



МИНОБРНАУКИ РОССИИ
*Федеральное государственное бюджетное образовательное
учреждение высшего образования*
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Отчет по выполнению практического задания №5
Тема: «Работа с данными из файла»
Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Петров А.
Фамилия И.О.

Группа: ИКБО 22-23
Номер группы

Москва 2024

Содержание

Содержание	2
Часть 5.1.	3
Цель работы.....	3
Ход Работы	3
Задание 1. Битовые операции.....	3
Задание 2. Сортировка с помощью битового массива	5
Задание 3. Сортировка файла чисел	8
Часть 5.2	11
Цель работы.....	11
Ход Работы	11
Задание 1. Разработать программу поиска записей с заданным ключом в двоичном файле с применением различных алгоритмов.....	11
Задание 2. Поиск в файле с применением линейного поиска	16
Задание 3. Поиск записи в файле с применением дополнительной структуры данных, сформированной в оперативной памяти	23
Анализ эффективности рассмотренных алгоритмов поиска в файле	28
Вывод.....	29
Список литературы	30

Часть 5.1.

Цель работы

Целью данной работы является освоение приёмов работы с битовым представлением целых чисел, а также реализация эффективного алгоритма сортировки числового файла с использованием битового массива.

Ход Работы

Задание 1. Битовые операции

Формулировка задачи

Необходимо освоить битовые операции для установки и проверки состояния отдельных битов числа. Реализовать примеры по работе с битовыми операциями, установить пятый и седьмой биты числа, а также вывести его побитово.

Математическая модель решения

Используются битовые операции сдвига и побитового И/ИЛИ для манипуляции отдельными битами числа. Для задания 1а необходимо установить 5-й бит числа в 0, для задания 1б — установить 7-й бит в 1, а для задания 1в — вывести двоичное представление числа побитово, начиная со старшего бита.

Код программы с комментариями

Задание 1а:

```
1  #include <iostream>
2  #include <bitset>
3
4  using namespace std;
5
6  int main()
7  {
8      unsigned char x = 255; // 11111111
9      unsigned char maska = 1; // 00000001
10     x = x & (~ (maska << 4)); // 11101111 (с реверсом 00010000)
11
12     cout << "Результат: " << (int)x << endl;
13     cout << "Результат в двоичном: " << bitset<8>(x) << endl;
14
15     return 0;
16 }
17
```

Задание 1б:

```
1  #include <iostream>
2  #include <bitset>
3
4  using namespace std;
5
6  int main()
7  {
8      unsigned char x = 0; // 00000000
9      unsigned char maska = 1; // 00000001
10     x = x | (maska << 6); // 01000000
11
12     cout << "Результат: " << (int)x << endl;
13     cout << "Результат в двоичном виде: " << bitset<8>(x) << endl;
14
15     return 0;
16 }
17
```

Задание 1в:

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <Windows.h>
4  #include <bitset>
5
6  using namespace std;
7
8  int main()
9  {
10
11     unsigned int x = 25; // 00000000 00000000 00000000 00011001
12     const int n = sizeof(int) * 8; // 32 бита
13     unsigned maska = (1 << (n - 1)); // 10000000 00000000 00000000 00000000
14
15     cout << "Начальный вид маски: " << bitset<n>(maska) << endl;
16     cout << "Результат: ";
17
18     for (int i = 1; i <= n; i++)
19     {
20         cout << ((x & maska) >> (n - i)); //
21         maska = maska >> 1; // сдвигаем маску на 1 бит вправо
22     }
23
24     cout << endl;
25     system("pause");
26     return 0;
27 }
28
```

Результаты тестирования

- Для задания 1а результат: число с 5-м битом, установленным в 0.
- Для задания 1б результат: число с 7-м битом, установленным в 1.
- Для задания 1в выводится двоичное представление числа 25 (с использованием битового поиска).

Задание 2. Сортировка с помощью битового массива

Формулировка задачи

Необходимо реализовать сортировку последовательности чисел с использованием битового массива, где каждый бит отображает наличие числа в наборе.

Математическая модель решения

Массив битов используется для представления набора чисел. Для каждого числа соответствующий бит в массиве устанавливается в 1. Для получения отсортированной последовательности программа проходит по массиву и выводит индексы, соответствующие установленным битам.

Код программы с комментариями

Задание 2а:

```
1  #include <iostream>
2  #include <bitset>
3
4  using namespace std;
5
6  int main()
7  {
8      unsigned char bitArray = 0; // 00000000
9      int numbers[8], n;
10
11     cout << "Введите количество чисел (не более 8): ";
12     cin >> n;
13
14     if (n > 8)
15     {
16         cout << "Чисел должно быть не более 8" << endl;
17         return 1;
18     }
19
20     cout << "Введите числа (от 0 до 7): ";
21     for (int i = 0; i < n; i++)
22     {
23         cin >> numbers[i];
24         if (numbers[i] >= 0 && numbers[i] <= 7)
25         {
26             bitArray |= (1 << numbers[i]); // 0001 сдвигаем на 1 бит влево(0010, 0100, 1000)
27         }
28         else
29         {
30             cout << "Число вне диапазона" << endl;
31             return 1;
32         }
33     }
34 }
```

```

35     cout << "Отсортированная последовательность: ";
36     for (int i = 0; i < 8; i++)
37     {
38         if (bitArray & (1 << i)) //выводим числа, которые есть в массиве(биты)
39         {
40             cout << i << " ";
41         }
42     }
43     cout << endl;
44
45     return 0;
46 }
47

```

Задание 2б:

```

1  #include <iostream>
2  #include <bitset>
3
4  using namespace std;
5
6  int main()
7  {
8      unsigned long long bitArray = 0;
9      int numbers[64], n;
10
11     cout << "Введите количество чисел (не более 64): ";
12     cin >> n;
13
14     if (n > 64)
15     {
16         cout << "Чисел должно быть не более 64!" << endl;
17         return 1;
18     }
19
20     cout << "Введите числа (от 0 до 63): ";
21     for (int i = 0; i < n; i++)
22     {
23         cin >> numbers[i];
24         if (numbers[i] >= 0 && numbers[i] <= 63)
25         {
26             bitArray |= (1ULL << numbers[i]); // 1 - unsigned long long
27         }
28     else
29     {
30         cout << "Число вне допустимого диапазона!" << endl;
31         return 1;
32     }
33 }
34

```

```

35     cout << "Отсортированная последовательность: ";
36     for (int i = 0; i < 64; i++)
37     {
38         if (bitArray & (1ULL << i))
39         {
40             cout << i << " ";
41         }
42     }
43     cout << endl;
44
45     return 0;
46 }
47

```

Задание 2в:

```

1  #include <iostream>
2  #include <bitset>
3
4  using namespace std;
5
6  int main()
7  {
8      unsigned char bitArray[8] = {0}; // 00000000
9      int numbers[64], n;
10
11     cout << "Введите количество чисел (не более 64): ";
12     cin >> n;
13
14     if (n > 64)
15     {
16         cout << "Чисел должно быть не более 64!" << endl;
17         return 1;
18     }
19
20     cout << "Введите числа (от 0 до 63): ";
21     for (int i = 0; i < n; i++)
22     {
23         cin >> numbers[i];
24         if (numbers[i] >= 0 && numbers[i] <= 63)
25         {
26             int byteIndex = numbers[i] / 8; // 0-7, 8-15..
27             int bitIndex = numbers[i] % 8; // 0-7, 0-7..
28             bitArray[byteIndex] |= (1 << bitIndex); // также как и в предыдущих заданиях
29         }
30         else
31         {
32             cout << "Число вне допустимого диапазона!" << endl;
33             return 1;
34         }
35     }

```

```

36
37     cout << "Отсортированная последовательность: ";
38     for (int i = 0; i < 64; i++)
39     {
40         int byteIndex = i / 8;
41         int bitIndex = i % 8;
42         if (bitArray[byteIndex] & (1 << bitIndex))
43         {
44             cout << i << " ";
45         }
46     }
47     cout << endl;
48
49     return 0;
50 }
51

```

Результаты тестирования:

Программа успешно сортирует последовательность чисел, разных размеров (изменённые в постановках задач 2а, 2б), как пример:

Задание 2а:

последовательность

{1, 0, 5, 7, 2, 4} -> {0, 1, 2, 4, 5, 7}

Задание 2б:

Так же сортирует последовательность и на больших значениях (64 числа)

Задание 2в:

И не меняет логики программы на примере массивов

Задание 3. Сортировка файла чисел

Формулировка задачи

Требуется реализовать сортировку числового файла, содержащего не более 10^7 неотрицательных целых чисел, с использованием битового массива и учётом ограничения на память (1 МБ).

Математическая модель решения

Каждое число представляется битом в битовом массиве. Для каждого числа в диапазоне от 1,000,000 до 9,999,999 соответствующий бит устанавливается в

единицу. После чтения файла программа проходит по битовому массиву и выводит отсортированные числа.

Код программы с комментариями

Задания 3а и 3б

```
1  #include <iostream>
2  #include <fstream>
3  #include <chrono>
4  #include <cstring>
5
6  using namespace std;
7  using namespace chrono;
8
9  int main()
10 {
11
12     auto start = high_resolution_clock::now();
13
14
15     const int MIN_NUMBER = 1000000;
16     const int MAX_NUMBER = 9999999;
17     const int TOTAL_BITS = MAX_NUMBER - MIN_NUMBER + 1;
18     const int BYTE_SIZE = 8;
19     const int ARRAY_SIZE = TOTAL_BITS / BYTE_SIZE + 1;
20
21     unsigned char* bitArray = new unsigned char[ARRAY_SIZE];
22     memset(bitArray, 0, ARRAY_SIZE); // устанавливаем все биты в 0
23
24     cout << "Память, занимаемая битовым массивом: " << ARRAY_SIZE << " байт" << endl;
25     cout << "Память, занимаемая битовым массивом: " << ARRAY_SIZE / 1024 << " килобайт" << endl;
26     cout << "Память, занимаемая битовым массивом: " << ARRAY_SIZE / (1024 * 1024) << " мегабайт" << endl;
27
28     ifstream inputFile("input.txt");
29     if (!inputFile.is_open())
30     {
31         cout << "Ошибка открытия файла для чтения!" << endl;
32         delete[] bitArray;
33         return 1;
34     }
35 }
```

```

35
36     int number;
37     while (inputFile >> number)
38     {
39         if (number >= MIN_NUMBER && number <= MAX_NUMBER)
40         {
41             int index = number - MIN_NUMBER;
42             bitArray[index / BYTE_SIZE] |= (1 << (index % BYTE_SIZE));
43         }
44     }
45     inputFile.close();
46
47
48     ofstream outputFile("output.txt");
49     if (!outputFile.is_open())
50     {
51         cout << "Ошибка открытия файла для записи!" << endl;
52         delete[] bitArray;
53         return 1;
54     }
55
56
57     for (int i = 0; i < TOTAL_BITS; ++i)
58     {
59         if (bitArray[i / BYTE_SIZE] & (1 << (i % BYTE_SIZE)))
60         {
61             outputFile << (i + MIN_NUMBER) << endl;
62         }
63     }

```

```

64     outputFile.close();
65
66
67     auto end = high_resolution_clock::now();
68     auto duration = duration_cast<seconds>(end - start);
69
70     cout << "Время работы программы: " << duration.count() << " секунд" << endl;
71
72
73     delete[] bitArray;
74
75     return 0;
76 }
77

```

Результаты тестирования

Память, занимаемая битовым массивом: 1125001 байт

Время работы программы: 8 секунд

Часть 5.2

Цель работы

Получить практический опыт по применению алгоритмов поиска в таблицах данных.

Ход Работы

Вариант № 1.

Алгоритм поиска:

Бинарный однородный без использования дополнительной таблицы.

Структура записи файла (ключ – подчеркнутое поле):

Читательский абонемент: номер читательского билета - целое пятизначное число, ФИО, Адрес.

Задание 1. Разработать программу поиска записей с заданным ключом в двоичном файле с применением различных алгоритмов.

Постановка задачи

Создать двоичный файл из записей, структура которых определена вариантом. Поле ключа записи в задании варианта подчеркнуто. Заполнить файл данными, используя для поля ключа датчик случайных чисел. Ключи записей в файле уникальны.

Описание подхода к решению

Определение структуры записи файла:

```
struct Reader {  
    int ticket_number; // номер читательского билета  
    char full_name[50]; // ФИО  
    char city[30]; // Город  
}
```

Определение размера записи в байтах:

Номер читательского билета: 4 байта (тип int).

ФИО: 50 байт (тип char[50]).

Город: 30 байт (тип char[30]).

каждая запись будет занимать - 84 байта.

Организация прямого доступа к записям в бинарном файле:

Поскольку каждая запись имеет фиксированный размер (84 байта), прямой доступ к любой записи можно осуществлять, зная её порядковый номер в файле. Для этого нужно умножить номер записи на размер одной записи (84 байта) и использовать `seekg()` для перемещения указателя файла к нужной позиции.

Алгоритмы, реализуемые в форме функций:

1. Функция для генерации текстового файла с уникальными ключами и случайными данными.
2. Функция для преобразования текстового файла в бинарный.
3. Функция для поиска записи по ключу с использованием бинарного поиска.
4. Функция для отображения записи.

Прототипы функций:

Генерация текстового файла

```
void generate_text_file(const std::string& filename, int count);  
// Предусловие: файл должен быть доступен для записи, count > 0  
// Постусловие: сгенерирован текстовый файл с count записями
```

Преобразование в бинарный файл

```
void convert_to_binary(const std::string& text_filename, const std::string&  
binary_filename);  
// Предусловие: текстовый файл должен существовать  
// Постусловие: сгенерирован бинарный файл на основе текстового файла
```

Бинарный поиск

```
bool binary_search_in_file(const std::string& binary_filename, int key, int&  
record_index);  
// Предусловие: бинарный файл должен быть отсортирован по ключам  
// Постусловие: возвращает true, если запись найдена, иначе false. Индекс  
записи записан в record_index
```

Отображение записи

```
void display_record(const std::string& binary_filename, int record_index);  
// Предусловие: бинарный файл существует, индекс записи корректен  
// Постусловие: выводит на экран содержимое записи
```

Код программы линейного поиска записи по ключу в листинг 1.

Листинг 1.

```
#include <iostream>  
#include <fstream>  
#include <string>  
#include <vector>  
#include <cstdlib>  
#include <ctime>  
#include <algorithm>  
#include <locale>  
  
using namespace std;  
  
struct Reader {  
    int ticket_number;    // Номер читательского билета (ключ)  
    char full_name[50];    // ФИО  
    char city[30];        // Город  
};  
  
// Функция для вывода всех записей из бинарного файла  
void display_all_records(const string& binary_filename) {  
    ifstream binary_file(binary_filename, ios::binary);  
  
    if (!binary_file.is_open()) {  
        cerr << "Ошибка открытия файла!" << endl;  
        return;  
    }  
  
    Reader reader;  
    int index = 0;  
  
    // Чтение записей до конца файла  
    while (binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader))) {  
        cout << "Запись #" << ++index << "\n";  
        cout << "Номер билета: " << reader.ticket_number << "\n";  
        cout << "ФИО: " << reader.full_name << "\n";  
        cout << "Город: " << reader.city << "\n";  
        cout << "-----\n";  
    }  
  
    binary_file.close();  
}  
  
// Предусловие: файл должен быть доступен для записи, count > 0  
// Постусловие: сгенерирован текстовый файл с count записями  
void generate_text_file(const string& filename, int count) {  
    ofstream file(filename);  
    if (!file.is_open()) {  
        cerr << "Ошибка открытия файла!" << endl;  
        return;  
    }
```

```

    }

    srand(time(nullptr));
    vector<int> used_keys;

    for (int i = 0; i < count; ++i) {
        int key;
        do {
            key = 10000 + rand() % 90000; // Генерация уникального ключа (5-
значный номер)
        } while ( find(used_keys.begin(), used_keys.end(), key) !=
used_keys.end());

        used_keys.push_back(key);

        string full_name = "Reader_" + to_string(i + 1);
        string city = "City_" + to_string(i % 10 + 1);

        file << key << " " << full_name << " " << city << "\n";
    }

    file.close();
}

// Предусловие: текстовый файл должен существовать
// Постусловие: сгенерирован бинарный файл на основе текстового файла
void convert_to_binary(const string& text_filename, const string&
binary_filename) {
    ifstream text_file(text_filename);
    ofstream binary_file(binary_filename, ios::binary);

    if (!text_file.is_open() || !binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    Reader reader;
    while (text_file >> reader.ticket_number >> reader.full_name >> reader.city) {
        binary_file.write(reinterpret_cast<char*>(&reader), sizeof(Reader));
    }

    text_file.close();
    binary_file.close();
}

// Предусловие: бинарный файл существует, индекс записи корректен
// Постусловие: выводит на экран содержимое записи
void display_record(const string& binary_filename, int record_index) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    binary_file.seekg(record_index * sizeof(Reader), ios::beg);

    Reader reader;
    binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader));

    cout << "Номер билета: " << reader.ticket_number << "\n";
    cout << "ФИО: " << reader.full_name << "\n";
    cout << "Город: " << reader.city << "\n";
}

```

```

        binary_file.close();
    }

    int main() {
        setlocale(LC_ALL, "RU");
        const string text_filename = "readers.txt";
        const string binary_filename = "readers.bin";
        const int record_count = 100;

        // 1. Генерация текстового файла
        generate_text_file(text_filename, record_count);

        // 2. Преобразование в бинарный файл
        convert_to_binary(text_filename, binary_filename);

        // 3. Вывод всех записей, чтобы узнать номер билета
        cout << "Все записи в файле:\n";
        display_all_records(binary_filename);

        return 0;
    }

```

Задание 2. Поиск в файле с применением линейного поиска

Постановка задачи

Разработать программу поиска записи по ключу в бинарном файле с применением алгоритма линейного поиска.

Описание подхода к решению

Псевдокод линейного поиска записи с ключом в файле:

Номер инструкции	Код алгоритма <code>binary_search_in_file</code>
1.	<code>binary_search_in_file(имя_файла, ключ, индекс_записи)</code>
2.	Открыть <code>binary_filename</code>
3.	Переместить указатель на конец файла
4.	<code>total_records = размер_файла / размер_записи_Reader</code>
5.	<code>left = 0</code>
6.	<code>right = total_records - 1</code>
7.	While <code>left <= right</code>
8.	<code>mid = (left + right) / 2</code> // найти середину диапазона
9.	Переместить указатель на позицию <code>mid * размер_записи_Reader</code> в файле
10.	Прочитать запись в переменную <code>reader</code>
11.	if номер_билета <code>reader == ключу</code>
12.	<code>индекс_записи = mid</code>
13.	Закрыть файл
14.	return ИСТИНА
15.	If else номер_билета <code>reader < ключа</code>
16.	<code>left = mid + 1</code> // искомый ключ в правой половине диапазона

17.	else right = mid - 1 // искомый ключ в левой половине диапазона
18.	End while
19.	Закреть файл

Код функции поиска:

```

bool binary_search_in_file(const string& binary_filename, int key, int& record_index) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return false;
    }

    binary_file.seekg(0, ios::end);
    int total_records = binary_file.tellg() / sizeof(Reader);
    int left = 0, right = total_records - 1;

    while (left <= right) {
        int mid = (left + right) / 2;
        binary_file.seekg(mid * sizeof(Reader), ios::beg);

        Reader reader;
        binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader));

        if (reader.ticket_number == key) {
            record_index = mid;
            binary_file.close();
            return true;
        }
        else if (reader.ticket_number < key) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }

    binary_file.close();
    return false;
}

```

Код программы бинарного поиска в листинг 2.

Листинг 2.

```

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <locale>
#include <chrono>
using namespace std;

```

```

struct Reader {
    int ticket_number;    // Номер читательского билета (ключ)
    char full_name[50];   // ФИО
    char city[30];        // Город
};

// Функция для вывода всех записей из бинарного файла
void display_all_records(const string& binary_filename) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    Reader reader;
    int index = 0;

    // Чтение записей до конца файла
    while (binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader))) {

        cout << "Запись #" << ++index << "\n";
        cout << "Номер билета: " << reader.ticket_number << "\n";
        cout << "ФИО: " << reader.full_name << "\n";
        cout << "Город: " << reader.city << "\n";
        cout << "-----\n";
    }

    binary_file.close();
}

// Предусловие: бинарный файл существует
// Постусловие: возвращает true, если запись найдена, иначе false. Индекс записи
// записан в record_index
bool linear_search_in_file(const string& binary_filename, int key, int&
record_index) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return false;
    }

    Reader reader;
    int index = 0;

    // Чтение записей и линейный поиск
    while (binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader))) {
        if (reader.ticket_number == key) {
            record_index = index;
            binary_file.close();
            return true;
        }
        ++index;
    }

    binary_file.close();
    return false;
}

// Предусловие: файл должен быть доступен для записи, count > 0
// Постусловие: сгенерирован текстовый файл с count записями

```

```

void generate_text_file(const string& filename, int count) {
    ofstream file(filename);
    if (!file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    srand(time(nullptr));
    vector<int> used_keys;

    for (int i = 0; i < count; ++i) {
        int key;
        do {
            key = 10000 + rand() % 90000; // Генерация уникального ключа (5-
            // значный номер)
        } while (find(used_keys.begin(), used_keys.end(), key) !=
            used_keys.end());

        used_keys.push_back(key);

        string full_name = "Reader_" + to_string(i + 1);
        string city = "City_" + to_string(i % 10 + 1);

        file << key << " " << full_name << " " << city << "\n";
    }

    file.close();
}

// Предусловие: текстовый файл должен существовать
// Постусловие: сгенерирован бинарный файл на основе текстового файла
void convert_to_binary(const string& text_filename, const string&
    binary_filename) {
    ifstream text_file(text_filename);
    ofstream binary_file(binary_filename, ios::binary);

    if (!text_file.is_open() || !binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    Reader reader;
    while (text_file >> reader.ticket_number >> reader.full_name >> reader.city) {
        // Гарантируем, что строки в массиве заканчиваются нулевым символом
        reader.full_name[sizeof(reader.full_name) - 1] = '\0';
        reader.city[sizeof(reader.city) - 1] = '\0';
        binary_file.write(reinterpret_cast<char*>(&reader), sizeof(Reader));
    }

    text_file.close();
    binary_file.close();
}

// Предусловие: бинарный файл существует, индекс записи корректен
// Постусловие: выводит на экран содержимое записи
void display_record(const string& binary_filename, int record_index) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }
}

```

```

        binary_file.seekg(record_index * sizeof(Reader), ios::beg);

        Reader reader;
        binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader));
        reader.full_name[sizeof(reader.full_name) - 1] = '\0';
        reader.city[sizeof(reader.city) - 1] = '\0';

        cout << "Номер билета: " << reader.ticket_number << "\n";
        cout << "ФИО: " << reader.full_name << "\n";
        cout << "Город: " << reader.city << "\n";

        binary_file.close();
    }

    // Предусловие: бинарный файл должен быть отсортирован по ключам
    // Постусловие: возвращает true, если запись найдена, иначе false. Индекс записи
    // записан в record_index
    bool binary_search_in_file(const string& binary_filename, int key, int&
        record_index) {
        ifstream binary_file(binary_filename, ios::binary);

        if (!binary_file.is_open()) {
            cerr << "Ошибка открытия файла!" << endl;
            return false;
        }

        binary_file.seekg(0, ios::end);
        int total_records = binary_file.tellg() / sizeof(Reader);
        int left = 0, right = total_records - 1;

        while (left <= right) {
            int mid = (left + right) / 2;
            binary_file.seekg(mid * sizeof(Reader), ios::beg);

            Reader reader;
            binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader));

            if (reader.ticket_number == key) {
                record_index = mid;
                binary_file.close();
                return true;
            }
            else if (reader.ticket_number < key) {
                left = mid + 1;
            }
            else {
                right = mid - 1;
            }
        }

        binary_file.close();
        return false;
    }

    int main() {
        setlocale(LC_ALL, "RU");
        const string text_filename = "readers.txt";
        const string binary_filename = "readers.bin";
        const int record_count = 10;
    }

```

```

// 1. Генерация текстового файла
generate_text_file(text_filename, record_count);

// 2. Преобразование в бинарный файл
convert_to_binary(text_filename, binary_filename);

// 3. Вывод всех записей, чтобы узнать номер билета
cout << "Все записи в файле:\n";
display_all_records(binary_filename);

// 4. Поиск записи по ключу
int key_to_search;
cout << "Введите номер читательского билета для поиска: ";
cin >> key_to_search;

int record_index;

// Измерение времени линейного поиска
auto start = chrono::high_resolution_clock::now();
binary_search_in_file(binary_filename, key_to_search, record_index);
display_record(binary_filename, record_index);
auto end = chrono::high_resolution_clock::now();
chrono::duration<double> elapsed = end - start;

cout << "Время выполнения линейного поиска: " << elapsed.count() << "
секунд\n";

return 0;
}

```

Таблица с замерами времени поиска записи:

Количество записей	Бинарный поиск (время в секундах)
100	0.0013124
1 000	0.0013699
10 000	0.0026451

Результаты тестирования:

100 записей

```

-----
Запись #100
Номер билета: 25548
ФИО: Reader_100
Город: City_10
-----
Введите номер читательского билета для поиска: 25548
Номер билета: 25548
ФИО: Reader_100
Город: City_10
Время выполнения линейного поиска: 0.0013124 секунд

```

1000 записей

```
Запись #1000
Номер билета: 24143
ФИО: Reader_1000
Город: City_10
-----
Введите номер читательского билета для поиска: 24143
Номер билета: 24143
ФИО: Reader_1000
Город: City_10
Время выполнения линейного поиска: 0.0013699 секунд
```

1000 записей

```
-----
Запись #10000
Номер билета: 17633
ФИО: Reader_10000
Город: City_10
-----
Введите номер читательского билета для поиска: 17633
Номер билета: 17633
ФИО: Reader_10000
Город: City_10
Время выполнения линейного поиска: 0.0026451 секунд
```

Задание 3. Поиск записи в файле с применением дополнительной структуры данных, сформированной в оперативной памяти

Постановка задачи

Требуется реализовать программу поиска записи по ключу в бинарном файле, используя дополнительную структуру данных, которая хранится в оперативной памяти. В данном случае это может быть массив или вектор с ключами (номераами читательских билетов), который будет загружаться в память перед поиском.

Описание подхода к решению

Таблица содержит две компоненты: ключ и ссылку (смещение) на запись в файле. Ссылка определяет смещение записи от начала файла в байтах.

Код функции поиска:

```
127 bool binary_search_in_file(const string& binary_filename, int key, int& record_index) {
128     ifstream binary_file(binary_filename, ios::binary);
129
130     if (!binary_file.is_open()) {
131         cerr << "Ошибка открытия файла!" << endl;
132         return false;
133     }
134
135     binary_file.seekg(0, ios::end);
136     int total_records = binary_file.tellg() / sizeof(Reader);
137     int left = 0, right = total_records - 1;
138
139     while (left <= right) {
140         int mid = (left + right) / 2;
141         binary_file.seekg(mid * sizeof(Reader), ios::beg);
142
143         Reader reader;
144         binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader));
145
146         if (reader.ticket_number == key) {
147             record_index = mid;
148             binary_file.close();
149             return true;
150         }
151         else if (reader.ticket_number < key) {
152             left = mid + 1;
153         }
154         else {
155             right = mid - 1;
156         }
157     }
158
159     binary_file.close();
160     return false;
161 }
```

Код программы бинарного поиска записи по ключу в листинг 3.

Листинг 3.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>
```

```

#include <ctime>
#include <algorithm>
#include <locale>
#include <chrono>
using namespace std;

struct Reader {
    int ticket_number;    // Номер читательского билета (ключ)
    char full_name[50];   // ФИО
    char city[30];        // Город
};
// структура данных в оперативной памяти
struct IndexEntry {
    int key;              // Номер читательского билета
    long offset;          // Смещение в бинарном файле
};
vector<IndexEntry> indexTable; // Таблица индексов

// функции для построения таблицы индексов
void build_index_table(const string& binary_filename) {
    ifstream binary_file(binary_filename, ios::binary);
    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    Reader reader;
    long offset = 0;

    while (binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader))) {
        indexTable.push_back({ reader.ticket_number, offset });
        offset += sizeof(Reader);
    }

    binary_file.close();
}

// функция поиска по таблице индексов
bool index_search(int key, long& offset) {
    for (const auto& entry : indexTable) {
        if (entry.key == key) {
            offset = entry.offset;
            return true;
        }
    }
    return false;
}

// Функция для чтения записи по смещению
Reader read_record_by_offset(const string& binary_filename, long offset) {
    ifstream binary_file(binary_filename, ios::binary);
    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        throw runtime_error("Ошибка открытия файла");
    }

    binary_file.seekg(offset, ios::beg);
    Reader reader;
    binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader));
    binary_file.close();
    return reader;
}

// Функция для вывода всех записей из бинарного файла

```



```

void display_all_records(const string& binary_filename) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    Reader reader;
    int index = 0;

    // Чтение записей до конца файла
    while (binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader))) {
        cout << "Запись #" << ++index << "\n";
        cout << "Номер билета: " << reader.ticket_number << "\n";
        cout << "ФИО: " << reader.full_name << "\n";
        cout << "Город: " << reader.city << "\n";
        cout << "-----\n";
    }

    binary_file.close();
}

// Предусловие: бинарный файл существует
// Постусловие: возвращает true, если запись найдена, иначе false. Индекс записи
// записан в record_index
bool linear_search_in_file(const string& binary_filename, int key, int&
record_index) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return false;
    }

    Reader reader;
    int index = 0;

    // Чтение записей и линейный поиск
    while (binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader))) {
        if (reader.ticket_number == key) {
            record_index = index;
            binary_file.close();
            return true;
        }
        ++index;
    }

    binary_file.close();
    return false;
}

// Предусловие: файл должен быть доступен для записи, count > 0
// Постусловие: сгенерирован текстовый файл с count записями
void generate_text_file(const string& filename, int count) {
    ofstream file(filename);
    if (!file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    srand(time(nullptr));
}

```

```

vector<int> used_keys;

for (int i = 0; i < count; ++i) {
    int key;
    do {
        key = 10000 + rand() % 90000; // Генерация уникального ключа (5-
значный номер)
    } while (find(used_keys.begin(), used_keys.end(), key) !=
used_keys.end());

    used_keys.push_back(key);

    string full_name = "Reader_" + to_string(i + 1);
    string city = "City_" + to_string(i % 10 + 1);

    file << key << " " << full_name << " " << city << "\n";
}

file.close();
}

// Предусловие: текстовый файл должен существовать
// Постусловие: сгенерирован бинарный файл на основе текстового файла
void convert_to_binary(const string& text_filename, const string&
binary_filename) {
    ifstream text_file(text_filename);
    ofstream binary_file(binary_filename, ios::binary);

    if (!text_file.is_open() || !binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    Reader reader;
    while (text_file >> reader.ticket_number >> reader.full_name >> reader.city) {
        binary_file.write(reinterpret_cast<char*>(&reader), sizeof(Reader));
    }

    text_file.close();
    binary_file.close();
}

// Предусловие: бинарный файл существует, индекс записи корректен
// Постусловие: выводит на экран содержимое записи
void display_record(const string& binary_filename, int record_index) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return;
    }

    binary_file.seekg(record_index * sizeof(Reader), ios::beg);

    Reader reader;
    binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader));

    cout << "Номер билета: " << reader.ticket_number << "\n";
    cout << "ФИО: " << reader.full_name << "\n";
    cout << "Город: " << reader.city << "\n";

    binary_file.close();
}

```

```

// Предусловие: бинарный файл должен быть отсортирован по ключам
// Постусловие: возвращает true, если запись найдена, иначе false. Индекс записи
записан в record_index
bool binary_search_in_file(const string& binary_filename, int key, int&
record_index) {
    ifstream binary_file(binary_filename, ios::binary);

    if (!binary_file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        return false;
    }

    binary_file.seekg(0, ios::end);
    int total_records = binary_file.tellg() / sizeof(Reader);
    int left = 0, right = total_records - 1;

    while (left <= right) {
        int mid = (left + right) / 2;
        binary_file.seekg(mid * sizeof(Reader), ios::beg);

        Reader reader;
        binary_file.read(reinterpret_cast<char*>(&reader), sizeof(Reader));

        if (reader.ticket_number == key) {
            record_index = mid;
            binary_file.close();
            return true;
        }
        else if (reader.ticket_number < key) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }

    binary_file.close();
    return false;
}

int main() {
    setlocale(LC_ALL, "RU");
    const string text_filename = "readers.txt";
    const string binary_filename = "readers.bin";
    const int record_count = 100;

    // 1. Генерация текстового файла
    generate_text_file(text_filename, record_count);

    // 2. Преобразование в бинарный файл
    convert_to_binary(text_filename, binary_filename);

    // 3. Построение таблицы индексов
    build_index_table(binary_filename);
    display_all_records("readers.bin");

    // 4. Поиск записи по ключу
    int key_to_search;
    cout << "Введите номер читательского билета для поиска: ";
    cin >> key_to_search;

    long offset;

```

```

auto start = chrono::high_resolution_clock::now();
index_search(key_to_search, offset);
Reader reader = read_record_by_offset(binary_filename, offset);
auto end = chrono::high_resolution_clock::now();

chrono::duration<double> elapsed = end - start;
cout << "Номер билета: " << reader.ticket_number << "\n";
cout << "ФИО: " << reader.full_name << "\n";
cout << "Город: " << reader.city << "\n";

cout << "Время выполнения поиска: " << elapsed.count() << " секунд\n";

return 0;
}

```

Результат тестирования программы для 100 записей:

```

Запись #100
Номер билета: 16227
ФИО: Reader_100
Город: City_10
-----
Введите номер читательского билета для поиска: 16227
Номер билета: 16227
ФИО: Reader_100
Город: City_10
Время выполнения поиска: 0.0001298 секунд

```

Таблица с замерами времени поиска записи:

Количество записей	Бинарный поиск (время в секундах)
100	0.0001298
1 000	0.0001218
10 000	0.0001473

Анализ эффективности рассмотренных алгоритмов поиска в файле

Линейный поиск имеет сложность $O(n)$, где n - количество записей в файле. Это означает, что время поиска растет линейно с увеличением количества записей в файле.

Поиск с использованием таблицы имеет сложность $O(1)$, так как время поиска не зависит от количества записей в файле. Это происходит потому, что таблица хранит ссылки на записи в файле, и поиск ключа в таблице выполняется за постоянное время.

Вывод

В ходе данной практической работы я освоил основные навыки управления битовыми операциями (сдвиги, маски, побитовые сравнения, инверсии) и использование их в файловых сортировках, также смог реализовать эффективные алгоритмы сортировок файлов на языке C++.

Список литературы

1. Бхаргава А. Грожаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб: Питер, 2017. – 288 с.
2. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.
3. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.
4. AlgoList – алгоритмы, методы, исходники [Электронный ресурс]. URL: <http://algotlist.manual.ru/> (дата обращения 15.03.2022).
5. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: <https://habr.com/ru/hub/algorithms/> (дата обращения 15.03.2022).
6. НОУ ИНТУИТ | Технопарк Mail.ru Group: Алгоритмы и структуры данных [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/3496/738/info> (дата обращения 15.03.2022).