Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Информатика»

Тема: Лабораторная работа №10

"Сохранение данных в файле с использованием потоков"

Семестр 2

Выполнил работу

Студенты группы РИС-24-1Б

Конькова С. С.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь-2025

# Постановка задачи

1. Создать пользовательский класс с минимальной функциональностью.

2. Написать функцию для создания объектов пользовательского класса (ввод исходной информации с клавиатуры) и сохранения их в потоке (файле).

3. Написать функцию для чтения и просмотра объектов из потока.

4. Написать функцию для удаления объектов из потока в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

5. Написать функцию для добавления объектов в поток в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

6. Написать функцию для изменения объектов в потоке в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

7. Для вызова функций в основной программе предусмотреть меню.

Задание:

* Удалить все записи меньшие заданного значения.
* Увеличить все записи с заданным значением на число L.
* Добавить K записей после элемента с заданным номером.

Анализ

1. Pair: Класс для работы с парой чисел (целое и дробное). Реализует операции сложения, вычитания, умножения, инкремента и сравнения.
2. Point: Структура для представления узла связанного списка, содержащая элемент типа T и указатель на следующий узел.
3. Iterator: Итератор для обхода связанного списка, поддерживающий операции сравнения и разыменования.
4. LinkedList: Класс для реализации связанного списка, поддерживающий операции добавления, удаления, копирования и индексирования элементов.
5. FileIO: Класс для работы с файлами, позволяющий загружать и выгружать данные из файла в связанный список, а также выполнять операции над записями (удаление, уменьшение, добавление).
6. fun\_int(): Функция для работы с double, демонстрирующая загрузку, уменьшение, удаление и добавление элементов в список.
7. fun\_Pair(): Функция для работы с Pair, аналогично демонстрирует операции с парами чисел.
8. В main() происходит установка кодировки консоли, выбор функции для работы с данными (либо с double, либо с Pair).

# Описание пользовательского класса

#pragma once

#include <iostream>

#include <algorithm> // для min/max

#include <fstream>

using namespace std;

class Pair

{

public:

Pair() {}

Pair(int a, double b);

int numINT = 0;

double numDOUBLE = 0;

Pair& operator =(const Pair& pair);

Pair& operator =(const int& k);

// вывод

private:

friend fstream& operator>>(fstream& fin, Pair& p)

{

fin >> p.numINT;

fin >> p.numDOUBLE;

return fin;

}

friend fstream& operator<<(fstream& fout, const Pair& p)

{

fout << p.numINT << "\n" << p.numDOUBLE << "\n";

return fout;

}

friend ostream& operator<<(ostream& out, const Pair& pair)

{

out << pair.numINT << " : " << pair.numDOUBLE;

return out;

}

friend istream& operator>>(istream& in, Pair& pair)

{

in >> pair.numINT >> pair.numDOUBLE;

return in;

}

public:

Pair operator \* (Pair& pair); // умножение двух пар чисел

Pair operator-(const Pair& p); // вычитание двух объектов

Pair& operator+(int a); // прибавление int числа

Pair& operator+(double b); // прибавление double числа

Pair& operator++(); // префиксный инкремент

Pair operator++(int); // постфиксный инкремент

bool operator<=(const Pair& p);

};

bool Pair::operator<=(const Pair& p)

{

return this->numINT <= p.numINT && this->numDOUBLE <= p.numDOUBLE;

}

Pair Pair::operator\*(Pair& pair)

{

Pair new\_pair;

new\_pair.numINT = this->numINT \* pair.numINT;

new\_pair.numDOUBLE = this->numDOUBLE \* pair.numDOUBLE;

return new\_pair;

}

Pair& Pair::operator =(const Pair& pair)

{

this->numINT = pair.numINT;

this->numDOUBLE = pair.numDOUBLE;

return \*this;

};

Pair& Pair::operator =(const int& k)

{

this->numINT = k;

this->numDOUBLE = k;

return \*this;

}

Pair::Pair(int a, double b)

{

numINT = a;

numDOUBLE = b;

}

Pair Pair::operator-(const Pair& p)

{

Pair res(numINT - p.numINT, numDOUBLE - p.numDOUBLE);

return res;

}

Pair& Pair::operator+(int a)

{

this->numINT += a;

return \*this;

}

Pair& Pair::operator+(double b)

{

this->numDOUBLE += b;

return \*this;

}

Pair& Pair::operator++()

{

++numINT; ++numDOUBLE;

return \*this;

}

Pair Pair::operator ++(int)

{

Pair temp = \*this;

this->numINT++; this->numDOUBLE++;

return temp;

}

//-------------------------//

// связь списка

//-------------------------//

template<typename T>

struct Point

{

T num; // int, float, double, Pair

Point\* next = nullptr; // указатель на следующий элемент списка

};

//-------------------------//

// итератор списка

//-------------------------//

template <typename T>

class Iterator

{

private:

Point<T>\* elem = nullptr; // указатель на связь

public:

Iterator() { elem = nullptr; }

Iterator(Point<T>\* point) { elem = point; }

Iterator(const Iterator& it) { elem = it.elem; };

bool operator == (const Iterator& it) { return elem == it.elem; }

bool operator != (const Iterator& it) { return elem != it.elem; }

void operator ++ () { elem = elem->next; }; // ++i - смещение вправо на 1

void operator ++ (int) { elem = elem->next; }; // i++ - смещение вправо на 1

//---------// i+n - смещение вправо на n

Iterator& operator + (int n)

{

for (int i = 0; i < n && elem->next != nullptr; i++)

elem = elem->next;

return \*this;

}

T operator \* () const { return elem->num; }; // разыменование итератора (вывод)

};

//-------------------------//

// связанный список

//-------------------------//

template <typename T>

class LinkedList

{

private:

size\_t size = 0; // размер списка

Point<T>\* head = nullptr; // указатель на первый элемент

Point<T>\* top = nullptr; // указатель на ласт элемент

//-------------------------------вывод----------------------------------------//

void output(Point<T>\* obj) const;

void show() const;

//---------------------------------------------------------------------------//

public:

friend ostream& operator<<(ostream& out, const LinkedList<T>& list)

{

list.show();

return out;

}

friend istream& operator>>(istream& in, LinkedList<T>& list)

{

T k;

in >> k;

list.push(k);

return in;

}

LinkedList(); // конструктор

LinkedList(size\_t s, T k); // констуктор: s - размер, k - значение по умолчанию

~LinkedList(); // деструктор

LinkedList(const LinkedList<T>& list); // конструктор копирования

T& operator [](int index) const; // доступ по индексу

LinkedList& operator =(const LinkedList<T>& list); // оператор присваивания

LinkedList operator \* (LinkedList<T>& list); // умножение списка на список

int operator ()() { return size; }; // возвращение длины списка

void push(T k); // добавление в конец списка

T pop(); // удаление первого элемента

Iterator<T> first() { return Iterator<T>(head); } // возвращает указатель на первый элемент

Iterator<T> last() { return Iterator<T>(nullptr); } // вовзращает указаетль на последний элемент

size\_t give\_size() { return size; } // длина списка

};

// конструктор

template <typename T>

LinkedList<T>::LinkedList()

{

head = nullptr;

top = nullptr;

size = 0;

}

// конструктор с параметрами

template <typename T>

LinkedList<T>::LinkedList(size\_t s, T k)

{

size = s;

if (size != 0) {

//---------создаем голову и передаем ей значение k---------//

Point<T>\* obj = new Point<T>;

obj->num = k;

head = obj;

//--------------------------------------------------------//

//-------создаем следующие size - 1 элементы--------------//

Point<T>\* temp = head;

for (int i = size - 1; i != 0; i--)

{

obj = new Point<T>;

temp->next = obj;

obj->num = k;

temp = obj;

}

// хвостик + итераторы

top = obj;

}

}

// добавление в конец

template <typename T>

void LinkedList<T>::push(T k)

{

//------------------------------------------//

// Если размер 0, то задаем голову = хвосту

//------------------------------------------//

if (size == 0)

{

head = new Point<T>;

head->num = k;

top = head;

}

//------------------------------------------//

// Иначе через хвост добавляем новый элемент

//------------------------------------------//

else {

Point<T>\* obj = new Point<T>;

top->next = obj;

obj->num = k;

top = obj;

}

// увеличиваем размер, итератор на конец

size++;

}

// удаление начального элемента

template <typename T>

T LinkedList<T>::pop()

{

// создаем промежуточную связь, удаляем голову, перемещаем голову

T k = head->num;

Point<T>\* temp = head;

head = head->next;

size--;

delete temp;

return k;

}

// деструктор

template <typename T>

LinkedList<T>::~LinkedList()

{

while (head != nullptr)

{

pop();

}

}

//----------------------------вывод----------------------------------//

template <typename T>

void LinkedList<T>::output(Point<T>\* obj) const

{

cout << obj->num << "\n";

if (obj == top) return;

output(obj->next);

}

template <typename T>

void LinkedList<T>::show() const

{

if (size == 0) cout << "Список пуст" << endl;

else output(head);

}

//-------------------------------------------------------------------//

// индекс

template <typename T>

T& LinkedList<T>::operator[](int index) const

{

if (index < size && index >= 0)

{

Point<T>\* temp = head;

int count = 0;

while (temp != nullptr)

{

if (count == index)

return temp->num;

temp = temp->next;

++count;

}

}

cout << endl << endl;

perror("Ошибка индексирования");

cout << endl;

exit(1);

}

// конструктор копирования

template <typename T>

LinkedList<T>::LinkedList(const LinkedList<T>& list)

{

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

{

this->push(list[i]);

}

}

// присваивание

template <typename T>

LinkedList<T>& LinkedList<T>::operator =(const LinkedList<T>& list)

{

// создаем size\_temp, т.к. pop уменьшает size, нужно застанить

int size\_temp = size;

// удаляем все элементы

for (int i = 0; i < size\_temp; i++)

{

this->pop();

}

// копируем через push()

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

{

this->push(list[i]);

}

return \*this;

}

// умножение

template <typename T>

LinkedList<T> LinkedList<T>::operator \* (LinkedList<T>& list)

{

// создаем новый список по макс рзамеру, обнуляем через второй параметр

T nulik;

nulik = 0;

LinkedList<T> new\_list;

for (int i = 0; i < min((int)size, list()); i++)

{

new\_list.push((\*this)[i] \* list[i]);

}

for (int i = 0; i < max((int)size, list()) - min((int)size, list()); i++)

{

new\_list.push(nulik);

}

return new\_list;

}

template <typename T>

class FileIO

{

LinkedList <T> list; // буферный список, очищается при каждом выводе списка get\_list();

string filename; // основной, куда будет сохраняться

string bufname = "temp.txt"; // буферный файл

public:

FileIO(string filename); // конструктор с параметрами

int load(LinkedList<T>& list); // загрузка в списка файл (возвращает число успешных загрузок)

int unload(); // выгрузка из файла в список (возвращает число успешных загрузок)

LinkedList<T> give\_list(); // вернуть список

void del\_less\_than(T obj); // удаление записей меньше заданного

void decrease\_records(T L); // уменьшение всех записей на L

void add\_records(int number, int k); // добавление k записей перед заданным номер n

};

template <typename T>

FileIO<T>::FileIO(string filename) : filename(filename) {}

template <typename T>

int FileIO<T>::load(LinkedList<T>& list)

{

fstream stream(filename, ios::out);

if (!stream) return -1;

auto it = list.first();

auto count = 0;

for (; it != list.last(); it++)

{

stream << \*it << "\n";

count++;

}

stream.close();

return count;

}

template <typename T>

int FileIO<T>::unload()

{

fstream stream(filename, ios::in);

if (!stream) return -1;

auto count = 0;

T k;

stream >> k;

while (!stream.eof())

{

count++;

list.push(k);

stream >> k;

}

stream.close();

return count;

}

template <typename T>

LinkedList<T> FileIO<T>::give\_list()

{

LinkedList<T> temp = list;

auto sizel = list.give\_size();

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

{

list.pop();

}

return temp;

}

template <typename T>

void FileIO<T>::del\_less\_than(T obj)

{

fstream in(filename, ios::in);

fstream out(bufname, ios::out);

T k;

in >> k;

while (!in.eof())

{

if (obj <= k)

{

out << k << "\n";

}

in >> k;

}

in.close();

out.close();

remove(filename.c\_str());

rename(bufname.c\_str(), filename.c\_str());

}

template <typename T>

void FileIO<T>::add\_records(int number, int k)

{

fstream in(filename, ios::in);

fstream out(bufname, ios::out);

T obj;

in >> obj;

int count = 1;

while (!in.eof())

{

out << obj << "\n";

in >> obj;

if (count == number)

{

cout << "Введите " << k << " элементов: " << endl;

T res;

for (int i = 1; i <= k; i++)

{

cin >> res;

out << res << "\n";

}

}

count++;

}

in.close();

out.close();

remove(filename.c\_str());

rename(bufname.c\_str(), filename.c\_str());

}

template <typename T>

void FileIO<T>::decrease\_records(T L)

{

fstream in(filename, ios::in);

fstream out(bufname, ios::out);

T k;

in >> k;

while (!in.eof())

{

out << k - L << "\n";

in >> k;

}

in.close();

out.close();

remove(filename.c\_str());

rename(bufname.c\_str(), filename.c\_str());

}

#include "LinkedList.h"

#include "windows.h"

#define SAVE\_VARIBLE\_NAME(varible) VARIBLE\_NAME = string(#varible) + ".txt";

// Для работы с n + Pair;

Pair& operator+(int n, Pair& pair)

{

return pair + n;

}

Pair& operator+(double n, Pair& pair)

{

return pair + n;

}

string VARIBLE\_NAME = "NULL"; // для получения имени переменной, выше директива (хз зачем, хочу)

void fun\_int()

{

LinkedList <double> list(3, 1.5);

SAVE\_VARIBLE\_NAME(list2);

FileIO <double> file(VARIBLE\_NAME);

list.push(1.7);

list.push(2.1);

list.push(2.5);

cout << "\n" << VARIBLE\_NAME << " загружается в файл: " << endl;

cout << list << endl;

file.load(list);

cout << "\n" << VARIBLE\_NAME << " уменьшаем на 0.5: " << endl;

file.decrease\_records(0.5);

if (file.unload() != -1)

cout << file.give\_list();

cout << "\n" << VARIBLE\_NAME << " удаляем меньшее 1.6: " << endl;

file.del\_less\_than(1.6);

if (file.unload() != -1)

cout << file.give\_list();

int n, k;

cout << "\n" << VARIBLE\_NAME << " добавляем после n= элемента k=" << endl;

cin >> n >> k;

file.add\_records(n, k);

if (file.unload() != -1)

cout << file.give\_list();

}

void fun\_Pair()

{

LinkedList <Pair> list(3, Pair(1, 1));

SAVE\_VARIBLE\_NAME(list2);

FileIO <Pair> file(VARIBLE\_NAME);

list.push(Pair(1, 1.7));

list.push(Pair(2, 2.1));

list.push(Pair(3, 2.5));

cout << "\n" << VARIBLE\_NAME << " загружается в файл: " << endl;

cout << list << endl;

file.load(list);

cout << "\n" << VARIBLE\_NAME << " уменьшаем на Pair(2,0.3) " << endl;

file.decrease\_records(Pair(2, 0.3));

if (file.unload() != -1)

cout << file.give\_list();

cout << "\n" << VARIBLE\_NAME << " удаляем меньшее Pair(1, 0.5) " << endl;

file.del\_less\_than(Pair(0, 1.4));

if (file.unload() != -1)

cout << file.give\_list();

int n, k;

cout << "\n" << VARIBLE\_NAME << " добавляем после n= элемента k=" << endl;

cin >> n >> k;

file.add\_records(n, k);

if (file.unload() != -1)

cout << file.give\_list();

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

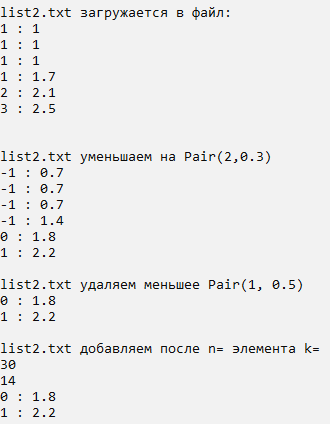
/\*fun\_int();\*/

fun\_Pair();

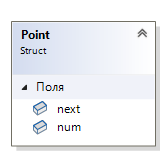
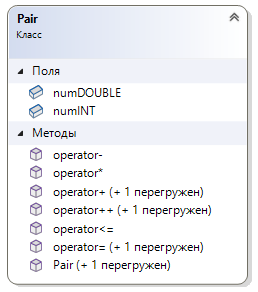
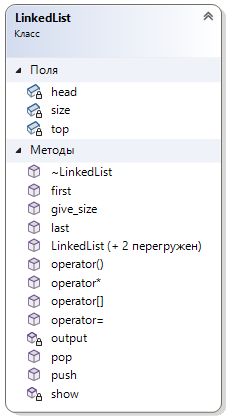
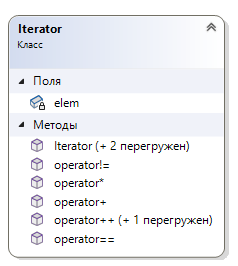
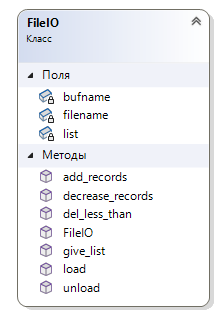
return 0;

}

# Работа программы



# UML-диаграмма



# Ответы на контрольные вопросы

1. Что такое поток?

Поток - определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти — буфер. Буфер накапливает байты, и фактическая передача данных выполняется после заполнения буфера. При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.

1. Какие типы потоков существуют?

Потоки бывают

* Стандартные: только однонаправленные, либо входные, либо выходные.
* Строковые: могут быть и однонаправленными и двунаправленными
* Файловые: могут быть и однонаправленными и двунаправленными.

1. Какую библиотеку надо подключить при использовании стандартных потоков?

Для использования стандартных потоков надо задать в программе директиву

#include <iostream>

В заголовочном файле iostream содержатся описания классов ввода/вывода и четыре стандартных системных объекта:

cin – объект класса istream, по умолчанию связан с клавиатурой. cout- объект класса ostream, по умолчанию связан с экраном.

clog - объект класса ostream, соответствующий стандартному выводу для ошибок, по умолчанию связан с экраном.

cerr - объект класса ostream, соответствующий стандартному выводу для ошибок, по умолчанию связан с экраном.

1. Какую библиотеку надо подключить при использовании файловых потоков?

Для использования файловых потоков надо задать в программе директиву

#include <fstream>

После этого в программе можно объявлять объекты – файловые потоки трех типов:

* входной ifstream;
* выходной ofstream;
* двунаправленный fstream;

1. Какую библиотеку надо подключить при использовании строковых потоков?

Для использования строковых потоков надо задать в программе директиву

#include <sstream>

После этого в программе можно объявлять объекты – строковые потоки трех типов:

* входной istringstream;
* выходной ostringstream;
* двунаправленный stringstream;

1. Какая операция используется при выводе в форматированный поток?

Для форматируемых потоков вывод, как правило, осуществляется перегруженной операцией сдвига влево operator<<.

stream<<3.4;stream<<’\n’;//вывод константы

1. Какая операция используется при вводе из форматированных потоков?

Для форматируемых потоков ввод перегруженной операцией сдвига вправо operator>>.

stream>>a>>b;

Ввод в переменную завершается, если очередной символ в потоке не соответствует типу вводимого значения. Стандартными разделителями в потоке являются: ‘ ‘(пробел), ‘\n’, ‘\t’.

1. Какие методы используются при выводе в форматированный поток?

Для вывода символов можно использовать методы:



1. Какие методы используется при вводе из форматированного потока?



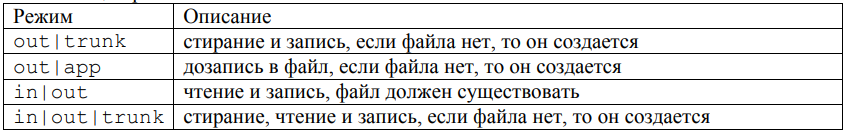


1. Какие режимы для открытия файловых потоков существуют?

Режимы открытия потока:

****

Комбинации режимов:



1. Какой режим используется для добавления записей в файл?

Для добавления записи в конец файла используется режим app.

Также, используется режим ate для чтения и/или записи в конец файла.

1. Какой режим (комбинация режимов) используется в конструкторе ifstream file(“f.txt”)?

Используется режим in - открыть поток для чтения (по умолчанию для ifstream).

1. Какой режим (комбинация режимов) используется в конструкторе fstream file(“f.txt”)?

Используется комбинация режимов in | out - чтение и запись, файл должен существовать.

1. Какой режим (комбинация режимов) используется в конструкторе ofstream file(“f.txt”)?

Используется режим out - открыть поток для записи в файл (по умолчанию для ofstream).

1. Каким образом открывается поток в режиме ios::out|ios::app?

Поток открывается для дозапись в файл, если файла нет, то он создаётся.

1. Каким образом открывается поток в режиме ios::out |ios::trunc?

Поток открывается для стирания и записи, если файла нет, то он создаётся.

1. Каким образом открывается поток в режиме ios::out |ios::in|ios::trunk?

Поток открывается для стирания, чтения и записи, если файла нет, то он создаётся.

1. Каким образом можно открыть файл для чтения?

Файл может быть открыт либо явно методом open(), либо неявно – конструктором при создании потока.

1. Каким образом можно открыть файл для записи?

Файл может быть открыт либо явно методом open(), либо неявно – конструктором при создании потока.

1. Привести примеры открытия файловых потоков в различных режимах.

Ниже приведены примеры открытия файловых потоков в различных режимах:

Открытие файла для записи: ofstream outputFile("example.txt");

Открытие файла для чтения: ifstream inputFile("example.txt");

Открытие файла для добавления записей в конец: ofstream appendFile("example.txt", ios::app);

Открытие файла для чтения и записи: fstream inOutFile("example.txt", ios::in | ios::out);

Открытие файла для чтения и записи, обрезание файла до нулевой длины: fstream truncFile("example.txt", ios::trunc | ios::in | ios::out);

1. Привести примеры чтения объектов из потока.

Чтение объектов из потока в C++ может быть выполнено с помощью операторов извлечения (>>). Ниже приведены примеры чтения различных типов данных из потока:

Чтение целого числа: int number; inputFile >> number;

Чтение строки: string str; getline(inputFile, str);

Чтение символа: char ch; inputFile >> ch;

Чтение числа с плавающей точкой: double value; inputFile >> value;

1. Привести примеры записи объектов в поток.

Запись объектов в поток в C++ может быть выполнена с помощью операторов вставки (<<). Ниже приведены примеры записи различных типов данных в поток:

Запись целого числа: int number = 10; outputFile << number;

Запись строки: string str = "example"; outputFile << str;

Запись символа: char ch = 'a'; outputFile << ch;

Запись числа с плавающей точкой: double value = 3.14; outputFile << value;

1. Сформулировать алгоритм удаления записей из файла.

Алгоритм удаления записей из файла в С++ может быть следующим:

1. Открыть файл для чтения и записи.

2. Создать временный файл.

3. Прочитать записи из исходного файла и проверять каждую запись на соответствие критерию удаления.

4. Записать нужные записи во временный файл.

5. Закрыть исходный файл и временный файл.

6. Удалить исходный файл с помощью функции.

7. Переименовать временный файл в имя исходного файла.

1. Сформулировать алгоритм добавления записей в файл.

1. Открыть файл для записи с помощью функции fstream::open().

2. Проверить, что файл успешно открыт с помощью функции fstream::is\_open().

3. Создать объект записи, который будет содержать данные для записи в файл.

4. Заполнить объект записи данными.

5. Записать объект записи в файл с помощью оператора << или функции fstream::write().

6. Проверить, что запись прошла успешно с помощью функции.

7. Повторять шаги 3-6 для каждой новой записи.

8. Закрыть файл с помощью функции fstream::close().

1. Сформулировать алгоритм изменения записей в файле.

1. Открыть файл для чтения и записи с помощью функции std::fstream.

2. Проверить, что файл успешно открыт.

3. Считать данные из файла в структуру или массив данных.

4. Изменить необходимые данные в структуре или массиве.

5. Перезаписать измененные данные в файл.

6. Закрыть файл.