

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего

образования

**«МИРЭА - Российский технологический университет»**

РТУ МИРЭА

Институт информационных технологий

Кафедра математического обеспечения и стандартизации информационных технологий

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2**

***«*Управление файлом*»***

по дисциплине

**«**Структуры и алгоритмы обработки данных**»**

Выполнил студент группы *ИКБО-03-22 Хохлинов Д.И.*

Принял *Сорокин А.В.*

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | « » 2023 г. |  |
|  |  |  |
| «Зачтено» | « » 2023 г. |  |

Москва 2023

**СОДЕРЖАНИЕ**

[ЗАДАНИЕ 1 3](#_Toc146577274)

[1.1 Формулировка и требования 3](#_Toc146577275)

[1.2 Тестовый пример 4](#_Toc146577276)

[1.3 Реализация приложения 5](#_Toc146577277)

[1.3.1 Функционал приложения 5](#_Toc146577278)

[1.3.2 Реализация приложения 6](#_Toc146577279)

[1.4 Результаты тестирования 11](#_Toc146577280)

[ЗАДАНИЕ 2 15](#_Toc146577281)

[2.1 Условие задания 15](#_Toc146577282)

[2.1.1 Формулировка 15](#_Toc146577283)

[2.1.2 Требования 15](#_Toc146577284)

[2.1.3 Формулировка задания варианта 16](#_Toc146577285)

[2.2 Тестовый пример 17](#_Toc146577286)

[2.3 Реализация приложения 19](#_Toc146577287)

[2.3.1 Структура используемой записи 19](#_Toc146577288)

[2.3.2 Структура двоичного файла 19](#_Toc146577289)

[2.2.3 Функционал приложения 20](#_Toc146577290)

[2.4 Код программы 24](#_Toc146577291)

[2.5 Тестирование 40](#_Toc146577292)

[ЗАКЛЮЧЕНИЕ 45](#_Toc146577293)

[ИНФОРМАЦИОННЫЙ ИСТОЧНИК 46](#_Toc146577294)

# **ЗАДАНИЕ 1**

## **1.1 Формулировка и требования**

Разработать программу, управления текстовым файлом.

Требования:

* Имя физического файла вводится пользователем и передается в функции обработки через параметр.
* При открытии файла выполнять контроль его существования и открытия. Примечание. При отладке программы можете имя физического файла определить через константу.
* Разработать функции для выполнения операций над текстовым файлом:

1) создание текстового файла средствами текстового редактора кодировки ASCII, содержащего десятичные числа по несколько чисел на строке;

2) вывод содержимого текстового файла;

3) добавление новой записи в конец файла;

4) прочитать значение числа, указав его порядковый номер в файле, и вернуть его значение;

5) определить количество чисел в файле.

* Разработать приложение и выполнить тестирование всех функций. Приложение должно содержать диалоговый интерфейс на основе текстового меню.
* Контроль открытия и существования файла выполнить в основной программе перед вызовом функции. Перед закрытием файла, проверить отсутствие ошибок ввода и вывода (метод good)
* Создать модуль и перенести в него все отлаженные функции. Исключить функции из приложения. Отладить приложение, подключив к нему модуль с функциями.
* Разработать функции для реализации дополнительных операций, определенных вариантом и сохранить их в модуле с остальными функциями.
* Выполнить тестирование приложения в полном объеме.

В соответствии с вариантом (25) дополнительная операция имеет следующий алгоритм: создать новый файл из значений исходного, переписав в него только простые числа, располагая каждое на отдельной строке текстового файла.

## **1.2 Тестовый пример**

Для тестирования программы использовался следующий файл (листинг 1):

Листинг 1. Содержание тестового файла

300 456 164 985

133 657

111 445 664 404 152 322

25 22 455

976

130

133 134 135

1024

-1023

307

311 313 317

## **1.3 Реализация приложения**

### **1.3.1 Функционал приложения**

Для выполнения задания использовались следующие методы модуля:

1. Метод открытия файла openFile. Принимает на вход строку path. Затем открывает файл, расположенный по пути path, в режиме чтения и записи. В случае неудачного открытия вызывает метод closeFile и возвращает «ложь», иначе возвращает «истину»;
2. Метод закрытия файла closeFile. Если ошибки во время работы с файлом не были обнаружены, то выводит соответствующее сообщение. Затем закрывает файл;
3. Метод повторного открытия файла reopenFile. Сначала закрывает файл, затем заново его открывает с помощью метода openFile;
4. Метод инициализации модуля init. Запрашивает у пользователя путь к рабочему файлу, затем пытается его открыть. Если инициализация успешна, возвращает «истину», иначе «ложь»;
5. Метод чтения содержимого showContent. Вызывает метод reopenFile, затем в случае успешного открытия считывает его содержимое и выводит его на экран;
6. Метод добавления записи append. Запрашивает у пользователя строку, которую необходимо записать в текущий файл, затем записывает ее. Если запись успешна, выводит соответствующее сообщение;
7. Метод поиска числа по индексу readByIndex. Принимает в качестве аргумента индекс index. Вызывает метод reopenFile; если файл не был успешно открыт, то возвращает соответствующий код ошибки. Если index-ое число существует в файле, то возвращает его; если нет, то возвращает код ошибки, сигнализирующий о том, что числа с таким индексом нет в файле;
8. Метод подсчета чисел в файле numberOfNumbers. Вызывает метод reopenFile; если файл не был успешно открыт, то возвращает соответствующий код ошибки. Возвращает количество чисел в файле;
9. Метод определения простоты числа isPrimal. Принимает в качестве аргумента целое число n. Если оно положительно и имеет только 2 делителя: 1 и n, то возвращает «истину», иначе – «ложь»;
10. Метод создания файла с простыми числами createFileWithPrimals. Вызывает метод reopenFile; если файл не был успешно открыт, то возвращает «ложь». Запрашивает у пользователя путь к файлу, куда будут сохранены простые числа; затем проходит по всем числам в исходном файле: если очередное число простое, то записывает его в новый файл с новой строки. В случае успешной записи выводит соответствующее сообщение и количество записанных чисел, а также возвращает «истину»; иначе возвращает «ложь».
11. Меню модуля process. Вызывает метод init; если он возвращает «ложь», то завершает работу. Затем в цикле предлагает пользователю одно из доступных действий. После выбора пользователем действия «выход» закрывает файл с помощью closeFile.

### **1.3.2 Реализация приложения**

Запишем программу на языке С++ (листинги 2-4):

Листинг 2 – Файл FileEditor.h

#ifndef FileEditorH

#define FileEditorH

#include <iostream>

#include <fstream>

#include <string>

#include <conio.h>

#include <cmath>

Продолжение листинга 2

using namespace std;

namespace FileEditor {

bool openFile(string path); //открыть файл

void closeFile();

bool reopenFile();

bool showContent(); //содержимое

void append(); //добавить запись в конец файла

int readByIndex(int index); //прочитать число по индексу

int numberOfNumbers(); //количество чисел

bool createFileWithPrimals();

void process(); //интерфейс для взаимодействия с файлом, который запускается из основной программы

bool init(); //инициализация

bool isPrimal(int n); //проверка простоты

}

#endif

Листинг 3 – Файл FileEditor.cpp

#ifndef FileEditorCpp

#define FileEditorCpp

#include "FileEditor.h"

int NO\_SUCH\_INDEX\_CODE = 2147483647;

int FILE\_READING\_ERROR\_CODE = 2147483646;

fstream file;

string \_path = "";

bool FileEditor::openFile(string path)

{

file = fstream(path.c\_str(), ios::in | ios::out);

if (!file)

{

closeFile();

return false;

}

return true;

}

bool FileEditor::reopenFile()

{

file.close();

return openFile(\_path);

}

void FileEditor::closeFile()

{

if (file.good())

{

Продолжение листинга 3

cout << "Ошибок ввода/вывода в файл не обнаружено." << endl;

}

file.close();

}

bool FileEditor::init()

{

string path;

cout << "Введите путь к файлу: ";

getline(cin, path);

\_path = path;

return openFile(path);

}

bool FileEditor::showContent()

{

if (!reopenFile())

{

cout << "Ошибка при чтении файла" << endl;

return false;

}

cout << "Содержимое файла:" << endl;

while (!file.eof())

{

string temp;

getline(file, temp);

cout << temp << endl;

}

return true;

}

void FileEditor::append()

{

file.seekg(0, ios::end);

cout << "Введите новую строку файла: ";

string temp;

getline(cin, temp);

file << "\n" << temp;

if (!file.fail())

cout << "Запись произведена успешно." << endl;

}

int FileEditor::readByIndex(int index)

{

if (index > numberOfNumbers())

{

cout << "Числа с индексом " << index << " не существует" << endl;

return NO\_SUCH\_INDEX\_CODE;

}

if (!reopenFile())

{

cout << "Ошибка при чтении файла" << endl;

return FILE\_READING\_ERROR\_CODE;

}

Продолжение листинга 3

int res = 0;

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

{

file >> res;

}

return res;

}

int FileEditor::numberOfNumbers()

{

if (!reopenFile())

{

cout << "Ошибка при чтении файла" << endl;

return FILE\_READING\_ERROR\_CODE;

}

int temp;

int amount = 0;

while (!file.eof())

{

file >> temp;

amount++;

}

return amount;

}

bool FileEditor::isPrimal(int n)

{

if (n <= 0) return false;

for (int i = 2; i\*i <= n; i++)

{

if ((n % i) == 0) return false;

}

return true;

}

bool FileEditor::createFileWithPrimals()

{

if (!reopenFile())

{

cout << "Ошибка при чтении файла" << endl;

return false;

}

int temp = 0;

int amount = 0;

string path;

cout << "Введите название файла, куда необходимо сохранить данные: ";

getline(cin, path);

fstream newfile = fstream(path.c\_str(), ios::out);

if (!newfile)

{

cout << "Не удалось создать файл" << endl;

}

else

{

Продолжение листинга 3

while (!file.eof())

{

file >> temp;

if (isPrimal(temp))

{

if (amount) newfile << "\n";

newfile << temp;

amount++;

}

}

cout << "Создан новый файл; записано простых чисел: " << amount << endl;

}

return true;

}

void FileEditor::process()

{

if (!init())

{

cout << "Не удалось открыть файл" << endl;

return;

}

char choose = ' ';

bool running = true;

int ind = 0;

int res;

while (running)

{

cout << "Выберите действие: " << endl <<

"[R] - вывести содержимое файла" << endl <<

"[A] - вывести количество чисел в файле" << endl <<

"[F] - найти число с заданным номером" << endl <<

"[W] - добавить запись в конец файла" << endl <<

"[N] - создать новый файл, записав в него только простые числа из исходного" << endl <<

"[Q] - выход" << endl;

choose = \_getch();

switch (tolower(choose))

{

case 'r':

if (!showContent()) running = false;

break;

case 'a':

if (!numberOfNumbers()) running = false;

else { cout << "Количество чисел в файле: " << numberOfNumbers() << endl; }

break;

case 'f':

cout << "Введите номер числа для поиска: ";

cin >> ind;

cin.ignore(32768, '\n');

res = readByIndex(ind);

if (res == NO\_SUCH\_INDEX\_CODE)

{

Продолжение листинга 3

}

else if (res == FILE\_READING\_ERROR\_CODE)

running = false;

else

{

cout << "Число с номером " << ind << ": " << res << endl;

}

break;

case 'w':

append();

break;

case 'n':

createFileWithPrimals();

break;

case 'q':

running = false;

break;

}

if (running)

{

system("pause");

system("cls");

}

}

closeFile();

}

#endif

Листинг 4 – Файл main.cpp

#include <iostream>

#include <locale.h>

#include "FileEditor.h"

int main()

{

setlocale(LC\_ALL, "ru");

FileEditor::process();

}

## **1.4 Результаты тестирования**

Результаты тестирования приведены на рисунках 1-7:

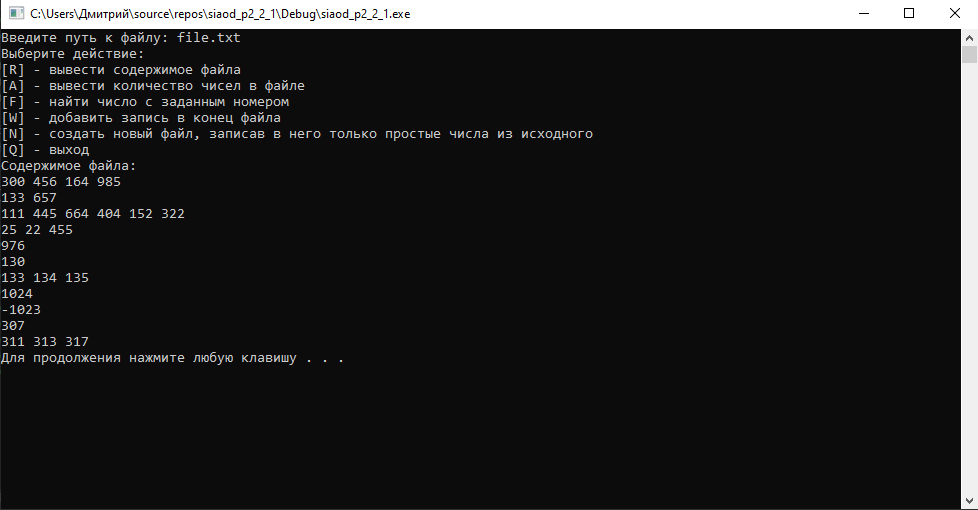


Рисунок 1 – Вывод содержимого файла

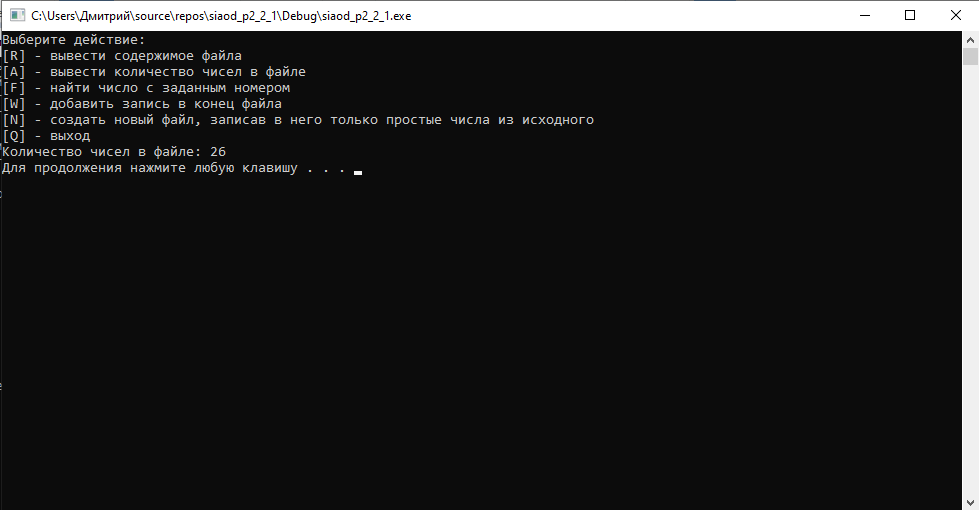


Рисунок 2 – Количество чисел в файле

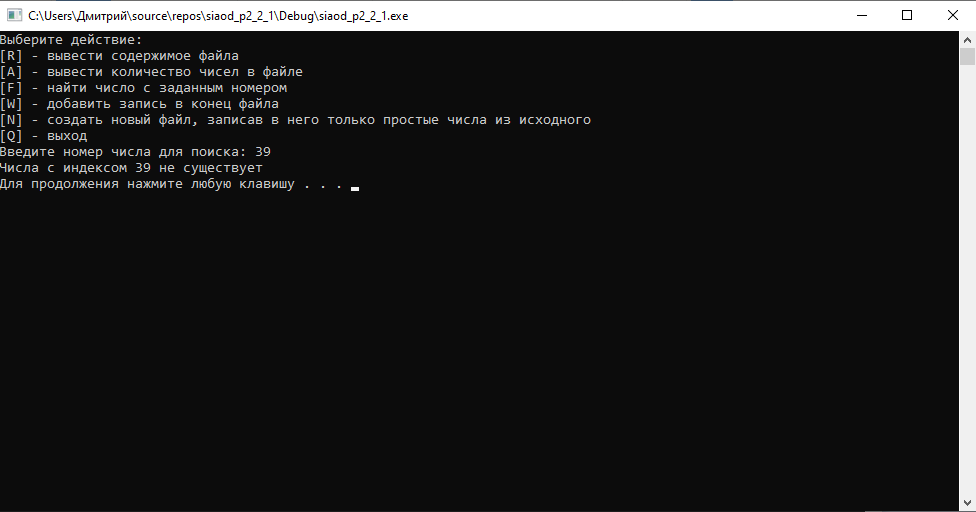


Рисунок 3 – Попытка поиска числа с несуществующим индексом

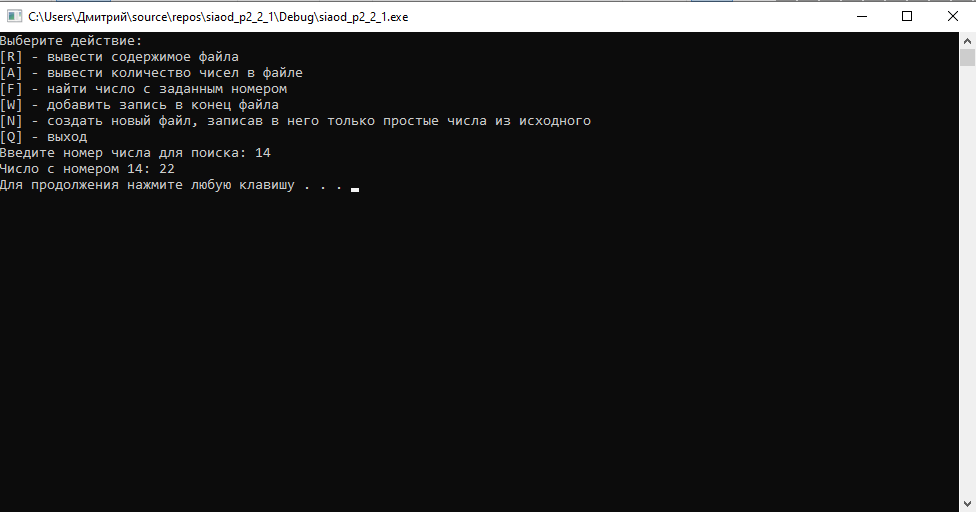


Рисунок 4 – Поиск числа с номером 14

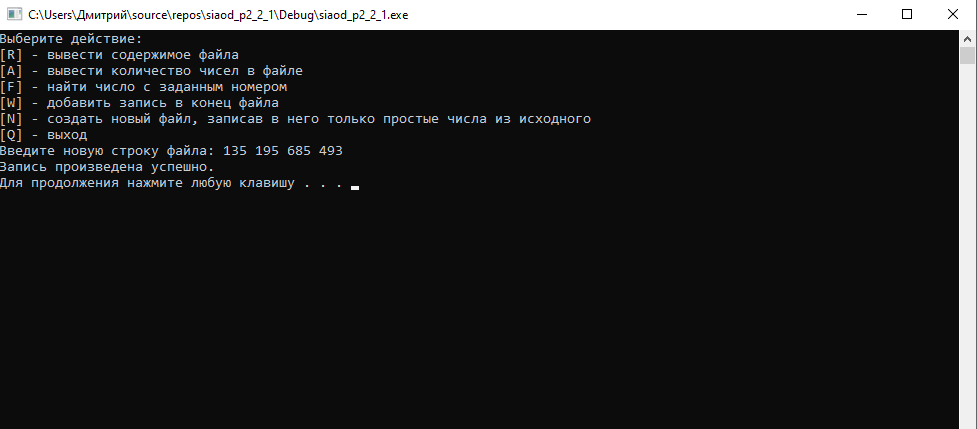


Рисунок 5 – Запись новой строки в файл

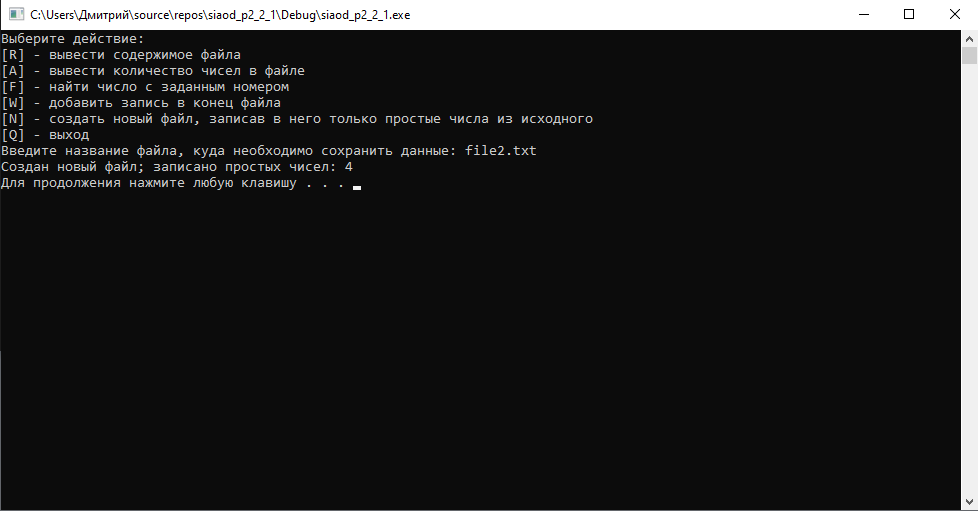


Рисунок 6 – Создание нового файла с простыми числами



Рисунок 7 – Содержимое нового файла

# **ЗАДАНИЕ 2**

## **2.1 Условие задания**

### **2.1.1 Формулировка**

Разработать программу управление двоичными файлами с записями фиксированной длины. Файл состоит из записей определенной структуры, согласно варианту. Записи имеют ключ, уникальный в пределах файла.

### **2.1.2 Требования**

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. Примечание. Реализация операций по чтению данных из файла будет проще, если значение для каждого поля записи размещать на отдельной строке текстового редактора.

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 25, а именно:

− сформировать список пациентов, которым поставлен диагноз с заданным кодом заболевания в новом двоичном файле;

− удалить сведения о пациенте с заданным ключом, сохраняя порядок следования остальных записей.

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.

### **2.1.3 Формулировка задания варианта**

Согласно варианту 25, используемая структура данных имеет следующие поля: номер полиса (ключ), фамилия, имя, отчество, код заболевания, дата установки диагноза, код врача.

Дополнительные операции:

− сформировать список пациентов, которым поставлен диагноз с заданным кодом заболевания в новом двоичном файле;

− удалить сведения о пациенте с заданным ключом, сохраняя порядок следования остальных записей.

## **2.2 Тестовый пример**

Для тестирования приложения использовался следующий текстовый файл (листинг 5) и полученный на его основе двоичный файл (рисунок 8):

Листинг 5 – Файл test.txt

123

Ivanov

Ivan

Ivanovich

X03

08.09.10

8800

124

Petrov

Ivan

Vasilyevich

M05

09.09.10

8800

125

Sidorov

Fedor

Ivanovich

X03

10.09.10

8800

126

Smirnov

Sergey

Sergeevich

A04

11.09.10

8800

127

Fedorov

Продолжение листинга 5

Valeriy

Ivanovich

X03

12.09.10

8800

128

Zaicev

Petr

Konstantinovich

X03

13.09.10

8815

129

Petrova

Marina

Ivanovna

D06

14.09.10

8876

130

Kuznetsov

Dmitriy

Anatolievich

E01

15.09.10

2646

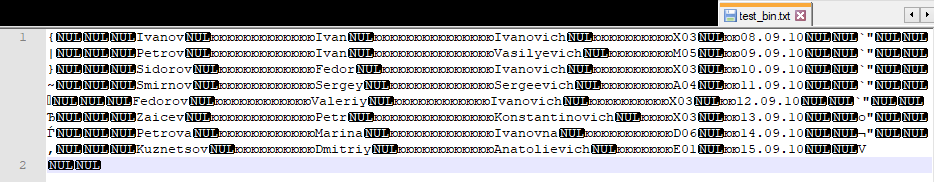


Рисунок 8 – Содержимое файла test\_bin.txt

## **2.3 Реализация приложения**

### **2.3.1 Структура используемой записи**

Для выполнения задания использовалась следующая структура:

struct record {

int oms\_code; //ключ

char surname[20]; //фамилия

char name[20]; //имя

char patronymic[20]; //отчество

char disease\_code[6]; //код болезни

char date[9]; //дата ДД.ММ.ГГ

int doctor\_id; //код врача

}

Количество памяти, занимаемое полями этой структуры:

* oms\_code – 4 байта;
* surname, name, patronymic – по 1 \* 20 = 20 байт;
* disease\_code – 6 байт;
* date – 9 байт;
* doctor\_id – 4 байта.

Итого один экземпляр занимает 83 байта. Проверим это системно (рисунок 9):



Рисунок 9 – Системное определение количества байт, занятых структурой

Расхождение в результатах обусловлено особенностями системы: количество выделяемых байт должно делиться на 4.

### **2.3.2 Структура двоичного файла**

Структура двоичного файла, используемого для хранения записей, изображена на рисунке 10.

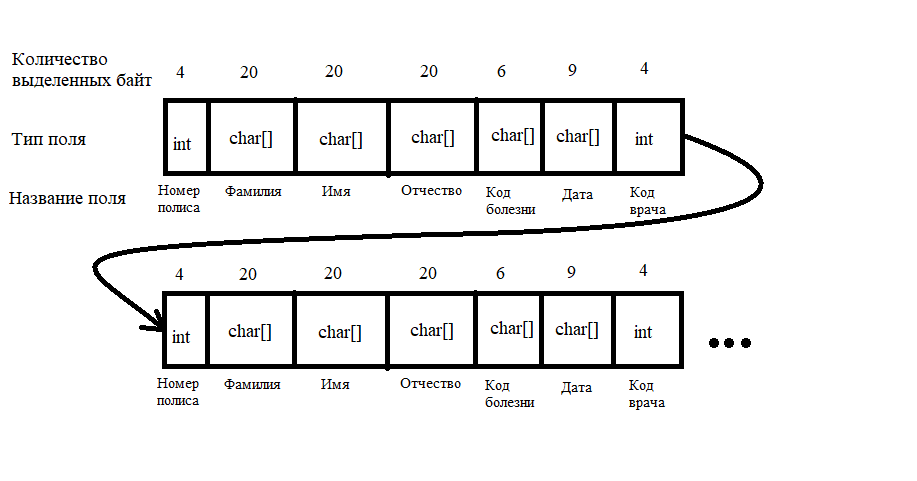


Рисунок 10 – Структура используемого двоичного файла

### **2.2.3 Функционал приложения**

Для выполнения задания использовались следующие функции и модули:

1. Структура record:
   1. Конструктор по умолчанию: заполняет структуру данными по умолчанию;
   2. Конструктор копии: создает новую структуру – копию исходной;
   3. Конструктор с параметрами: заполняет структуру данными из аргументов;
   4. Оператор ==: если все поля структур равны друг другу, то возвращает «истину», иначе – «ложь»;
   5. Оператор !=: возвращает инверсию результата сравнения структур.
2. Структура двунаправленного списка:
   1. Поля: указатели на первый и последний элементы first и last соответственно, длина списка;
   2. Конструктор: устанавливает указатели first, last равными nullptr, длину списка -–равной 0;
   3. Деструктор: пока список содержит элементы, вызывает метод delete\_last;
   4. Оператор доступа к элементу по индексу: проверяет, есть ли в списке элемент с индексом, переданным в аргументе; если да, возвращает указатель на него, если нет, возвращает nullptr;
   5. Метод empty: если указатель last – nullptr, возвращает «истину»;
   6. Метод append\_to\_end: добавляет новый элемент из аргумента в список;
   7. Метод delete\_last: удаляет последний элемент списка и обновляет указатель last;
   8. Метод delete\_element: удаляет элемент списка с индексом index, обновляет указатели предыдущего и следующего элемента, если такие есть;
   9. Метод clear: пока список содержит элементы, вызывает метод delete\_last;
   10. Метод show: выводит на экран содержимое списка;
   11. Метод is\_in: проверяет, есть ли в списке элемент, равный переданному в аргументе элементу;
   12. Метод get\_element\_index: проверяет, есть ли в списке элемент, равный переданному в аргументе элементу; если да, возвращает его индекс в списке, если нет, возвращает -1;
   13. Метод get\_element: проверяет, есть ли в списке элемент, равный переданному в аргументе элементу; если да, возвращает указатель на него, если нет, возвращает nullptr;
   14. Метод get\_element\_by\_index: действие аналогично действию оператора доступа к элементу по индексу;
   15. Метод get\_element\_by\_key: проверяет, есть ли в списке элемент с ключом, равным переданному в аргументе; если да, возвращает указатель на него, если нет, возвращает nullptr;
   16. Метод get\_first: возвращает указатель first;
   17. Метод get\_last: возвращает указатель last;
   18. Метод size: возвращает длину списка;
3. Модуль BinFileEditor:
   1. Метод openFile. Принимает в качестве аргумента путь к файлу. Открывает файл, расположенный по заданному пути, в текстовом или двоичном формате. Если файл открыт успешно, возвращает «истину», иначе «ложь»;
   2. Метод reopenFile. Закрывает файл с помощью метода close, затем вызывает метод openFile;
   3. Метод закрытия файла closeFile. Если ошибки во время работы с файлом не были обнаружены, то выводит соответствующее сообщение. Затем закрывает файл;
   4. Метод init. Запрашивает у пользователя тип файла и путь к нему. Затем открывает его; если файл открыт успешно, то считывает данные из него в структуру списка и возвращает «истину», иначе «ложь»;
   5. Метод saveToBin. Запрашивает у пользователя путь к создаваемому двоичному файлу. Затем создает файл с таким путем и, в случае успешного открытия, записывает в него данные списка и возвращает «истину», иначе «ложь»;
   6. Метод saveToText. Запрашивает у пользователя путь к создаваемому текстовому файлу. Затем создает файл с таким путем и, в случае успешного открытия, записывает в него данные списка и возвращает «истину», иначе «ложь»;
   7. Метод saveBin. Перезаписывает текущий файл текущими данными из списка и закрывает его, затем снова открывает файл;
   8. Метод saveText. Перезаписывает текущий файл текущими данными из списка и закрывает его, затем снова открывает файл;
   9. Метод deleteRecordByKey. Принимает в качестве аргумента номер полиса. Если запись с таким номером полиса найдена, то удаляет ее в списке и перезаписывает файл с помощью saveBin или saveText (в зависимости от формата открытого файла), после чего возвращает «истину», иначе «ложь»;
   10. Метод searchByDisease. Принимает в качестве аргумента код болезни. Запрашивает у пользователя путь к файлу, куда необходимо сохранить данные. Затем открывает файл по этому пути; если открытие успешно, записывает в файл все записи с переданным кодом болезни и возвращает «истину», иначе «ложь»;
   11. Метод deleteRecord. Принимает в качестве аргумента номер полиса. Если запись с таким номером полиса найдена, то заменяет ее последней записью в списке и перезаписывает файл с помощью saveBin или saveText (в зависимости от формата открытого файла), после чего возвращает «истину», иначе «ложь»;
   12. Метод readByIndex. Принимает в качестве аргументов индекс и ссылку на переменную типа record. Если переданный индекс меньше 0 или больше числа элементов в списке, то печатает соответствующее сообщение и возвращает «ложь»; в противном случае вызывает метод reopenFile; если он возвращает значение «ложь», то возвращает «ложь»; иначе считывает записи из файла в переменную, ссылка на которую была передана в аргументе, пока не будет достигнута запись с индексом, переданным в аргументе, после чего возвращает «истину»;
   13. Метод showContent. Вызывает метод reopenFile; если он возвращает «ложь», то возвращает «ложь»; если нет, то вызывает метод clear списка и считывает записи из файла в список. Затем вызывает метод show списка и возвращает «истину».
   14. Меню модуля process. Вызывает метод init; если он возвращает «ложь», то завершает работу. Затем в цикле предлагает пользователю одно из доступных действий. После выбора пользователем действия «выход» закрывает файл с помощью closeFile.

## **2.4 Код программы**

Запишем код программы на языке С++ (листинги 6-10):

Листинг 6 – list\_2way.h

#pragma once

#ifndef \_\_list2way\_h\_\_

#define \_\_list2way\_h\_\_

#include <iostream>

#include <string>

using namespace std;

struct record {

int oms\_code; //ключ

char surname[20]; //фамилия

char name[20]; //имя

char patronymic[20]; //отчество

char disease\_code[6]; //код болезни

char date[9]; //дата ДД.ММ.ГГ

int doctor\_id; //код врача

record()

{

oms\_code = -1;

strcpy\_s(this->surname, "generic");

strcpy\_s(this->name, "generic");

strcpy\_s(this->patronymic, "generic");

strcpy\_s(this->disease\_code, "XXXXX");

strcpy\_s(this->date, "--.--.--");

doctor\_id = -1;

}

record(const record& copy)

{

this->oms\_code = copy.oms\_code;

strcpy\_s(this->surname, copy.surname);

strcpy\_s(this->name, copy.name);

strcpy\_s(this->patronymic, copy.patronymic);

strcpy\_s(this->disease\_code, copy.disease\_code);

strcpy\_s(this->date, copy.date);

this->doctor\_id = copy.doctor\_id;

}

record(int oms\_code, char\* surname, char\* name, char\* patr, char\* disease\_code, char\* date, int doctor\_id) :

oms\_code(oms\_code), doctor\_id(doctor\_id)

{

strcpy\_s(this->surname, surname);

strcpy\_s(this->name, name);

strcpy\_s(this->patronymic, patronymic);

strcpy\_s(this->disease\_code, disease\_code);

strcpy\_s(this->date, date);

}

bool operator == (record right)

{

if (this->oms\_code != right.oms\_code) return false;

if (strcmp(this->surname, right.surname)) return false;

if (strcmp(this->name, right.name)) return false;

Продолжение листинга 6

if (strcmp(this->patronymic, right.patronymic)) return false;

if (strcmp(this->disease\_code, right.disease\_code)) return false;

if (strcmp(this->date, right.date)) return false;

if (this->doctor\_id != right.doctor\_id) return false;

return true;

}

bool operator != (record right)

{

return !(\*this == right);

}

};

using std::cout; using std::endl;

struct list\_chain\_2way

{

void show();

record data;

list\_chain\_2way\* prev = nullptr;

list\_chain\_2way\* next = nullptr;

};

struct list\_2way

{

private:

list\_chain\_2way\* first;

list\_chain\_2way\* last;

int length = 0;

public:

list\_2way();

~list\_2way();

list\_chain\_2way\* operator [] (int index);

bool empty();

void append\_to\_end(record new\_record);

void delete\_last();

void delete\_element(int index);

void clear();

void show();

bool is\_in(record data);

int get\_element\_index(record data);

list\_chain\_2way\* get\_element(record data);

list\_chain\_2way\* get\_element\_by\_key(int key);

list\_chain\_2way\* get\_element\_by\_index(int index);

Продолжение листинга 6

list\_chain\_2way\* get\_first();

list\_chain\_2way\* get\_last();

int size();

};

#endif

Листинг 7 – list\_2way.cpp

#include "list\_2way.h"

void list\_chain\_2way::show()

{

cout << "{ Номер полиса: " << this->data.oms\_code << ", ФИО: " << this->data.surname << " " << this->data.name <<

" " << this->data.patronymic << ", код болезни: " << this->data.disease\_code << ", дата обращения - " << this->data.date

<< ", ИД врача: " << this->data.doctor\_id << "}" << endl;

}

list\_2way::list\_2way()

{

first = nullptr;

last = nullptr;

}

list\_2way::~list\_2way()

{

while (!empty())

delete\_last();

}

void list\_2way::clear()

{

while (!empty())

delete\_last();

}

list\_chain\_2way\* list\_2way::operator [] (int index)

{

if (empty()) return nullptr;

list\_chain\_2way\* cur = first;

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

{

cur = cur->next;

if (!cur)

return nullptr;

}

return cur;

}

bool list\_2way::empty()

{

return (last == nullptr);

Продолжение листинга 7

}

void list\_2way::append\_to\_end(record new\_record)

{

list\_chain\_2way\* temp = new list\_chain\_2way;

temp->data = new\_record;

length++;

if (empty())

{

first = temp;

last = temp;

return;

}

temp->prev = last;

last->next = temp;

last = temp;

}

void list\_2way::delete\_last()

{

if (empty()) return;

list\_chain\_2way\* temp = last;

last = temp->prev;

if (last)

last->next = nullptr;

delete temp;

length--;

}

void list\_2way::delete\_element(int index)

{

if (empty()) return;

if (index >= this->size()) return;

list\_chain\_2way\* temp = this->get\_element\_by\_index(index);

list\_chain\_2way\* old\_prev = temp->prev;

list\_chain\_2way\* old\_next = temp->next;

if ((old\_prev == nullptr) && (old\_next == nullptr))

{

first = nullptr;

last = nullptr;

}

else if (old\_prev == nullptr)

{

first = old\_next;

old\_next->prev = old\_prev;

}

else if (old\_next == nullptr)

{

last = old\_prev;

old\_prev->next = old\_next;

}

else

{

old\_prev->next = old\_next;

Продолжение листинга 7

old\_next->prev = old\_prev;

}

delete temp;

length--;

}

void list\_2way::show()

{

if (empty()) { cout << "Список пуст" << endl; return; }

list\_chain\_2way\* temp = first;

cout << "[" << endl;

while (temp != nullptr)

{

temp->show();

temp = temp->next;

}

cout << "] (длина: " << length << ")" << endl;

}

bool list\_2way::is\_in(record data)

{

if (empty()) return false;

list\_chain\_2way\* temp = first;

while ((temp != nullptr) && (temp->data != data))

{

temp = temp->next;

}

if (temp == nullptr)

return false;

else

return true;

}

int list\_2way::get\_element\_index(record data)

{

if (empty()) return -2;

list\_chain\_2way\* temp = first;

int index = 0;

while ((temp != nullptr) && (temp->data != data))

{

temp = temp->next;

index++;

}

if (temp == nullptr)

return -1;

else

return index;

}

list\_chain\_2way\* list\_2way::get\_element(record data)

{

if (empty()) return nullptr;

list\_chain\_2way\* temp = first;

while ((temp != nullptr) && (temp->data != data))

{

Продолжение листинга 7

temp = temp->next;

}

if (temp == nullptr)

return nullptr;

else

return temp;

}

list\_chain\_2way\* list\_2way::get\_element\_by\_key(int key)

{

if (empty()) return nullptr;

list\_chain\_2way\* temp = first;

while ((temp != nullptr) && (temp->data.oms\_code != key))

{

temp = temp->next;

}

if (temp == nullptr)

return nullptr;

else

return temp;

}

list\_chain\_2way\* list\_2way::get\_element\_by\_index(int index)

{

if (empty()) return nullptr;

list\_chain\_2way\* temp = first;

int i = 0;

while ((temp != nullptr) && (i < index))

{

temp = temp->next;

i++;

}

if (temp == nullptr)

return nullptr;

else

return temp;

}

list\_chain\_2way\* list\_2way::get\_first()

{

return this->first;

}

list\_chain\_2way\* list\_2way::get\_last()

{

return this->last;

}

int list\_2way::size()

{

return this->length;

}

Листинг 8 – BinFileEditor.h

#pragma once

#ifndef \_\_BinFileEditor\_H\_\_

#define \_\_BinFileEditor\_H\_\_

#include <iostream>

#include <string>

#include <conio.h>

#include <fstream>

#include "list\_2way.h"

using namespace std;

namespace BinFileEditor {

bool openFile(string path);

bool reopenFile();

void closeFile();

void saveBin(); //служебное сохранение в двоичный файл

void saveText(); //служебное сохранение в текстовый файл

bool deleteRecordByKey(int key); //удалить пациента, сохраняя порядок остальных записей

bool searchByDisease(string disease\_code); //сформировать новый двоичный файл из людей с заболеванием

bool deleteRecord(int key); //удалить запись по индексу, последнюю запись переместить на место удаленной

bool readByIndex(int index, record &res); //прочитать запись по индексу

bool showContent(); //содержимое

bool saveToText(); //создание текстового файла на основе текущих записей

bool saveToBin(); //создание двоичного файла на основе текущих записей

void process(); //интерфейс для взаимодействия с файлом, который запускается из основной программы

bool init(); //инициализация

}

#endif

Листинг 9 – BinFileEditor.cpp

#include "list\_2way.h"

#include "BinFileEditor.h"

list\_2way entries; //нужно для записи в файл

fstream file;

bool isBinary;

string \_path;

bool BinFileEditor::openFile(string path)

{

file = fstream(path.c\_str(), ios::in | ios::out | (isBinary \* ios::binary));

Продолжение листинга 9

if (!file)

{

closeFile();

return false;

}

return true;

}

bool BinFileEditor::reopenFile()

{

file.close();

return openFile(\_path);

}

void BinFileEditor::closeFile()

{

if (file.good())

{

cout << "Ошибки ввода-вывода не обнаружены." << endl;

}

file.close();

}

bool BinFileEditor::init()

{

char filetype = ' ';

cout << "Выберите тип файла, который необходимо прочесть ([T] - текстовый, [B] - двоичный):";

while (!((tolower(filetype) == 'b') || (tolower(filetype) == 't')))

{

filetype = \_getch();

}

string path;

cout << endl << "Введите путь к файлу: ";

getline(cin, path);

\_path = path;

if (tolower(filetype) == 'b')

{

file = fstream(\_path, ios::in | ios::binary);

isBinary = true;

}

else

{

file = fstream(\_path, ios::in);

isBinary = false;

}

if (!file)

{

cout << "Ошибка при открытии файла" << endl;

return false;

}

else

{

while (!file.eof())

Продолжение листинга 9

{

int temp\_i;

string temp\_s;

record temp\_r = record();

if (isBinary)

{

file.read((char\*)&temp\_r, sizeof(record));

}

else

{

getline(file, temp\_s); temp\_r.oms\_code = strtol(temp\_s.c\_str(), nullptr, 0);

file.getline(temp\_r.surname, 20);

file.getline(temp\_r.name, 20);

file.getline(temp\_r.patronymic, 20);

file.getline(temp\_r.disease\_code, 6);

file.getline(temp\_r.date, 9);

getline(file, temp\_s); temp\_r.doctor\_id = strtol(temp\_s.c\_str(), nullptr, 0);

}

entries.append\_to\_end(temp\_r);

}

if (isBinary)

entries.delete\_last();

}

file.close();

return true;

}

bool BinFileEditor::saveToBin()

{

string path;

cout << "Введите путь к создаваемому двоичному файлу: ";

getline(cin, path);

fstream file2(path, ios::out | ios::binary);

if (!file2)

{

cout << "Произошла ошибка при записи в файл." << endl;

return false;

}

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

{

file2.write((char\*)(entries.get\_element\_by\_index(i)), sizeof(record));

}

file2.close();

return true;

}

bool BinFileEditor::saveToText()

{

string path;

cout << "Введите путь к создаваемому текстовому файлу: ";

getline(cin, path);

fstream file2(path, ios::out);

Продолжение листинга 9

if (!file2)

{

cout << "Произошла ошибка при записи в файл." << endl;

return false;

}

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

{

file2 << entries[i]->data.oms\_code << "\n";

file2 << entries[i]->data.surname << "\n";

file2 << entries[i]->data.name << "\n";

file2 << entries[i]->data.patronymic << "\n";

file2 << entries[i]->data.disease\_code << "\n";

file2 << entries[i]->data.date << "\n";

file2 << entries[i]->data.doctor\_id;

if (i < entries.size() - 1)

file2 << "\n";

}

file2.close();

return true;

}

void BinFileEditor::saveBin()

{

file.close();

file = fstream(\_path.c\_str(), ios::out | ios::binary);

if (!file)

{

cout << "Произошла ошибка при записи в файл." << endl;

return;

}

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

{

file.write((char\*)(entries.get\_element\_by\_index(i)), sizeof(record));

}

file.close();

file = fstream(\_path.c\_str(), ios::in | ios::out | ios::binary);

}

void BinFileEditor::saveText()

{

file.close();

file = fstream(\_path, ios::out);

if (!file)

{

cout << "Произошла ошибка при записи в файл." << endl;

return;

}

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

{

file << entries[i]->data.oms\_code << "\n";

file << entries[i]->data.surname << "\n";

file << entries[i]->data.name << "\n";

file << entries[i]->data.patronymic << "\n";

file << entries[i]->data.disease\_code << "\n";

Продолжение листинга 9

file << entries[i]->data.date << "\n";

file << entries[i]->data.doctor\_id;

if (i < entries.size() - 1)

file << "\n";

}

file.close();

file = fstream(\_path.c\_str(), ios::in | ios::out);

}

bool BinFileEditor::deleteRecordByKey(int key)

{

if (!entries.get\_element\_by\_key(key))

{

cout << "Запись с таким номером полиса не найдена" << endl;

return false;

}

else

{

int index = entries.get\_element\_index(entries.get\_element\_by\_key(key)->data);

entries.delete\_element(index);

if (isBinary)

{

saveBin();

}

else

{

saveText();

}

cout << "Запись успешно удалена" << endl;

}

return true;

}

bool BinFileEditor::searchByDisease(string disease\_code)

{

string path;

cout << "Введите путь, по которому будут сохранены данные: ";

getline(cin, path);

fstream file2 = fstream(path, ios::out | ios::binary);

if (!file2)

{

cout << "Произошла ошибка при записи в файл." << endl;

return false;

}

else

{

int total = 0;

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

{

if (entries[i]->data.disease\_code == disease\_code)

{

file2.write((char\*)(entries.get\_element\_by\_index(i)), sizeof(record));

Продолжение листинга 9

total++;

}

}

cout << "Создан двоичный файл; перенесено " << total << " записей" << endl;

}

return true;

}

bool BinFileEditor::deleteRecord(int key)

{

if (!entries.get\_element\_by\_key(key))

{

cout << "Запись с таким номером полиса не найдена" << endl;

return false;

}

else

{

int index = entries.get\_element\_index(entries.get\_element\_by\_key(key)->data);

record last = record(entries.get\_last()->data);

entries[index]->data = record(last);

if (isBinary)

{

saveBin();

}

else

{

saveText();

}

cout << "Запись успешно заменена на последнюю запись" << endl;

}

return true;

}

bool BinFileEditor::readByIndex(int index, record &res)

{

if ((entries.size() < index) || (index < 0))

{

cout << "Записи с индексом " << index << " не существует" << endl;

return false;

}

if (!reopenFile())

{

cout << "Ошибка при чтении файла" << endl;

return false;

}

string temp\_s;

res = record();

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

{

if (isBinary)

{

Продолжение листинга 9

file.read((char\*)&res, sizeof(record));

}

else

{

getline(file, temp\_s); res.oms\_code = strtol(temp\_s.c\_str(), nullptr, 0);

file.getline(res.surname, 20);

file.getline(res.name, 20);

file.getline(res.patronymic, 20);

file.getline(res.disease\_code, 6);

file.getline(res.date, 9);

getline(file, temp\_s); res.doctor\_id = strtol(temp\_s.c\_str(), nullptr, 0);

}

}

return true;

}

bool BinFileEditor::showContent()

{

if (!reopenFile())

{

cout << "Ошибка при чтении файла" << endl;

return false;

}

cout << "Содержимое файла: " << endl;

entries.clear();

while (!file.eof())

{

int temp\_i;

string temp\_s;

record temp\_r = record();

if (isBinary)

{

file.read((char\*)&temp\_r, sizeof(record));

}

else

{

getline(file, temp\_s); temp\_r.oms\_code = strtol(temp\_s.c\_str(), nullptr, 0);

file.getline(temp\_r.surname, 20);

file.getline(temp\_r.name, 20);

file.getline(temp\_r.patronymic, 20);

file.getline(temp\_r.disease\_code, 6);

file.getline(temp\_r.date, 9);

getline(file, temp\_s); temp\_r.doctor\_id = strtol(temp\_s.c\_str(), nullptr, 0);

}

entries.append\_to\_end(temp\_r);

}

if (isBinary)

entries.delete\_last();

entries.show();

return true;

}

Продолжение листинга 9

void BinFileEditor::process()

{

if (!init()) return;

char choose = ' ';

int n;

record t;

string disease\_code\_search = "";

bool running = true;

while (running)

{

cout << "Выберите действие: " << endl

<< "[T] - сохранить текущие данные в текстовый файл" << endl

<< "[B] - сохранить текущие данные в двоичный файл" << endl

<< "[C] - вывести содержимое текущего файла" << endl

<< "[I] - получить запись по индексу" << endl

<< "[D] - удалить запись по ключу путем замены на последнюю запись" << endl

<< "[K] - удалить запись по ключу, сохраняя порядок остальных записей" << endl

<< "[S] - сформировать новый двоичный файл из записей с определенным кодом болезни" << endl

<< "[Q] - выход" << endl;

choose = \_getch();

switch (tolower(choose))

{

case 'q':

running = false;

break;

case 't':

saveToText();

break;

case 'b':

saveToBin();

break;

case 'c':

showContent();

break;

case 'i':

cout << "Введите индекс:" << endl;

cin >> n;

cin.ignore(32768, '\n');

if (readByIndex(n, t))

{

cout << "Запись с индексом " << n << ": " << "{ Номер полиса: " << t.oms\_code << ", ФИО: " << t.surname << " " << t.name <<

" " << t.patronymic << ", код болезни: " << t.disease\_code << ", дата обращения - " << t.date

<< ", ИД врача: " << t.doctor\_id << "}" << endl;

}

break;

Продолжение листинга 9

case 'd':

cout << "Введите номер полиса:" << endl;

cin >> n;

cin.ignore(32768, '\n');

if (entries.get\_element\_by\_key(n))

{

t = entries.get\_element\_by\_key(n)->data;

deleteRecord(t.oms\_code);

}

else

{

cout << "Запись с таким номером полиса не найдена" << endl;

}

break;

case 'k':

cout << "Введите номер полиса:" << endl;

cin >> n;

cin.ignore(32768, '\n');

if (entries.get\_element\_by\_key(n))

{

t = entries.get\_element\_by\_key(n)->data;

deleteRecordByKey(t.oms\_code);

}

else

{

cout << "Запись с таким номером полиса не найдена" << endl;

}

break;

case 's':

cout << "Введите код болезни, который необходимо найти: " << endl;

cin >> disease\_code\_search;

cin.ignore(32768, '\n');

searchByDisease(disease\_code\_search);

break;

default:

cout << "Неизвестное действие." << endl;

}

if (running)

{

system("pause");

system("cls");

}

}

closeFile();

}

Листинг 10 – main.cpp

#include <iostream>

#include <locale.h>

#include "list\_2way.h"

#include "BinFileEditor.h"

int main()

{

setlocale(LC\_ALL, "ru");

BinFileEditor::process();

}

## **2.5 Тестирование**

Проведем тестирование написанной программы (рисунки 11-):

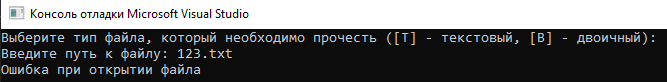


Рисунок 11 – Попытка открытия несуществующего файла

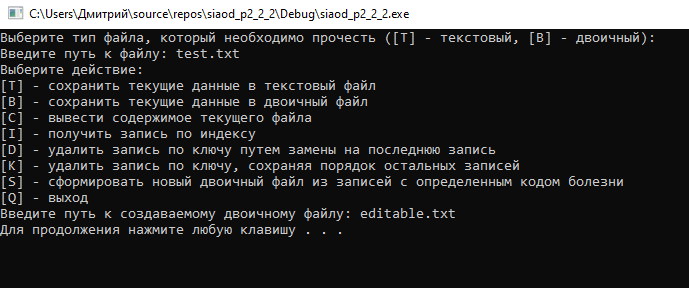


Рисунок 12 – Создание нового двоичного файла, с которым будет проводиться дальнейшая работа

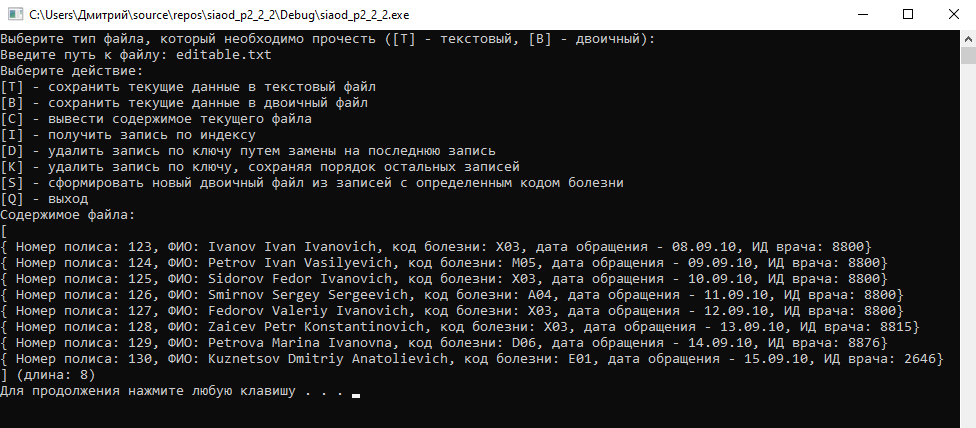


Рисунок 13 – Вывод содержимого двоичного файла

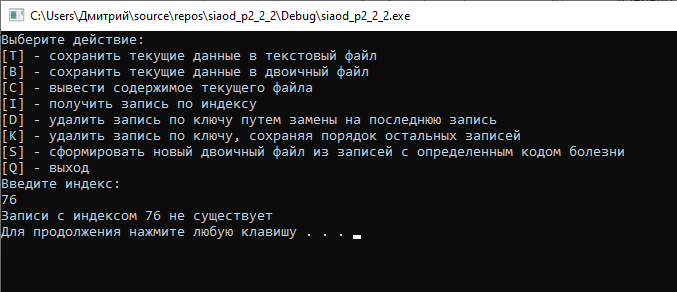


Рисунок 14 – Попытка получения записи с несуществующим индексом

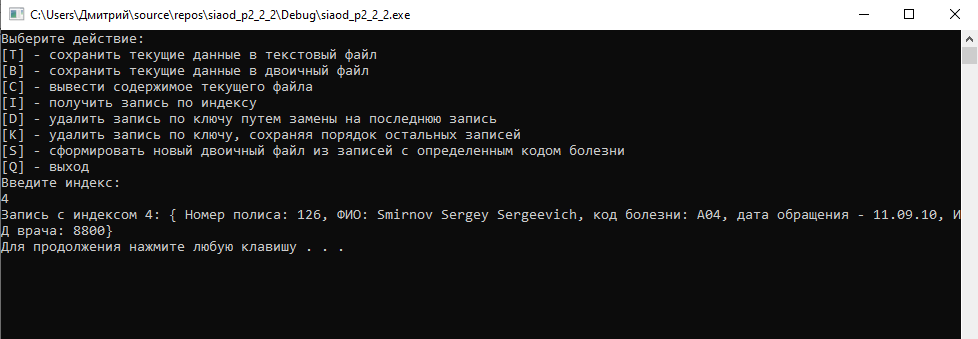


Рисунок 15 – Получение записи с индексом 4

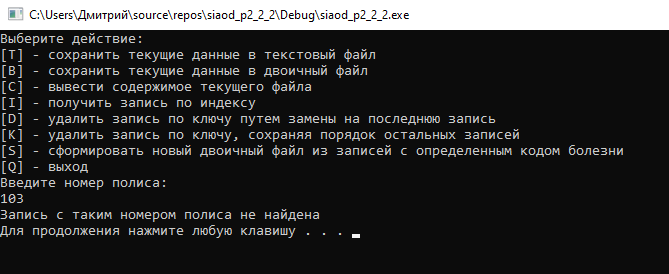


Рисунок 16 – Попытка удаления записи с несуществующим номером полиса

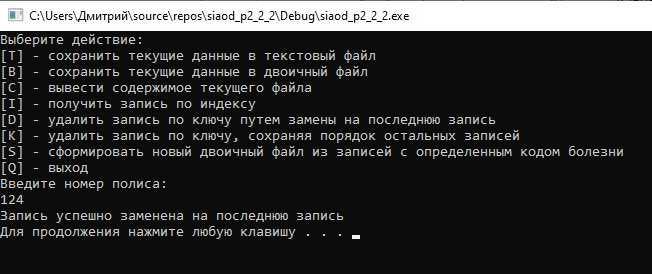


Рисунок 17 – Замена записи с номером 124 на последнюю запись

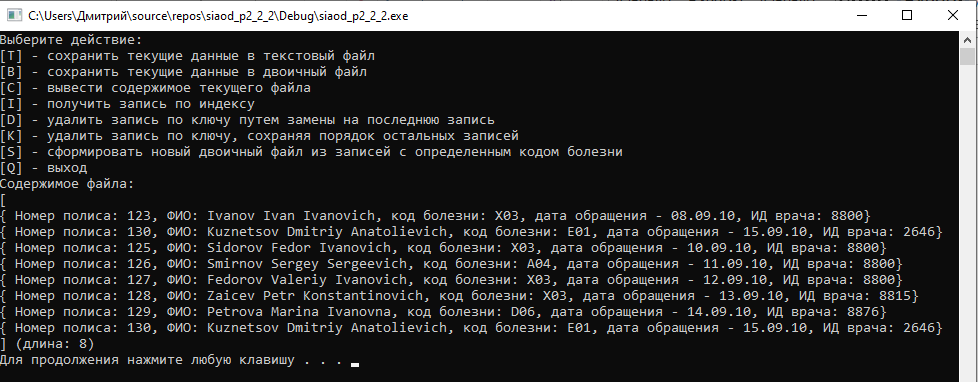


Рисунок 18 – Содержимое файла editable.txt после замены, проведенной на рисунке 17

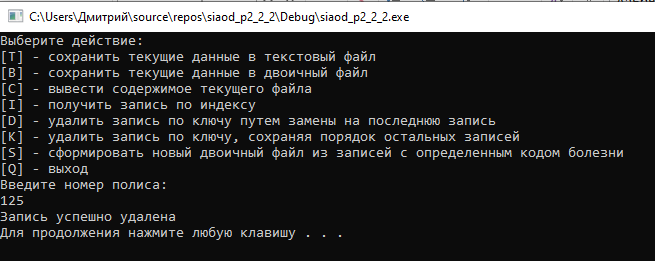


Рисунок 19 – Удаление записи с номером 125

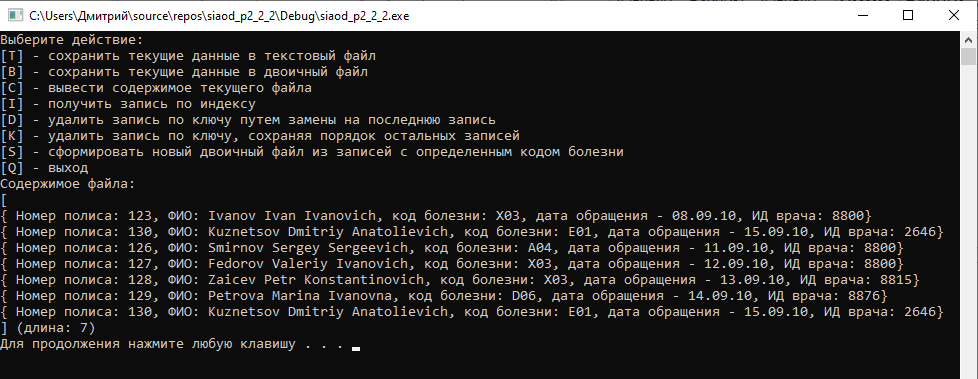


Рисунок 20 - Содержимое файла editable.txt после удаления, проведенного на рисунке 19

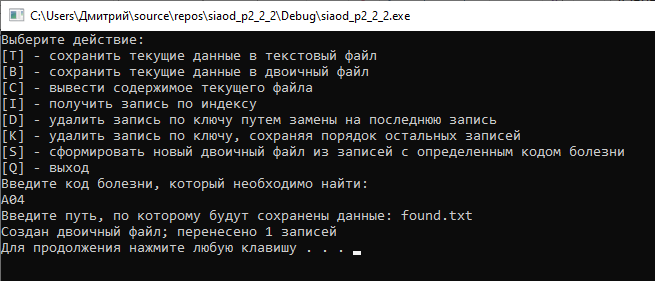


Рисунок 21 – Сохранение записей с кодом болезни А04 в новый файл



Рисунок 22 – Содержимое файла found.txt

# **ЗАКЛЮЧЕНИЕ**

В ходе работы были выполнены два задания по применению файловых потоков языка С++, по управлению текстовым и двоичным файлами, проведены тестирования программ на наборе тестовых данных. Все программы работают корректно, следовательно, можно говорить о правильности алгоритмов, реализованных с помощью программ.

# **ИНФОРМАЦИОННЫЙ ИСТОЧНИК**

Задания для самостоятельной работы №2. URL: <https://online-edu.mirea.ru/pluginfile.php?file=%2F1137402%2Fmod_assign%2Fintroattachment%2F0%2FСиАОД%20Самостоятельная%20работа%202%20%28управление%20файлом%29.pdf&amp;forcedownload=1>, дата обращения: 23.09.23