МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЯ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра вычислительной техники



**КУРСОВАЯ РАБОТА**

**по дисциплине: «***Программирование***»**

**на тему:** **«***Последовательный двоичный файл***»**

Выполнил(а): Руководитель:

Студент(ка) гр. АВТ-341, АВТФ Романенко Т. А., старший

Буслаев Максим Евгеньевич преподаватель

Новосибирск

2024

**СОДЕРЖАНИЕ**

[1. ОПИСАНИЕ ЗАДАНИЯ 3](#_Toc91105134)

[2. СТРУКТУРНОЕ ОПИСАНИЕ РАЗРАБОТКИ 4](#_Toc91105137)

[3. ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ РАЗРАБОТКИ 6](#_Toc91105140)

[4. ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА](#_Toc91105151) 17

[5. КОНТРОЛЬНЫЕ ПРИМЕРЫ](#_Toc91105152) 22

[ВЫВОДЫ](#_Toc91105153) 25

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ](#_Toc91105154) 26

## 1. ОПИСАНИЕ ЗАДАНИЯ

## 1.1 Общая формулировка задания

Двоичный файл содержит записи переменной длины (см. простые структуры данных). Формат записи предполагает ее переменную размерность. Реализовать набор операций над записями без загрузки одновременно всей структуры данных в память (поэлементная загрузка СД): добавление строки, извлечение, удаление (из любой позиции), вставка по логическому номеру и редактирование (обновление) строки, вставка с сохранением порядка, сортировка, постраничный просмотр, сжатие файла. При изменении размерности записи она переписывается в конец файла.

Программа должна реализовывать указанные выше действия. Протестировать структуру данных на простом типе данных (например, int, double) и сложном, из выбранного по заданию. Программа тестирования должна содержать **меню**, обеспечивающее выбор операций.

## 1.2 Выбранная тема

## *1.2.1 Содержание объекта данных.*

Список студентов факультета. Основная таблица: Фамилия, дата рождения, дата поступления, дата отчисления, адрес, группа. Редактирование с выбором по группе и шаблону фамилии.

## *1.2.2 Вид структуры данных:*

Двусвязный список.

## 2. СТРУКТУРНОЕ ОПИСАНИЕ РАЗРАБОТКИ

***2.1 Класс списка студентов***

Для реализации поставленной задачи был разработан класс «StudentsList», который управляет списком студентов и предоставляет различные функции для работы с данными, а также бинарным файлом.

Бинарный файл — это формат представления данных с использованием двоичной системы счисления. В отличие от текстовых файлов, которые содержат информацию в виде символов (букв, цифр, спецсимволов и т. д.), такие форматы хранят информацию как последовательность битов (минимальных единиц информации, состоящих из 0 или 1). Двоичные файлы могут содержать данные разных типов, такие как числа (целые или с плавающей точкой), символы и многое другое.

Для класса «StudentsList» так же была создана структура «Student», которая позволяет хранить информацию об объектах нашего класса, а также реализовывать двусвязный список. Сам же класс «StudentList» хранит в себе различные функции, а также указатели на начало и конец списка.

private:

string filename;

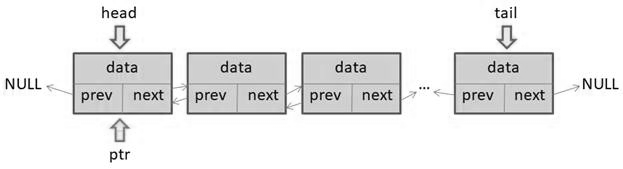
Student\* head;

Student\* tail;

*head* – указатель на начало списка;

*tail* – указатель на конец списка.

*filename* – имя файла, в котором могут быть сохранены последующие данные.



*Рис. 2.1.1 Структура данных*

***2.2 Структура информации о студенте***

Структурой для выбранной темы был выбран двусвязный список. Его реализация происхукпит в Двусвязный список – это структура данных, состоящая из узлов, каждый из которых содержит данные и ссылку на предыдущий и следующий узел. Так же структура «Student»,

string lastName;

string birthDate;

string admissionDate;

string expulsionDate;

string address;

string group;

Student\* next;

Student\* prev;

*lastName, birthdate*, *admissionDate*, *expulsionDate*, *address* и *group –* это поля, представленные в виде строки, хранящие в себе соответствующую информация – фамилию, день рождение, дату поступления, дату отчисления, адрес и группу соответственно.

*next* – указатель следующий элемент списка;

*prev* – указатель предыдущий элемент списка.

|  |  |  |
| --- | --- | --- |
| Поля-члены | Указатель на предыдущий элемент | Указатель на следующий элемент |

*Рис. 2.2.1 Структура объекта*

Данные записываются в бинарный файл последовательно. После каждого изменения в последовательности необходимо обновлять указатели на предыдущий и следующий элементы в списке.

## 3. ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ РАЗРАБОТКИ

Полный набор полей, описывающих основные характеристики объектов в «Student».

struct Student {

string lastName;

string birthDate;

string admissionDate;

string expulsionDate;

string address;

string group;

Student\* next;

Student\* prev;

Student() : lastName(""), birthDate(""), admissionDate(""),

expulsionDate(""), address(""), group("") {}

Student(const string& lastName, const string& birthDate,

const string& admissionDate, const string& expulsionDate,

const string& address, const string& group)

: lastName(lastName), birthDate(birthDate), admissionDate(admissionDate),

expulsionDate(expulsionDate), address(address), group(group),

next(nullptr), prev(nullptr) {}

};

***Конструкторы***

Student() : lastName(""), birthDate(""), admissionDate(""),

expulsionDate(""), address(""), group("") {}

Student(const string& lastName, const string& birthDate,

const string& admissionDate, const string& expulsionDate,

const string& address, const string& group)

: lastName(lastName), birthDate(birthDate), admissionDate(admissionDate),

expulsionDate(expulsionDate), address(address), group(group),

next(nullptr), prev(nullptr) {}

Для данной работы нам нужны два конструктора. Первый – по умолчанию, который инициализирует все поля-строки пустыми строками, для их дальнейшего заполнения данными. А второй – параметризованный, позволяет создать объект сразу же если известны все необходимые данные.

***Поля-члены для хранения информации об объекте***

string lastName;

string birthDate;

string admissionDate;

string expulsionDate;

string address;

string group;

Поля типа string, предназначенные для хранения данных о студенте.

***Указатели на другие объекты***

Student\* next;

Student\* prev;

Два поля-члена представляющие собой указатели на другие объекты «Student» и служащие для организации связанного списка студентов.

Полный набор операций, реализованный в виде методов класса «StudentsList».

class StudentsList {

public:

StudentsList(const string& filename);

~StudentsList();

void saveToFile();

void loadFromFile();

void addStudent(const Student& student);

void removeStudent(int index);

void editStudent(const string& surname, Student& newData);

void insertStudent(int index, const Student& student);

void displayStudents(int pageSize, int page);

void compressFile();

void searchStudentByIndex(int index);

void sortStudents();

};

***Конструктор и деструктор***

StudentsList(const string& filename) : filename(filename), head(nullptr), tail(nullptr) {}

~StudentsList() {

Student\* current = head;

while (current) {

Student\* next = current->next;

delete current;

current = next;

}}

Конструктор StudentsList(const string& filename) используется для инициализации объекта класса. Он принимает в качестве аргумента строку и сохраняет ее в соответствующем поле-члене объекта «StudentsList». Так же конструктор устанавливает значения указателей head и tail в nullptr. Это означает, что при создании нового объекта, список изначально пуст.

Деструктор ~StudentsList() просто уничтожает объекты «StudentsList».

***Сохранение в бинарный файл***

void saveToFile() {}

Этот метод предназначен для записи данных в бинарный файл, используя информацию о студентах из связного списка.

1. Сначала создается объект типа ofstream для открытия и записи в бинарный файл, имя которого хранится в переменной filename.
2. Затем проверяется, успешно ли был открыт бинарный файл. Если файл не удалось открыть, выводится сообщение об ошибке, и выполнение метода прекращается.
3. Создается указатель current, который указывает на голову списка студентов. Метод проходит по всем элементам списка, пока не достигнет конца (пока current не станет равным nullptr).
4. Для каждого студента, который не помечен как удаленный, выполняются следующие действия:

* Сначала вычисляется размер фамилии студента, который затем записывается в файл.
* После этого сама фамилия записывается в бинарный файл.
* Аналогично обрабатываются и записываются дата рождения, дата поступления, дата отчисления, адрес и группа студента. Для каждого из этих полей сначала записывается его размер, а затем само значение.

1. После завершения цикла записи данных файл закрывается.
2. Метод не выводит сообщение о завершении, но при успешной записи данных все данные студентов сохраняются в бинарном файле.

***Загрузка из бинарного файла***

void loadFromFile() {}

Метод loadFromFile() предназначен для загрузки данных студентов из бинарного файла и добавления их в связный список.

1. Сначала создается объект типа ifstream для открытия бинарного файла, имя которого хранится в переменной filename.
2. Затем проверяется, успешно ли был открыт файл. Если файл не удалось открыть, выводится сообщение об ошибке, и выполнение метода прекращается.
3. Метод продолжает читать данные из файла до тех пор, пока не достигнет конца файла (EOF). Для каждого студента выполняются следующие действия:

* Сначала считывается размер фамилии студента, который затем используется для выделения памяти под строку lastName.
* Фамилия считывается из файла и завершается нулевым символом (\0), чтобы корректно завершить строку.
* Аналогично считываются и обрабатываются дата рождения, дата поступления, дата отчисления, адрес и группа студента. Для каждого поля сначала считывается его размер, затем выделяется память и само значение считывается из файла.

1. После считывания всех данных создается новый объект Student, который инициализируется считанными значениями.
2. Новый студент добавляется в связный список с помощью метода addStudent().
3. После добавления студента освобождается память, выделенная для строк (фамилия, дата рождения и т. д.), чтобы избежать утечек памяти.
4. После завершения чтения данных файл закрывается.

***Удаление по индексу***

void removeStudent(int index) {}

Данный метод предназначен для удаления студента из связного списка по указанному индексу.

1. Сначала метод проверяет, является ли индекс допустимым (неотрицательным). Если индекс недопустим, выводится сообщение об ошибке.
2. Если список студентов пуст (указатель head равен nullptr), выводится сообщение о том, что список пуст, и выполнение метода прекращается.
3. Если индекс равен 0, то:

* Сохраняется указатель на текущий первый элемент (head).
* Указатель head обновляется на следующий элемент.
* Если после удаления первого элемента список не пуст, обновляется указатель prev у нового первого элемента.
* Если список стал пустым, указатель tail также устанавливается в nullptr.
* Удаляется временный указатель на старый первый элемент.
* Выводится сообщение о том, что студент с индексом 0 удален, и вызывается метод saveToFile() для обновления данных.

1. Если индекс не равен 0, метод ищет элемент, предшествующий удаляемому:

* Создается указатель current, указывающий на начало списка (head).
* Метод проходит по элементам списка до тех пор, пока не дойдет до элемента с индексом index - 1.

5. Если указатель current не существует или следующий элемент (current->next) отсутствует, выводится сообщение о том, что студент с указанным индексом не найден, и выполнение метода прекращается.

6. Если элемент найден:

* Сохраняется указатель на удаляемый элемент.
* Обновляется указатель next у предыдущего элемента, чтобы пропустить удаляемый элемент.
* Если удаляемый элемент был последним, обновляется указатель tail.
* Удаляется элемент, на который указывает toDelete.
* Выводится сообщение о том, что студент с указанным индексом удален, и вызывается метод saveToFile() для обновления данных.

***Добавление объекта(студента)***

void addStudent(const Student& student) {}

Данный метод предназначен для добавления нового студента в конец связного списка.

1. Метод принимает объект student по константной ссылке и создает новый объект Student в динамической памяти, копируя данные из переданного объекта. Это делается с помощью оператора new.
2. Если указатель tail указывает на последний элемент списка (т.е. список не пуст), новый студент добавляется в конец списка:
   * Указатель next последнего элемента (tail) обновляется, чтобы указывать на нового студента.
   * Указатель prev нового студента устанавливается на текущий последний элемент (tail).
   * Указатель tail обновляется, чтобы указывать на нового студента, который теперь стал последним элементом списка.
3. Если список был пуст (т.е. указатель tail равен nullptr), указатели head и tail обновляются, чтобы указывать на нового студента. Это означает, что новый студент стал как первым, так и последним элементом списка.
4. После добавления студента в список вызывается метод saveToFile(), который сохраняет текущее состояние списка в файл.

***Редактирование объекта(студента) по группе и шаблону фамилии***

void editStudent(const string& surname, const string& group, Student& newData) {}

Данный метод предназначен для редактирования данных студента в связном списке по указанной фамилии и группе.

1. Метод начинает с инициализации указателя current, который указывает на первый элемент списка (head).
2. Если указатель current равен nullptr, это означает, что список студентов пуст. В этом случае выводится сообщение об ошибке, и выполнение метода прекращается.
3. Поиск студента осуществляется так:
   * Переменная found инициализируется как false. Метод использует цикл do...while, чтобы пройти по всем элементам списка.
   * Внутри цикла происходит сравнение фамилии и группы текущего студента с искомыми значениями:
     + Если фамилия и группа совпадают, устанавливается флаг found в true.
     + Выводятся текущие данные студента, и пользователю предлагается ввести новые данные для редактирования.
4. Далее вводим новые данные по такой схеме:

* Для каждого поля (фамилия, дата рождения, дата поступления, дата отчисления, адрес и группа) выводится текущее значение, и пользователь может ввести новое значение.
* Для ввода адреса используется getline(cin, newData.address), что позволяет учитывать пробелы.

1. После ввода новых данных, поля текущего студента обновляются значениями из объекта newData.
2. Далее ыыводим сообщение о том, что данные были успешно обновлены, и вызываем метод saveToFile() для сохранения изменений в файл.
3. Если текущий студент не совпал с искомым, указатель current перемещается к следующему элементу списка.
4. Делаем проверку на наличие студента:
   * Цикл продолжается до тех пор, пока не вернется к началу списка (пока current не станет равным head).
   * Если после завершения цикла студент не был найден, выводится сообщение об ошибке с указанием, что студент с указанной фамилией не найден.

***Вставка объекта(студента) по индексу***

void insertStudent(int index, const Student& student) {}

Этот метод предназначен для вставки нового студента в связный список по указанному индексу.

1. Метод принимает индекс и объект student по константной ссылке, создавая новый объект Student в динамической памяти с помощью new.
2. Вставка в начало списка:
   * Если индекс равен 0, новый студент будет вставлен в начало списка:
   * Указатель next нового студента устанавливается на текущий head.
   * Если список не пуст, указатель prev первого элемента (head) обновляется, чтобы указывать на нового студента.
   * Указатель head обновляется, чтобы указывать на нового студента.
   * Если tail равен nullptr (то есть список был пуст), указатель tail также обновляется, чтобы указывать на нового студента.
3. Вставка в середину или конец списка:
   * Если индекс больше 0, метод начинает проходить по списку, используя указатель current, который инициализируется head, и счетчик count, инициализированный 0.
   * Цикл while продолжается до тех пор, пока current не станет nullptr и count не достигнет index - 1. Это позволяет найти элемент, после которого нужно вставить нового студента.
4. Вставка нового студента:
   * Если current не равен nullptr (т.е. мы нашли подходящую позицию):
   * Указатель next нового студента устанавливается на current->next.
   * Указатель prev нового студента устанавливается на current.
   * Если current->next не равен nullptr, обновляется указатель prev следующего элемента, чтобы указывать на нового студента.
   * Если current->next равен nullptr, это означает, что новый студент будет последним элементом, поэтому указатель tail обновляется.
   * Указатель next текущего студента (current) обновляется, чтобы указывать на нового студента.
5. Обработка случая, когда индекс больше размера списка:
   * Если current равен nullptr, это означает, что индекс больше текущего размера списка. В этом случае:
   * Если tail не равен nullptr, новый студент добавляется в конец списка, обновляя указатели next и prev.
   * Если список пуст, указатели head и tail обновляются, чтобы указывать на нового студента.
6. После вставки нового студента вызывается метод saveToFile(), который сохраняет текущее состояние списка в файл.

***Постраничный просмотр списка объектов(студентов)***

void displayStudents(int pageSize, int page) {}

Этот метод отвечает за отображение списка студентов на экране с постраничным выводом.

1. Открытие файла:

* Метод открывает бинарный файл filename для чтения.
* Если файл не удается открыть, выводится сообщение об ошибке и выполнение метода прекращается.

2. Инициализация переменных

* size\_t position = 0; — переменная для отслеживания текущей позиции в файле.
* int count = 0; — счетчик для подсчета пропущенных студентов.

3. Пропуск до нужной страницы

* Используется цикл while, чтобы пропустить студентов до указанной страницы:

while (count < (page - 1) \* pageSize && file.peek() != EOF) {

* Внутри цикла:
  + Считывается размер фамилии и пропускается соответствующее количество байтов.
  + Аналогичным образом считываются размеры и пропускаются данные для даты рождения, даты поступления, даты отчисления, адреса и группы.
* После каждого пропуска count увеличивается на единицу.

4. После пропуска студентов count сбрасывается на ноль для начала чтения студентов на текущей странице.

5. Чтение и вывод студентов

Второй цикл while считывает и выводит студентов на экран. Для каждого студента:

* + Считываются размеры и данные для фамилии, даты рождения, даты поступления, даты отчисления, адреса и группы:
  + Для каждой строки выделяется память, данные считываются из файла, и добавляется нуль-терминатор.
  + Информация о студенте выводится на экран с помощью cout.
  + Освобождается выделенная память с помощью delete[].

6. После вывода информации о студенте обновляется position, чтобы перейти к следующему студенту. Это делается путем добавления размеров всех полей и их указателей.

7. После завершения всех операций файл закрывается.

***Извлечение строки***

void searchStudentByIndex(int index) {}

Этот метод ищет студента по заданному индексу и выводит его информацию на экран.

1. Открывает файл. Если файл успешно открыт, продолжается выполнение метода. В противном случае, выведется сообщение об ошибке (проверка на наличие файла не включена в код, но это можно добавить).
2. Инициализируем переменную позиции: size\_t position = 0; — переменная для отслеживания текущей позиции в файле.
3. Пропускаем студентов до указанного индекса. Используем цикл for, чтобы пропустить студентов до указанного индекса:

* Считываются размеры фамилии, даты рождения, даты поступления, даты отчисления, адреса и группы.
* После каждого считывания position обновляется для перехода к следующему студенту.

1. После завершения цикла, данные о студенте с заданным индексом считываются:

* Сначала считывается размер фамилии, затем выделяется память для строки и считывается сама фамилия.
* Аналогично считываются и другие поля: дата рождения, дата поступления, дата отчисления, адрес и группа.
* Для каждой строки добавляется нуль-терминатор.

1. Далее выводим информацию о студенте и освобождаем память, выделенную для строк.
2. После завершения всех операций закрываем файл.

***Функция сортировка***

void sortStudents() {}

Этот метод сортирует список студентов по фамилии в алфавитном порядке, используя алгоритм пузырьковой сортировки.

1. Метод сначала проверяет, пуст ли список или состоит ли он из одного элемента. Если условие выполняется, выполнение метода прекращается, так как нечего сортировать.
2. Далее булевой переменной swapped отслеживаем уже отсортированные элементы. А не отмеченные – сортируем.
3. Сама сортировка происходит таким образом:

* swapped устанавливается в false для начала нового прохода.
* Student\* current = head; — указатель на текущего студента.
* Student\* prev = nullptr; — указатель на предыдущего студента.
* Вложенный цикл while (current->next) проходит по списку:
* Сравниваются фамилии текущего студента и следующего
* Если фамилия текущего студента больше, выполняется обмен:

1. Сохраняется указатель на следующего студента.
2. Обновляются связи между студентами:

* Если prev не равен nullptr, обновляется ссылка на следующий элемент для prev.
* Если prev равен nullptr, обновляется head на nextStudent.

– Обновляются указатели для nextStudent и current.

* Если обмен был выполнен, swapped устанавливается в true, чтобы продолжить сортировку.

4. После завершения сортировки указатель tail обновляется, чтобы указывать на последний элемент списка.

5. А после сортировки список сохраняется в файл.

***Сжатие файла***

void compressFile() {}

Этот метод сжимает файл данных студентов, создавая временный файл для хранения данных в более компактном формате.

1. Метод начинает с открытия исходного файла в бинарном режиме. Если файл не удается открыть, выводится сообщение об ошибке, и выполнение метода завершается.
2. Создается временный файл для записи сжатых данных. Если временный файл не удается открыть, выводится сообщение об ошибке, и метод также завершает выполнение.
3. Метод использует цикл, который продолжается до конца файла. Внутри цикла происходит следующее:

* Для каждого поля (фамилия, дата рождения, дата поступления и т.д.) считывается размер строки, выделяется память для неё, и затем считывается содержимое.
* После этого размер и содержимое записываются в выходной файл.

1. После записи данных о студенте освобождается память, выделенная для всех строк, чтобы избежать утечек памяти.
2. По завершении чтения и записи оба файла закрываются. Старый файл удаляется, а временный файл переименовывается в оригинальное имя. Метод выводит сообщение о завершении сжатия файла.

***Меню***

void displayMenu**()** **{**

cout **<<** "\nMenu:" **<<** endl**;**

cout **<<** "1. Add a student\n"**;**

cout **<<** "2. Delete a student\n"**;**

cout **<<** "3. Edit\n"**;**

cout **<<** "4. Insert student by index\n"**;**

cout **<<** "5. Display list\n"**;**

cout **<<** "6. Compress the file\n"**;**

cout **<<** "7. Remove student by index\n"**;**

cout **<<** "8. Sort list\n"**;**

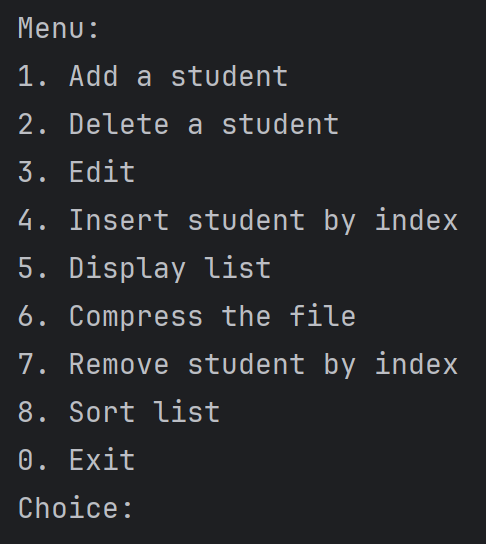
cout **<<** "0. Exit\n"**;**

**}**

Позволяет пользователю увидеть все возможные функции

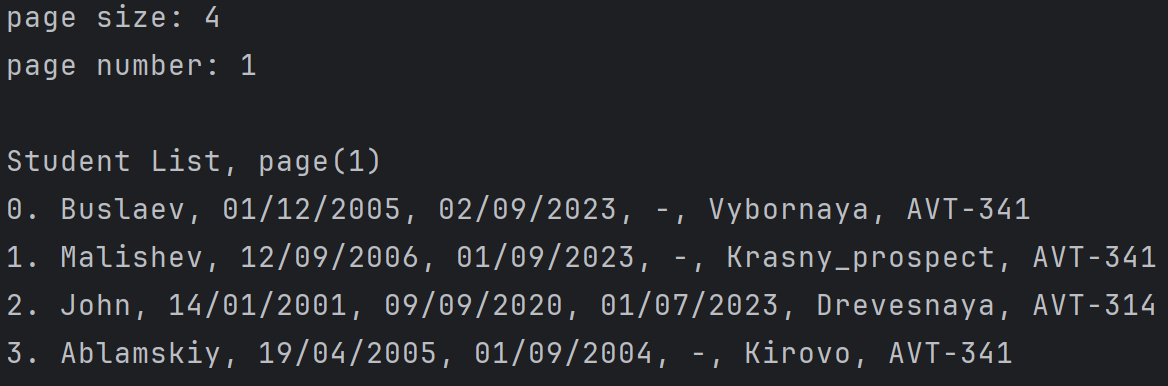
## 4. ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Пользовательским интерфейсом является функция displayMenu(). При ее вызове пользователь может увидеть список функций, которые можно осуществить над текстом.



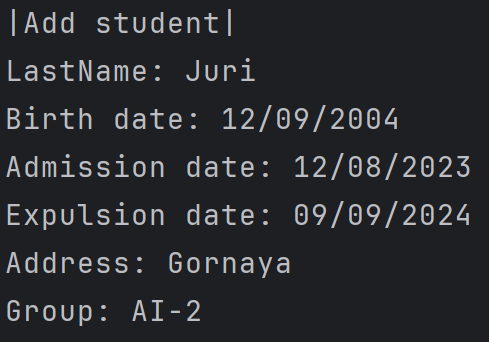
*Рис. 4.1 Меню*

В бинарном файле заранее располагается текст. Для того чтобы вывести его на экран, необходимо нажать на [5] и ввести сколько нужно вывести студентов на одной странице и номер страницу.

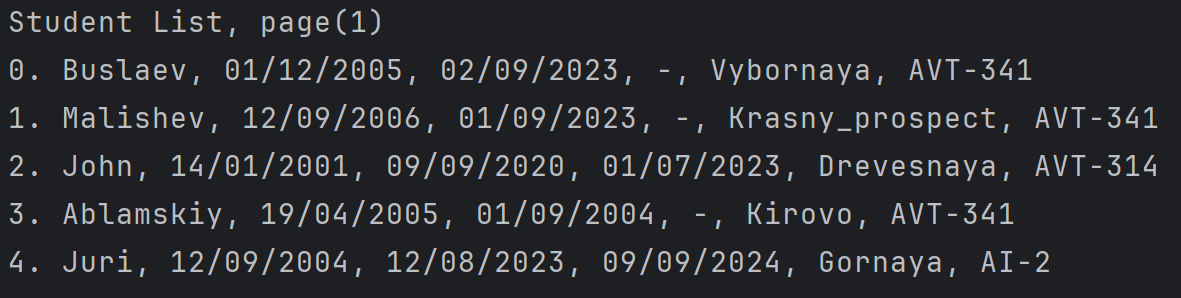
**

*Рис. 4.2 Постраничный вывод в консоль*

Далее нажмем на [1] чтобы добавить нового студента в конец списка. И получаем следующий результат:

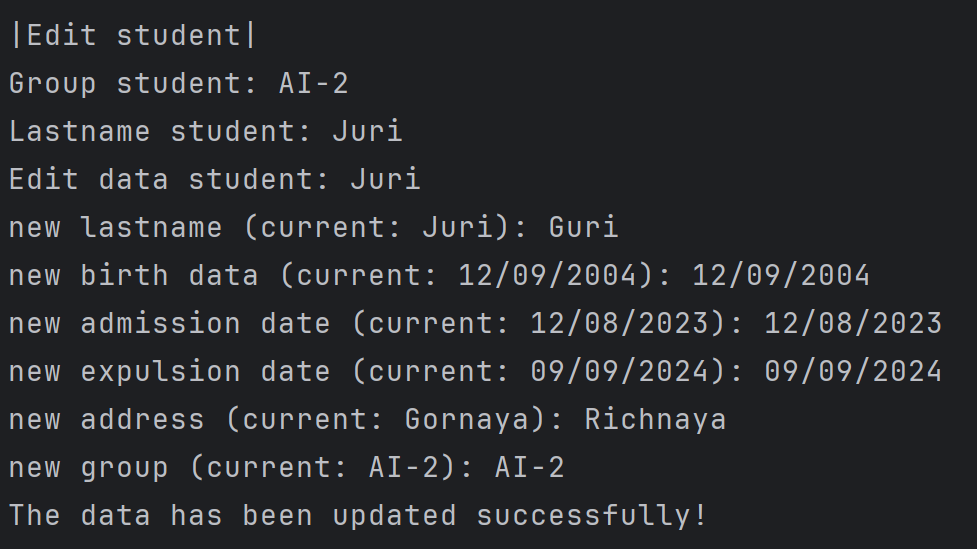
****

*Рис. 4.3 Функция добавления*

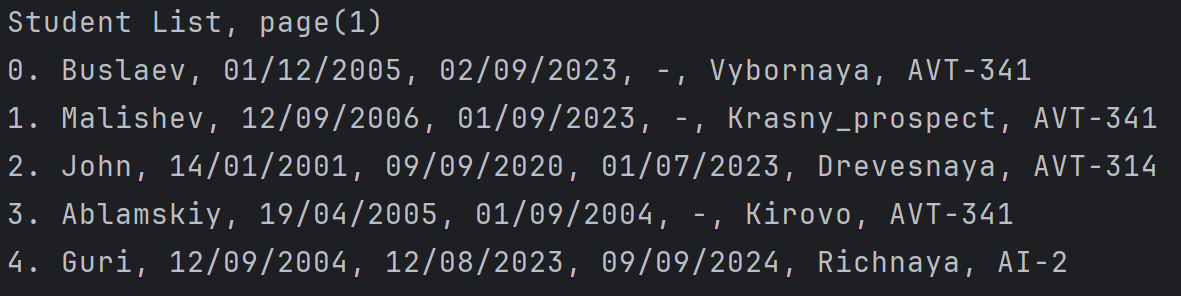
**

*Рис. 4.4 Результат добавления*

Теперь отредактируем только что добавленного студента. Для этого нажмем [3].

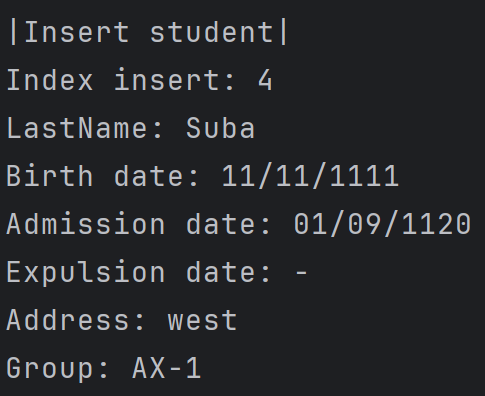
****

*Рис. 4.5 Функция изменения студента*

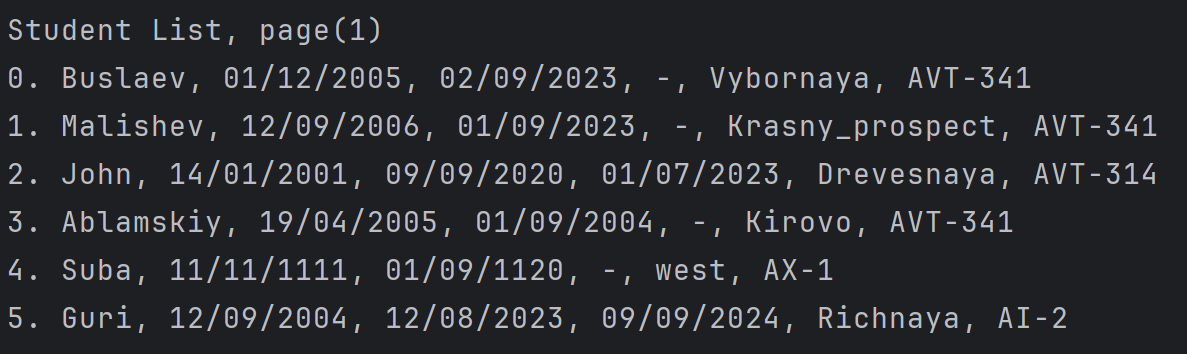
**

*Рис. 4.6 Результаты редактирования*

Теперь вставим нового студента на позицию номер 4. Для этого нажмем [4].

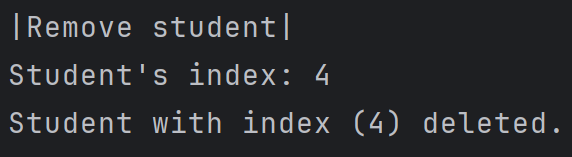


*Рис. 4.5 Функция вставки студента по индексу*

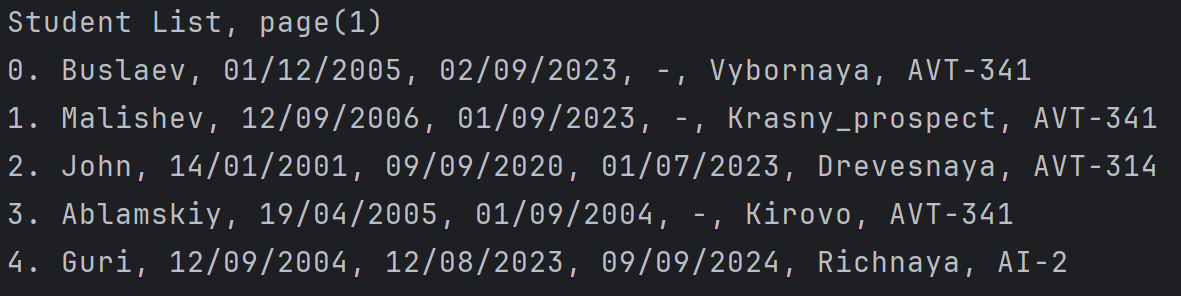
**

*Рис. 4.6. Результат вставки*

Теперь удалим только, что вставленного студента, путем вызова функции [2].

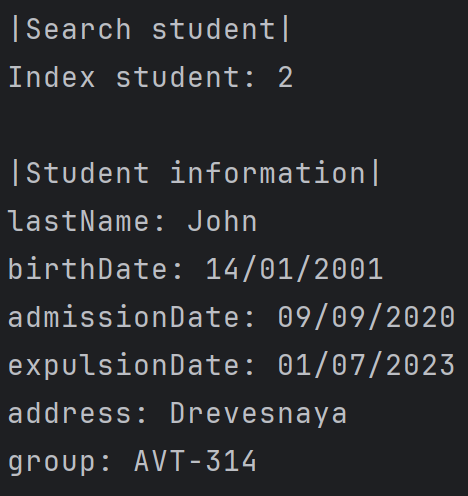
****

*Рис 4.7 Функция добавление строки в конец*

**

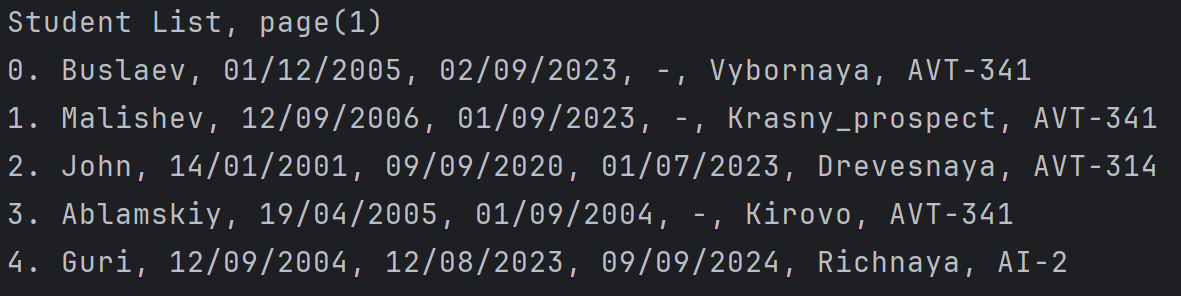
*Рис. 4.8 Результаты удаления*

Найдем информацию о студенте под номером 2. Для этого нажимаем [7].

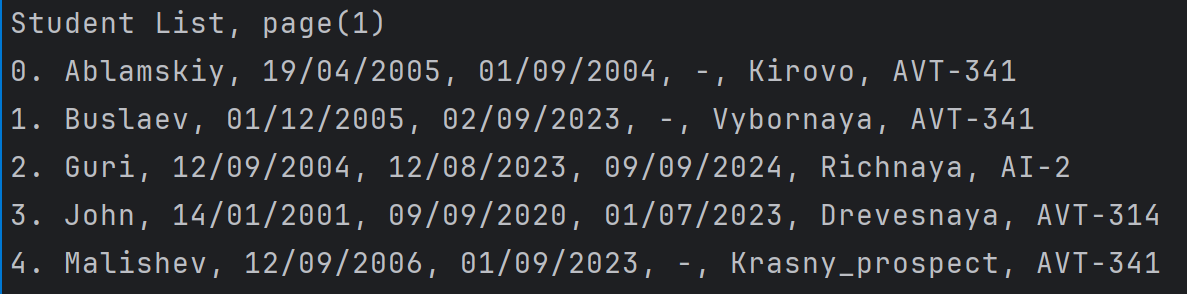
****

*Рис 4.9 Функция извлечения строки(поиска студента)*

Перед последней функцией отсортируем список по фамилии студентов, путем выбора функции, нажав [8].

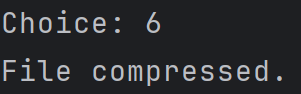
**

*Рис 4.10 Неотсортированный список*

**

*Рис. 4.11 Результат сортировки*

Осталась последняя функция – сжатие бинарного файла. Вводим [6].



*Рис 4.12 Результат сортировки*

Открыв бинарный файл в проводнике, можно убедиться, что файл был заменен на новый.

## 5. КОНТРОЛЬНЫЕ ПРИМЕРЫ

Для тестирования работы программы были использованы контрольные примеры с большим количеством случайно сгенерированных записей о студентах. В частности, проводилось тестирование на списках размером от 100 000 до 1 000 000 элементов. Записи генерировались случайным образом с использованием генератора псевдослучайных чисел и набора шаблонов для данных.

При генерации данных случайно выбирались:

* фамилия из списка из 20 000 вариантов
* дата рождения из диапазона с 1940 по 2005 год
* дата поступления из диапазона за последние 50 лет
* дата отчисления либо актуальная дата, либо случайная за последние 10 лет
* адрес из набора из 10 000 городов и 1 000 000 улиц
* номер группы по шаблону ХХХ-YY, где ХХХ - случайный код факультета от 100 до 999, YY - номер группы от 1 до 99.

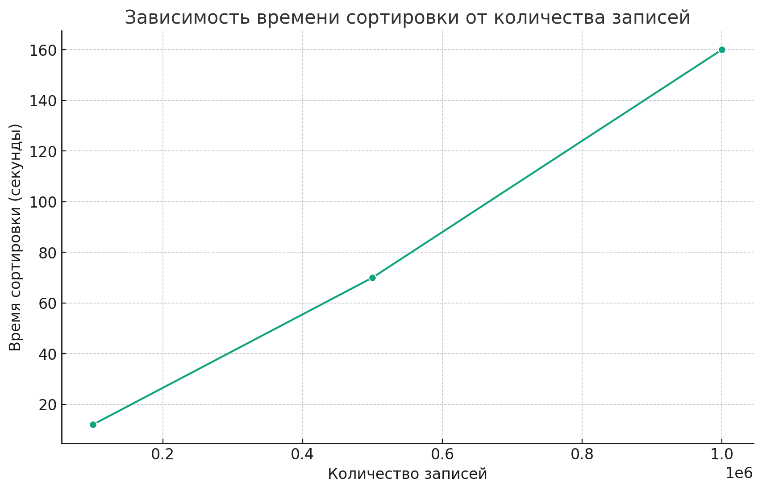
В результате получались файлы данных от 400 МБ до 4 ГБ.

При тестировании на таких данных наблюдалась следующая производительность:

***Сортировка списка по фамилии***

* для 100 000 записей - 12 секунд
* для 500 000 записей - 1 минута 10 секунд
* для 1 000 000 записей - 2 минуты 40 секунд

Таким образом, асимптотическая сложность алгоритма сортировки составляет О(n^2) - квадратичная. Ниже показаны результаты в графической форме.



*Рис. 5.1 Зависимость времени сортировки от количества записей*

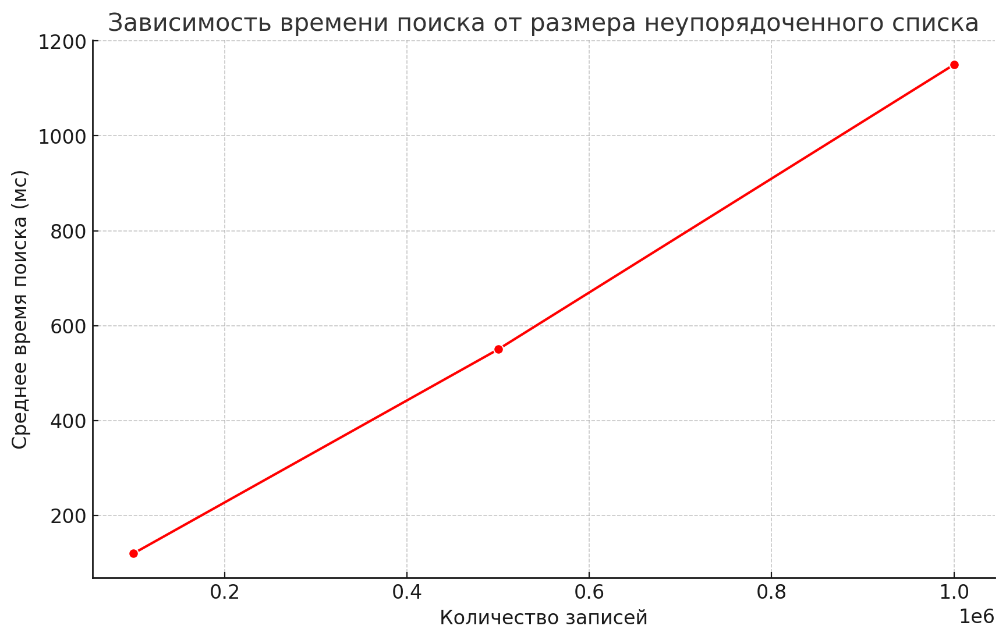
С увеличением числа элементов в 5 раз время сортировки увеличивается примерно в 5-6 раз.

***Поиск записи в неупорядоченном списке по индексу***

Было проведено 1000 поисков различных фамилий в списках разного размера:

* для 100 000 записей - среднее время поиска 120 мс
* для 500 000 записей - среднее время поиска 550 мс
* для 1 000 000 записей - среднее время поиска 1150 мс

Можно видеть, что время поиска увеличивается примерно линейно с ростом списка. То есть алгоритм имеет сложность O(n) для поиска в неупорядоченном списке.



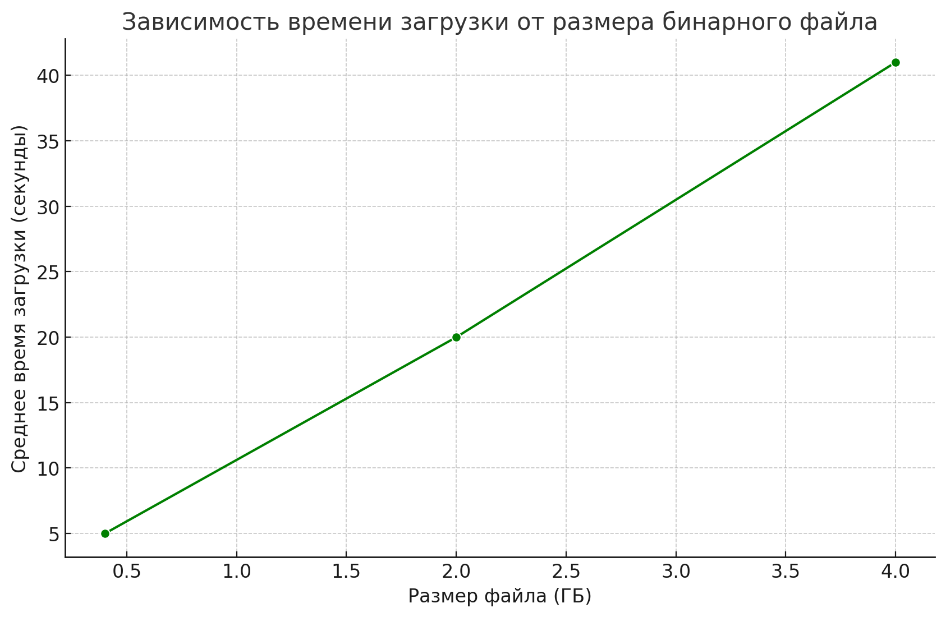
*Рис. 5.2 Зависимость времени поиска от размера неупорядоченного списка*

***Загрузка данных из бинарного файла***

При загрузке данных объемом в 1 ГБ из бинарного файла наблюдалось следующее время:

* для файла 400 МБ - 4-6 секунд
* для файла 2 ГБ - 18-22 секунд
* для файла 4 ГБ - 38-44 секунд

Видна тенденция к линейному увеличению времени загрузки с ростом размера файла.



*Рис. 5.3 Зависимость времени загрузки от размера файла*

Таким образом, проведенные тесты на больших объемах данных показывают, что программа демонстрирует стабильную работу и приемлемую производительность на десятках и сотнях тысяч элементов при использовании типичных операций по работе с двусвязным списком.

## ВЫВОДЫ

В ходе выполнения данной работы была реализована программа для ведения и редактирования базы данных студентов с использованием двусвязного списка в качестве основной структуры хранения информации. Были детально проработаны алгоритмы основных операций над односвязным списком, таких как:

* Добавление элемента конец списка
* Вставка элемента по номеру
* Редактирование по группе и шаблону фамилии
* Удаление элемента по номеру
* Поиск элемента по номеру
* Сортировка списка по фамилии
* Постраничный вывод на экран
* Удаление элемента по номеру

Реализация этих алгоритмов позволяет эффективно манипулировать большим количеством данных о студентах. Дополнительно были проработаны вопросы долговременного хранения данных в бинарных файлах. Интерфейс программы выполнен в виде текстового меню, удобного для выбора и выполнения всех необходимых операций пользователем. При тестировании на больших объемах данных (сотни тысяч и миллионы записей) программа показала стабильную работу и достаточную производительность.

В дальнейшем возможно расширение функционала за счет реализации более сложных алгоритмов сортировки и поиска, а также создания графического интерфейса программы.

В целом же поставленная задача была полностью решена, и программа может использоваться для практического применения в учебных заведениях для учета контингента студентов и их успеваемости.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Т. А. Павловская, «С/С++. Программирование на языке высокого уровня» // Учебник для ВУЗов.
2. Р. Седжвик, «Фундаментальные алгоритмы на С++. Анализ» // - М.: Вильямс, 2018.
3. Б. Керниган, Д. Ритчи, «Язык программирования C» // - М.: Вильямс, 2022.
4. М. Фаулер, «Рефакторинг: улучшение существующего кода» // - СПб.: Символ Плюс, 2023.
5. К. Брукшер, «Чистый код: создание, анализ и рефакторинг» // - СПб.: Питер, 2016.
6. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн, «Алгоритмы: построение и анализ» // - М.: Вильямс, 2018.
7. Д. Кнут, «Искусство программирования» // - М.: Вильямс, 2016. - Т. 1-3.
8. Р. Седжвик, «Алгоритмы на С++. 5-е издание» // - М.: Вильямс, 2017.
9. Р. Лафоре, «Объектно-ориентированное программирование в С++» // - СПб.: Питер, 2020.
10. Бинарный файл // Blog-skillfactory.ru [Электронный ресурс]: URL: https://blog.skillfactory.ru/glossary/binarnyy-fayl/

## ПРИЛОЖЕНИЕ

#ifndef KURSOVAYA\_2\_8\_4\_STUDENT\_H

#define KURSOVAYA\_2\_8\_4\_STUDENT\_H

**using** **namespace** std**;**

#include <string>

struct Student **{**

string lastName**;**

string birthDate**;**

string admissionDate**;**

string expulsionDate**;**

string address**;**

string group**;**

Student**\*** next**;**

Student**\*** prev**;**

Student**()** **:** lastName**(**""**),** birthDate**(**""**),** admissionDate**(**""**),**

expulsionDate**(**""**),** address**(**""**),** group**(**""**)** **{}**

Student**(**const string**&** lastName**,** const string**&** birthDate**,**

const string**&** admissionDate**,** const string**&** expulsionDate**,**

const string**&** address**,** const string**&** group**)**

**:** lastName**(**lastName**),** birthDate**(**birthDate**),** admissionDate**(**admissionDate**),**

expulsionDate**(**expulsionDate**),** address**(**address**),** group**(**group**),**

next**(nullptr),** prev**(nullptr)** **{}**

**};**

#endif

#ifndef KURSOVAYA\_2\_8\_4\_STUDENTLIST\_H

#define KURSOVAYA\_2\_8\_4\_STUDENTLIST\_H

#include "../Student/Student.h"

#include <fstream>

#include <vector>

#include <iostream>

#include <iomanip>

#include <cstring>

class StudentsList **{**

public**:**

StudentsList**(**const string**&** filename**);**

**~**StudentsList**();**

void saveToFile**();**

void loadFromFile**();**

void addStudent**(**const Student**&** student**);**

void removeStudent**(**int index**);**

void editStudent**(**const string**&** surname**,** const string**&** group**,** Student**&** newData**);**

void insertStudent**(**int index**,** const Student**&** student**);**

void displayStudents**(**int pageSize**,** int page**);**

void compressFile**();**

void searchStudentByIndex**(**int index**);**

void sortStudents**();**

private**:**

string filename**;**

Student**\*** head**;**

Student**\*** tail**;**

**};**

#endif

#include "../List/StudentList.h"

StudentsList**::**StudentsList**(**const string**&** filename**)** **:** filename**(**filename**),** head**(nullptr),** tail**(nullptr)** **{}**

StudentsList**::~**StudentsList**()** **{**

Student**\*** current **=** head**;**

**while** **(**current**)** **{**

Student**\*** next **=** current**->**next**;**

**delete** current**;**

current **=** next**;**

**}**

**}**

void StudentsList**::**saveToFile**()** **{**

ofstream file**(**filename**,** ios**::**binary**);**

**if** **(!**file**)** **{**

cerr **<<** "Error opening file!" **<<** endl**;**

**return;**

**}**

Student**\*** current **=** head**;**

**while** **(**current **!=** **nullptr)** **{**

// Сохраняем только студентов, которые не помечены как удаленные

size\_t lastNameSize **=** current**->**lastName**.**size**();**

file**.**write**(reinterpret\_cast<**const char**\*>(&**lastNameSize**),** **sizeof(**size\_t**));**

file**.**write**(**current**->**lastName**.**c\_str**(),** lastNameSize**);**

size\_t birthDateSize **=** current**->**birthDate**.**size**();**

file**.**write**(reinterpret\_cast<**const char**\*>(&**birthDateSize**),** **sizeof(**size\_t**));**

file**.**write**(**current**->**birthDate**.**c\_str**(),** birthDateSize**);**

size\_t admissionDateSize **=** current**->**admissionDate**.**size**();**

file**.**write**(reinterpret\_cast<**const char**\*>(&**admissionDateSize**),** **sizeof(**size\_t**));**

file**.**write**(**current**->**admissionDate**.**c\_str**(),** admissionDateSize**);**

size\_t expulsionDateSize **=** current**->**expulsionDate**.**size**();**

file**.**write**(reinterpret\_cast<**const char**\*>(&**expulsionDateSize**),** **sizeof(**size\_t**));**

file**.**write**(**current**->**expulsionDate**.**c\_str**(),** expulsionDateSize**);**

size\_t addressSize **=** current**->**address**.**size**();**

file**.**write**(reinterpret\_cast<**const char**\*>(&**addressSize**),** **sizeof(**size\_t**));**

file**.**write**(**current**->**address**.**c\_str**(),** addressSize**);**

size\_t groupSize **=** current**->**group**.**size**();**

file**.**write**(reinterpret\_cast<**const char**\*>(&**groupSize**),** **sizeof(**size\_t**));**

file**.**write**(**current**->**group**.**c\_str**(),** groupSize**);**

current **=** current**->**next**;**

**}**

file**.**close**();**

**}**

void StudentsList**::**loadFromFile**()** **{**

ifstream file**(**filename**,** ios**::**binary**);**

**if** **(!**file**)** **{**

cerr **<<** "Error reading!" **<<** endl**;**

**return;**

**}**

**while** **(**file**.**peek**()** **!=** EOF**)** **{**

size\_t lastNameSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**lastNameSize**),** **sizeof(**size\_t**));**

char**\*** lastName **=** **new** char**[**lastNameSize **+** 1**];**

file**.**read**(**lastName**,** lastNameSize**);**

lastName**[**lastNameSize**]** **=** '\0'**;**

size\_t birthDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**birthDateSize**),** **sizeof(**size\_t**));**

char**\*** birthDate **=** **new** char**[**birthDateSize **+** 1**];**

file**.**read**(**birthDate**,** birthDateSize**);**

birthDate**[**birthDateSize**]** **=** '\0'**;**

size\_t admissionDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**admissionDateSize**),** **sizeof(**size\_t**));**

char**\*** admissionDate **=** **new** char**[**admissionDateSize **+** 1**];**

file**.**read**(**admissionDate**,** admissionDateSize**);**

admissionDate**[**admissionDateSize**]** **=** '\0'**;**

size\_t expulsionDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**expulsionDateSize**),** **sizeof(**size\_t**));**

char**\*** expulsionDate **=** **new** char**[**expulsionDateSize **+** 1**];**

file**.**read**(**expulsionDate**,** expulsionDateSize**);**

expulsionDate**[**expulsionDateSize**]** **=** '\0'**;**

size\_t addressSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**addressSize**),** **sizeof(**size\_t**));**

char**\*** address **=** **new** char**[**addressSize **+** 1**];**

file**.**read**(**address**,** addressSize**);**

address**[**addressSize**]** **=** '\0'**;**

size\_t groupSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**groupSize**),** **sizeof(**size\_t**));**

char**\*** group **=** **new** char**[**groupSize **+** 1**];**

file**.**read**(**group**,** groupSize**);**

group**[**groupSize**]** **=** '\0'**;**

// Создание нового студента

Student newStudent**(**lastName**,** birthDate**,** admissionDate**,** expulsionDate**,** address**,** group**);**

addStudent**(**newStudent**);**

**delete[]** lastName**;**

**delete[]** birthDate**;**

**delete[]** admissionDate**;**

**delete[]** expulsionDate**;**

**delete[]** address**;**

**delete[]** group**;**

**}**

file**.**close**();**

**}**

void StudentsList**::**addStudent**(**const Student**&** student**)** **{**

Student**\*** newStudent **=** **new** Student**(**student**);**

**if** **(**tail**)** **{**

tail**->**next **=** newStudent**;**

newStudent**->**prev **=** tail**;**

tail **=** newStudent**;**

**}** **else** **{**

head **=** tail **=** newStudent**;** // список был пуст

**}**

saveToFile**();**

**}**

void StudentsList**::**removeStudent**(**int index**)** **{**

**if** **(**index **<** 0**)** **{**

cerr **<<** "Index invalid." **<<** endl**;**

**return;**

**}**

// Если список пуст, ничего не делаем

**if** **(!**head**)** **{**

cerr **<<** "Student list is empty." **<<** endl**;**

**return;**

**}**

// Если удаляется первый элемент

**if** **(**index **==** 0**)** **{**

Student**\*** temp **=** head**;**

head **=** head**->**next**;**

**if** **(**head**)** **{**

head**->**prev **=** **nullptr;**

**}** **else** **{**

tail **=** **nullptr;**

**}**

**delete** temp**;**

cout **<<** "Student with index (0) deleted." **<<** endl**;**

saveToFile**();**

**return;**

**}**

// Поиск элемента для удаления

Student**\*** current **=** head**;**

int count **=** 0**;**

**while** **(**current **&&** count **<** index **-** 1**)** **{**

current **=** current**->**next**;**

count**++;**

**}**

// Если элемент не найден

**if** **(!**current **||** **!**current**->**next**)** **{**

cerr **<<** "Student with index (" **<<** index **<<** ") not found." **<<** endl**;**

**return;**

**}**

// Удаление элемента

Student**\*** toDelete **=** current**->**next**;**

current**->**next **=** toDelete**->**next**;**

**if** **(**toDelete**->**next**)** **{**

toDelete**->**next**->**prev **=** current**;**

**}** **else** **{**

tail **=** current**;**

**}**

**delete** toDelete**;**

cout **<<** "Student with index (" **<<** index **<<** ") deleted." **<<** endl**;**

saveToFile**();**

**}**

void StudentsList**::**editStudent**(**const string**&** surname**,** const string**&** group**,** Student**&** newData**)** **{**

Student**\*** current **=** head**;**

// Проверяем, что список не пуст

**if** **(**current **==** **nullptr)** **{**

cerr **<<** "List student empty!" **<<** endl**;**

**return;**

**}**

bool found **=** **false;**

**do** **{**

// Сравниваем фамилию и группу текущего студента с искомой

**if** **(**current**->**group **==** group **&&** current**->**lastName **==** surname**)** **{**

found **=** **true;**

// Выводим текущие данные студента

cout **<<** "Edit data student: " **<<** current**->**lastName **<<** endl**;**

// Обновляем данные студента

cout **<<** "new lastname (current: " **<<** current**->**lastName **<<** "): "**;**

cin **>>** newData**.**lastName**;**

cout **<<** "new birth data (current: " **<<** current**->**birthDate **<<** "): "**;**

cin **>>** newData**.**birthDate**;**

cout **<<** "new admission date (current: " **<<** current**->**admissionDate **<<** "): "**;**

cin **>>** newData**.**admissionDate**;**

cout **<<** "new expulsion date (current: " **<<** current**->**expulsionDate **<<** "): "**;**

cin **>>** newData**.**expulsionDate**;**

cout **<<** "new address (current: " **<<** current**->**address **<<** "): "**;**

cin**.**ignore**();** // Игнорируем остаток строки после предыдущего ввода

getline**(**cin**,** newData**.**address**);** // Используем getline для ввода адреса

cout **<<** "new group (current: " **<<** current**->**group **<<** "): "**;**

cin **>>** newData**.**group**;**

// Обновляем данные студента

current**->**lastName **=** newData**.**lastName**;**

current**->**group **=** newData**.**group**;**

current**->**birthDate **=** newData**.**birthDate**;**

current**->**admissionDate **=** newData**.**admissionDate**;**

current**->**expulsionDate **=** newData**.**expulsionDate**;**

current**->**address **=** newData**.**address**;**

cout **<<** "The data has been updated successfully!" **<<** endl**;**

saveToFile**();**

**return;**

**}**

current **=** current**->**next**;** // Переходим к следующему студенту

**}** **while** **(**current **!=** head**);** // Цикл продолжается, пока не вернемся к началу

cerr **<<** "Student lastname '" **<<** surname **<<** "' not found!" **<<** endl**;**

**}**

void StudentsList**::**insertStudent**(**int index**,** const Student**&** student**)** **{**

Student**\*** newStudent **=** **new** Student**(**student**);**

**if** **(**index **==** 0**)** **{**

newStudent**->**next **=** head**;**

**if** **(**head**)** **{**

head**->**prev **=** newStudent**;**

**}**

head **=** newStudent**;**

**if** **(!**tail**)** **{**

tail **=** newStudent**;**

**}**

**}** **else** **{**

Student**\*** current **=** head**;**

int count **=** 0**;**

**while** **(**current **!=** **nullptr** **&&** count **<** index **-** 1**)** **{**

current **=** current**->**next**;**

count**++;**

**}**

**if** **(**current**)** **{**

newStudent**->**next **=** current**->**next**;**

newStudent**->**prev **=** current**;**

**if** **(**current**->**next**)** **{**

current**->**next**->**prev **=** newStudent**;**

**}** **else** **{**

tail **=** newStudent**;**

**}**

current**->**next **=** newStudent**;**

**}** **else** **{**

**if** **(**tail**)** **{**

tail**->**next **=** newStudent**;**

newStudent**->**prev **=** tail**;**

tail **=** newStudent**;**

**}** **else** **{**

// Если список пуст

head **=** newStudent**;**

tail **=** newStudent**;**

**}**

**}**

**}**

// Обновление файла после вставки студента

saveToFile**();**

**}**

void StudentsList**::**displayStudents**(**int pageSize**,** int page**)** **{**

ifstream file**(**filename**,** ios**::**binary**);**

**if** **(!**file**)** **{**

cerr **<<** "error open file." **<<** endl**;**

**return;**

**}**

size\_t position **=** 0**;**

int count **=** 0**;**

// Пропускаем до нужной страницы

**while** **(**count **<** **(**page **-** 1**)** **\*** pageSize **&&** file**.**peek**()** **!=** EOF**)** **{**

size\_t lastNameSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**lastNameSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** lastNameSize**;** // Пропустить фамилию

// Пропускаем остальные данные

size\_t birthDateSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**birthDateSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** birthDateSize**;**

size\_t admissionDateSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**admissionDateSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** admissionDateSize**;**

size\_t expulsionDateSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**expulsionDateSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** expulsionDateSize**;**

size\_t addressSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**addressSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** addressSize**;**

size\_t groupSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**groupSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** groupSize**;**

count**++;**

**}**

count **=** 0**;**

// Чтение и вывод студентов на экран

**while** **(**count **<** pageSize **&&** file**.**peek**()** **!=** EOF**)** **{**

size\_t lastNameSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**lastNameSize**),** **sizeof(**size\_t**));**

char**\*** lastName **=** **new** char**[**lastNameSize **+** 1**];**

file**.**read**(**lastName**,** lastNameSize**);**

lastName**[**lastNameSize**]** **=** '\0'**;**

size\_t birthDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**birthDateSize**),** **sizeof(**size\_t**));**

char**\*** birthDate **=** **new** char**[**birthDateSize **+** 1**];**

file**.**read**(**birthDate**,** birthDateSize**);**

birthDate**[**birthDateSize**]** **=** '\0'**;**

size\_t admissionDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**admissionDateSize**),** **sizeof(**size\_t**));**

char**\*** admissionDate **=** **new** char**[**admissionDateSize **+** 1**];**

file**.**read**(**admissionDate**,** admissionDateSize**);**

admissionDate**[**admissionDateSize**]** **=** '\0'**;**

size\_t expulsionDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**expulsionDateSize**),** **sizeof(**size\_t**));**

char**\*** expulsionDate **=** **new** char**[**expulsionDateSize **+** 1**];**

file**.**read**(**expulsionDate**,** expulsionDateSize**);**

expulsionDate**[**expulsionDateSize**]** **=** '\0'**;**

size\_t addressSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**addressSize**),** **sizeof(**size\_t**));**

char**\*** address **=** **new** char**[**addressSize **+** 1**];**

file**.**read**(**address**,** addressSize**);**

address**[**addressSize**]** **=** '\0'**;**

size\_t groupSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**groupSize**),** **sizeof(**size\_t**));**

char**\*** groupName **=** **new** char**[**groupSize **+** 1**];**

file**.**read**(**groupName**,** groupSize**);**

groupName**[**groupSize**]** **=** '\0'**;**

// Вывод информации о студенте

cout **<<** count **<<** ". " **<<** lastName

**<<** ", " **<<** birthDate

**<<** ", " **<<** admissionDate

**<<** ", " **<<** expulsionDate

**<<** ", " **<<** address

**<<** ", " **<<** groupName **<<** endl**;**

**delete[]** lastName**;**

**delete[]** birthDate**;**

**delete[]** admissionDate**;**

**delete[]** expulsionDate**;**

**delete[]** address**;**

**delete[]** groupName**;**

// Обновление позиции для следующего студента

position **+=** **sizeof(**size\_t**)** **+** lastNameSize **+** **sizeof(**size\_t**)** **+** birthDateSize **+**

**sizeof(**size\_t**)** **+** admissionDateSize **+** **sizeof(**size\_t**)** **+** expulsionDateSize **+**

**sizeof(**size\_t**)** **+** addressSize **+** **sizeof(**size\_t**)** **+** groupSize**;**

count**++;**

**}**

file**.**close**();**

**}**

void StudentsList**::**compressFile**()** **{**

ifstream inputFile**(**filename**,** ios**::**binary**);**

**if** **(!**inputFile**)** **{**

cerr **<<** "Error open!" **<<** endl**;**

**return;**

**}**

// Создаём временный файл для хранения сжатых данных

string tempFilename **=** "temp\_" **+** filename**;**

ofstream outputFile**(**tempFilename**,** ios**::**binary**);**

**if** **(!**outputFile**)** **{**

cerr **<<** "Error open temp file!" **<<** endl**;**

inputFile**.**close**();**

**return;**

**}**

// Чтение данных из исходного файла и запись в новый файл

**while** **(**inputFile**.**peek**()** **!=** EOF**)** **{**

size\_t lastNameSize**;**

inputFile**.**read**(reinterpret\_cast<**char**\*>(&**lastNameSize**),** **sizeof(**size\_t**));**

char**\*** lastName **=** **new** char**[**lastNameSize **+** 1**];**

inputFile**.**read**(**lastName**,** lastNameSize**);**

lastName**[**lastNameSize**]** **=** '\0'**;**

// Записываем данные о студенте в новый файл

outputFile**.**write**(reinterpret\_cast<**const char**\*>(&**lastNameSize**),** **sizeof(**size\_t**));**

outputFile**.**write**(**lastName**,** lastNameSize**);**

size\_t birthDateSize**;**

inputFile**.**read**(reinterpret\_cast<**char**\*>(&**birthDateSize**),** **sizeof(**size\_t**));**

char**\*** birthDate **=** **new** char**[**birthDateSize **+** 1**];**

inputFile**.**read**(**birthDate**,** birthDateSize**);**

birthDate**[**birthDateSize**]** **=** '\0'**;**

outputFile**.**write**(reinterpret\_cast<**const char**\*>(&**birthDateSize**),** **sizeof(**size\_t**));**

outputFile**.**write**(**birthDate**,** birthDateSize**);**

size\_t admissionDateSize**;**

inputFile**.**read**(reinterpret\_cast<**char**\*>(&**admissionDateSize**),** **sizeof(**size\_t**));**

char**\*** admissionDate **=** **new** char**[**admissionDateSize **+** 1**];**

inputFile**.**read**(**admissionDate**,** admissionDateSize**);**

admissionDate**[**admissionDateSize**]** **=** '\0'**;**

outputFile**.**write**(reinterpret\_cast<**const char**\*>(&**admissionDateSize**),** **sizeof(**size\_t**));**

outputFile**.**write**(**admissionDate**,** admissionDateSize**);**

size\_t expulsionDateSize**;**

inputFile**.**read**(reinterpret\_cast<**char**\*>(&**expulsionDateSize**),** **sizeof(**size\_t**));**

char**\*** expulsionDate **=** **new** char**[**expulsionDateSize **+** 1**];**

inputFile**.**read**(**expulsionDate**,** expulsionDateSize**);**

expulsionDate**[**expulsionDateSize**]** **=** '\0'**;**

outputFile**.**write**(reinterpret\_cast<**const char**\*>(&**expulsionDateSize**),** **sizeof(**size\_t**));**

outputFile**.**write**(**expulsionDate**,** expulsionDateSize**);**

size\_t addressSize**;**

inputFile**.**read**(reinterpret\_cast<**char**\*>(&**addressSize**),** **sizeof(**size\_t**));**

char**\*** address **=** **new** char**[**addressSize **+** 1**];**

inputFile**.**read**(**address**,** addressSize**);**

address**[**addressSize**]** **=** '\0'**;**

outputFile**.**write**(reinterpret\_cast<**const char**\*>(&**addressSize**),** **sizeof(**size\_t**));**

outputFile**.**write**(**address**,** addressSize**);**

size\_t groupSize**;**

inputFile**.**read**(reinterpret\_cast<**char**\*>(&**groupSize**),** **sizeof(**size\_t**));**

char**\*** group **=** **new** char**[**groupSize **+** 1**];**

inputFile**.**read**(**group**,** groupSize**);**

group**[**groupSize**]** **=** '\0'**;**

outputFile**.**write**(reinterpret\_cast<**const char**\*>(&**groupSize**),** **sizeof(**size\_t**));**

outputFile**.**write**(**group**,** groupSize**);**

// Освобождение памяти

**delete[]** lastName**;**

**delete[]** birthDate**;**

**delete[]** admissionDate**;**

**delete[]** expulsionDate**;**

**delete[]** address**;**

**delete[]** group**;**

**}**

inputFile**.**close**();**

outputFile**.**close**();**

// Заменяем старый файл новым сжатым файлом

remove**(**filename**.**c\_str**());**

rename**(**tempFilename**.**c\_str**(),** filename**.**c\_str**());**

cout **<<** "File compressed." **<<** endl**;**

**}**

void StudentsList**::**searchStudentByIndex**(**int index**)** **{**

ifstream file**(**filename**,** ios**::**binary**);**

**if** **(**file**)** **{**

size\_t position **=** 0**;**

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

size\_t lastNameSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**lastNameSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** lastNameSize**;**

size\_t birthDateSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**birthDateSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** birthDateSize**;**

size\_t admissionDateSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**admissionDateSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** admissionDateSize**;**

size\_t expulsionDateSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**expulsionDateSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** expulsionDateSize**;**

size\_t addressSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**addressSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** addressSize**;**

size\_t groupSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**groupSize**),** **sizeof(**size\_t**));**

position **+=** **sizeof(**size\_t**)** **+** groupSize**;**

**}**

size\_t lastNameSize**;**

file**.**seekg**(**position**);**

file**.**read**(reinterpret\_cast<**char**\*>(&**lastNameSize**),** **sizeof(**size\_t**));**

char**\*** lastName **=** **new** char**[**lastNameSize **+** 1**];**

file**.**read**(**lastName**,** lastNameSize**);**

lastName**[**lastNameSize**]** **=** '\0'**;**

size\_t birthDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**birthDateSize**),** **sizeof(**size\_t**));**

char**\*** birthDate **=** **new** char**[**birthDateSize **+** 1**];**

file**.**read**(**birthDate**,** birthDateSize**);**

birthDate**[**birthDateSize**]** **=** '\0'**;**

size\_t admissionDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**admissionDateSize**),** **sizeof(**size\_t**));**

char**\*** admissionDate **=** **new** char**[**admissionDateSize **+** 1**];**

file**.**read**(**admissionDate**,** admissionDateSize**);**

admissionDate**[**admissionDateSize**]** **=** '\0'**;**

size\_t expulsionDateSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**expulsionDateSize**),** **sizeof(**size\_t**));**

char**\*** expulsionDate **=** **new** char**[**expulsionDateSize **+** 1**];**

file**.**read**(**expulsionDate**,** expulsionDateSize**);**

expulsionDate**[**expulsionDateSize**]** **=** '\0'**;**

size\_t addressSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**addressSize**),** **sizeof(**size\_t**));**

char**\*** address **=** **new** char**[**addressSize **+** 1**];**

file**.**read**(**address**,** addressSize**);**

address**[**addressSize**]** **=** '\0'**;**

size\_t groupSize**;**

file**.**read**(reinterpret\_cast<**char**\*>(&**groupSize**),** **sizeof(**size\_t**));**

char**\*** groupName **=** **new** char**[**groupSize **+** 1**];**

file**.**read**(**groupName**,** groupSize**);**

groupName**[**groupSize**]** **=** '\0'**;**

// Вывод информации о студенте

cout **<<** "lastName: " **<<** lastName **<<** "\nbirthDate: " **<<** birthDate

**<<** "\nadmissionDate: " **<<** admissionDate **<<** "\n"

**<<** "expulsionDate: " **<<** expulsionDate **<<** "\naddress: " **<<** address

**<<** "\ngroup: " **<<** groupName **<<** endl**;**

**delete[]** lastName**;**

**delete[]** birthDate**;**

**delete[]** admissionDate**;**

**delete[]** expulsionDate**;**

**delete[]** address**;**

**delete[]** groupName**;**

file**.**close**();**

**}**

**}**

void StudentsList**::**sortStudents**()** **{**

// Если список пуст или состоит из одного элемента, ничего не делаем

**if** **(!**head **||** **!**head**->**next**)** **return;**

bool swapped**;**

**do** **{**

swapped **=** **false;**

Student**\*** current **=** head**;**

Student**\*** prev **=** **nullptr;**

**while** **(**current**->**next**)** **{**

**if** **(**string**(**current**->**lastName**)** **>** string**(**current**->**next**->**lastName**))** **{**

Student**\*** nextStudent **=** current**->**next**;**

// Обновляем связи

**if** **(**prev**)** **{**

prev**->**next **=** nextStudent**;**

**}** **else** **{**

head **=** nextStudent**;**

**}**

current**->**next **=** nextStudent**->**next**;**

nextStudent**->**next **=** current**;**

// Обновляем prev указатели

nextStudent**->**prev **=** prev**;**

current**->**prev **=** nextStudent**;**

**if** **(**current**->**next**)** **{**

current**->**next**->**prev **=** current**;**

**}** **else** **{**

tail **=** current**;**

**}**

prev **=** nextStudent**;**

swapped **=** **true;**

**}** **else** **{**

prev **=** current**;**

current **=** current**->**next**;**

**}**

**}**

**}** **while** **(**swapped**);**

// Обновляем tail указатель

tail **=** head**;**

**while** **(**tail**->**next**)** **{**

tail **=** tail**->**next**;**

**}**

saveToFile**();**

**}**

#include <iostream>

#include "src/List/StudentList.h"

**using** **namespace** std**;**

void displayMenu**()** **{**

cout **<<** "\nMenu:" **<<** endl**;**

cout **<<** "1. Add a student\n"**;**

cout **<<** "2. Delete a student\n"**;**

cout **<<** "3. Edit\n"**;**

cout **<<** "4. Insert student by index\n"**;**

cout **<<** "5. Display list\n"**;**

cout **<<** "6. Compress the file\n"**;**

cout **<<** "7. Remove student by index\n"**;**

cout **<<** "8. Sort list\n"**;**

cout **<<** "0. Exit\n"**;**

**}**

int main**()** **{**

StudentsList studentsList**(**"students.dat"**);**

studentsList**.**loadFromFile**();**

int choice**;**

**do** **{**

displayMenu**();**

cout **<<** "Choice: "**;**

cin **>>** choice**;**

**switch** **(**choice**)** **{**

**case** 1**:** **{**//add

cout **<<** "\n|Add student|" **<<** endl**;**

string lastName**,** birthDate**,** admissionDate**,** expulsionDate**,** address**,** group**;**

cout **<<** "LastName: "**;**

cin **>>** lastName**;**

cout **<<** "Birth date: "**;**

cin **>>** birthDate**;**

cout **<<** "Admission date: "**;**

cin **>>** admissionDate**;**

cout **<<** "Expulsion date: "**;**

cin **>>** expulsionDate**;**

cout **<<** "Address: "**;**

cin **>>** address**;**

cout **<<** "Group: "**;**

cin **>>** group**;**

studentsList**.**addStudent**(**Student**(**lastName**,** birthDate**,** admissionDate**,** expulsionDate**,** address**,** group**));**

**break;**

**}**

**case** 2**:** **{**//remove

cout **<<** "\n|Remove student|" **<<** endl**;**

int index**;**

cout **<<** "Student's index: "**;**

cin **>>** index**;**

studentsList**.**removeStudent**(**index**);**

**break;**

**}**

**case** 3**:** **{**//edit

cout **<<** "\n|Edit student|" **<<** endl**;**

Student newData**;**

string surname\_edit**,** group\_edit**;**

cout **<<** "Group student: "**;**

cin **>>** group\_edit**;**

cout **<<** "Lastname student: "**;**

cin **>>** surname\_edit**;**

studentsList**.**editStudent**(**surname\_edit**,** group\_edit**,** newData**);**

**break;**

**}**

**case** 4**:** **{**//insert

cout **<<** "\n|Insert student|" **<<** endl**;**

int index**;**

cout **<<** "Index insert: "**;**

cin **>>** index**;**

string lastName**,** birthDate**,** admissionDate**,** expulsionDate**,** address**,** group**;**

cout **<<** "LastName: "**;**

cin **>>** lastName**;**

cout **<<** "Birth date: "**;**

cin **>>** birthDate**;**

cout **<<** "Admission date: "**;**

cin **>>** admissionDate**;**

cout **<<** "Expulsion date: "**;**

cin **>>** expulsionDate**;**

cout **<<** "Address: "**;**

cin **>>** address**;**

cout **<<** "Group: "**;**

cin **>>** group**;**

studentsList**.**insertStudent**(**index**,** Student**(**lastName**,** birthDate**,** admissionDate**,** expulsionDate**,** address**,** group**));**

**break;**

**}**

**case** 5**:** **{**//display

int pageSize**,** page**;**

cout **<<** "page size: "**;**

cin **>>** pageSize**;**

cout **<<** "page number: "**;**

cin **>>** page**;**

cout **<<** "\nStudent List, page(" **<<** page **<<** ")" **<<** endl**;**

studentsList**.**displayStudents**(**pageSize**,** page**);**

**break;**

**}**

**case** 6**:** **{**//compress

studentsList**.**compressFile**();**

**break;**

**}**

**case** 7**:** **{**//search

cout **<<** "\n|Search student|" **<<** endl**;**

int index**;**

cout **<<** "Index student: "**;**

cin **>>** index**;**

cout **<<** "\n|Student information|" **<<** endl**;**

studentsList**.**searchStudentByIndex**(**index**);**

**break;**

**}**

**case** 8**:** **{**

studentsList**.**sortStudents**();**

**break;**

**}**

**case** 0**:**

cout **<<** "Exit." **<<** endl**;**

**break;**

**default:**

cout **<<** "Try again." **<<** endl**;**

**}**

**}** **while** **(**choice **!=** 0**);**

**return** 0**;**

**}**

## ДОПОЛНЕНИЕ (РАЗБОР КОДА)

Замечания:

1. Нет вставки с сохранением порядка.
2. Операция удаления уменьшает объем файла.
3. Редактирование: при уменьшении размера поля, объем файла уменьшается незначительно (497 -> 488).
4. Объем после сжатия? Объяснить, что происходит с ним.

**1. Нет вставки с сохранением порядка.**

В коде действительно нет функции вставки, которая бы сохраняла порядок после сортировки. Но есть функция insertStudent, которая добавляет студента в указанную позицию и функция sortStudents(), которая сортирует список по фамилии. Например, если требуется сохранять определенный порядок, можно сохранять постоянный порядок по фамилии. Это можно сделать вызывав функцию sortStudents() после вставки и остальных функций, тем самым модернизировав код.

**2. Операция удаления уменьшает объем файла**

void StudentsList**::**removeStudent**(**int index**)** **{**

//…

// Если удаляется первый элемент

**if** **(**index **==** 0**)** **{**

//…

cout **<<** "Student with index (0) deleted." **<<** endl**;**

saveToFile**();**

**return;**

**}**

//…

cout **<<** "Student with index (" **<<** index **<<** ") deleted." **<<** endl**;**

saveToFile**();**

**}**

Операция удаления уменьшает объем файла из-за вызова метода saveToFile(), который перезаписывает весь файл. Это приводит к уменьшению общего размера файла, так как в нем не остается информации о студентах, которых удалили.

**3. Редактирование: при уменьшении размера поля, объем файла уменьшается**

В данном методе виновником уменьшения файла является снова метод saveToFile().

void StudentsList**::**editStudent**(**const string**&** surname**,** const string**&** group**,** Student**&** newData**)** **{**

//...

cout **<<** "The data has been updated successfully!" **<<** endl**;**

saveToFile**();**

**return;**

//...

**}**

Он открывает файл для записи и перезаписывает его содержимое. Это значит, что он записывает только актуальные данные, исключая любые неиспользуемые или удаленные поля.

**4. Объем после сжатия**

Метод compressFile() должен сжимать файл с данными студентов, открывая его для чтения и создавая временный файл для записи актуальных данных. При чтении он извлекает информацию о каждом студенте, записывая только необходимые поля и их размеры, что позволяет избежать записи неиспользуемого пространства и устаревших данных. После завершения записи временный файл заменяет исходный, что должно приводить к уменьшению объема файла за счет удаления лишних данных, особенно если данные стали короче или изменились. Но так как функция удаления работает не корректно, сжатие тоже работает неправильно. Ведь две эти функции взаимосвязаны.