



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

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных
технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2

по дисциплине

«Структуры и алгоритмы обработки данных»

Тема. Внешние структуры данных: текстовый и двоичный файлы.

Выполнил студент группы ИКБО-04-22

Основин А.И.

Принял старший преподаватель

Скворцова Л.А.

Москва 2023

СОДЕРЖАНИЕ

1	Задание 1.....	4
1.1	Постановка задачи	4
1.1.1	Требования по выполнению	4
1.1.2	Формулировка индивидуального задания.....	5
1.2	Тестовый пример.....	5
1.2.1	Копия содержания текстового файла на примере 20 записей.	5
1.2.2	Вывод содержимого текстового файла на экран	6
1.2.3	Добавление новой записи в конец файла	6
1.2.4	Получение значения числа из файла по порядковому номеру	6
1.2.5	Определение количества чисел в файле	7
1.2.6	Дополнительная операция	7
1.3	Реализация приложения	8
1.3.1	Функционал приложения	8
1.3.2	Код основной программы	9
1.4	Результаты тестирования	12
2	ЗАДАНИЕ 2	16
2.1	Постановка задачи	16
2.1.1	Требования по выполнению	16
2.1.2	Формулировка индивидуального задания.....	17
2.2	Тестовый пример.....	17
2.3	Реализация приложения	18
2.3.1	Структура записи двоичного файла.....	18
2.3.2	Изображение структуры двоичного файла с записями фиксированной длины	19

2.3.3	Функционал приложения	20
2.3.4	Код основной программы	22
2.4	Результаты тестирования	28
3	Вывод.....	31

1 ЗАДАНИЕ 1

1.1 Постановка задачи

1.1.1 Требования по выполнению

Разработать программу управления текстовым файлом.

Требования по выполнению:

1. Создать текстовый файл средствами текстового редактора кодировки ASCII, содержащего десятичные числа по несколько чисел на строке. Количество чисел на разных строках может отличаться.
2. Реализация ввода-вывода на основе файловых потоков C++: ofstream, ifstream.
3. Имя физического файла вводится пользователем и передается в функции обработки через параметр.
4. При открытии файла выполнять контроль его существования и открытия.
5. Разработать функции для выполнения операций над текстовым файлом:
6. вывод содержимого текстового файла на экран;
7. добавление новой записи в конец файла;
8. прочесть значение числа, указав его порядковый номер в файле, и вернуть его значение при успешном выполнении и код завершения если значение числа превышает количество чисел в файле;
9. определение количества чисел в файле.
10. Разработать программу и выполнить тестирование всех функций. Программа должна содержать диалоговый интерфейс на основе текстового меню.

11. Контроль открытия и существования файла выполнить в основной программе перед вызовом функции. Перед закрытием файла, проверить отсутствие ошибок ввода и вывода (метод good).
12. Создать файл заголовка и перенести в него все отлаженные функции. Исключить функции из основной программы. Отладить приложение, подключив к нему модуль с функциями.
13. Разработать функции для реализации дополнительных операций, определенных вариантом и сохранить их в модуле с остальными функциями.
14. Выполнить тестирование приложения в полном объеме.

1.1.2 Формулировка индивидуального задания

Дополнительная операция №17. Создать новый файл из значений исходного, размещая в каждой строке три числа: исходное, количество цифр в числе, сумму цифр в числе.

1.2 Тестовый пример

1.2.1 Копия содержания текстового файла на примере 20 записей

Копия содержания текстового файла на примере 20 записей (по несколько записей на строке) построчно:

- 1) 10 12 31
- 2) 21 2
- 3) 32 15
- 4) 437 68
- 5) 52 4444
- 6) 66 2345
- 7) 77 555 3432
- 8) 85 88 643 222

1.2.2 Вывод содержимого текстового файла на экран

При данном содержании текстового файла в консоль должно быть выведено 20 чисел через пробел:

10 12 31 21 2 32 15 437 68 52 4444 66 2345 77 555 3432 85 88 643 222

В случае возникновения ошибки чтения из файла должно быть выведено сообщение: «Ошибка при выводе чисел из файла в консоль.»

1.2.3 Добавление новой записи в конец файла

После добавления числа «156» в текстовый файл, содержание текстового файла должно измениться следующим образом построчно:

- 1) 10 12 31
- 2) 21 2
- 3) 32 15
- 4) 437 68
- 5) 52 4444
- 6) 66 2345
- 7) 77 555 3432
- 8) 85 88 643 222
- 9) 156

При успешном добавлении нового числа в файл в консоль должно быть выведено сообщение «Число добавлено в файл.», в противном случае – «Ошибка при добавлении нового числа в файл.»

1.2.4 Получение значения числа из файла по порядковому номеру

При запросе 20 элемента исходного файла в консоль должно быть выведено число «222». В случае ввода индекса несуществующего элемента или при ошибке чтения из файла в консоль должно быть выведено сообщение «Ошибка при поиске числа по порядковому номеру в файле.»

1.2.5 Определение количества чисел в файле

При определении количества чисел для исходного файла в консоль должно быть выведено число 20. В случае ошибки чтения из файла в консоль должно быть выведено сообщение «Ошибка при подсчёте количества чисел в файле.»

1.2.6 Дополнительная операция

Поскольку в исходном файле 20 чисел, для каждого из которых должно быть выведено три новых числа, новый файл будет состоять из 20 строк – по три числа на каждой строке. Для исходного файла содержимое нового файла будет следующим построчно:

- 1) 10 2 1
- 2) 12 2 3
- 3) 31 2 4
- 4) 21 2 3
- 5) 2 1 2
- 6) 32 2 5
- 7) 15 2 6
- 8) 437 3 14
- 9) 68 2 14
- 10) 52 2 7
- 11) 4444 4 16
- 12) 66 2 12
- 13) 2345 4 14
- 14) 77 2 14
- 15) 555 3 15
- 16) 3432 4 12
- 17) 85 2 13
- 18) 88 2 16

19) 643 3 13

20) 222 3 6

При успешном создании нового файла в консоль будет выведено сообщение «Новый файл успешно создан.», в противном случае – «Ошибка при создании нового файла.»

1.3 Реализация приложения

1.3.1 Функционал приложения

В Листинге 1 представлены прототипы функций, реализующих операции задания.

Листинг 1 – file_methods.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

void print(istream& file);
void append(ostream& file, int new_number);
int get_number(istream& file, int index);
int count(istream& file);
void stats_file(istream& file, ostream& new_file);
```

Функция `print(istream& file)` реализует вывод чисел из файла, связанного с потоком `file` в консоль. Предусловие: `file` – ссылка на поток чтения из файла. Постусловие: в консоль выведены числа из файла; возвращаемое значение отсутствует.

Функция `append(ostream& file, int new_number)` реализует добавление нового числа в конец файла, связанного с потоком `file`. Предусловие: `file` – ссылка на поток записи в конец файла; `new_number` – число для записи, переменная целого типа. Постусловие: в файл добавлено число; возвращаемое значение отсутствует.

Функция `get_number(istream& file, int index)` реализует получение значения числа из файла, связанного с потоком `file`, по порядковому номеру. Предусловие: `file` – ссылка на поток чтения из файла; `number` – порядковый номер числа, переменная целого типа. Постусловие: возвращаемое значение –

целое число под порядковым номером из файла или вердикт о несуществовании числа с таким порядковым номером в файле.

Функция `count(istream& file)` реализует определение количества чисел в файле, связанном с потоком `file`. Предусловие: `file` – ссылка на поток чтения из файла. Постусловие: возвращаемое значение – целое число, определяющее количество чисел в файле.

Функция `stats_file(istream& file, ostream& new_file)` реализует индивидуальное задание – дополнительную операцию: создаёт новый файл в который записывает на отдельной строке каждое число, количество цифр в нем и сумму цифр исходного числа. Предусловие: `file` – ссылка на поток чтения из файла; `new_file` – ссылка на поток записи в файл. Постусловие: в новый файл записаны по три числа на каждой строке через пробел – исходное число, количество цифр в исходном числе и сумма цифр в исходном числе; возвращаемое значение отсутствует.

1.3.2 Код основной программы

В Листинге 2 представлена реализация функций модуля.

Листинг 2 – file_methods.cpp

```
#include "file_methods.h"

void print(istream& file) {
    int number;
    while (!file.eof()) {
        file >> number;
        cout << number << " ";
    }
}

void append(ostream& file, int new_number) {
    file << "\n" << new_number;
}

int get_number(istream& file, int index) {
    int num = INT32_MAX, i = 0;
    while (i <= index && !file.eof()) {
        file >> num;
        i++;
    }

    if (i != index) {
        return INT32_MAX;
    }
    return num;
}
```

```

}

int count(istream& file) {
    int average = 0, current;
    while (!file.eof()) {
        file >> current;
        ++average;
    }

    return average;
}

void stats_file(istream& file, ostream& new_file) {
    int number, sum;
    while (!file.eof()) {
        file >> number;
        new_file << number << " " << to_string(number).length() << " ";

        sum = 0;
        while (number > 0) {
            sum += number % 10;
            number /= 10;
        }

        new_file << sum << endl;
    }
}

```

В Листинге 3 представлена реализация диалогового интерфейса на основе текстового меню в функции main.

Листинг 3 – main.cpp

```

#include "file_methods.h"

int main()
{
    system("chcp 1251");
    string file, new_file;
    ifstream fin;
    ofstream fout;

    int action;
    while (true) {
        cout << "Выберите действие:" << endl;
        cout << "1. Напечатать числа из файла" << endl;
        cout << "2. Добавить новое число в конец файла" << endl;
        cout << "3. Найти число в файле по порядковому номеру" << endl;
        cout << "4. Определить количество чисел в файле" << endl;
        cout << "5. Создать новый файл из значений исходного, размещая в каждой строке три числа: исходное, количество цифр в числе, сумму цифр в числе" << endl;
        cout << "6. Выйти" << endl;

        cin >> action;
        if (action != 6) {
            cout << "Введите имя файла: ";
            cin >> file;
        }

        switch (action) {
            case 1:

```

```

        fin.open(file, ios::in);
        if (fin.is_open()) {
            print(fin);
            if (!fin) {
                cout << "Ошибка при выводе чисел из файла в консоль." <<
endl;
            }
            fin.close();
        }
        else {
            cout << "Файл не найден или не существует." << endl;
        }
        break;

    case 2:
        fout.open(file, ios::out | ios::app);
        if (fout.is_open()) {
            cout << "Введите число для добавления в файл: ";
            int number;
            cin >> number;
            append(fout, number);
            if (!fout) {
                cout << "Ошибка при добавлении нового числа в файл." <<
endl;
            }
            else {
                cout << "Число добавлено в файл." << endl;
            }
            fout.close();
        }
        else {
            cout << "Файл не найден или не существует." << endl;
        }
        break;

    case 3:
        fin.open(file, ios::in);
        if (fin.is_open()) {
            cout << "Введите порядковый номер числа: ";
            int number;
            cin >> number;
            number = get_number(fin, number);

            if (number != INT32_MAX) {
                cout << "Найденное число: " << number << endl;
            }
            else {
                cout << "Ошибка при поиске числа по порядковому номеру в
файле." << endl;
            }
            fin.close();
        }
        else
            cout << "Файл не найден или не существует." << endl;
        break;

    case 4:
        fin.open(file, ios::in);
        if (fin.is_open()) {
            cout << "Количество чисел в файле: " << count(fin) << endl;
            if (!fin) {
                cout << "Ошибка при подсчёте количества чисел в файле." <<
endl;
            }
        }
    }
}

```

```

        }
        fin.close();
    }
    else
        cout << "Файл не найден или не существует." << endl;
    break;

case 5: {
    cout << "Введите имя нового файла: ";
    cin >> new_file;
    fin.open(file, ios::in);
    if (fin.is_open()) {
        fout.open(new_file, ios::out);
        stats_file(fin, fout);
        if (!fin || !fout) {
            cout << "Ошибка при создании нового файла." << endl;
        }
        else {
            cout << "Новый файл успешно создан." << endl;
        }
        fin.close();
        fout.close();
    }
    else
        cout << "Файл не найден или не существует." << endl;
    break;
}
default:
    return 0;
}
}
}

```

1.4 Результаты тестирования

На Рисунке 1 представлено содержимое исходного файла.

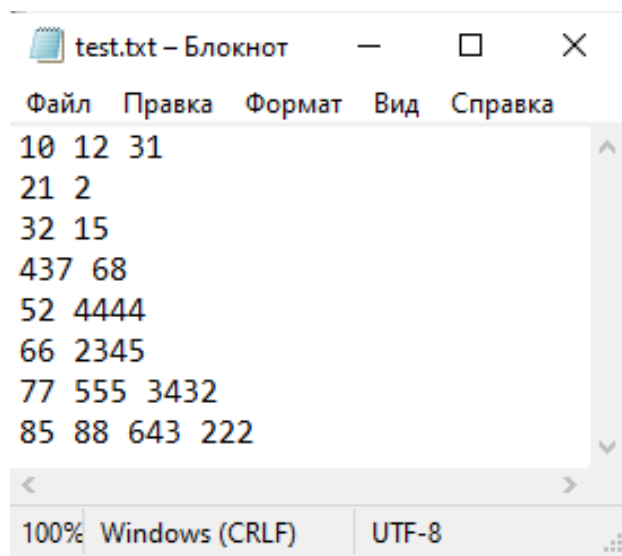
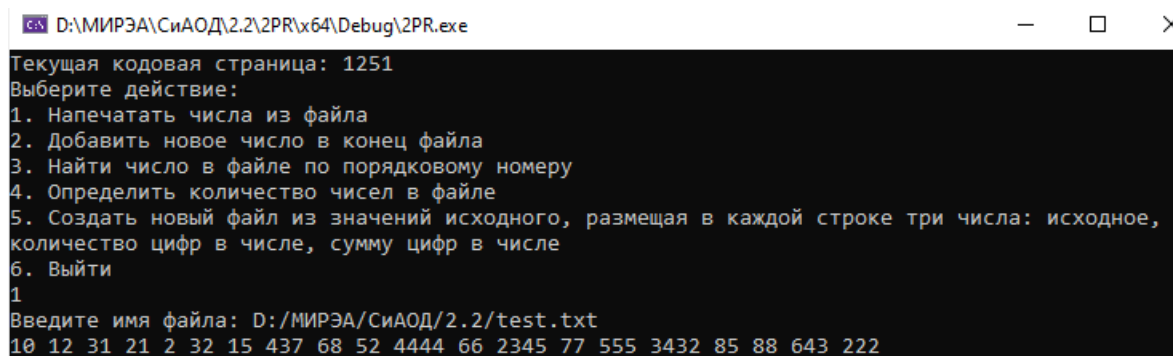


Рисунок 1 – Содержимое исходного файла

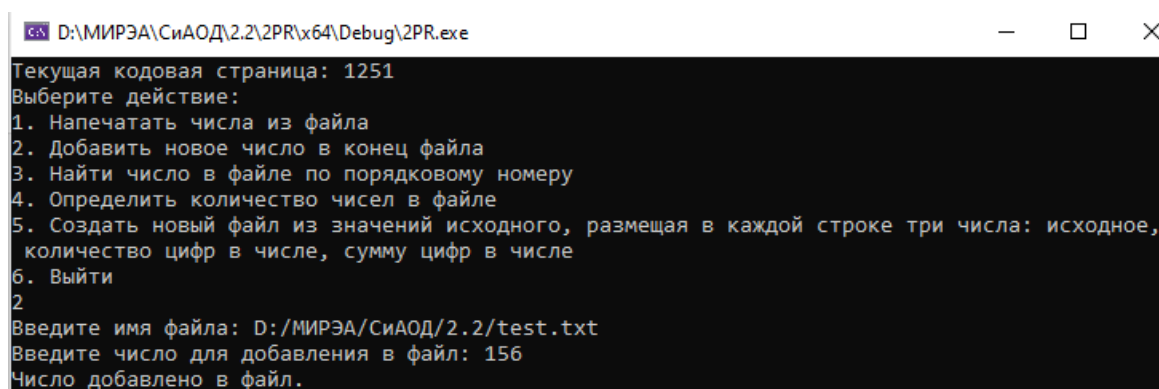
Результаты тестирования функции вывода чисел из файла в консоль представлены на Рисунке 2.



```
Текущая кодовая страница: 1251
Выберите действие:
1. Напечатать числа из файла
2. Добавить новое число в конец файла
3. Найти число в файле по порядковому номеру
4. Определить количество чисел в файле
5. Создать новый файл из значений исходного, размещая в каждой строке три числа: исходное,
   количество цифр в числе, сумму цифр в числе
6. Выйти
1
Введите имя файла: D:/МИРЭА/СиАОД/2.2/test.txt
10 12 31 21 2 32 15 437 68 52 4444 66 2345 77 555 3432 85 88 643 222
```

Рисунок 2 – Тестирование функции вывода содержимого файла в консоль

Результаты тестирования функции добавления числа в конец файла представлены на Рисунках 3 – 4.



```
Текущая кодовая страница: 1251
Выберите действие:
1. Напечатать числа из файла
2. Добавить новое число в конец файла
3. Найти число в файле по порядковому номеру
4. Определить количество чисел в файле
5. Создать новый файл из значений исходного, размещая в каждой строке три числа: исходное,
   количество цифр в числе, сумму цифр в числе
6. Выйти
2
Введите имя файла: D:/МИРЭА/СиАОД/2.2/test.txt
Введите число для добавления в файл: 156
Число добавлено в файл.
```

Рисунок 3 – Вывод вердикта операции добавления нового числа в конец файла в консоль

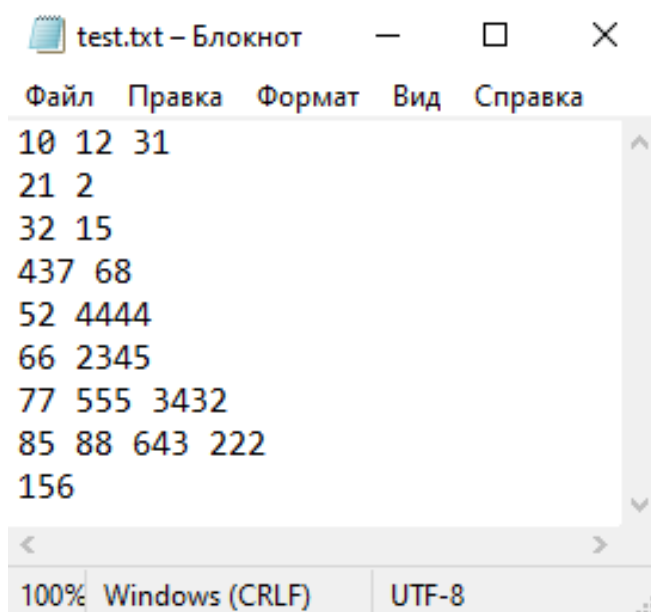
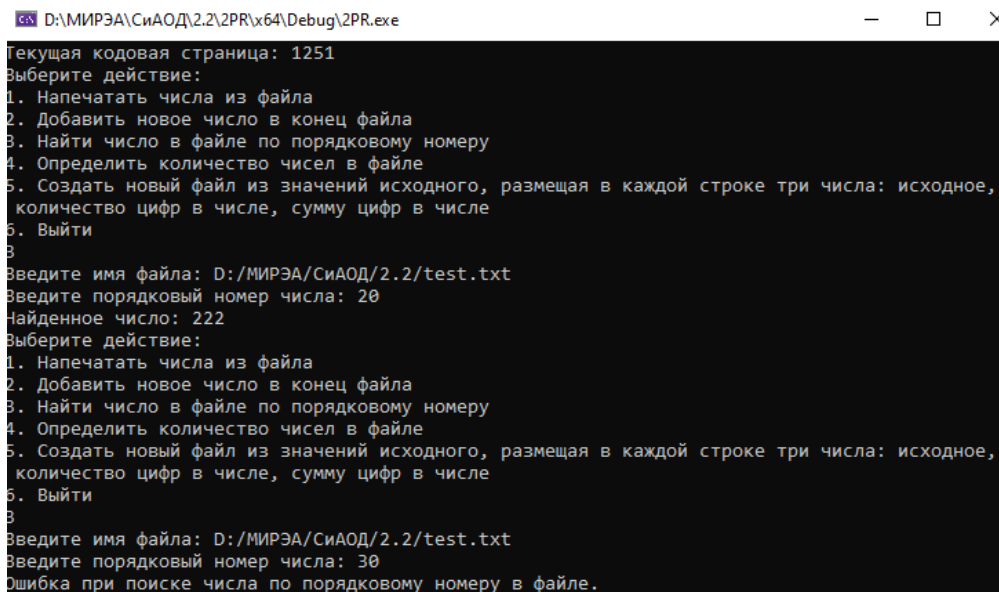


Рисунок 4 – Вид файла после операции добавления нового числа

Результаты тестирования функции поиска числа в файле по порядковому номеру представлены на Рисунке 5: сначала в консоль выводится 20 число из исходного файла; затем при попытке запросить из того же исходного файла 30

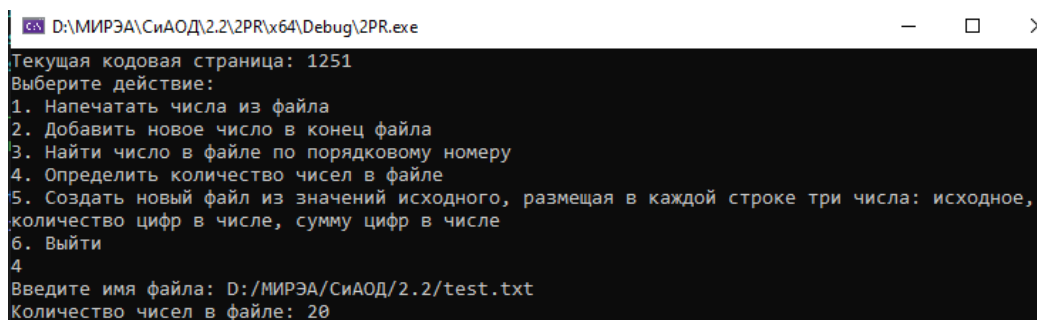
число в консоль выводится сообщение о несуществовании в исходном файле числа с таким индексом.



```
D:\МИРЭА\СиАОД\2.2\2PR\x64\Debug\2PR.exe
Текущая кодовая страница: 1251
Выберите действие:
1. Напечатать числа из файла
2. Добавить новое число в конец файла
3. Найти число в файле по порядковому номеру
4. Определить количество чисел в файле
5. Создать новый файл из значений исходного, размещая в каждой строке три числа: исходное, количество цифр в числе, сумму цифр в числе
6. Выйти
3
Введите имя файла: D:/МИРЭА/СиАОД/2.2/test.txt
Введите порядковый номер числа: 20
Найденное число: 222
Выберите действие:
1. Напечатать числа из файла
2. Добавить новое число в конец файла
3. Найти число в файле по порядковому номеру
4. Определить количество чисел в файле
5. Создать новый файл из значений исходного, размещая в каждой строке три числа: исходное, количество цифр в числе, сумму цифр в числе
6. Выйти
4
Введите имя файла: D:/МИРЭА/СиАОД/2.2/test.txt
Введите порядковый номер числа: 30
Ошибка при поиске числа по порядковому номеру в файле.
```

Рисунок 5 – Тестирование функции поиска числа в файле по порядковому номеру

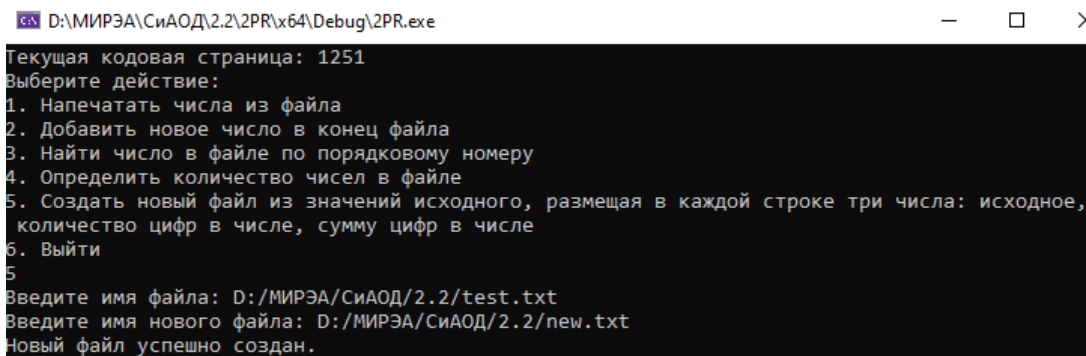
Результаты тестирования функции определения количества чисел в файле представлены на Рисунке 6.



```
D:\МИРЭА\СиАОД\2.2\2PR\x64\Debug\2PR.exe
Текущая кодовая страница: 1251
Выберите действие:
1. Напечатать числа из файла
2. Добавить новое число в конец файла
3. Найти число в файле по порядковому номеру
4. Определить количество чисел в файле
5. Создать новый файл из значений исходного, размещая в каждой строке три числа: исходное, количество цифр в числе, сумму цифр в числе
6. Выйти
4
Введите имя файла: D:/МИРЭА/СиАОД/2.2/test.txt
Количество чисел в файле: 20
```

Рисунок 6 – Тестирование функции определения количества чисел в файле

Результаты тестирования функции для создания нового файла на основе чисел из старого представлены на Рисунках 7 – 8.



```
D:\МИРЭА\СиАОД\2.2\2PR\x64\Debug\2PR.exe
Текущая кодовая страница: 1251
Выберите действие:
1. Напечатать числа из файла
2. Добавить новое число в конец файла
3. Найти число в файле по порядковому номеру
4. Определить количество чисел в файле
5. Создать новый файл из значений исходного, размещая в каждой строке три числа: исходное, количество цифр в числе, сумму цифр в числе
6. Выйти
5
Введите имя файла: D:/МИРЭА/СиАОД/2.2/test.txt
Введите имя нового файла: D:/МИРЭА/СиАОД/2.2/new.txt
Новый файл успешно создан.
```

Рисунок 7 – Вывод вердикта операции создания нового файла на основе чисел из старого в консоль

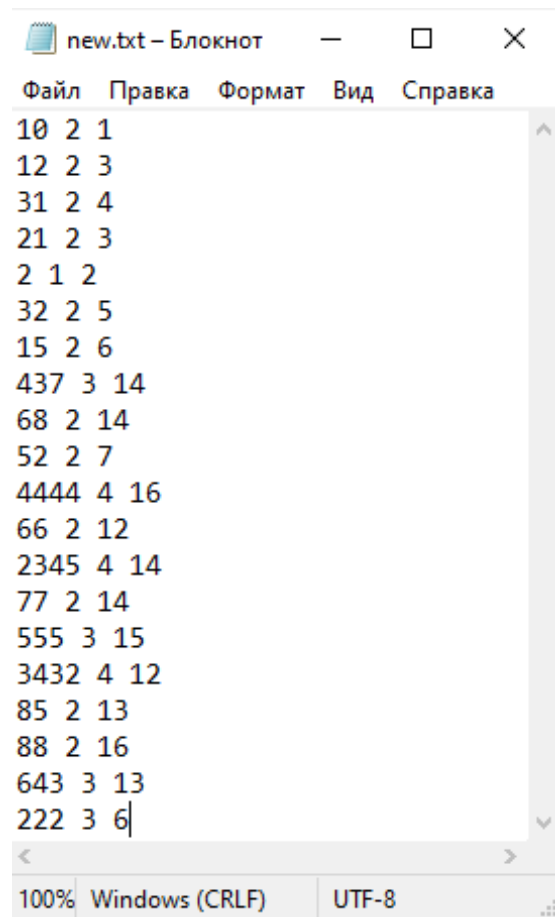


Рисунок 8 – Вид нового файла, созданного на основе чисел исходного

2 ЗАДАНИЕ 2

2.1 Постановка задачи

2.1.1 Требования по выполнению

Разработать программу управления двоичными файлами с записями фиксированной длины.

Общие требования: файл состоит из записей определенной структуры, согласно варианту. Записи имеют ключ, который уникален в пределах файла.

Требования к подготовке и выполнению задания.

1. Разработать структуру записи двоичного файла согласно варианту задания.
2. Подготовить тестовые данные в текстовом файле с кодировкой ASCII, в соответствии со структурой записи варианта. При открытии файла выполнить контроль его существования и открытия.
3. Имя файла вводит пользователь.
4. При открытии файла обеспечить контроль существования и открытия файла.
5. При применении механизма прямого доступа к записи файла выполнить контроль присутствия записи с заданным номером в файле.
6. Разработать функции для выполнения операций:
 - преобразование тестовых данных из текстового файла в двоичный файл;
 - сохранение данных двоичного файла в текстовом, так, чтобы используя их можно было восстановить двоичный файл;
 - вывод всех записей двоичного файла;
 - доступ к записи по ее порядковому номеру в файле, используя механизм прямого доступа к записи в двоичном файле;

- удаление записи с заданным значением ключа, выполнить путем замены на последнюю запись;
 - манипулирование записями в двоичном файле согласно дополнительным операциям, определенным в варианте;
7. Сохраните функции в новом модуле.
 8. Разработать приложение, демонстрирующее выполнение всех операций, подключив к нему модуль с функциями.
 9. Выполнить тестирование приложения, продемонстрировав выполнение всех операций.

2.1.2 Формулировка индивидуального задания

Индивидуальный вариант №17.

Структура записи: частотный словарь: слово, количество вхождений в текст.

Дополнительная операция:

1. Определить, какое слово встречалось чаще всего в тексте.
2. Добавить в файл новую запись по слову
3. Обновить количество вхождений некоторых слов, увеличив их количество на 1.

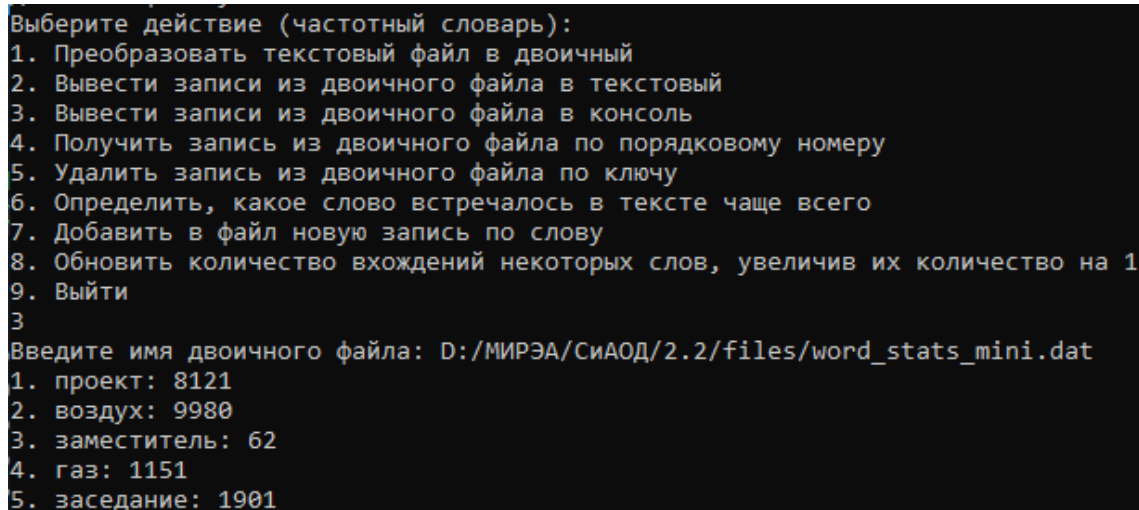
2.2 Тестовый пример

Копия содержания текстового файла на примере 5 записей для преобразования в двоичный файл построчно:

1. проект
2. 8121
3. воздух
4. 9980
5. заместитель
6. 62
7. газ
8. 1151
9. заседание

10.1901

Содержание двоичного файла (результат операции вывода двоичного файла на экран), полученного из данных текстового файла представлено на Рисунке 9.



```
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
3
Введите имя двоичного файла: D:/МИРЭА/СиАОД/2.2/files/word_stats_mini.dat
1. проект: 8121
2. воздух: 9980
3. заместитель: 62
4. газ: 1151
5. заседание: 1901
```

Рисунок 9 – Содержание двоичного файла, полученного на основе данных из текстового файла

2.3 Реализация приложения

2.3.1 Структура записи двоичного файла

Структура записи представлена в Листинге 4.

Листинг 4 – Структура записи

```
struct word {
    char name[30];
    unsigned int count;
};
```

На первый взгляд, размер одной записи данной структуры равен сумме её полей, то есть 30 Байт для массива типа `char` и 4 Байта для переменной типа `int` – 34 Байта в сумме, однако это не так. Язык программирования C++ для обеспечения скорости доступа к элементам памяти и безопасности при работе с памятью применяет Байты заполнения: неиспользуемые байты памяти, которые вставляются между переменными в структуре, чтобы гарантировать, что каждая переменная начинается с адреса памяти, выровненного с ее типом данных. В данном случае размер массива `name` должен быть кратен 4 (размеру переменной типа `int`). По этой причине в структуре происходит выравнивание:

между массивом name и переменной count есть два неиспользуемых Бита, вследствие чего одна запись данной структуры весит 36 Байтов.

В целях экономии места можно сократить размер массива char на два символа, тогда вес каждой записи такой структуры станет меньше на целых 4 Бита. Также можно увеличить размер массива char на два элемента при этом не увеличивая затраты памяти. Однако, размер структуры принят равным 36 Байтам при любом случае. Система также показывает размер 36 Байт, как показано на Рисунке 10.

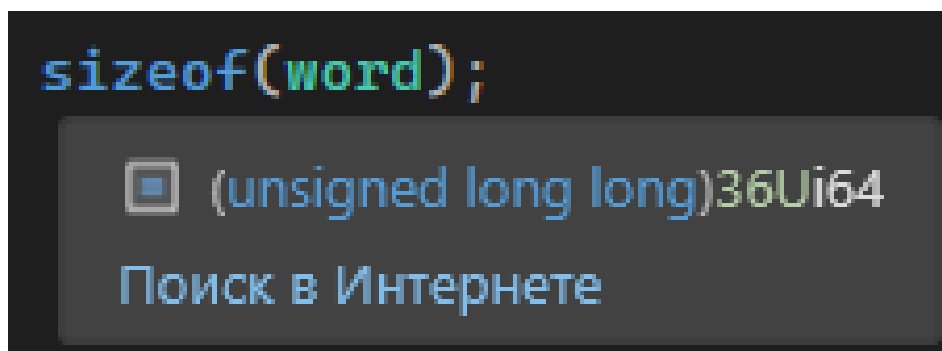


Рисунок 10 – Размер одного экземпляра структуры word

2.3.2 Изображение структуры двоичного файла с записями фиксированной длины

Структура двоичного файла, состоящего из записей типа word, представлена на Рисунке 11.

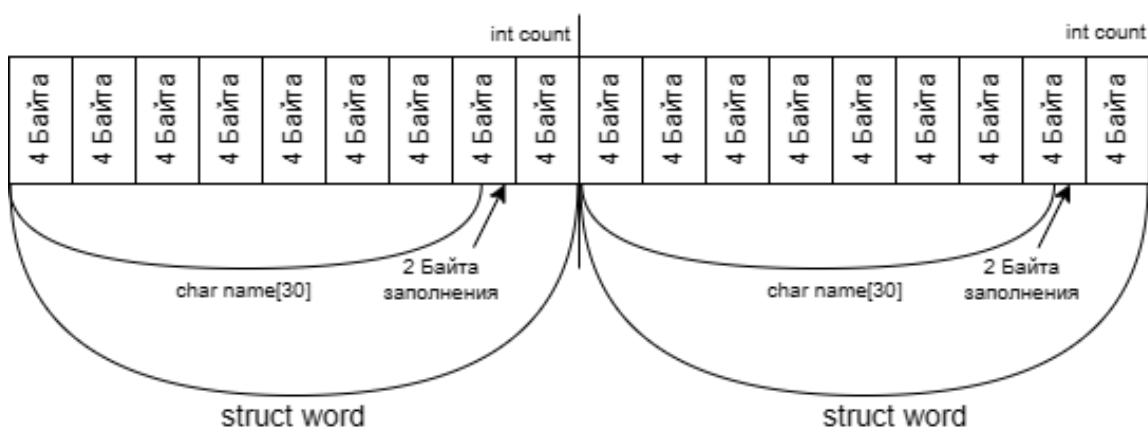


Рисунок 11 – Структура двоичного файла, состоящего из записей типа word

2.3.3 Функционал приложения

В Листинге 5 представлены прототипы функций, реализующих операции задания.

Листинг 5 – bin_file_methods.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <io.h>
#include <fcntl.h>
using namespace std;

struct word {
    char name[30];
    unsigned int count;
};

void text2bin(istream& text_file, ostream& bin_file);
void bin2text(istream& bin_file, ostream& text_file);
void print_bin(istream& file);
word get_word(istream& file, int index);
bool delete_word(fstream& file, string key, string file_path);
word get_widest(istream& file);
void add_word(ostream& file, string new_word);
void count_words(fstream& file, vector<string> words);
```

Функция `text2bin(istream& text_file, ostream& bin_file)` реализует запись данных из файла, связанного с потоком `text_file`, в двоичный файл, связанный с потоком `bin_file`. Предусловие: `text_file` — ссылка на поток чтения из текстового файла; `bin_file` — ссылка на поток записи в бинарный файл. Постусловие: в двоичном файле записаны данные из текстового файла; возвращаемое значение отсутствует.

Функция `bin2text(istream& bin_file, ostream& text_file)` реализует запись данных из файла, связанного с потоком `bin_file`, в текстовый файл, связанный с потоком `text_file`. Предусловие: `bin_file` — ссылка на поток чтения из двоичного файла; `text_file` — ссылка на поток записи в текстовый файл. Постусловие: в текстовом файле записаны данные из двоичного файла; возвращаемое значение отсутствует.

Функция `print_bin(istream& file)` реализует вывод в консоль записей из двоичного файла, связанного с потоком `bin_file`. Предусловие: `file` — ссылка

на поток чтения из двоичного файла. Постусловие: в консоль выведены данные из двоичного файла; возвращаемое значение отсутствует.

Функция `get_word(istream& file, int index)` реализует получение записи с порядковым номером `index` из двоичного файла, связанного с потоком `file`. Предусловие: `file` — ссылка на поток чтения из двоичного файла, `index` — целочисленная переменная, порядковый номер. Постусловие: возвращаемое значение — объект структуры `word`.

Функция `delete_word(fstream& file, string key, string file_path)` реализует удаление записи о слове `key` из двоичного файла, связанного с потоком `file`. Предусловие: `file` — ссылка на файловый поток, связанный с двоичным файлом; `key` — строка, хранит искомое слово; `file_path` — строка, хранит путь до двоичного файла. Постусловие: из двоичного файла удалена запись с нужным ключом; возвращаемое значение — булева переменная, показывает успешность удаления записи.

Функция `get_widest(istream& file)` реализует получение самого часто встречающегося в тексте слова. Предусловие: `file` — ссылка на поток чтения из двоичного файла. Постусловие: возвращаемое значение — объект структуры `word`.

Функция `add_word(ostream& file, string new_word)` реализует добавление новой записи о слове в конец двоичного файла. Предусловие: `file` — ссылка на поток записи в двоичный файл, `new_word` — строковая переменная, слово, которое необходимо записать. Постусловие: в конец двоичного файла записано новое слово; возвращаемое значение отсутствует.

Функция `count_words(fstream& file, vector<string> words)` реализует обновление количества вхождений некоторых слов в текст. Предусловие: `file` — ссылка на файловый поток, связанный с двоичным файлом, `words` — контейнер типа `vector` элементов строкового типа, слова, для которых необходимо обновить количество вхождений. Постусловие: в двоичном файле обновлено количество вхождений переданных слов в текст; возвращаемое значение отсутствует.

2.3.4 Код основной программы

В Листинге 6 представлена реализация функций модуля.

Листинг 6 – *bin_file_methods.cpp*

```
#include "bin_file_methods.h"

void text2bin(istream& text_file, ostream& bin_file) {
    while (!text_file.eof()) {
        word current;
        int i = 0;

        do {
            text_file.get(current.name[i]);
        } while (current.name[i++] != '\n');
        current.name[i - 1] = '\0';

        text_file >> current.count;
        text_file.get();

        bin_file.write((char*)&current, sizeof(word));
    }
}

void bin2text(istream& bin_file, ostream& text_file) {
    word current;
    bin_file.read((char*)&current, sizeof(word));
    while (!bin_file.eof()) {
        text_file << current.name << "\n" << current.count;
        bin_file.read((char*)&current, sizeof(word));

        if (!bin_file.eof()) {
            text_file << "\n";
        }
    }
}

void print_bin(istream& file) {
    word current;
    int n = 1;
    file.read((char*)&current, sizeof(word));
    while (!file.eof()) {
        cout << n++ << ". " << current.name << ": " << current.count << endl;
        file.read((char*)&current, sizeof(word));
    }
}

word get_word(istream& file, int index) {
    word current;
    file.seekg((index - 1) * sizeof(word), ios::beg);
    file.read((char*)&current, sizeof(word));

    if (file.bad() || file.fail()) {
        current.name[0] = '\0';
    }
    return current;
}

bool delete_word(fstream& file, string key, string file_path) {
    word last, current;
    bool status = false;
```

```

        file.seekg(-(int)sizeof(word), ios::end);
        file.read((char*)&last, sizeof(word));

        // Record couldn't be deleted if pointer would still be in that part of
file
        if (last.name != key) {
            file.seekg(ios::beg);
            file.read((char*)&current, sizeof(word));
            while (!file.eof()) {
                if (current.name == key) {
                    file.seekp(-(int)sizeof(word), ios::cur);
                    file.write((char*)&last, sizeof(word));
                    status = true;
                    break;
                }
                file.read((char*)&current, sizeof(word));
            }
        }
        else {
            status = true;
        }

        if (status) {
            int fh;
            if (_open_s(&fh, file_path.c_str(), _O_RDWR, _SH_DENYNO, _S_IREAD |
_S_IWRITE) == 0) {
                if (!(_chsize(fh, (_filelength(fh) - sizeof(word))) == 0)) {
                    status = false;
                }
                _close(fh);
            }
            /*
            * FOR UNIX: CLOSE FILE, THAN RESIZE IT
            #include <filesystem>
            auto p = filesystem::path(file_path);
            filesystem::resize_file(p, (size - 1) * sizeof(word));
            */
        }
        return status;
    }

word get_widest(istream& file) {
    word current, best;
    best.count = 0;

    file.read((char*)&current, sizeof(word));
    while (!file.eof()) {
        if (current.count > best.count) {
            best.count = current.count;
            strcpy_s(best.name, current.name);
        }
        file.read((char*)&current, sizeof(word));
    }

    return best;
}

void add_word(ostream& file, string new_word) {
    word current;
    int pos = new_word.size();
    strncpy(current.name, new_word.c_str(), pos);
    current.name[pos] = '\\0';
}

```

```

        current.count = 0;
        file.write((char*)&current, sizeof(word));
    }
    void count_words(fstream& file, vector<string> words) {
        word current;
        file.read((char*)&current, sizeof(word));
        while (!file.eof()) {
            for (string w : words) {
                if (w == current.name) {
                    ++current.count;
                    file.seekp(-(int)sizeof(word), ios::cur);
                    file.write((char*)&current, sizeof(word));
                    file.seekg(sizeof(word), ios::cur);
                    cout << current.name << " " << current.count << endl;
                    break;
                }
            }
            file.read((char*)&current, sizeof(word));
        }
    }
}

```

В Листинге 7 представлена реализация диалогового интерфейса на основе текстового меню в функции main.

Листинг 7 – main.cpp

```

#include "bin_file_methods.h"

int main() {
    system("chcp 1251");
    string text_file, bin_file;
    ifstream fin;
    ofstream fout;
    fstream fios;

    int action;
    while (true) {
        cout << "Выберите действие (частотный словарь):" << endl;
        cout << "1. Преобразовать текстовый файл в двоичный" << endl;
        cout << "2. Вывести записи из двоичного файла в текстовый" << endl;
        cout << "3. Вывести записи из двоичного файла в консоль" << endl;
        cout << "4. Получить запись из двоичного файла по порядковому номеру"
        << endl;
        cout << "5. Удалить запись из двоичного файла по ключу" << endl;
        cout << "6. Определить, какое слово встречалось в тексте чаще всего"
        << endl;
        cout << "7. Добавить в файл новую запись по слову" << endl;
        cout << "8. Обновить количество вхождений некоторых слов, увеличив их
        количество на 1" << endl;
        cout << "9. Выйти" << endl;

        cin >> action;
        switch (action) {
            case 1: {
                cout << "Введите имя текстового файла: ";
                cin >> text_file;
                cout << "Введите имя двоичного файла: ";
                cin >> bin_file;
                fin.open(text_file, ios::in);
                if (fin.is_open()) {
                    fout.open(bin_file, ios::binary | ios::out);
                    text2bin(fin, fout);
                    if (!fout) {

```



```

        cout << "Ошибка при записи в файл." << endl;
        return 1;
    }
    cout << "Двоичный файл успешно записан." << endl;
    fin.close();
    fout.close();
}
else {
    cout << "Файл не найден или не существует." << endl;
}
break;
}

case 2: {
    cout << "Введите имя двоичного файла: ";
    cin >> bin_file;
    cout << "Введите имя текстового файла: ";
    cin >> text_file;
    fin.open(bin_file, ios::binary | ios::in);
    if (fin.is_open()) {
        fout.open(text_file, ios::out);
        bin2text(fin, fout);
        if (fout.bad() || fout.fail()) {
            cout << "Ошибка при записи в файл." << endl;
            return 1;
        }
        cout << "Текстовый файл успешно записан." << endl;
        fin.close();
        fout.close();
    }
    else {
        cout << "Файл не найден или не существует." << endl;
    }
    break;
}

case 3: {
    cout << "Введите имя двоичного файла: ";
    cin >> bin_file;
    fin.open(bin_file, ios::binary | ios::in);
    if (fin.is_open()) {
        print_bin(fin);
        if (fin.bad()) {
            cout << "Ошибка при чтении файла." << endl;
            return 1;
        }
        fin.close();
    }
    else {
        cout << "Файл не найден или не существует." << endl;
    }
    break;
}

case 4: {
    cout << "Введите имя двоичного файла: ";
    cin >> bin_file;
    cout << "Введите порядковый номер записи: ";
    int number;
    cin >> number;

    fin.open(bin_file, ios::binary | ios::in);
    if (fin.is_open()) {

```

```

        word found = get_word(fin, number);
        if (found.name[0] == '\0') {
            cout << "Введённый порядковый номер превышает количество
записей в файле." << endl;
        }
        else {
            cout << number << "-ое слово '" << found.name << "'
встречено в тексте " << found.count << " раз(-а)." << endl;
        }

        if (fin.bad() || fin.fail()) {
            cout << "Ошибка при чтении файла." << endl;
            return 1;
        }
        fin.close();
    }
    else {
        cout << "Файл не найден или не существует." << endl;
    }
    break;
}

case 5: {
    cout << "Введите имя двоичного файла: ";
    cin >> bin_file;
    cout << "Введите ключ (слово): ";
    string key;
    cin >> key;

    fios.open(bin_file, ios::binary | ios::in | ios::out);
    if (fios.is_open()) {
        bool status = delete_word(fios, key, bin_file);
        if (fios.bad()) {
            cout << "Ошибка при чтении файла." << endl;
            return 1;
        }

        if (!status) {
            cout << "Не удалось удалить запись по ключу." << endl;
        }
        else {
            cout << "Запись удалена." << endl;
        }
        fios.close();
    }
    else {
        cout << "Файл не найден или не существует." << endl;
    }
    break;
}

case 6: {
    cout << "Введите имя двоичного файла: ";
    cin >> bin_file;
    fin.open(bin_file, ios::binary | ios::in);
    if (fin.is_open()) {
        word best = get_widest(fin);
        if (fin.bad()) {
            cout << "Ошибка при чтении файла." << endl;
            return 1;
        }

        cout << "Слово '" << best.name << "' встречалось в тексте
чаще всего: " << best.count << " раз(-а)." << endl;
    }
}

```

```

        fin.close();
    }
    else {
        cout << "Файл не найден или не существует." << endl;
    }
    break;
}

case 7: {
    cout << "Введите имя двоичного файла: ";
    cin >> bin_file;
    cout << "Введите новое слово для добавления в частотный словарь: ";

    string new_word;
    cin >> new_word;

    fout.open(bin_file, ios::binary | ios::out | ios::app);
    if (fout.is_open()) {
        add_word(fout, new_word);
        if (fout.bad() || fout.fail()) {
            cout << "Ошибка при чтении файла." << endl;
            return 1;
        }

        cout << "Слово " << new_word << " успешно записано в конец
бинарного файла." << endl;
        fout.close();
    }
    else {
        cout << "Файл не найден или не существует." << endl;
    }
    break;
}

case 8: {
    cout << "Введите имя двоичного файла: ";
    cin >> bin_file;
    cout << "Вводите слова, которые необходимо посчитать (чтобы
прекратить ввод введите 'end'): " << endl;
    string new_word = "";
    vector <string> words;
    while (new_word != "end") {
        cin >> new_word;
        words.push_back(new_word);
    }
    words.pop_back();

    fios.open(bin_file, ios::binary | ios::in | ios::out);
    if (fios.is_open()) {
        count_words(fios, words);
        if (fios.bad()) {
            cout << "Ошибка при чтении файла или записи в него." <<
endl;

            return 1;
        }

        cout << "Количество вхождений слов успешно обновлено в
бинарном файле. " << endl;
        fout.close();
    }
    else {
        cout << "Файл не найден или не существует." << endl;
    }
}

```

```

        break;
    }

    default: {
        return 0;
    }
}
}
}

```

2.4 Результаты тестирования

Входные данные представлены на Рисунке 9. На Рисунке 12 представлено тестирование функции получения записи из двоичного файла по порядковому номеру.

```

C:\ D:\МИРЭА\СиАОД\2.2\2PR_bin\x64\Debug\2PR_bin.exe
Текущая кодовая страница: 1251
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
4
Введите имя двоичного файла: D:\МИРЭА\СиАОД\2.2\files\word_stats_mini.dat
Введите порядковый номер записи: 3
3-ое слово 'заместитель' встречено в тексте 62 раз(-а).

```

Рисунок 12 – Тестирование функции получения записи из двоичного файла по порядковому номеру

На Рисунке 13 представлено тестирование функции удаления записи из двоичного файла по ключу.

```

C:\ D:\МИРЭА\СиАОД\2.2\2PR_bin\x64\Debug\2PR_bin.exe
Текущая кодовая страница: 1251
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
5
Введите имя двоичного файла: D:\МИРЭА\СиАОД\2.2\files\word_stats_mini.dat
Введите ключ (слово): воздух
Запись удалена.

```

Рисунок 13 – Тестирование функции удаления записи из двоичного файла по ключу

На Рисунке 14 представлено содержимое двоичного файла после удаления записи.

```
D:\МИРЭА\СиАОД\2.2\2PR_bin\x64\Debug\2PR_bin.exe
Текущая кодовая страница: 1251
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
3
Введите имя двоичного файла: D:/МИРЭА/СиАОД/2.2/files/word_stats_mini.dat
1. проект: 8121
2. заседание: 1901
3. заместитель: 62
4. газ: 1151
```

Рисунок 14 – Содержимое двоичного файла после удаления записи

На Рисунке 15 представлено тестирование функции определения наиболее часто встречаемого слова.

```
D:\МИРЭА\СиАОД\2.2\2PR_bin\x64\Debug\2PR_bin.exe
Текущая кодовая страница: 1251
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
6
Введите имя двоичного файла: D:/МИРЭА/СиАОД/2.2/files/word_stats_mini.dat
Слово 'проект' встречалось в тексте чаще всего: 8121 раз(-a).
```

Рисунок 15 – Тестирование функции определения наиболее часто встречаемого слова

На Рисунке 16 представлено тестирование функции добавления в файл новой записи по слову.

```
D:\МИРЭА\СиАОД\2.2\2PR_bin\x64\Debug\2PR_bin.exe
Текущая кодовая страница: 1251
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
7
Введите имя двоичного файла: D:/МИРЭА/СиАОД/2.2/files/word_stats_mini.dat
Введите новое слово для добавления в частотный словарь: красота
Слово красота успешно записано в конец бинарного файла.
```

Рисунок 16 – Тестирование функции добавления в файл новой записи по слову

На Рисунке 17 представлено содержимое двоичного файла после добавления в файл новой записи по слову.

```
cs D:\МИРЭА\СиАОД\2.2\2PR_bin\x64\Debug\2PR_bin.exe
Текущая кодовая страница: 1251
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
3
Введите имя двоичного файла: D:/МИРЭА/СиАОД/2.2/files/word_stats_mini.dat
1. проект: 8121
2. заседание: 1901
3. заместитель: 62
4. газ: 1151
5. красота: 0
```

Рисунок 17 – Содержимое двоичного файла после добавления в файл новой записи по слову

На Рисунке 18 представлено тестирование функции обновления количества вхождений некоторых слов.

```
cs D:\МИРЭА\СиАОД\2.2\2PR_bin\x64\Debug\2PR_bin.exe
Текущая кодовая страница: 1251
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
8
Введите имя двоичного файла: D:/МИРЭА/СиАОД/2.2/files/word_stats_mini.dat
Вводите слова, которые необходимо посчитать (чтобы прекратить ввод введите 'end'):
красота
газ проект
заместитель
end
проект 8122
заместитель 63
красота 1
```

Рисунок 18 – Тестирование функции обновления количества вхождений некоторых слов

На Рисунке 19 представлено содержимое двоичного файла после обновления количества вхождений некоторых слов.

```
cs D:\МИРЭА\СиАОД\2.2\2PR_bin\x64\Debug\2PR_bin.exe
Текущая кодовая страница: 1251
Выберите действие (частотный словарь):
1. Преобразовать текстовый файл в двоичный
2. Вывести записи из двоичного файла в текстовый
3. Вывести записи из двоичного файла в консоль
4. Получить запись из двоичного файла по порядковому номеру
5. Удалить запись из двоичного файла по ключу
6. Определить, какое слово встречалось в тексте чаще всего
7. Добавить в файл новую запись по слову
8. Обновить количество вхождений некоторых слов, увеличив их количество на 1
9. Выйти
3
Введите имя двоичного файла: D:/МИРЭА/СиАОД/2.2/files/word_stats_mini.dat
1. проект: 8122
2. заседание: 1901
3. заместитель: 63
4. газ: 1151
5. красота: 1
Выберите действие (частотный словарь):
```

Рисунок 19 – Содержимое двоичного файла после обновления количества вхождений некоторых слов

3 ВЫВОД

В ходе выполнения данной практической работы были приобретены навыки работы с файловыми потоками `ifstream` и `ofstream` в C++ и навыки обработки текстовых и двоичных файлов.