 |  |
|  | Некорректный ввод ФИО (используется та же функция проверки, что и во втором пункте) | - | - |
| Некорректный ввод должности врача (функция проверки должности ничем не отличается функции проверки ФИО во втором пункте) | - | - |
| Некорректный ввод номера кабинета врача | 0 | Ошибка! Повторите ввод! |
| 29 | Ошибка! Повторите ввод! |
| Некорректный ввод графика работы | 23:00-09:00 | Ошибка! Повторите ввод |
| 00:00-09:00 | Ошибка! Повторите ввод |
| Корректное добавление врача | Казакова П.Н.  Кардиолог  20  09:00-17:00 | Данные о новых врачах добавлены |
| 9 | **Просмотр списка врачей** | Пункт 3 | Вывод таблицы с информацией о добавленных врачах |
| 10 | **Удаление сведений о враче** | Пункт 2 |  |
| Некорректный ввод ФИО (используется та же функция проверки, что и во втором пункте) | - | - |
| Ввод ФИО, которого нет в БД | Агутин Н.Н. | Доктор не найден. |
| Корректный ввод ФИО | Сидоров П.П. | Данные о докторе успешно удалены. |
| Ввод ФИО врача, к которому есть направление | Казакова П.Н. | Удаление невозможно! К доктору имеется направление! |
| 11 | **Поиска врача по ФИО** | Пункт 4 |  |
| Некорректный ввод ФИО (используется та же функция проверки, что и во втором пункте) | - | - |
| Ввод ФИО врача, которого нет в БД | Агутин Н.Н. | Доктор с веденным ФИО не найден. |
| Корректный ввод ФИО врача | Казакова П.Н. | Вывод информации о враче и регистрационный номер пациента (если у пациента есть направление к данному врачу) |
| 12 | **Поиск врача по фрагменту должность** | Пункт 5 |  |
| Ввод фрагмента, которого нет в БД | чьъ | Вывод таблицы с найденными врачами (в данном случае пустой таблицы) |
| Ввод фрагмента, который есть в БД | лог | Вывод таблицы с найденными врачами |
| 13 | **Очистка базы данных врачей** | Пункт 6 |  |
| Если есть хотя бы один врач с выданным на него направлением |  | Базу данных очистить нельзя! Имеются выданные направления! |
| Если направлений к врачу нет |  | База данных врачей полностью очищена |
| 14 | **Выдача направлений пациенту (меню направлений)** | Пункт 1 |  |
| Ввод регистрационного номера (осуществляются все те же проверки что и в пункте 2) | - | - |
| Ввод ФИО врача (осуществляются все те же проверки что и в пункте 8) | - | - |
| Ввод времени приема врача (осуществляются все те же проверки что и в пункте 8) | - | - |
| Некорректный ввод даты приема у врача | 12.02.2020 | Ошибка! Повторите ввод! |
| вамапип | Ошибка! Повторите ввод! |
| 06689 | Ошибка! Повторите ввод! |
| 12.01.3000 | Ошибка! Повторите ввод! |
| 12.02.2022 | Ошибка! Повторите ввод! |
| Корректная выдача направления | 01-000001  Казакова П.Н.  02.06.2022  10:00 | Направление добавлено |
| 15 | **Возврат направления у пациента** | Пункт 2 | Вывод БД направлений |
|  | Ввод регистрационного номера (осуществляются все те же проверки что и в пункте 2) | - | - |
|  | Ввод ФИО врача (осуществляются все те же проверки что и в пункте 8) | - | - |
|  | Ввод даты приема врача (осуществляются все те же проверки что и в пункте 8) | - | - |
|  | Корректный возврат направления | 01-000001  Казакова П.Н.  02.06.2022  10:00 | Направление успешно удалено. |
| 16 | **Просмотр выданных направлений** | Пункт 3 | Вывод информации о выданных направлениях |

Изображение выглядит как текст

Автоматически созданное описание

Рис.1. Меню (проверка на ввод пункта)

Изображение выглядит как текст

Автоматически созданное описание

Рис.2. Меню больного

Изображение выглядит как текст

Автоматически созданное описание

Рис. 3. Меню врача

Изображение выглядит как текст

Автоматически созданное описание

Рис.4. Меню выдачи и возврата направления

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рис 5. Добавление больного (с проверкой на ввод)

Изображение выглядит как текст

Автоматически созданное описание

Рис 6. База больных

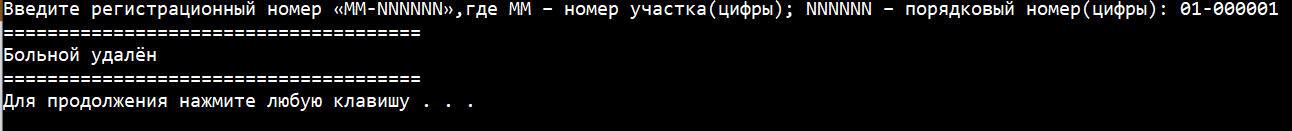


Рис 7. Удаление существующего больного

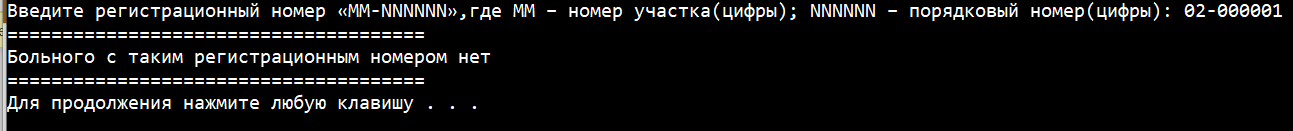


Рис 8. Удаление несуществующего больного

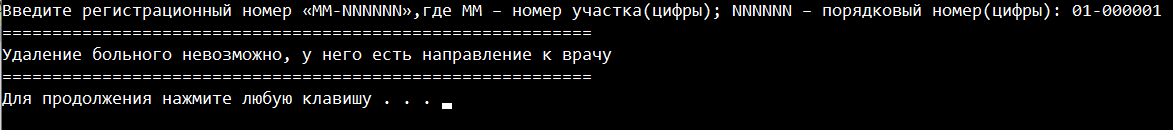


Рис 9. Удаление больного, у которого есть направление

Изображение выглядит как текст

Автоматически созданное описание

Рис 10. Поиск больного по регистрационному номеру

Изображение выглядит как текст

Автоматически созданное описание

Рис 11. Поиск больного по ФИО

Изображение выглядит как текст

Автоматически созданное описание

Рис 12. Поиск несуществующего больного

Изображение выглядит как текст

Автоматически созданное описание

Рис 13. Удаление всех больных

Изображение выглядит как текст

Автоматически созданное описание

Рис 14. Добавление врача (с проверкой на ввод)

Изображение выглядит как текст

Автоматически созданное описание

Рис 15. База с врачами

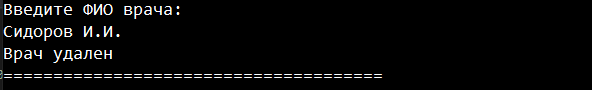


Рис 16. Удаление врача

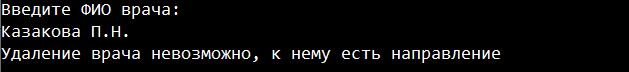


Рис 17. Удаление врача, к которому выдано направление

Изображение выглядит как текст

Автоматически созданное описание

Рис 18. Поиск врача по ФИО

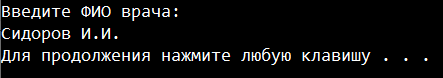


Рис 19. Поиск несуществующего врача

Изображение выглядит как текст

Автоматически созданное описание

Рис 20. Поиск врача по фрагменту должности



Рис 21. Удаление всех врачей

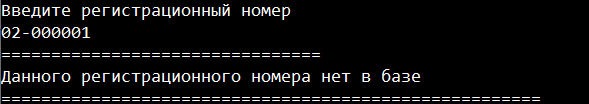


Рис 22. Выдача направления несуществующему больному

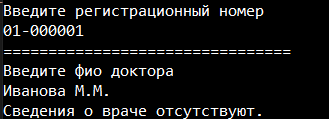


Рис 23. Выдача направления к несуществующему врачу

Изображение выглядит как текст

Автоматически созданное описание

Рис 24. Выдача направления (с проверкой на ввод даты)

Изображение выглядит как текст

Автоматически созданное описание

Рис 25. Выдача направления на занятое время

Изображение выглядит как текст

Автоматически созданное описание

Рис 26. Возврат несуществующего направления

Изображение выглядит как текст

Автоматически созданное описание

Рис 27. Возврат направления

Изображение выглядит как текст, черный, табличка

Автоматически созданное описание

Рис 28. База с направлениями

Изображение выглядит как текст

Автоматически созданное описание

Рис 29. Хранение в файле данных о пациентах

Изображение выглядит как текст

Автоматически созданное описание

Рис 30. Хранение в файле данных о врачах

Изображение выглядит как текст

Автоматически созданное описание

Рис 31. Хранение в файле данных о выданных направлениях

### Заключение

В ходе выполнения курсовой работы, была разработана информационная система по регистрации больных в поликлинике. Данная система удовлетворяет заданным требованиям и решает все поставленные задачи.

Во время выполнения работы были повторены используемые алгоритмы, а также реализации структур данных, необходимых для организации информационной системы.

### Список литературы

1) Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / ГУАП. СПб., 2004.

2) Вирт Н. Алгоритмы и структуры данных. М: Мир, 1989.

### Приложение

#include <iostream>

#include <string>

#include <Windows.h>

#include <time.h>

#include<fstream>

#include <stdlib.h>

#include <stdio.h>

using namespace std;

int SIZE\_TABLE=100; // размер таблицы

int s=0;

// Основное меню

void menu\_main()

{

cout << "Меню\n"

<< "1. Данные о больном\n"

<< "2. Данные о врачах\n"

<< "3. Данные о выдаче и возврате направлений к врачу\n"

<< "0. Завершение работы\n";

cout << "=====================\nВыберите действие: ";

}

// Меню больного

void menu\_patients()

{

cout << "Больной\n"

<< "1. Регистрация больного\n"

<< "2. Удаление больного\n"

<< "3. Просмотр больных\n"

<< "4. Поиск больного по регистрационному номеру\n"

<< "5. Поиск больного по ФИО\n"

<< "6. Удаление всех больных\n"

<< "7. Вывести больных из файла\n"

<< "0. Возврат в меню\n";

cout << "=======================================\nВыберите действие: ";

}

// Меню врача

void menu\_doctors()

{

cout << "Врач\n"

<< "1. Добавление врача\n"

<< "2. Удаление врача\n"

<< "3. Просмотр всех врачей\n"

<< "4. Поиск врача по ФИО\n"

<< "5. Поиск врача по должности\n"

<< "6. Удаление всех врачей\n"

<< "7. Вывести врачей из файла\n"

<< "0. Возврат в меню\n";

cout << "==============================\nВыберите действие: ";

}

// Меню для выдачи и возврата направления

void menu\_issue\_and\_refund()

{

cout << "Выдача и возврат направления к врачу\n"

<< "1. Выдача направления\n"

<< "2. Возврат направления\n"

<<"3.Посмотреть все направления\n"

<<"4.Загрузить направления из файла\n"

<< "0. Возврат в меню\n";

cout << "=============================\nВыберите действие: ";

}

//больной

struct Patient

{

string regist\_number; // «MM-NNNNNN»,где MM – номер участка(цифры); NNNNNN – порядковый номер(цифры)

string all\_name; // ФИО

int year\_of\_birth; // Год рождения

string address; // Адрес

string place\_of\_work; // место работы

};

// Структура хэш-таблицы

struct Hash

{

int collision = 0; // Кол-во коллизий

int position = 0; // Позиция

Patient\* patient=NULL; // больной

Hash\* next = NULL; // Следующий элемент двусвязного списка

Hash\* prev = NULL; // Предыдущий элемент двусвязного списка

string metka;

};

//врач

struct Doctor

{

string fio\_doctor; //ФИО врача

string position; //

int number\_cabinet; //

string time; //

string time\_1;

};

//структура дерева

struct Three

{

int height; // Высота поддерева

Doctor\* doctor = NULL; // Данные о враче

Three\* left = NULL; // Левый элемент поддерева

Three\* right = NULL; // Правый элемент поддерева

};

//направление

struct Ticket

{

string registr;

string fio\_doctor;

string date;

string time;

};

//структура списка

struct Data

{

Ticket\* ticket;

Data\* next = NULL;

Data\* prev = NULL;

};

// Создание хэш-таблицы

void create\_hash\_table(Hash\*& start\_ht, Hash\*& end\_ht)

{

for (int i = 0; i < SIZE\_TABLE; i++)

{

if (i == 0)

{

end\_ht = start\_ht;

continue;

}

Hash\* append = new Hash;

append->position = i;

append->next = start\_ht;

append->prev = end\_ht;

end\_ht->next = append;

end\_ht = append;

append->metka = "";

append->patient = NULL;

}

}

//получение хеша

int HashFunctionHorner(string s, int key)

{

int hash\_result = 0;

for (int i = 0; i < s.size(); i++)

hash\_result = (key \* hash\_result + s[i]) % SIZE\_TABLE;

hash\_result = (hash\_result \* 2 + 1) % SIZE\_TABLE;

return hash\_result;

}

// первая хеш функция

int hash\_position(string r)

{

return HashFunctionHorner(r, SIZE\_TABLE - 1);

}

//вторая хеш функция

int hash\_position2(string r)

{

return HashFunctionHorner(r, SIZE\_TABLE + 1);

}

// Проверка на ввод регистр. номера

void checkRegist(string& regist)

{

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, regist);

while (true)

{

//«MM-NNNNNN» MM – номер участка(цифры), NNNNNN – порядковый номер(цифры)

if ((int)regist[0] >= 48 && (int)regist[0] <= 57 && (int)regist[1] >= 48 && (int)regist[1] <= 57

&& (int)regist[2] == 45 && (int)regist[3] >= 48 && (int)regist[3] <= 57

&& (int)regist[4] >= 48 && (int)regist[4] <= 57 && (int)regist[5] >= 48 && (int)regist[5] <= 57

&& (int)regist[6] >= 48 && (int)regist[6] <= 57 && (int)regist[7] >= 48 && (int)regist[7] <= 57

&& (int)regist[8] >= 48 && (int)regist[8] <= 57 && regist.size() == 9)

{

break;

}

else

{

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cout << "Введите регистрационный номер повторно «MM-NNNNNN»,где MM – номер участка(цифры); NNNNNN – порядковый номер(цифры): ";

getline(cin, regist);

}

}

}

// Проверка ввода номера меню общего

int check\_menu(int& m)

{

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> m;

while ((cin.fail()) || (cin.get() != '\n') || (m < 0) || (m > 3))

{

cout << "Введите номер меню повторно: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> m;

}

return m;

}

//проверка на ввод пункта меню больных и врачей

int check\_menu1(int& m)

{

cin >> m;

while ((cin.fail()) || (cin.get() != '\n') || (m < 0) || (m > 7))

{

cout << "Введите номер меню повторно: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> m;

}

return m;

}

//проверка пункта меню направлений

int check\_menu2(int& m)

{

cin >> m;

while ((cin.fail()) || (cin.get() != '\n') || (m < 0) || (m > 4))

{

cout << "Введите номер меню повторно: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> m;

}

return m;

}

//проверка на ввод года рождения

int check\_year(int& m)

{

cin >> m;

while ((cin.fail()) || (cin.get() != '\n') || (m < 1920) || (m > 2022))

{

cout << "Введите год рождения повторно: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> m;

}

return m;

}

//проверка введенных ФИО больного

string check\_fio(string z)

{

int check = 0;

int kolvo = 0;

for (int i = 0; i < z.length(); i++) {

if (int('А') <= int(z[i]) && int(z[i]) <= int('п') ||

int('р') <= int(z[i]) && int(z[i]) <= int('я') ||

int(z[i]) == int('ё') ||

int(z[i]) == int('Ё') ||

int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length()+1; i++) {

if (int(z[i]) == 32) kolvo++;

if (int(z[i]) == 32 && int(z[i + 1]) == 32 || int(z[0]) == int('\0') || int(z[0])==int(' '))

check = 0;

if (int(z[i]) == 32 && !(int('А') <= int(z[i+1]) && int(z[i+1]) <= int('Я')|| int(z[i]) == int('Ё')) || (!(int('А') <= int(z[0]) && int(z[0]) <= int('Я')|| int(z[i]) == int('Ё'))))

check = 0;

}

while ((cin.fail()) || check == 0 || kolvo != 2)

{

kolvo = 0;

cout << "Некорректный ввод!. Повторите: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

for (int i = 0; i < z.length(); i++) {

if (int('А') <= int(z[i]) && int(z[i]) <= int('п') ||

int('р') <= int(z[i]) && int(z[i]) <= int('я') ||

int(z[i]) == int('ё') ||

int(z[i]) == int('Ё') ||

int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length(); i++) {

if (int(z[i]) == 32) kolvo++;

if (int(z[i]) == 32 && int(z[i + 1]) == 32 || int(z[0]) == int('\0') || int(z[0]) == int(' '))

check = 0;

if (int(z[i]) == 32 && !(int('А') <= int(z[i + 1]) && int(z[i + 1]) <= int('Я') || int(z[i+1]) == int('Ё')) || (!(int('А') <= int(z[0]) && int(z[0]) <= int('Я')|| int(z[0]) == int('Ё'))))

check = 0;

}

}

return z;

}

//проверка введенного места работы

string check\_work(string z)

{

int check = 0;

int kolvo = 0;

for (int i = 0; i < z.length(); i++) {

if (int('a') <= int(z[i]) && int(z[i]) <= int('z') ||

int('A') <= int(z[i]) && int(z[i]) <= int('Z') ||

int('А') <= int(z[i]) && int(z[i]) <= int('п') ||

int('р') <= int(z[i]) && int(z[i]) <= int('я') ||

int(z[i]) == int('ё') ||

int(z[i]) == int('Ё') ||

int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length() + 1; i++) {

if (int(z[0]) == int('\0'))

{

check = 0; break;

}

else if (int(z[i]) == 32 && int(z[i + 1]) == 32 || (int(z[z.length() - 1] == 32)) || int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

}

while ((cin.fail()) || check == 0)

{

kolvo = 0;

cout << "Некорректный ввод!. Повторите: " << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

for (int i = 0; i < z.length(); i++) {

if (int('a') <= int(z[i]) && int(z[i]) <= int('z') ||

int('A') <= int(z[i]) && int(z[i]) <= int('Z') ||

int('А') <= int(z[i]) && int(z[i]) <= int('п') ||

int('р') <= int(z[i]) && int(z[i]) <= int('я') ||

int(z[i]) == int('ё') ||

int(z[i]) == int('Ё') ||

int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length(); i++) {

if (int(z[0]) == int('\0'))

{

check = 0; break;

}

else if (int(z[i]) == 32 && int(z[i + 1]) == 32 || (int(z[z.length() - 1] == 32)) || int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

}

}

return z;

}

//проверка на ввод адреса

string check\_adress(string z)

{

int check = 0;

for (int i = 0; i < z.length(); i++) {

if (int('А') <= int(z[i]) && int(z[i]) <= int('п') ||

int('р') <= int(z[i]) && int(z[i]) <= int('я') ||

int(z[i]) == int('ё') || int(z[i]) == int('Ё') ||

int(z[i]) >= int('0') && int(z[i]) <= ('9') ||

int (z[i])==int('.')|| int(z[i])==int(',')||int (z[i])==int('-')||

int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length() + 1; i++) {

if (int(z[0]) == int('\0'))

{

check = 0; break;

}

else if (int(z[i]) == 32 && int(z[i + 1]) == 32 || (int(z[0] == 32)) || int(z[0]) == int('\0') ||

//проверка первого символа на недопустимые символы и последний символ обязательно номер (дома, например)

int(z[0]) == int('.') || int(z[0]) == int(',') || int(z[0]) == int('-') || int(z[z.length() - 1]) < int('0') || int(z[z.length() - 1]) > int('9'))

{

check = 0; break;

}

}

while ((cin.fail()) || check == 0)

{

cout << "Некорректный ввод!. Повторите: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

for (int i = 0; i < z.length(); i++) {

if (int('А') <= int(z[i]) && int(z[i]) <= int('п') ||

int('р') <= int(z[i]) && int(z[i]) <= int('я') ||

int(z[i]) == int('ё') || int(z[i]) == int('Ё') ||

int (z[i])>=int('0')&& int (z[i])<=('9')||

int(z[i]) == int('.') || int(z[i]) == int(',') || int(z[i]) == int('-') ||

int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length(); i++) {

if (int(z[0]) == int('\0'))

{

check = 0; break;

}

else if (int(z[i]) == 32 && int(z[i + 1]) == 32 || (int(z[0] == 32))||

int(z[0]) == int('.') || int(z[0]) == int(',') || int(z[0]) == int('-') || int(z[z.length() - 1]) < int('0') || int(z[z.length() - 1]) > int('9'))

check = 0;

}

}

return z;

}

//для проверки одинаковых регистр номеров

int find\_to\_registr(Hash\*& start\_ht, Hash\*& end\_ht, int k)

{

// если хэштаблица пустая

if (start\_ht->next == NULL)

{

return 0;

}

int count\_searching = 0; // для проверки на наличие совпадений

Hash\* help = start\_ht;

for (int i = 0; i < SIZE\_TABLE; i++)

{

// если ячейка пуста

if (help->patient == NULL)

{

help = help->next;

continue;

}

// если ячейка не пуста

if (help->patient != NULL)

{

string regist=help->patient->regist\_number;

string num = regist.erase(0, 3);

int reg = stoi(num);

if (reg==k)

{

count\_searching++;

}

}

help = help->next;

}

if (count\_searching == 0)

{

return 0;

}

else return 1;

}

//проверка на ввод участка у регистрационного номера

void checkM(string& regist)

{

while (true)

{

if ((int)regist[0] >= 48 && (int)regist[0] <= 57 && (int)regist[1] >= 48 && (int)regist[1] <= 57

&& regist.size() == 2 && regist != "00")

{

break;

}

else

{

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cout << "Введите номер участка (ММ - цифры) повторно; ";

getline(cin, regist);

}

}

}

//генерация регистрационного номера по порядку

string generate\_regist\_number(string key, int counter\_patient)

{

key += "-";

if (counter\_patient < 10)

{

key += "00000";

key += to\_string(counter\_patient);

}

else if (counter\_patient >= 10 && counter\_patient < 100)

{

key += "0000";

key += to\_string(counter\_patient);

}

else if (counter\_patient >= 100 && counter\_patient < 1000)

{

key += "000";

key += to\_string(counter\_patient);

}

else if (counter\_patient >= 1000 && counter\_patient < 10000)

{

key += "00";

key += to\_string(counter\_patient);

}

else if (counter\_patient >= 10000 && counter\_patient < 100000)

{

key += "0";

key += to\_string(counter\_patient);

}

else

{

key += to\_string(counter\_patient);

}

return key;

}

//проверка заполнена ли таблица

bool isfull(Hash\*& start\_ht)

{

Hash\* help = start\_ht;

for (int i = 0; i < SIZE\_TABLE; i++)

{

if (help->patient == NULL)

return 1;

help = help->next;

}

return 0;

}

// Добавление больного

void add\_patient(Hash\*& start\_ht, Hash\*& end\_ht, int& k)

{

Patient\* bolnoy = new Patient;

int hash1;

string key;

cout << "Введите номер участка (ММ-цифры); ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, key);

checkM(key);

if (isfull(start\_ht) == 0)

{

cout << "Table is full" << endl;

system("pause");

return;

}

bolnoy->regist\_number = generate\_regist\_number(key,k);

cout << "Введите ФИО больного: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, bolnoy->all\_name);

bolnoy->all\_name=check\_fio(bolnoy->all\_name);

cout << "Введите год рождения больного: ";

check\_year(bolnoy->year\_of\_birth);

cout << "Введите адрес проживания: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, bolnoy->address);

bolnoy->address = check\_adress(bolnoy->address);

cout << "Введите место работы (учебы): ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, bolnoy->place\_of\_work);

bolnoy->place\_of\_work=check\_work(bolnoy->place\_of\_work);

system("cls");

hash1 = hash\_position(bolnoy->regist\_number); // Вычисление позиции в таблице

// если таблица не создана

if (start\_ht->next == NULL)

{

create\_hash\_table(start\_ht, end\_ht); // создаём таблицу

Hash\* help = start\_ht;

// помещаем в таблицу больного

for (int i = 0; i <= hash1; i++)

{

if (help->position == hash1)

{

// помещаем в таблицу и выходим из функции

help->patient = bolnoy;

return;

}

help = help->next;

}

}

// создаём вспомогательную переменную для итераций по таблице

Hash\* help = start\_ht;

// ищем место в таблице и обрабатываем коллизии

for (int i = 0; i <= (hash1+1); i++)

{

// если хэш совпал и по этому адресу не хранится элемент

if (help->position == hash1 && help->patient == NULL)

{

help->patient = bolnoy;

if (help->metka == "del")

help->metka = "";

return;

}

// иначе если хэш совпал и по этому адресу хранится элемент

else if (help->position == hash1 && help->patient != NULL)

{

help->collision++; // добавляем коллизию

int hash2 = hash\_position2(bolnoy->regist\_number);

for (int j = 0;; j++)

{

hash1 = (hash1 + j \* hash2) % SIZE\_TABLE; // двойное хеширование

help = start\_ht; // создаём переменную для поисковых итераций

for (int n = 0; n <= (hash1+1); n++)

{

// если хэш совпал и по этому адресу не хранится элемент

if (help->position == hash1 && help->patient == NULL)

{

help->patient = bolnoy;

if (help->metka == "del")

help->metka = "";

return;

}

// если хэш совпал и по этому адрусу хранится элемент

else if (help->position == hash1 && help->patient != NULL)

{

help->collision++;

break;

}

help = help->next;

}

}

return;

}

help = help->next;

}

}

// Вывод больных

void show\_patient(Hash\*& start\_ht, Hash\*& end\_ht)

{

Hash\* help = start\_ht;

int count = 0; // для проверки наличия больных в списке

for (int i = 0; i < SIZE\_TABLE; i++)

{

if (start\_ht->next == NULL)

{

cout << "Больных нет\n";

cout << "================================" << endl;

system("pause");

return;

}

if (help->patient != NULL)

{

count++;

cout << "ФИО больного: " << help->patient->all\_name << endl;

cout << "Регистрационный номер: " << help->patient->regist\_number << endl;

cout << "Год рождения: " << help->patient->year\_of\_birth << endl;

cout << "Адрес: " << help->patient->address << endl;

cout << "Место работы (учебы): " << help->patient->place\_of\_work << endl;

cout << "position number: " << help->position << endl;

cout << "================================" << endl;

}

help = help->next;

}

if (!count)

{

cout << "Больных нет\n";

cout << "================================" << endl;

}

system("pause");

}

//вывод направлений с этим регистрационным номером

void search\_registr\_in\_data(Data\*& head, Data\*&tail, string registr)

{

Data\* help = head;

if (help == 0)

{

return;

}

else

{

do

{

if (help->ticket->registr == registr)

{

cout << "======================================" << endl;

cout << "Направление к " << help->ticket->fio\_doctor << endl;

}

help = help->next;

} while (help != head);

}

}

//Поиск больного по регистрационному номеру

void search\_to\_registr(Hash\*& start\_ht, Hash\*& end\_ht, Data\*&head,Data\*&tail)

{

// если хэштаблица пустая

if (start\_ht->next == NULL)

{

cout << "Данных нет\n";

cout << "======================================" << endl;

system("pause");

return;

}

int count\_searching = 0; // для проверки на наличие совпадений

int metka = 0;

Hash\* help = start\_ht;

string registr\_in\_function;

cout << "Введите регистрационный номер «MM-NNNNNN»,где MM – номер участка(цифры); NNNNNN – порядковый номер(цифры): ";

checkRegist(registr\_in\_function);

cout << "================================" << endl;

int hash1 = hash\_position(registr\_in\_function);

int hash2 = hash\_position2(registr\_in\_function);

for (int i = 0; i <= hash1; i++)

{

if (help->position == hash1 && help->patient != NULL && help->patient->regist\_number == registr\_in\_function)

{

cout << "ФИО больного: " << help->patient->all\_name << endl;

cout << "Регистрационный номер: " << help->patient->regist\_number << endl;

cout << "Год рождения: " << help->patient->year\_of\_birth << endl;

cout << "Адрес: " << help->patient->address << endl;

cout << "Место работы (учебы): " << help->patient->place\_of\_work << endl;

search\_registr\_in\_data(head, tail, help->patient->regist\_number);

cout << "================================" << endl;

count\_searching++;

break;

}

else if (help->position == hash1 && (help->patient == NULL || help->patient != NULL && help->patient->regist\_number != registr\_in\_function))

{

for (int j = 0;j<SIZE\_TABLE\*11; j++)

{

hash1 = (hash1 + j \* hash2) % SIZE\_TABLE;

help = start\_ht;

for (int n = 0; n <= hash1; n++)

{

if (help->position == hash1 && help->patient != NULL && help->patient->regist\_number == registr\_in\_function)

{

cout << "ФИО больного: " << help->patient->all\_name << endl;

cout << "Регистрационный номер: " << help->patient->regist\_number << endl;

cout << "Год рождения: " << help->patient->year\_of\_birth << endl;

cout << "Адрес: " << help->patient->address << endl;

cout << "Место работы (учебы): " << help->patient->place\_of\_work << endl;

search\_registr\_in\_data(head, tail, help->patient->regist\_number);

cout << "================================" << endl;

count\_searching++;

}

else if (help->position == hash1 && help->metka == "del")

{

metka++;

}

help = help->next;

}

if (count\_searching != 0||metka!=0)

break;

}

break;

}

help=help->next;

}

if (count\_searching == 0)

{

cout << "Совпадений нет\n";

cout << "======================================" << endl;

}

system("pause");

}

// Поиск больного по ФИО

void search\_to\_FUO\_patient(Hash\*& start\_ht, Hash\*& end\_ht)

{

if (start\_ht->next == NULL)

{

cout << "Больных нет\n";

cout << "======================================" << endl;

system("pause");

return;

}

string name; // храним тут ФИО

cout << "Введите ФИО больного: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, name);

name=check\_fio(name);

Hash\* help = start\_ht;

int count = 0; // для проверки наличия больного в списке

for (int i = 0; i < SIZE\_TABLE; i++)

{

if (help->patient != NULL)

{

// если ФИО совпадает

if (help->patient->all\_name == name)

{

cout << "ФИО больного: " << help->patient->regist\_number << endl;

cout << "Регистрационный номер: " << help->patient->all\_name << endl;

cout << "================================" << endl;

count++;

}

}

help = help->next;

}

if (!count)

{

cout << "Больного с таким ФИО нету\n";

cout << "================================" << endl;

}

system("pause");

}

//поиск регистрационного номера в списке направлений для проверки записи к врачу

bool check\_registr\_in\_data(Data\*& head, Data\*& tail, string registr)

{

Data\* help = head;

if (help == 0)

{

return 0;

}

else

{

do

{

if (help->ticket->registr == registr)

return 1;

help = help->next;

} while (help != head);

}

return 0;

}

// Удаление больного

void del\_patient(Hash\*& start\_ht, Hash\*& end\_ht,Data\*& head, Data\*& tail)

{

if (start\_ht->next == NULL)

{

cout << "======================================" << endl;

cout << "Больных нет\n";

cout << "======================================" << endl;

system("pause");

return;

}

string registr; // тут храним регистрационнный номер

cout << "Введите регистрационный номер «MM-NNNNNN»,где MM – номер участка(цифры); NNNNNN – порядковый номер(цифры): ";

checkRegist(registr); // вводим регистрационный номер корректно

if (check\_registr\_in\_data(head, tail, registr))

{

cout << "===========================================================" << endl;

cout << "Удаление больного невозможно, у него есть направление к врачу\n";

cout << "===========================================================" << endl;

system("pause");

return;

}

Hash\* help = start\_ht;

int count = 0; // для проверки наличия больного в списке

int hash1 = hash\_position(registr);

int hash2 = hash\_position2(registr);

for (int i = 0; i <= hash1; i++)

{

if (help->position == hash1 && help->patient != NULL && help->patient->regist\_number == registr)

{

count++;

delete help->patient;

help->patient = nullptr;

help->patient = NULL;

help->collision = 0;

help->metka = "del";

cout << "======================================" << endl;

cout << "Больной удалён\n";

cout << "======================================" << endl;

system("pause");

return;

}

else if (help->position == hash1 && (help->patient == NULL || help->patient != NULL && help->patient->regist\_number != registr))

{

for (int j = 0; j < SIZE\_TABLE \* 11; j++)

{

hash1 = (hash1 + j \* hash2) % SIZE\_TABLE;

help = start\_ht;

for (int n = 0; n <= hash1; n++)

{

if (help->position == hash1 && help->patient != NULL && help->patient->regist\_number == registr)

{

count++;

delete help->patient;

help->patient = nullptr;

help->patient = NULL;

help->collision = 0;

help->metka = "del";

cout << "======================================" << endl;

cout << "Больной удалён\n";

cout << "======================================" << endl;

system("pause");

return;

}

help = help->next;

}

if (count != 0)

break;

}

break;

}

help = help->next;

}

if (count==0)

{

cout << "======================================" << endl;

cout << "Больного с таким регистрационным номером нет\n";

cout << "======================================" << endl;

}

system("pause");

}

// Удаление всех больных

void delete\_all\_patients(Hash\*& start\_ht, Hash\*& end\_ht, Data\*&head)

{

if (head != NULL)

{

cout << "Нельзя удалить всех больных, у них есть направления к врачу\n";

cout << "======================================================" << endl;

system("pause");

return;

}

if (start\_ht->next == NULL)

{

cout << "Больных нет\n";

cout << "======================================" << endl;

system("pause");

return;

}

Hash\* help = start\_ht;

for (int i = 0; i < SIZE\_TABLE; i++)

{

// если в ячейке пусто

if (help->patient == NULL)

{

help = help->next;

continue;

}

// если в ячейке не пусто

if (help->patient != NULL)

{

delete help->patient;

help->patient = nullptr;

}

help = help->next;

}

cout << "Больные удалены\n";

cout << "======================================" << endl;

system("pause");

}

// Запись данных из файла в список

void ReadHelp(Patient\* bolnoy, Hash\*& start\_ht, Hash\*& end\_ht)

{

if (isfull(start\_ht) == 0)

{

cout << "Table is full" << endl;

system("pause");

return;

}

int hash1 = hash\_position(bolnoy->regist\_number);

if (start\_ht->next == NULL)

{

create\_hash\_table(start\_ht, end\_ht); // создаём таблицу

Hash\* help = start\_ht;

// помещаем в таблицу больного

for (int i = 0; i <= hash1; i++)

{

if (help->position == hash1)

{

// помещаем в таблицу и выходим из функции

help->patient = bolnoy;

return;

}

help = help->next;

}

}

// создаём вспомогательную переменную для итераций по таблице

Hash\* help = start\_ht;

// ищем место в таблице и обрабатываем коллизии

for (int i = 0; i <= (hash1 + 1); i++)

{

// если хэш совпал и по этому адресу не хранится элемент

if (help->position == hash1 && help->patient == NULL)

{

help->patient = bolnoy;

if (help->metka == "del")

help->metka = "";

return;

}

// иначе если хэш совпал и по этому адресу хранится элемент

else if (help->position == hash1 && help->patient != NULL)

{

help->collision++; // добавляем коллизию

int hash2 = hash\_position2(bolnoy->regist\_number);

for (int j = 0;; j++)

{

hash1 = (hash1 + j \* hash2) % SIZE\_TABLE; // двойное хеширование

help = start\_ht; // создаём переменную для поисковых итераций

for (int n = 0; n <= (hash1 + 1); n++)

{

// если хэш совпал и по этому адресу не хранится элемент

if (help->position == hash1 && help->patient == NULL)

{

help->patient = bolnoy;

if (help->metka == "del")

help->metka = "";

return;

}

// если хэш совпал и по этому адрусу хранится элемент

else if (help->position == hash1 && help->patient != NULL)

{

help->collision++;

break;

}

help = help->next;

}

}

return;

}

help = help->next;

}

}

// Чтение из файла

void Read(Hash\*& Start, Hash\*& End, int &n)

{

Hash\* st = Start;

ifstream read\_file("patient.txt");

string str, fio, work, year, registr, adress;

char txt[10050];

int x;

if (!read\_file.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else {

do {

Patient\* patient = new Patient;

read\_file.getline(txt, 1050);

for (int i = 0; i < 1050; i++)

{

str += txt[i];

}

x = 0;

while (str[x] != '|')

{

registr += str[x];

x++;

patient->regist\_number = registr;

}

x++;

while (str[x] != '|')

{

fio += str[x];

x++;

patient->all\_name=fio;

}

x++;

while (str[x] != '|')

{

year += str[x];

x++;

patient->year\_of\_birth = atoi(year.c\_str());

}

x++;

while (str[x] != '|')

{

adress += str[x];

x++;

patient->address = adress;

}

x++;

while (str[x] != ';')

{

work += str[x];

x++;

patient->place\_of\_work = work;

}

x++;

ReadHelp(patient,Start, End);

str.clear();

fio.clear();

registr.clear();

year.clear();

adress.clear();

work.clear();

n++;

} while (!read\_file.eof());

read\_file.close();

}

}

// Запись в файл

void Record(Hash\*& Start)

{

cout << "Это изменит существующую базу данных. Хотите продолжить?\n1.Да\n2.Нет" << endl;

int choce;

cin >> choce;

switch (choce) {

case 1: {

Hash\* st = Start;

ofstream record\_file;

record\_file.open("patient.txt", ios\_base::trunc);

if (!record\_file.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else {

for (int i=0; i < SIZE\_TABLE; i++)

{

if (st->patient == NULL)

{

st = st->next; continue;

}

else

{

record\_file << st->patient->regist\_number << "|" << st->patient->all\_name << "|" << st->patient->year\_of\_birth

<< "|" << st->patient->address << "|" << st->patient->place\_of\_work << ";";

Hash\* help = st->next;

for (int j = st->position; j < SIZE\_TABLE; j++)

{

if (help->patient != NULL)

{

record\_file << endl; break;

}

help = help->next;

}

}

st = st->next;

}

cout << endl << "Таблица сохранена" << endl << endl;

break;

}

record\_file.close();

}

case 2:

{

break;

}

}

}

//для проверки высоты

int height(Three\*& element)

{

if (element == NULL)

{

return -1;

}

else

{

return element->height;

}

}

//малый левый поворот

Three\* single\_left\_rotate(Three\*& element)

{

Three\* help = element->right;

element->right = help->left;

help->left = element;

element->height = max(height(element->left), height(element->right)) + 1;

help->height = max(height(element->left), element->height) + 1;

return help;

}

//малый правый поворот

Three\* single\_right\_rotate(Three\*& element)

{

Three\* help = element->left;

element->left = help->right;

help->right = element;

element->height = max(height(element->left), height(element->right)) + 1;

help->height = max(height(help->left), element->height) + 1;

return help;

}

//большой левый поворот

Three\* big\_left\_rotate(Three\*& element)

{

element->right = single\_right\_rotate(element->right);

return single\_left\_rotate(element);

}

//большой правый поворот

Three\* big\_right\_rotate(Three\*& element)

{

element->left = single\_left\_rotate(element->left);

return single\_right\_rotate(element);

}

//проверка введенных ФИО

string check\_fio\_doctor()

{

string z;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

int check = 0;

int kolvo = 0;

for (int i = 0; i < z.length(); i++) {

if (int('А') <= int(z[i]) && int(z[i]) <= int('п') || int('р') <= int(z[i]) && int(z[i]) <= int('я') ||int(z[i]) == int('ё') ||int(z[i]) == int('Ё') ||int(z[i])==int('.')|| int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length() + 1; i++) {

if (int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

if (int(z[z.length() - 1] != int('.')) && int(z[z.length() - 3] != int('.')))

{

check = 0; break;

}

if (int(z[i]) == 32 && !(int('А') <= int(z[i + 1]) && int(z[i + 1]) <= int('Я') ||

int(z[i]) == int('.') && !(int('А') <= int(z[i + 1]) && int(z[i + 1]) <= int('Я') || int(z[i]) == int('Ё')) || (!(int('А') <= int(z[0]) && int(z[0]) <= int('Я') || int(z[i]) == int('Ё')))))

check = 0;

}

while ((cin.fail()) || check == 0 || z.length()==26)

{

kolvo = 0;

cout << "Некорректный ввод!. Повторите: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

for (int i = 0; i < z.length(); i++) {

if (int('А') <= int(z[i]) && int(z[i]) <= int('п') ||int('р') <= int(z[i]) && int(z[i]) <= int('я') ||int(z[i]) == int('ё') ||int(z[i]) == int('Ё') ||int(z[i]) == int ('.')|| int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length(); i++) {

if (int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

if (int(z[z.length() - 1] != int('.')) && int(z[z.length() - 3] != int('.')))

{

check = 0; break;

}

if (int(z[i]) == 32 && !(int('А') <= int(z[i + 1]) && int(z[i + 1]) <= int('Я') ||

int(z[i]) == int('.') && !(int('А') <= int(z[i + 1]) && int(z[i + 1]) <= int('Я') || int(z[i]) == int('Ё')) || (!(int('А') <= int(z[0]) && int(z[0]) <= int('Я') || int(z[i]) == int('Ё')))))

{

check = 0; break;

}

}

}

return z;

}

//проверка на ввод должности

string check\_position()

{

string z;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

int check = 0;

for (int i = 0; i < z.length(); i++) {

if (int('А') <= int(z[i]) && int(z[i]) <= int('п') || int('р') <= int(z[i]) && int(z[i]) <= int('я') || int(z[i]) == int('ё') || int(z[i]) == int('Ё') || int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length() + 1; i++) {

if (int(z[0]) == int('\0'))

{

check = 0; break;

}

else if (int(z[i]) == 32 && int(z[i + 1]) == 32 || (int(z[z.length() - 1] == 32)) || int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

}

while ((cin.fail()) || check == 0)

{

cout << "Некорректный ввод!. Повторите: " << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

for (int i = 0; i < z.length(); i++) {

if (int('А') <= int(z[i]) && int(z[i]) <= int('п') ||int('р') <= int(z[i]) && int(z[i]) <= int('я') ||int(z[i]) == int('ё') ||int(z[i]) == int('Ё') || int(z[i]) == 32)

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length(); i++) {

if (int(z[0]) == int('\0'))

{

check = 0; break;

}

else if (int(z[i]) == 32 && int(z[i + 1]) == 32 || (int(z[z.length() - 1] == 32)) || int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

}

}

return z;

}

//проверка на ввод кабинета

int check\_cabinet()

{

int m;

cin >> m;

while ((cin.fail()) || (cin.get() != '\n') || m<=0)

{

cout << "Введите повторно: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> m;

}

return m;

}

//проверка на промежуток времени

string check\_time()

{

string z;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

int check = 0, time10, time20;

string time1, time2;

for (int i = 0; i < z.length(); i++) {

if (int('0') <= int(z[i]) && int(z[i]) <= int('9') || int(z[i]) == int(':')||int(z[i]) == int('-'))

check = 1;

else {

check = 0; break;

}

if (i < 2)

{

time1 += z[i];

}

if (i > 5 && i < 8)

time2 += z[i];

}

for (int i = 0; i < z.length() + 1; i++) {

if (int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

if (int (z[2])!=int(':') || int(z[8]) != int(':') || int(z[5]) != int('-') || int('3') <= int(z[0]) && int(z[0]) <= int('9')

|| int('2') == int(z[0]) && int(z[1]) >= int('1')||int('3') <= int(z[6]) && int(z[6]) <= int('9')

|| int('2') == int(z[6]) && int(z[7]) >= int('1'))

{

check = 0; break;

}

}

time10 = stoi(time1);

time20 = stoi(time2);

if (time10 > time20)

check = 0;

while ((cin.fail()) || check == 0)

{

cout << "Некорректный ввод!. Повторите: " << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

time1.clear();

time2.clear();

for (int i = 0; i < z.length(); i++) {

if (int('0') <= int(z[i]) && int(z[i]) <= int('9') || int(z[i]) == int(':') || int(z[i]) == int('-'))

check = 1;

else {

check = 0; break;

}

if (i < 2)

{

time1 += z[i];

}

if (i > 5 && i < 8)

time2 += z[i];

}

for (int i = 0; i < z.length() + 1; i++) {

if (int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

if (int(z[2]) != int(':') || int(z[8]) != int(':') || int(z[5]) != int('-'))

{

check = 0; break;

}

if (int(z[2]) != int(':') || int(z[8]) != int(':') || int(z[5]) != int('-') || int('3') <= int(z[0]) && int(z[0]) <= int('9')

|| int('2') == int(z[0]) && int(z[1]) >= int('1') || int('3') <= int(z[6]) && int(z[6]) <= int('9')

|| int('2') == int(z[6]) && int(z[7]) >= int('1'))

{

check = 0; break;

}

}

time10 = stoi(time1);

time20 = stoi(time2);

if (time10 > time20)

check = 0;

}

return z;

}

// Добавление элементов в дерево

void add\_doctor(Three\*& element, string value)

{

if (element == NULL)

{

Doctor\* r = new Doctor;

r->fio\_doctor = value;

cout << "Введите должность: ";

r->position = check\_position();

cout << "Введите номер кабинета: ";

r->number\_cabinet= check\_cabinet();

cout << "Введите график работы:";

r->time = check\_time();

element = new Three;

element->height = 0;

element->doctor = r;

}

else

{

if (value < element->doctor->fio\_doctor)

{

add\_doctor(element->left, value);

//Если произошла разбалансировка

if (height(element->left) - height(element->right) == 2)

{

if (value < element->left->doctor->fio\_doctor)

{

element = single\_right\_rotate(element);

}

else

{

element = big\_right\_rotate(element);

}

}

}

else if (value > element->doctor->fio\_doctor)

{

add\_doctor(element->right, value);

if (height(element->right) - height(element->left) == 2)

{

if (value > element->right->doctor->fio\_doctor)

{

element = single\_left\_rotate(element);

}

else

{

element = big\_left\_rotate(element);

}

}

}

}

element->height = max(height(element->left), height(element->right)) + 1;

}

// Вывод врачей

void show\_doctors(const Three\* element)

{

if (element == NULL)

{

return;

}

show\_doctors(element->left); // Обошли левое поддерево

cout << "ФИО врача: " << element->doctor->fio\_doctor << endl;

cout << "Должность: " << element->doctor->position << endl;

cout << "Номер кабинета: " << element->doctor->number\_cabinet << endl;

cout << "График работы: " << element->doctor->time << endl;

cout << "====================================\n";

show\_doctors(element->right); // Обошли правое поддерево

}

// Удаление минимального значения

Doctor\* deletemin(Three\*& element)

{

Doctor\* a;

if (element->left == NULL)

{

a = element->doctor;

element = element->right;

return a;

}

else

{

a = deletemin(element->left);

return a;

}

}

//есть ли выданные направления к этому врачу

bool check\_fio\_in\_data(Data\*& head, Data\*& tail, string fio)

{

Data\* help = head;

if (help == 0)

{

return 0;

}

else

{

do

{

if (help->ticket->fio\_doctor == fio)

return 1;

help = help->next;

} while (help != head);

}

return 0;

}

// Удаление элемента из дерева

void del\_doctor(Three\*& element, string value, Data\*& head, Data\*& tail)

{

if (check\_fio\_in\_data(head, tail, value))

{

cout << "Удаление врача невозможно, к нему есть направление\n";

cout << "===========================================================" << endl;

return;

}

Three\* d;

if (element == NULL)

{

cout << "Такого врача нет" << endl;

cout << "======================================" << endl;

}

else if (value < element->doctor->fio\_doctor)

{

del\_doctor(element->left, value, head, tail);

}

else if (value > element->doctor->fio\_doctor)

{

del\_doctor(element->right, value, head, tail);

}

else if ((element->left == NULL) && (element->right == NULL))

{

d = element;

free(element->doctor);

element->doctor = nullptr;

free(d);

d = nullptr;

element = nullptr;

cout << "Врач удален" << endl;

cout << "======================================" << endl;

}

else if (element->left == NULL)

{

Three\* help = element->right;

d = element;

delete element->doctor;

element->doctor = nullptr;

delete d;

d = nullptr;

element = help;

cout << "Врач удален" << endl;

cout << "======================================" << endl;

}

else if (element->right == NULL)

{

d = element;

element = element->left;

free(element->doctor);

element->doctor = nullptr;

free(d);

d = nullptr;

cout << "Врач удален" << endl;

cout << "======================================" << endl;

}

else

{

free(element->doctor);

element->doctor = nullptr;

element->doctor = deletemin(element->right);

cout << "Врач удален" << endl;

cout << "======================================" << endl;

}

}

//поиск выданных направлений к этому врачу

void search\_fio\_in\_data(Data\*& head, Data\*& tail, string fio)

{

Data\* help = head;

if (help == 0)

{

return;

}

else

{

do

{

if (help->ticket->fio\_doctor == fio)

{

cout << "Больной: " << help->ticket->registr << endl;

cout << "======================================" << endl;

}

help = help->next;

} while (help != head);

}

}

// Поиск врача по ФИО

void search\_doctor(const Three\* element, string fio, Data\*& head, Data\*& tail)

{

if (element == NULL)

{

return;

}

search\_doctor(element->left, fio, head, tail); // Обошли левое поддерево

if (element->doctor->fio\_doctor == fio)

{

cout << "ФИО врача: " << element->doctor->fio\_doctor << endl;

cout << "Должность: " << element->doctor->position << endl;

cout << "Номер кабинета: " << element->doctor->number\_cabinet << endl;

cout << "График работы: " << element->doctor->time << endl;

cout << "====================================\n";

search\_fio\_in\_data(head, tail, fio);

}

search\_doctor(element->right,fio, head, tail); // Обошли правое поддерево

}

// Удаление всех врачей

void delete\_all\_doctors(Three\*& element, Data\*& head)

{

if (head != NULL)

{

cout << "Удаление врачей невозможно, к ним есть направления\n";

cout << "===========================================================" << endl;

return;

}

if (element == NULL)

{

return;

}

delete\_all\_doctors(element->left, head); // Обошли левое поддерево

delete\_all\_doctors(element->right, head); // Обошли правое поддерево

delete element->doctor;

element->doctor = nullptr;

delete element;

element = nullptr;

}

//Алгоритм поиска

bool straigth\_search(string str, string substr)

{

int length\_sub = substr.length(); // длина подстроки

int length\_str = str.length(); // длина строки

int check = 0;

if (length\_str == 0 || length\_sub == 0 || length\_sub == 1 || length\_str == 1)

{

return false;

}

else

{

for (int i = 0; i <= length\_str - length\_sub; i++)

{

for (int j = 0; j < length\_sub; j++)

if (int(str[j + i]) != int(substr[j]))

break;

else if (j == length\_sub - 1)

return true;

}

}

return false;

}

// Поиск по должности

void position\_search(const Three\* element, string pos)

{

if (element == NULL)

{

return;

}

position\_search(element->left, pos); // Обошли левое поддерево

if (straigth\_search(element->doctor->position, pos))

{

cout << "ФИО врача: " << element->doctor->fio\_doctor << endl;

cout << "Должность: " << element->doctor->position << endl;

cout << "Номер кабинета: " << element->doctor->number\_cabinet << endl;

cout << "Время работы: " << element->doctor->time << endl;

cout << "====================================\n";

}

position\_search(element->right, pos); // Обошли правое поддерево

}

//Запись данных из файла

void ReadHelp1(Doctor\* doctor, Three\*& element, string value)

{

if (element == NULL)

{

element = new Three;

element->height = 0;

element->doctor = doctor;

}

else

{

if (value < element->doctor->fio\_doctor)

{

ReadHelp1(doctor, element->left, value);

//Если произошла разбалансировка

if (height(element->left) - height(element->right) == 2)

{

if (value < element->left->doctor->fio\_doctor)

{

element = single\_right\_rotate(element);

}

else

{

element = big\_right\_rotate(element);

}

}

}

else if (value > element->doctor->fio\_doctor)

{

ReadHelp1(doctor, element->right, value);

if (height(element->right) - height(element->left) == 2)

{

if (value > element->right->doctor->fio\_doctor)

{

element = single\_left\_rotate(element);

}

else

{

element = big\_left\_rotate(element);

}

}

}

}

element->height = max(height(element->left), height(element->right)) + 1;

}

// Чтение из файла

void Read1(Three\*& Start)

{

ifstream read\_file("doctors.txt");

string str, fio, cabinet, time, pos;

char txt[1050];

int x;

if (!read\_file.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else {

do {

Doctor\* doctor = new Doctor;

if (!read\_file.getline(txt, 1050))

break;

for (int i = 0; i < 1050; i++)

{

str += txt[i];

}

x = 0;

while (str[x] != '|')

{

fio += str[x];

x++;

doctor->fio\_doctor = fio;

}

x++;

while (str[x] != '|')

{

pos += str[x];

x++;

doctor->position = pos;

}

x++;

while (str[x] != '|')

{

cabinet += str[x];

x++;

doctor->number\_cabinet = atoi(cabinet.c\_str());

}

x++;

while (str[x] != ';')

{

time += str[x];

x++;

doctor->time = time;

}

x++;

ReadHelp1(doctor, Start, doctor->fio\_doctor);

str.clear();

fio.clear();

cabinet.clear();

pos.clear();

time.clear();

} while (!read\_file.eof());

read\_file.close();

}

}

// Запись в файл

void Record1(Three\* element)

{

ofstream record\_file;

record\_file.open("doctors.txt", ios::in | ios::app);

if (!record\_file.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else {

if (element == NULL)

{

return;

}

Record1(element->left); // Обошли левое поддерево

record\_file << element->doctor->fio\_doctor << "|"<< element->doctor->position << "|" << element->doctor->number\_cabinet <<"|" << element->doctor->time << ";";

Record1(element->right); // Обошли правое поддерево

}

record\_file << endl;

record\_file.close();

}

//существует ли такой регистрационный номер в базе пациентов

bool search\_data(Hash\*& start\_ht, Hash\*& end\_ht, string registr)

{

Hash\* help = start\_ht;

cout << "================================" << endl;

for (int i = 0; i < SIZE\_TABLE; i++)

{

// если хэш-таблица пустая

if (start\_ht->next == NULL)

{

return false;

}

// если в ячейке ничего нету, переходим к следующей

if (help->patient == NULL)

{

help = help->next;

continue;

}

// если в ячейке что-то есть

if (help->patient != NULL)

{

if (help->patient->regist\_number == registr)

{

return true;

}

}

help = help->next;

}

return false;

}

//существует ли такой доктор

Doctor\* search\_doc(const Three\* element, string fio)

{

if (element == NULL)

{

return 0;

}

search\_doc(element->left, fio); // Обошли левое поддерево

if (element->doctor->fio\_doctor == fio)

{

return element->doctor;

}

search\_doc(element->right, fio); // Обошли правое поддерево

}

//проверка на запись в одно и то же время к одному и тому же врачу

bool searchtime(Ticket\*& data, Data\*& head,string time)

{

Data\* help = head;

if (head == NULL)

return 1;

else {

do

{

if (data->registr == help->ticket->registr && data->date == help->ticket->date && time == help->ticket->time && data->fio\_doctor == help->ticket->fio\_doctor)

{

return 0;

break;

}

if (data->date == help->ticket->date && time == help->ticket->time && data->fio\_doctor == help->ticket->fio\_doctor)

{

return 0;

break;

}

if (data->date == help->ticket->date && time == help->ticket->time && data->registr == help->ticket->registr)

{

return 0;

break;

}

help = help->next;

} while (help != head);

return 1;

}

}

//сортировка слиянием

Data\* merge(Data\* a, Data\* f, int m, int rem)

{

if (!a) return f;

if (!f) return a;

Data\* c = 0;

if (a->ticket->fio\_doctor < f->ticket->fio\_doctor) {

c = a;

c->next = merge(a->next, f, m, rem);

}

else {

c = f;

c->next = merge(a, f->next, m, rem);

}

return c;

}

Data\* sort(Data\* head, Data\* tail, int l)

{

if (head == 0 || head->next == 0)

return head;

int m = (l) / 2;

int rem = l - m;

int k = 0;

Data\* help = head->prev;

Data\* a = head, \* f = head->next;

while ((f != 0) && (f->next != 0))

{

head = head->next;

f = f->next->next;

}

f = head->next;

head->next = NULL;

return merge(sort(a, help, m), sort(f, tail, rem), m, rem);

}

//проверка на ввод даты

bool Date(string a)//проверка на корректность ввода даты выдачи

{

bool bolt = false;

string day = " ", month = " " , year = " ";

int day1, month1, year1;

if (a.length() != 10)

bolt = true;//длина

else

{

if (a[2] != 46 || a[5] != 46)

bolt = true;//точки

else

for (int i = 0; i < 10; i++)

{

if (i != 2 && i != 5)

if (a[i] < 48 || a[i]>57) {

bolt = true;

return bolt;

}//числа, неотрицательность

}

day.append (a,0,2);

month.append(a,3,2);

year.append(a,6,4);

day1 = stoi(day);

month1 = stoi(month);

year1 = stoi(year);

if ((month1 == 1 || month1 == 3 || month1 == 5 || month1 == 7 || month1 == 8 || month1 == 10 || month1 == 12) && day1 > 31)

bolt = true;

else if (month1 == 2 && day1 > 29)

bolt = true;//февраль

else if ((day1 > 30) && (month1 == 4 || month1 == 6 || month1 == 9 || month1 == 11))

bolt = true;

else if (year1 < 2022 || month1 < 5 && year1 == 2022 || year1 > 2022)

bolt = true;

}

return bolt;

}

//проверка на ввод времени

string time1()

{

string z;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

int check = 0, time10, time20;

string time1, time2;

for (int i = 0; i < z.length(); i++) {

if (int('0') <= int(z[i]) && int(z[i]) <= int('9') || int(z[i]) == int(':'))

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length() + 1; i++) {

if (int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

if (int(z[2]) != int(':') || int('3') <= int(z[0]) && int(z[0]) <= int('9')

|| int('2') == int(z[0]) && int(z[1]) >= int('1')|| int('0') == int(z[0]) && int(z[1]) <= int('9'))

{

check = 0; break;

}

}

while ((cin.fail()) || check == 0)

{

cout << "Некорректный ввод!. Повторите: " << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, z);

time1.clear();

time2.clear();

for (int i = 0; i < z.length(); i++) {

if (int('0') <= int(z[i]) && int(z[i]) <= int('9') || int(z[i]) == int(':'))

check = 1;

else {

check = 0; break;

}

}

for (int i = 0; i < z.length() + 1; i++) {

if (int(z[0]) == int('\0') || int(z[0]) == int(' '))

{

check = 0; break;

}

if (int(z[2]) != int(':'))

{

check = 0; break;

}

if (int(z[2]) != int(':') || int('3') <= int(z[0]) && int(z[0]) <= int('9')

|| int('2') == int(z[0]) && int(z[1]) >= int('1')||int('0') == int(z[0]) && int(z[1]) <= int('9'))

{

check = 0; break;

}

}

}

return z;

}

//добавление направления в список

void Insert(Data\*& head, Data\*& tail, Hash\*& start\_ht, Hash\*& end\_ht, Three\*& element)

{

Ticket\* data2 = new Ticket;

string registr;

cout << "Введите регистрационный номер\n";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, registr);

data2->registr = registr;

if (search\_data(start\_ht, end\_ht, registr))

{

s++;

cout << "Введите фио доктора\n";

string fio;

fio = check\_fio\_doctor();

while (!search\_doc(element, fio))

{

cout << "Сведения о враче отсутствуют." << endl;

cout << "Введите фио доктора повторно\n";

fio = check\_fio\_doctor();

}

data2->fio\_doctor = fio;

string date1;

cout << "Введите дату выдачи номерка(пример: 03.06.2022): ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, date1);

while (Date(date1))

{

cin.sync();

if (Date(date1))

{

cout << "Некорректный ввод, введите дату приема: ";

getline(cin, date1);

}

}

data2->date = date1;

cout << "Введите время приема\n";

string time;

time = time1();

Data\* st = head;

while (!searchtime(data2, st,time))

{

cout << "Запись на это время занята. Введите другое время" << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, time);

}

data2->time = time;

Data\* help = new Data;

help->ticket = data2;

if (head == NULL)

{

head = help; // установить указатель первого элемента на новый элемент

tail = help; // установить указатель текущего элемента на новый элемент

tail->next = head; // установить указатель следующего элемента на новый элемент

tail->prev = help; // установить указатель предыдущего элемента на новый элемент

head->prev = tail; // перенаправляем предыдущий добавляемого на текущий

}

else

{

help->next = tail->next; // перенаправляем указатель следующего элемента в добавляемом

help->next->prev = help; // перенаправляем указатель следующего элемента на добавляемый

tail->next = help; // перенаправляем следующий указатель на добавляемый

help->prev = tail; // перенаправляем предыдущий добавляемого на текущий

tail = help;

if (s > 1)

{

Data\* st = head;

st->prev = NULL;

tail->next = NULL;

head = sort(st, tail, s);

Data\* help = head;

while (help->next != 0)

{

tail = help->next;

tail->prev = help;

help = help->next;

}

head->prev = tail;

tail->next = head;

}

}

}

else

{

cout << "Данного регистрационного номера нет в базе\n";

cout << "======================================================" << endl;

return;

}

}

//удаление направления

void Delete(Data\*& head, Data\*& tail, Hash\*& start\_ht, Hash\*& end\_ht, Three\*& element)

{

Ticket\* data2 = new Ticket;

string registr;

cout << "Введите регистрационный номер\n";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, registr);

data2->registr = registr;

if (search\_data(start\_ht, end\_ht, registr))

{

cout << "Введите фио доктора\n";

string fio;

fio = check\_fio\_doctor();

while (!search\_doc(element, fio))

{

cout << "Сведения о враче отсутствуют." << endl;

cout << "Введите фио доктора повторно\n";

fio = check\_fio\_doctor();

}

data2->fio\_doctor = fio;

cout << "Введите дату приема\n";

string date;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, date);

data2->date = date;

cout << "Введите время приема\n";

string time;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, time);

Data\* help = head;

if (searchtime(data2, help, time))

{

cout << "Записи нет в базе\n";

cout << "======================================================" << endl;

return;

}

else

{

s--;

data2->time = time;

Data\* help = head;

// если всего 1 элемент

if (head == tail)

{

delete head;

head = nullptr;

tail = nullptr;

cout << "Направление возвращено\n";

cout << "======================================================" << endl;

return;

}

// если удаляем элемент откуда-то из центра

else

{

// бежим, пока не встретим следующим элементом удаляемый

do

{

if (data2->registr == help->ticket->registr && data2->date == help->ticket->date && time == help->ticket->time && data2->fio\_doctor == help->ticket->fio\_doctor)

{

help->prev->next = help->next; // переставляем указатель

help->next->prev = help->prev; // переставляем указатель

delete help;

help = nullptr;

break;

}

help = help->next;

} while (help != head);

cout << "Направление возвращено\n";

cout << "======================================================" << endl;

return;

}

}

}

else

{

cout << "Данного регистрационного номера нет в базе\n";

cout << "======================================================" << endl;

return;

}

}

//просмотр всех направлений

void show(Data\*& head, Data\*& tail)

{

Data\* help = head;

if (head == NULL)

{

cout << "Записей нет.\n";

return;

}

do

{

cout << "Регистрационный номер: " << help->ticket->registr << endl;

cout << "ФИО врача: "<< help->ticket->fio\_doctor << endl;

cout <<"Дата приема: "<< help->ticket->date << endl;

cout <<"Время приема: "<< help->ticket->time << endl;

cout << "====================================\n";

help = help->next;

} while (help != head);

}

// Запись данных из файла в список

void ReadHelp2(Ticket\*ticket, Data\*& head, Data\*& tail)

{

Data\* help = new Data;

help->ticket = ticket;

if (head == 0)

{

head = help;

tail = help;

tail->next = head;

tail->prev = help;

head->prev = tail;

return;

}

else

{

help->next = tail->next;

help->next->prev = help;

tail->next = help;

help->prev = tail;

tail = help;

if (s > 1)

{

Data\* st = head;

st->prev = NULL;

tail->next = NULL;

head = sort(st, tail, s);

Data\* help = head;

while (help->next != 0)

{

tail = help->next;

tail->prev = help;

help = help->next;

}

head->prev = tail;

tail->next = head;

}

}

}

// Чтение из файла

void Read2(Data\*& Start, Data\*& End)

{

Data\* st = Start;

ifstream read\_file("ticket.txt");

string str, fio, date, time, registr;

char txt[10050];

int x;

if (!read\_file.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else {

do {

Ticket\* ticket = new Ticket;

read\_file.getline(txt, 1050);

for (int i = 0; i < 1050; i++)

{

str += txt[i];

}

x = 0;

while (str[x] != '|')

{

registr += str[x];

x++;

ticket->registr = registr;

}

x++;

while (str[x] != '|')

{

fio += str[x];

x++;

ticket->fio\_doctor = fio;

}

x++;

while (str[x] != '|')

{

date += str[x];

x++;

ticket->date=date;

}

x++;

while (str[x] != ';')

{

time += str[x];

x++;

ticket->time=time;

}

x++;

s++;

ReadHelp2(ticket, Start, End);

str.clear();

fio.clear();

registr.clear();

time.clear();

date.clear();

} while (!read\_file.eof());

read\_file.close();

}

}

// Запись в файл

void Record2(Data\*& Start, Data\*&tail)

{

cout << "Это изменит существующую базу данных. Хотите продолжить?\n1.Да\n2.Нет" << endl;

int choce;

cin >> choce;

switch (choce) {

case 1: {

Data\* st = Start;

ofstream record\_file;

record\_file.open("ticket.txt");

if (!record\_file.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else {

while (st->next != Start)

{

record\_file << st->ticket->registr << "|" << st->ticket->fio\_doctor << "|" << st->ticket->date << "|" << st->ticket->time << ";" << endl;

st = st->next;

}

record\_file << st->ticket->registr << "|" << st->ticket->fio\_doctor << "|" << st->ticket->date << "|" << st->ticket->time << ";";

cout << endl << "Таблица сохранена" << endl << endl;

break;

}

record\_file.close();

}

case 2:

{

break;

}

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "rus");

Hash\* start\_ht = new Hash; // указатель на начало таблицы

Hash\* end\_ht; // указатель на конец таблицы

Three\* start = NULL; // Создаём указатель на дерево

Data\* head = NULL; // указатель на начало списка

Data\* tail = NULL; // указатель на конец списка\*/

while (true)

{

menu\_main();

int k;

int num=1;

check\_menu(k);

cout << k;

switch (k)

{

case 1:

while (true)

{

system("cls");

menu\_patients();

int k1;

check\_menu1(k1);

int l = 0;

switch (k1)

{

case 1:

system("cls");

add\_patient(start\_ht, end\_ht,num);

Record(start\_ht);

show\_patient(start\_ht, end\_ht);

break;

case 2:

system("cls");

del\_patient(start\_ht, end\_ht,head, tail);

Record(start\_ht);

break;

case 3:

system("cls");

show\_patient(start\_ht, end\_ht);

break;

case 4:

system("cls");

search\_to\_registr(start\_ht, end\_ht, head,tail);

break;

case 5:

system("cls");

search\_to\_FUO\_patient(start\_ht, end\_ht);

break;

case 6:

system("cls");

delete\_all\_patients(start\_ht, end\_ht,head);

Record(start\_ht);

num=1;

break;

case 7:

system("cls");

Read(start\_ht, end\_ht, num);

show\_patient(start\_ht, end\_ht);

break;

case 0:

l++;

break;

}

if (l == 1)

{

system("cls");

break;

}

}

break;

case 2:

while (true)

{

system("cls");

menu\_doctors();

int k2;

check\_menu1(k2);

int l = 0;

switch (k2)

{

case 1:

{

system("cls");

cout << "Введите ФИО врача: " << endl;

string value = check\_fio\_doctor();

add\_doctor(start, value);

cout << "Это изменит существующую базу данных. Хотите продолжить?\n1.Да\n2.Нет" << endl;

int choce;

cin >> choce;

switch (choce) {

case 1: {

ofstream ("doctors.txt");

Record1(start); break; }

case 2: {break; }

}

system("pause");

break;

}

case 2:

{

system("cls");

cout << "Введите ФИО врача: " << endl;

string value = check\_fio\_doctor();

del\_doctor(start, value,head,tail);

cout << "Это изменит существующую базу данных. Хотите продолжить?\n1.Да\n2.Нет" << endl;

int choce;

cin >> choce;

switch (choce) {

case 1: {

ofstream("doctors.txt");

Record1(start); break; }

case 2: {break; }

}

system("pause");

break;

}

case 3:

system("cls");

show\_doctors(start);

system("pause");

break;

case 4:

{

system("cls");

cout << "Введите ФИО врача: " << endl;

string value=check\_fio\_doctor();

search\_doctor(start, value,head,tail);

system("pause");

break;

}

case 5:

{

system("cls");

string value;

cout << "Введите должность: ";

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

getline(cin, value);

position\_search(start, value);

system("pause");

break;

}

case 6:

system("cls");

delete\_all\_doctors(start,head);

cout << "Это изменит существующую базу данных. Хотите продолжить?\n1.Да\n2.Нет" << endl;

int choce;

cin >> choce;

switch (choce) {

case 1: {

ofstream("doctors.txt");

break; }

case 2: {break; }

}

system("pause");

break;

case 7:

system("cls");

Read1(start);

show\_doctors(start);

system("pause");

break;

case 0:

l++;

break;

}

if (l == 1)

{

system("cls");

break;

}

}

break;

case 3:

while (true)

{

system("cls");

menu\_issue\_and\_refund();

int k3;

check\_menu2(k3);

int l = 0;

switch (k3)

{

case 1:

{

system("cls");

Insert(head, tail, start\_ht, end\_ht, start);

Record2(head, tail);

show(head, tail);

system("pause");

break;

}

case 2:

system("cls");

Delete(head, tail, start\_ht, end\_ht, start);

Record2(head, tail);

show(head, tail);

system("pause");

break;

case 3:

system("cls");

show(head, tail);

system("pause");

case 4:

system("cls");

Read2(head, tail);

show(head, tail);

system("pause");

break;

case 0:

{

l++;

break; }

}

if (l == 1)

{

system("cls");

break;

}

}

break;

case 0:

return 0;

}

}

}