МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра МОиСИТ

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №5.2

«Алгоритмы поиска в таблице при работе с данными из файла»

по дисциплине

«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-10-24 Бикташев И. И.

Практическую работу выполнил «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва 2025

Цель работы:

Получить практический опыт по применению алгоритмов поиска в таблицах данных.

Задание 1

Задача: Создать двоичный файл из записей. Заполнить файл данными. Ключи записей в файле уникальны.

В качестве записей используют данные пациента состоящие из следующих полей: номер карточки, код заболевания и фамилия врача. Для работы с записями была создана структура Patient (листинг 1)

Листинг 1 - Структура Patient

```
// Данные пациента
struct Patient {
    unsigned long cardNum;
    char* diseaseNum;
    char* doctorName;

    // Переопределяем оператор для сортировки
    bool operator<(const Patient& other) const {
        return cardNum < other.cardNum;
    }
};
```

Размер записи в байтах определил с помощью констант типа size_t (листинг 2).

Листинг 2 - Размеры полей структуры Patient

```
// Размеры полей структуры Patient
const size_t cardNumSize = sizeof(unsigned long);
const size_t diseaseNumSize = 4;
const size_t doctorNameSize = 20;
```

Описание алгоритма: Для начала создаётся текстовый файл из записей, поля записи разделены символом прямого слэша. Сами случайным образом создаются генератором случайных чисел. После данный текстовый файл читается и данные из него конвертируются в бинарный вид и записываются в бинарный файл. Для быстрой конвертации ключа записи (номер карточки) из

типа unsigned long в массив байтов было создано объединение LongByte (листинг 3).

Листинг 3 - Объединение LongByte

```
// Для быстрого конвертирования
union LongByte {
    unsigned long asLong;
    char asBytes[4];
};
```

Доступные коды заболеваний, фамилии врачей и количество записей будут хранятся в соответствующий константах (листинг 4)

Листинг 4 - Константы

```
const string DISEASES[] = {"A01", "B14", "C08", "A02", "A21", "A14", "B01",
"B32", ”C11", "B21", "B45", "C06", "A12", "B20", "B09", "C06", "C26"};

const string DOCTORS[] = {"Лубеников", "Горбунов", "Бандурина", "Шатрова",
"Ардашева", "Гурчиани", "Рябов", "Зыкова", "Калмыков", "Петрунина",
"Ильина", "Максимов"};

const int N = 1000000;
```

Для конвертации текстового файла в бинарный построчно читаем текстовый файл и каждую строчку разделяем на 3 разные строчки: первая строка – номер карточки, конвертируем его в сначала в unsinged long а после с помощью объединения в массив байтов, вторая и третья строчки - это код заболевания и фамилия врача соответственно, их конвертируем в массив байтов и записываем в бинарный файл в соответствии с их заданными размерами.

Код: Были реализованы функции writeTxtFile для записи данных в текстовый файл и convertToBin для конвертирования текстового файла в бинарный (листинг 5).

Листинг 5 - Запись данных и их конвертация

```
// Записывает данные в текстовый файл
void writeTxtFile() {
    ofstream out("patients.txt");
    random_device rd;
    mt19937 gen(rd()); // Генератор случайных чисел

    // Создание диапазона чисел
    vector<unsigned long> cards;
    for (unsigned long i = 0; i < N; i++) {
        cards.push_back(i);
    }

    shuffle(cards.begin(), cards.end(), gen);
```

```
    for (unsigned long i = 0; i < N; i++) {
        string disease = DISEASES[gen() % 17];
        string doctor = DOCTORS[gen() % 12];

        out << cards[i] << "|" << disease << "|" << doctor << endl;
    }

    out.close();
}

// Читает данные из текстого файла и конвертирует их в бинарный файл
void convertToBin() {
    ifstream in("patients.txt");
    ofstream out("patients.bin", ios::binary);

    // Буфер
    string line;
    size_t prev;
    size_t next;
    while (getline(in, line)) {
        // Находит первую подстроку до разделителя
        next = line.find("|");
        auto cardNumStr = line.substr(0, next);
        LongByte cardNum;
        cardNum.asLong = stoul(cardNumStr);
        prev = next + 1;

        next = line.find("|", prev);
        auto diseaseNum = line.substr(prev, next - prev);
        prev = next + 1;

        auto doctorName = line.substr(prev);

        // Записывает в файл блок данных заданного размером
        out.write(cardNum.asBytes, cardNumSize);
        out.write(diseaseNum.c_str(), sizeof(char) * diseaseNumSize);
        out.write(doctorName.c_str(), sizeof(char) * doctorNameSize);
    }

    in.close();
    out.close();
}
```

Результаты тестирования представлены на рисунках 1 и 2

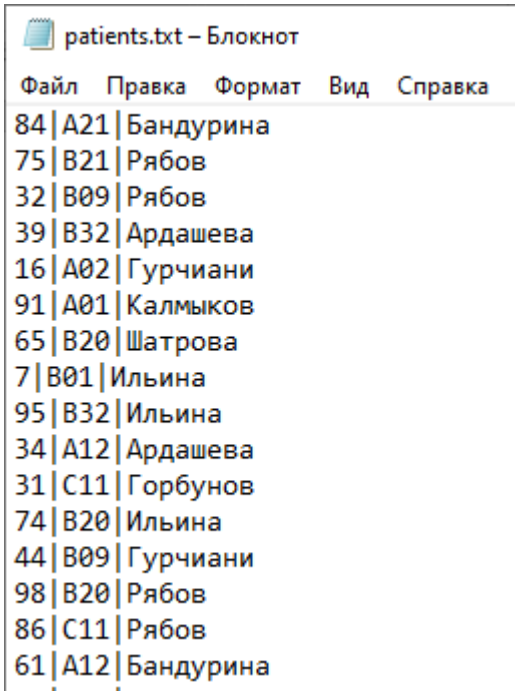

Рисунок 1 - Содержание текстового файла

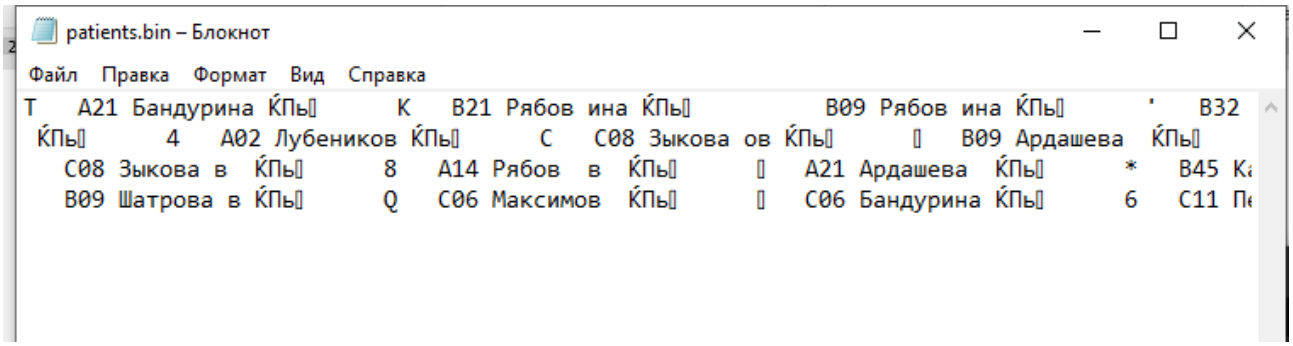

Рисунок 2 - Содержание бинарного файла

Задание 2

Задача: Реализовать поиск в файле с применением линейного поиска.

Для реализации данного алгоритма надо дополнительно конвертировать данные из бинарного файла в массив экземпляров структуры Patient. Для этого мы читаем по несколько байтов и записываем их в поля структуры (листинг 6).

Листинг 6 - Функция чтения из бинарного файла

```
// Читает данные из бинарника
Patient* readFromBin() {
    ifstream in("patients.bin", ios::binary);
    Patient* result = new Patient[N];

    for (int i = 0; i < N; i++) {
        // Создаем указатель на данные пациента
        Patient patient;
        // Инициализирую байты каждого свойства структуры
        LongByte cardNum;
        patient.diseaseNum = new char[diseaseNumSize];
        patient.doctorName = new char[doctorNameSize];
        // Читаем каждое поле побайтово
        in.read(cardNum.asBytes, cardNumSize);
        patient.cardNum = cardNum.asLong;
        in.read(patient.diseaseNum, diseaseNumSize);
        in.read(patient.doctorName, doctorNameSize);
        // Записывает в массив указатель на данные пациента
        result[i] = patient;
    }

    in.close();
    return result;
}
```

Линейный поиск перебирает все элементы массива записей и ищет среди них искомый (по ключу). Если ключ i-ого элемента совпадает с искомым ключом то функция линейного поиска возвращает данный элемент. Для удобства использования других методов поиска был объявлен тип searchFunc и функция searchNum для поиска записи по ключу, выводу записи на экран и замера времени поиска с помощью библиотеки chrono (листинг 7).

```
// Функция поиска
typedef int (*searchFunc)(Patient*, unsigned long);

// Линейный поиск
int linearSearch(Patient* array, unsigned long toFind) {
    for (int i = 0; i < N; i++) {
        if (array[i].cardNum == toFind) {
            return i;
        }
    }

    return -1; // Not found
}

// Поиск значения по ключу
void searchNum(Patient* patients, searchFunc search) {
    int cardNum;
    cout << "Введите нужный номер карты: ";
    cin >> cardNum;
    auto begin = chrono::steady_clock::now();
    int found = search(patients, cardNum);
    auto end = chrono::steady_clock::now();
```

```
    if (found != -1) {
        Patient patient = patients[found];
        cout << patient.cardNum << " " <<
                patient.diseaseNum << " " <<
                patient.doctorName << endl;
    } else {
        cout << "Карта не найдена";
    }

    cout << "Прошло: " << chrono::duration_cast<chrono::milliseconds>(end -
begin).count() << " мс" << endl;
}
```

Результаты тестирования программы с замерами времени предоставлены на рисунках 3 и 4.

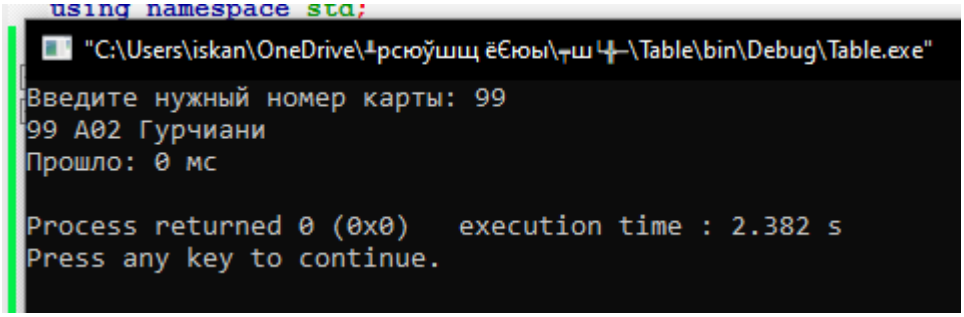

Рисунок 3 - Результат работы при N = 100

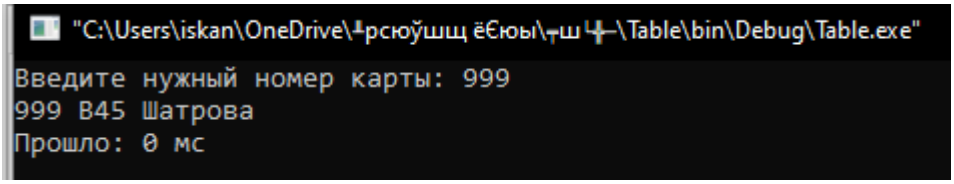

Рисунок 4 - Результат работы при N = 1000

Задание 3

Задача: Поиск записи в файле с применением интерполяционного поиска

Интерполяционный поиск – это улучшенная версия бинарного поиска для отсортированных массивов с равномерным распределением. Вместо деления пополам вычисляет вероятную позицию элемента с помощью формулы интерполяции (листинг 7).

```
// Интерполяционный поиск
int interpolationSearch(Patient* array, unsigned long toFind) {
    int mid;
    // Возвращает индекс элемента со значением toFind или -1, если такого
элемента не существует
```

```
    int low = 0;
    int high = N - 1;

    while (array[low].cardNum < toFind && array[high].cardNum > toFind) {
        if (array[high].cardNum == array[low].cardNum) // Защита от деления на
0
            break;
        mid = low + ((toFind - array[low].cardNum) * (high - low)) /
(array[high].cardNum - array[low].cardNum);

        if (array[mid].cardNum < toFind)
            low = mid + 1;
        else if (array[mid].cardNum > toFind)
            high = mid - 1;
        else
            return mid;
    }

    if (array[low].cardNum == toFind)
        return low;
    if (array[high].cardNum == toFind)
        return high;

    return -1; // Not found
}
```

Результат работы программы при N = 10000 предоставлен на рисунке 5.

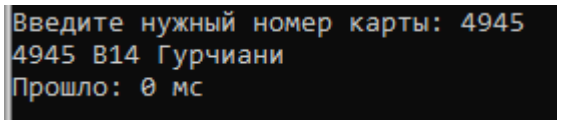

Рисунок 5 - Результат работы интерполяционного поиска

Таблица 1 - Время работы поиска при разных N

N	Время работы программы, мс
100	0
1000	0
10000	0
100000	0
500000	2

Вывод

Поставленные задачи выполнены: реализованы функции поиска в массиве данных по ключу, кодирования данных в текстовый или бинарный файл и чтение из них.

Литература

Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/ (дата обращения 01.09.2021).

Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).