**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра АПУ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Программирование»**

**Тема: Разработка базы данных «Студенческий поток»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3371 |  | Манешов Д.В. |
| Преподаватель |  | Писарев А.С. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

|  |  |  |
| --- | --- | --- |
| Студент Манешов Д.В. | | |
| Группа 3371 | | |
| Тема работы: **Разработка базы данных «Студенческий поток»** | | |
| Исходные данные:  Общая часть задания:  Студент потока характеризуется следующими данными:   * ФИО (до 50 символов); * номер группы; * набор из пяти оценок за последнюю сессию (без указания предметов); * размер стипендии. * дополнительная информация (выбирается индивидуально каждым студентом, например, телефон, эл. почта, год рождения и т.п.)   Необходимо:   1. разработать (и программно реализовать) динамические структуры данных и алгоритмы их обработки, позволяющие поддерживать выполнение следующих функций:    * консольный ввод/вывод данных о всех студентах потока;    * файловый ввод/вывод данных о потоке;    * редактирование данных о студентах и группах потока, включающее операции добавления/удаления групп и студентов; 2. Вывести перечень групп потока с указанием (для каждой группы) процента суммарной стипендии группы по отношению к суммарной стипендии потока   Общие требования к программам:   1. Программа должна поддерживать систему меню, пункты которых соответствуют выполнению функций, предусмотренных общей частью задания. 2. Предлагаемые структуры данных должны учитывать изначальную неопределенность возможного количества групп и студентов в группах, а также обеспечивать максимальную скорость процессов обработки данных, предусмотренных заданием. 3. Тексты программ должны содержать комментарии, объясняющие назначение основных функций, типов и объектов данных, функциональных блоков и т.п. 4. Представляемые тексты программ должны обеспечивать возможность их компиляции и построения в среде MS Visual Studio. | | |
| Дата выдачи задания: 21.04.2024 | | |
| Дата сдачи курсовой: | | |
| Дата защиты курсовой: | | |
| Студент |  | Манешов Д.В. |
| Преподаватель |  | Писарев А.С. |

**АННОТАЦИЯ**

В данной курсовой работе разработана программа на языке C++ для управления данными о студентах учебного потока. Основной функционал программы включает консольный и файловый ввод/вывод данных, редактирование информации о студентах и группах, а также фильтрацию студентов по заданным критериям. Для реализации использованы динамические структуры данных: двусвязный список для хранения групп и хеш-таблица для хранения студентов в группах. В результате работы программы обеспечивается эффективное управление данными студентов, включая расчет общей суммы стипендий и управление данными через различные фильтры.

**SUMMARY**

In this course work, a C++ programme is developed to manage data on students of a study stream. The main functionality of the programme includes console and file data input/output, editing information about students and groups, as well as filtering students by specified criteria. Dynamic data structures are used for implementation: a doubly linked list for storing groups and a hash table for storing students in groups. As a result, the programme provides efficient management of student data, including calculation of the total amount of scholarships and data management through various filters.

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc168013260)

[1. Описание блоков программы 8](#_Toc168013261)

[1.1. Описание файла student.h 8](#_Toc168013262)

[1.2. Описание файла student.cpp 11](#_Toc168013263)

[1.3. Описание файла group.h 18](#_Toc168013264)

[1.4. Описание файла group.cpp 23](#_Toc168013265)

[1.5. Описание файла course-work-2sem.cpp 28](#_Toc168013266)

[1.6. Структуры Student, StudentNode и StudentHashTable 38](#_Toc168013267)

[1.6.1. Методы для управления студентами 41](#_Toc168013268)

[1.6.2. Методы для работы с файлами 47](#_Toc168013269)

[1.6.3. Метод для фильтрации студентов 50](#_Toc168013270)

[1.7. Структура Group 52](#_Toc168013271)

[1.7.1. Методы для управления группами 56](#_Toc168013272)

[1.7.2. Методы для работы с файлами 61](#_Toc168013273)

[1.7.3. Методы для печати информации и вычисления данных 64](#_Toc168013274)

[1.8. Функции взаимодействия с пользователем 68](#_Toc168013275)

[2. Тестирование программы 71](#_Toc168013276)

[2.1. Работа программы 71](#_Toc168013277)

[Заключение 73](#_Toc168013278)

[Список использованных источников 74](#_Toc168013279)

[Приложение A 75](#_Toc168013280)

[Код написанной программы 75](#_Toc168013281)

[Приложение B 94](#_Toc168013282)

# Введение

Цель данной курсовой работы состоит в разработке и реализации программы на языке C++, предназначенной для управления данными о студентах учебного потока. Основные задачи работы включают:

* создание динамических структур данных для хранения и обработки информации о студентах и группах с использованием указателей и динамического выделения памяти;
* реализацию функций для консольного и файлового ввода и вывода данных, а также для интерактивного редактирования информации о студентах и группах;
* разработку алгоритмов для выполнения индивидуального задания, включающего вывод переченья групп потока с указанием (для каждой группы) процента суммарной стипендии группы по отношению к суммарной стипендии потока;
* соблюдение общих требований к программе, таких как поддержка системы меню и подменю, комментирование кода, и отказ от использования готовых структур данных;

Методы решения задач основаны на применении знаний и навыков программирования на языке C++. Для реализации блоков программы использованы следующие функции:

* функции управления группами. Включают создание и удаление групп, добавление и удаление студентов из группы, а также поиск и редактирование информации о группах;
* функции управления студентами. Включают добавление, удаление, редактирование и фильтрацию студентов по различным параметрам (ФИО, номер телефона, оценки, стипендия, номер группы);
* функции ввода/вывода. Обеспечивают консольный ввод/вывод данных, а также сохранение и загрузку информации о группах и студентах из файлов;
* функции для вычислений. Включают расчет общей суммы стипендий и процентного соотношения стипендий группы к суммарной стипендии потока;

# 1. Описание блоков программы

## 1.1. Описание файла student.h

Файл student.h (рис. 1, 2) содержит объявления структур и функций, используемых для управления данными о студентах. В этом файле определены основные константы, структуры и функции, обеспечивающие создание, хранение, редактирование и обработку информации о студентах.

Основные компоненты файла student.h:

1. Препроцессорные директивы:

* #ifndef STUDENT\_H
* #define STUDENT\_H

Эти директивы обеспечивают защиту от многократного включения заголовочного файла, предотвращая ошибки компиляции.

1. Подключаемые библиотеки:

* #include <iostream>
* #include <fstream>
* #include <cstring>

Эти библиотеки необходимы для ввода-вывода, работы с файлами и строками.

1. Константы:

* MAX\_NAME\_LENGTH - максимальная длина имени студента (50 символов).
* PHONE\_NUMBER\_LENGTH - максимальная длина номера телефона студента (13 символов, включая код страны).
* NUM\_GRADES - количество оценок у студента (5 оценок).

1. Функции:

* unsigned int hashStudentName(const char\* name) - функция для вычисления хэш-значения имени студента, используемая для хранения студентов в хэш-таблице.

1. Структуры:

* Student - структура для хранения информации о студенте:
  + int id - идентификационный номер студента.
  + char name[MAX\_NAME\_LENGTH] - имя студента.
  + int grades[NUM\_GRADES] - массив оценок студента.
  + float scholarship - размер стипендии студента.
  + char phoneNumber[PHONE\_NUMBER\_LENGTH] - номер телефона студента.
  + int groupNumber - номер группы, к которой принадлежит студент.
  + Конструктор структуры Student с параметрами по умолчанию.
* StudentNode - структура узла связного списка для студентов:
  + Student student - информация о студенте.
  + StudentNode\* next - указатель на следующий узел в списке.
  + Конструктор структуры StudentNode с параметрами.
* StudentHashTable - структура хэш-таблицы для хранения студентов:
  + StudentNode\*\* table - массив указателей на узлы.
  + int tableSize - размер таблицы.
  + int numElements - количество элементов в таблице.
  + int nextId - следующий идентификатор для нового студента.
  + Конструктор структуры StudentHashTable с инициализацией размера таблицы.
  + Деструктор для освобождения памяти.
  + Функции для добавления, поиска, удаления студентов, а также вычисления общей суммы стипендий, печати всех студентов и сохранения/загрузки данных из файла.
  + Дополнительные функции для проверки пустоты таблицы, удаления всех студентов, увеличения размера таблицы и фильтрации студентов по различным параметрам.
  + Приватная функция unsigned int hashFunction(const char\* name) для вычисления индекса на основе имени студента.

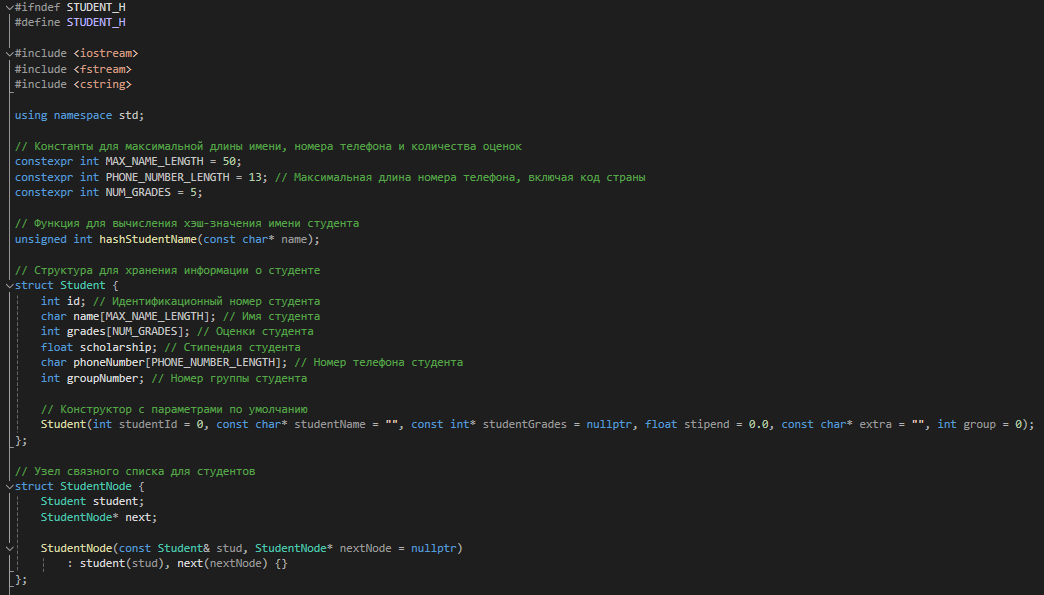


Рисунок 1 - Файл student.h

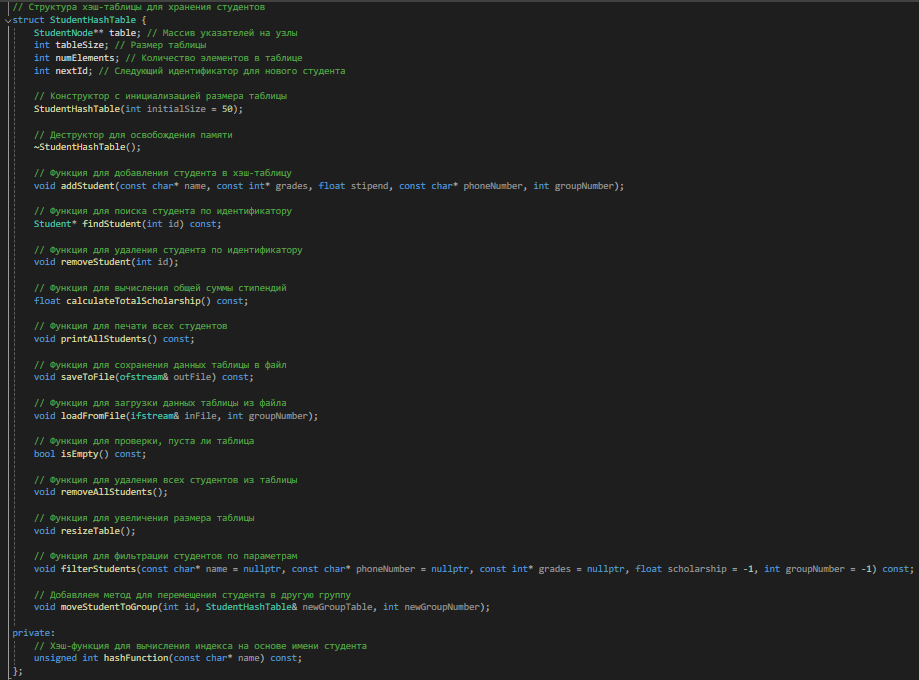


Рисунок 2 - Файл student.h

## 1.2. Описание файла student.cpp

Файл student.cpp содержит определения функций и методов, объявленных в файле student.h, которые используются для управления данными о студентах. Этот файл реализует функциональность, обеспечивающую создание, хранение, редактирование и обработку информации о студентах.

Основные компоненты файла student.cpp:

1. Подключение заголовочного файла:

* #include "student.h" (рис. 3).

Файл student.cpp включает заголовочный файл student.h, в котором объявлены структуры и функции для работы с данными о студентах.



Рисунок 3 – Подключение заголовочного файла student.h

1. Функции:

* Функция для вычисления хэш-значения имени студента (рис. 4).

Эта функция вычисляет хэш-значение для имени студента, что используется в хэш-таблице для хранения студентов.

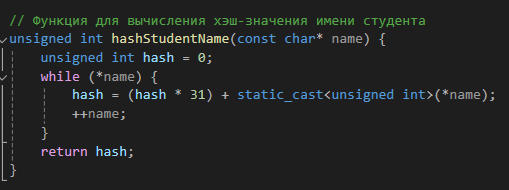


Рисунок 4 – Функция hashStudentName

* Конструктор структуры Student (рис. 5).

Этот конструктор инициализирует объект Student заданными параметрами, включая идентификационный номер, имя, оценки, стипендию, номер телефона и номер группы.

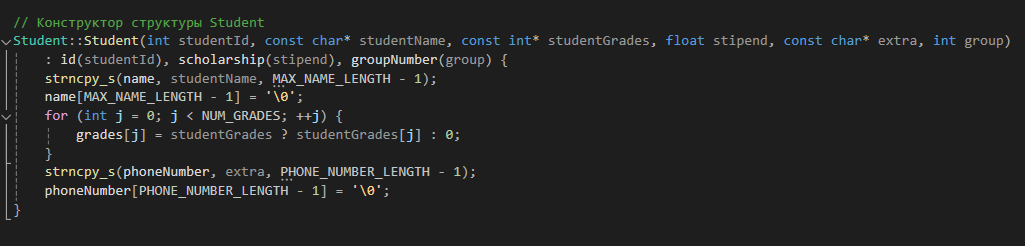


Рисунок 5 – Конструктор структуры Student

* Конструктор и деструктор хэш-таблицы студентов StudentHashTable (рис. 6).

Конструктор инициализирует хэш-таблицу с заданным размером, а деструктор освобождает память, занятую хэш-таблицей.

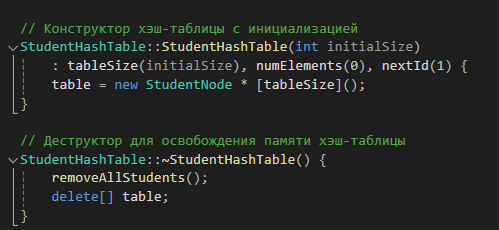


Рисунок 6 – Конструктор и деструктор StudentHashTable

* Функция для добавления студента в хэш-таблицу (рис. 7).

Эта функция добавляет нового студента в хэш-таблицу, увеличивая размер таблицы при необходимости.

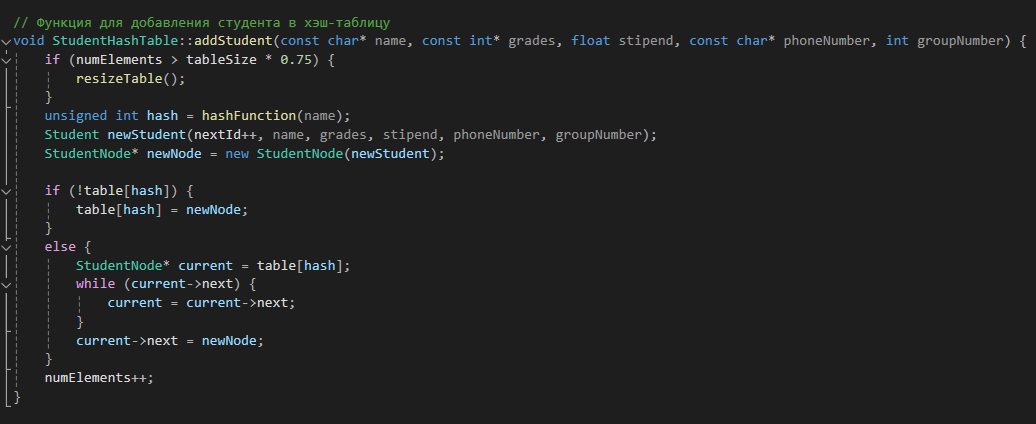


Рисунок 7 – Функция addStudent

* Функция для увеличения размера хэш-таблицы (рис. 8).

Эта функция увеличивает размер хэш-таблицы, пересчитывая хэш-значения для всех студентов.

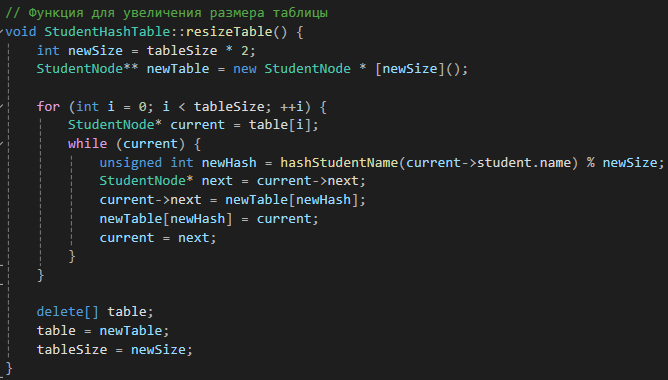


Рисунок 8 – Функция resizeTable

* Функции для поиска, удаления и печати студентов, а также вычисления общей суммы стипендий (рис. 9 - 12).

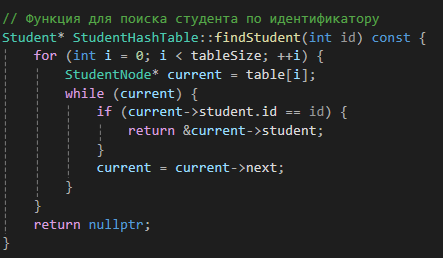


Рисунок 9 – Функция findStudent

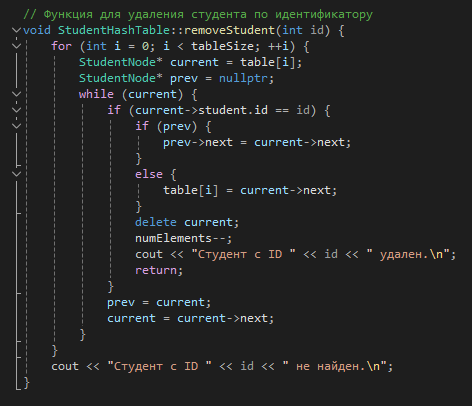


Рисунок 10 – Функция removeStudent

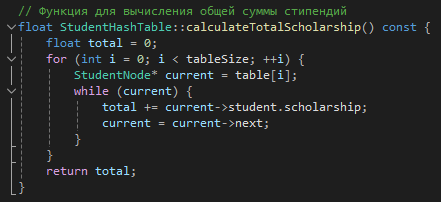


Рисунок 11 – Функция calculateTotalScholarship

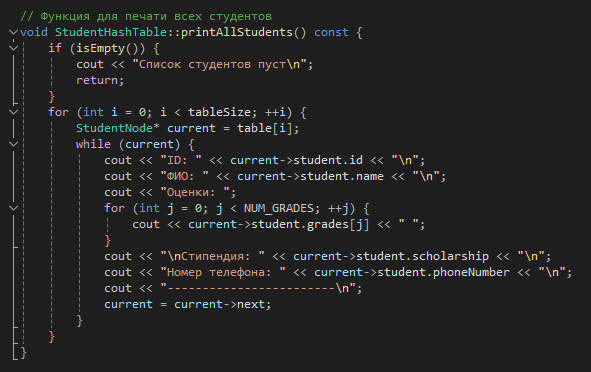


Рисунок 12 – Функция printAllStudents

* Функции для сохранения и загрузки данных студентов в/из файла (рис. 13, 14).

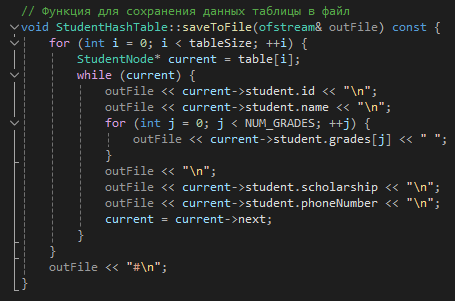


Рисунок 13 – Функция saveToFile



Рисунок 14 – Функция loadFromFile

* Функции для фильтрации студентов по параметрам, перемещения студентов между группами и проверку, пуста ли таблица (рис. 15, 16).

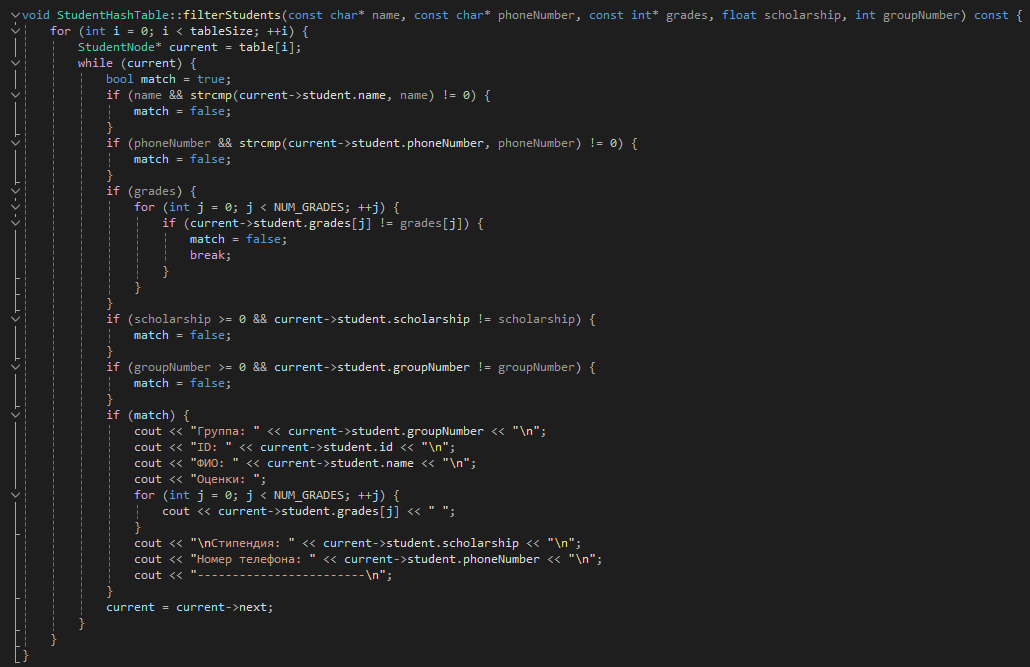


Рисунок 15 – Функция filterStudents

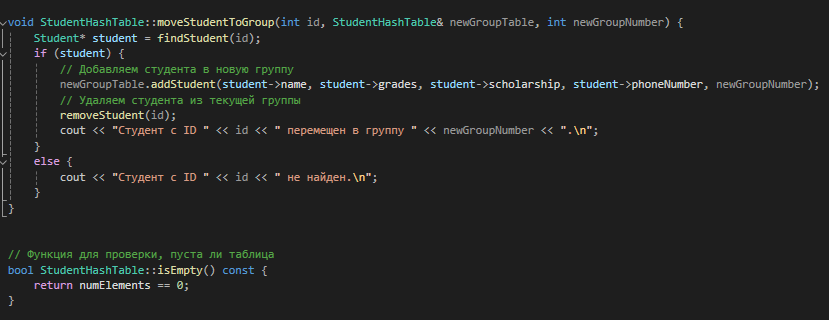


Рисунок 16 – Функция moveStudentToGroup и isEmpty

## 1.3. Описание файла group.h

Файл group.h (рис. 17, 18) содержит объявления структур и методов для работы с группами студентов. Он включает определения структур GroupNode и GroupList, а также функций, обеспечивающих управление данными о группах и студентах. Ниже приведено подробное описание содержимого файла.

Основные разделы файла group.h:

1. Предварительные объявления и включения

* #ifndef GROUP\_H / #define GROUP\_H:
  + Эти директивы предотвращают многократное включение файла.
* #include "student.h":
  + Включает файл student.h, который содержит определения структуры Student и методов для работы с данными студентов.

1. Структура GroupNode

* Представляет собой узел, содержащий информацию о конкретной группе студентов.
* Поля:
  + int groupNumber — номер группы.
  + StudentHashTable students — хэш-таблица студентов, принадлежащих группе.
  + GroupNode\* next — указатель на следующий узел в двусвязном списке.
  + GroupNode\* prev — указатель на предыдущий узел в двусвязном списке.
* Конструктор:
  + GroupNode(int num) — инициализирует номер группы и устанавливает указатели на nullptr.
* Методы:
* void printStudents() — выводит список всех студентов в группе.
* float calculateTotalScholarship() const — вычисляет общую сумму стипендий в группе.
* void saveToFile(ofstream& outFile) const — сохраняет данные группы в файл.
* void loadFromFile(ifstream& inFile, int groupNumber) — загружает данные группы из файла.

1. Структура GroupList

* Представляет собой двусвязный список групп.
* Поля:
  + GroupNode\* head — указатель на первый узел списка.
  + GroupNode\* tail — указатель на последний узел списка.
* Конструктор:
  + GroupList() — инициализирует пустой список групп.
* Методы:
  + void addGroup(int groupNumber) — добавляет новую группу в список.
  + void removeGroup(int groupNumber) — удаляет группу по номеру.
  + GroupNode\* findGroup(int groupNumber) const — ищет группу по номеру.
  + void addStudentToGroup(int groupNumber, const char\* studentName, const int\* grades, float stipend, const char\* phoneNumber) — добавляет студента в группу.
  + void removeStudentFromGroup(int groupNumber, int studentId) — удаляет студента из группы по его идентификатору.
  + void removeAllStudentsFromGroup(int groupNumber) — удаляет всех студентов из группы.
  + void removeAllGroup() — удаляет все группы и студентов.
  + void printAllGroupNumbers() const — выводит номера всех групп.
  + void printAllGroups() const — выводит информацию обо всех группах и студентах.
  + float calculateTotalScholarship() const — вычисляет общую сумму стипендий во всех группах.
  + bool isEmpty() const — проверяет, пуст ли список групп.
  + void printGroupScholarshipPercentage() const — выводит процент стипендий по каждой группе.
  + void saveToFile(const char\* filename) const — сохраняет данные всех групп в файл.
  + void loadFromFile(const char\* filename) — загружает данные всех групп из файла.

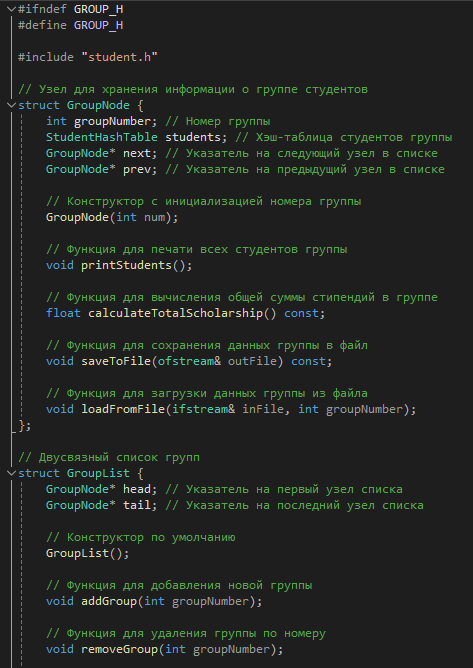


Рисунок 17 – Файл group.h

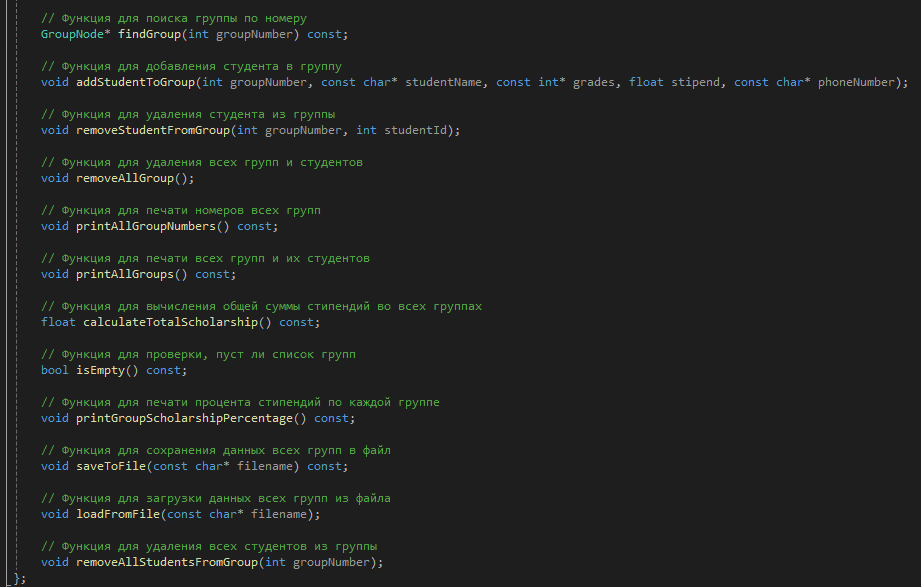


Рисунок 18 – Файл group.h

## 1.4. Описание файла group.cpp

Файл group.cpp содержит определения функций и методов, объявленных в файле group.h, которые используются для управления данными о группах студентов. Этот файл реализует функциональность, обеспечивающую создание, хранение, редактирование и обработку информации о группах и их студентах.

Основные компоненты файла group.cpp:

1. Подключение заголовочного файла:

* #include "group.h" (рис. 19).

Файл group.cpp включает заголовочный файл group.h, в котором объявлены структуры и функции для работы с данными о группах студентов.



Рисунок 19 – Подключение заголовочного файла group.h

1. Методы структуры GroupNode:

* Конструктор и функции для работы с группой (рис. 20).

Конструктор инициализирует объект GroupNode заданным номером группы. Функции включают печать всех студентов группы, вычисление общей суммы стипендий, сохранение данных группы в файл и загрузку данных группы из файла.

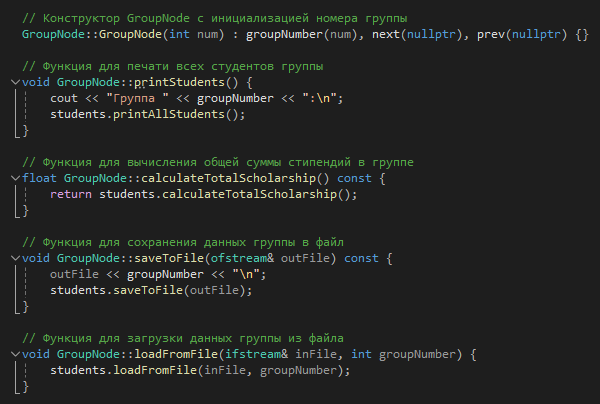


Рисунок 20 – Методы структуры GroupNode

1. Методы структуры GroupList:

* Конструктор и управление группами (рис. 21 - 23).

Конструктор инициализирует пустой список групп. Включает функции добавления и удаления групп, поиска группы по номеру, печати номеров всех групп и печати информации обо всех группах и их студентах.



Рисунок 21 – Конструктор и функции добавления и удаления групп

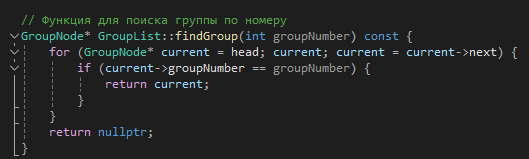


Рисунок 22 – Функция поиска группы по номеру

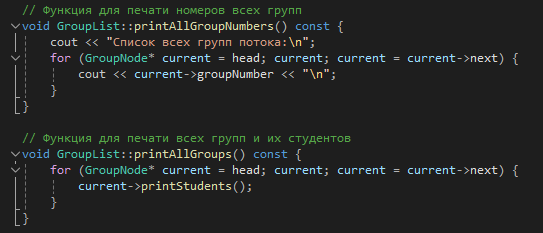


Рисунок 23 – Функции печати номеров всех групп и печати информации обо всех группах и их студентах

* Работа со студентами (рис. 24).

Включает функции добавления студента в группу, удаления студента из группы, удаления всех студентов из группы, а также удаления всех студентов из группы.

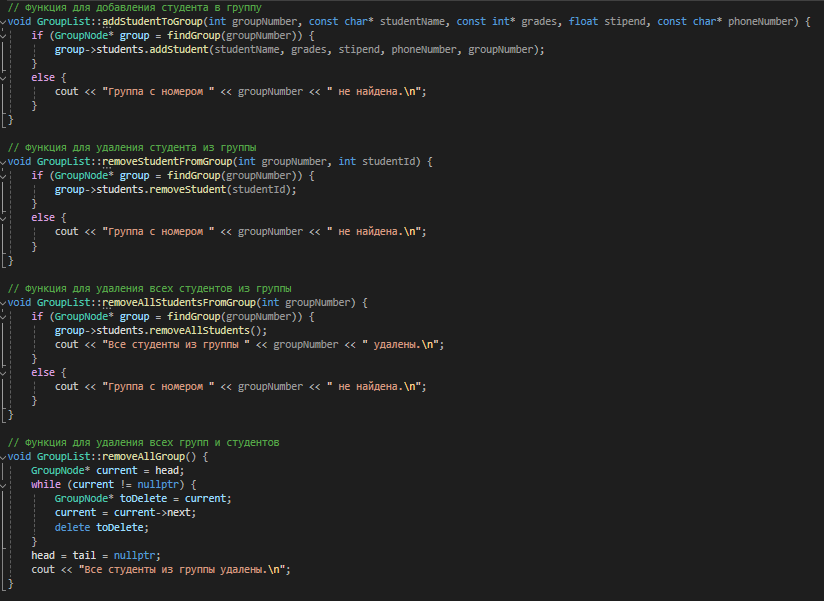


Рисунок 24 – Работа со студентами

* Вычисления и проверки (рис. 25).

Включает функции вычисления общей суммы стипендий во всех группах, проверки, пуст ли список групп, и вычисления процента стипендий по каждой группе.

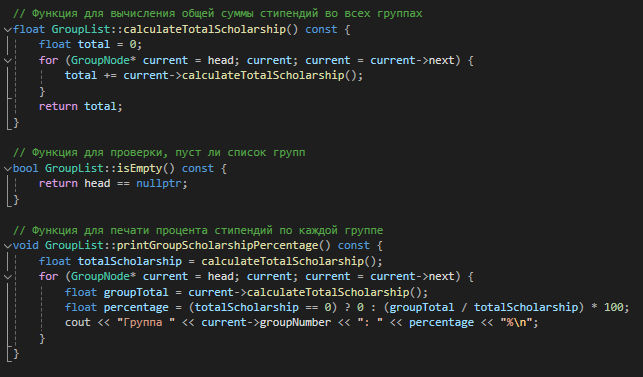


Рисунок 25 – Вычисления и проверки

* Сохранение и загрузка данных (рис. 26).

Включает функции сохранения данных всех групп в файл и загрузки данных всех групп из файла.

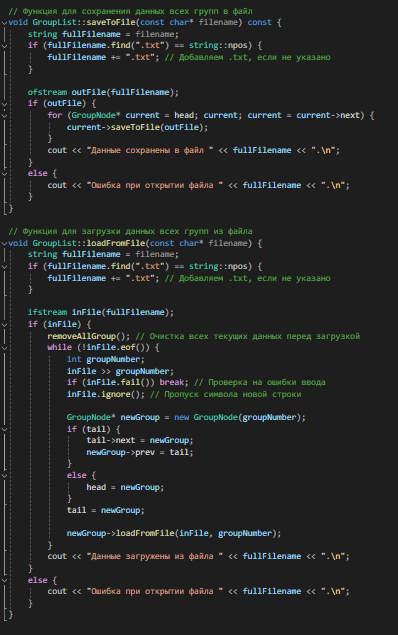


Рисунок 26 – Сохранение и загрузка данных

## 1.5. Описание файла course-work-2sem.cpp

Файл course-work-2sem.cpp содержит основную функцию и меню для взаимодействия с пользователем, позволяющее управлять данными о студентах и группах. В этом файле реализованы функции для добавления, редактирования, удаления студентов и групп, а также для фильтрации и сохранения данных. Ниже приведено подробное описание содержимого файла.

Основные компоненты файла course-work-2sem.cpp:

1. Подключение заголовочного файла:

* #include <iostream>
* #include <windows.h>
* #include "group.h" (рис. 27).

Файл course-work-2sem.cpp включает заголовочный файл group.h, содержащий объявления структур и функций для работы с данными о группах студентов, а также подключает библиотеки для работы с вводом/выводом и функциями Windows.



Рисунок 27 – Подключение заголовочного файла и библиотек

1. Определение пространства имен:

* using namespace std; (рис. 28).

Определяется пространство имен std для упрощения использования стандартной библиотеки C++.



Рисунок 28 – Определение пространства имен

1. Функции для взаимодействия с пользователем

* Функция safeReadString (рис. 29).

Эта функция безопасно считывает строку из ввода, предотвращая переполнение буфера.

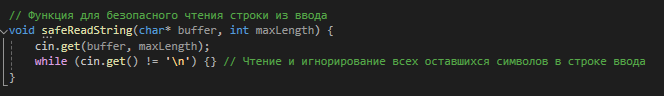


Рисунок 29 – Функция safeReadString

* Меню для фильтрации студентов (рис. 30).

Функция filterMenu предоставляет пользователю возможность фильтрации студентов по различным критериям: ФИО, номер телефона, оценки, размер стипендии и номер группы.



Рисунок 30 – Меню для фильтрации студентов

* Меню для редактирования данных студента (рис. 31).

Функция editStudentMenu позволяет пользователю редактировать данные студента, включая ФИО, оценки, размер стипендии, номер телефона и перемещение в другую группу.

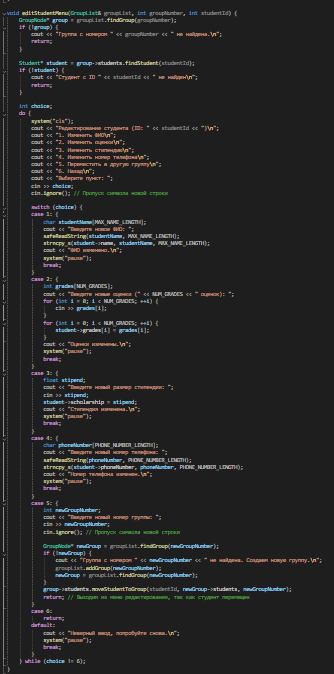


Рисунок 31 – Меню для редактирования данных студента

1. Главное меню программы:

* Функция menu (рис. 32 - 37).

Главное меню программы предоставляет пользователю следующие возможности:

* + Добавление и удаление групп.
  + Удаление всех студентов группы.
  + Показ процентов стипендий по группам.
  + Удаление всех групп.
  + Добавление и удаление студентов в группе.
  + Редактирование данных студента.
  + Показ всех групп и студентов.
  + Фильтрация студентов.
  + Сохранение данных в файл и загрузка данных из файла.
  + Завершение работы программы.

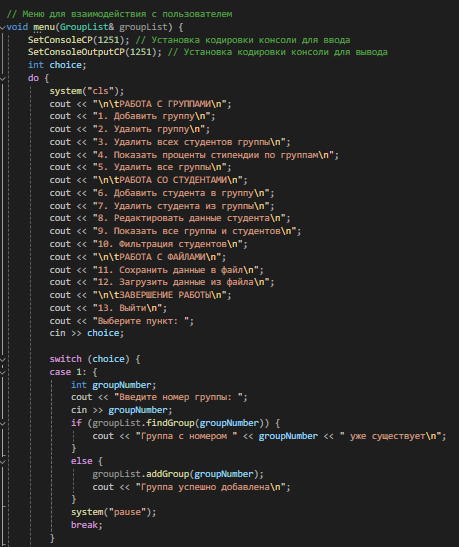


Рисунок 32 – Главное меню программы



Рисунок 33 – Главное меню программы

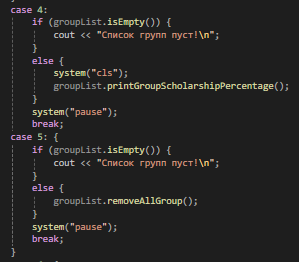


Рисунок 34 – Главное меню программы



Рисунок 35 – Главное меню программы

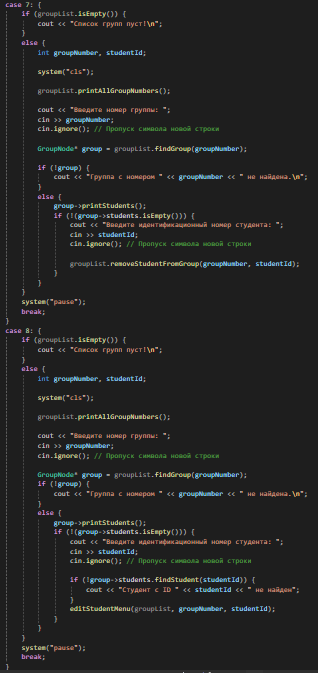


Рисунок 36 – Главное меню программы



Рисунок 37 – Главное меню программы

1. Основная функция программы:

* Функция main (рис. 38).

Эта функция инициализирует список групп и вызывает главное меню для взаимодействия с пользователем.

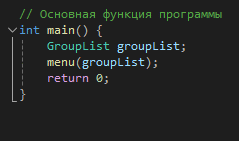


Рисунок 38 – Основная функция программы.

## 1.6. Структуры Student, StudentNode и StudentHashTable

Структура Student (рис. 38) представляет собой модель данных для хранения информации о студенте. Она определена в заголовочном файле student.h и реализована в файле student.cpp. Структура включает несколько полей, каждое из которых предназначено для хранения определенного аспекта информации о студенте. В частности, структура содержит идентификационный номер студента (поле int id), имя студента (поле char name[MAX\_NAME\_LENGTH]), массив оценок студента (поле int grades[NUM\_GRADES]), размер стипендии студента (поле float scholarship), номер телефона студента (поле char phoneNumber[PHONE\_NUMBER\_LENGTH]) и номер группы, к которой принадлежит студент (поле int groupNumber). Эти поля позволяют полноценно представлять информацию о каждом студенте в системе.

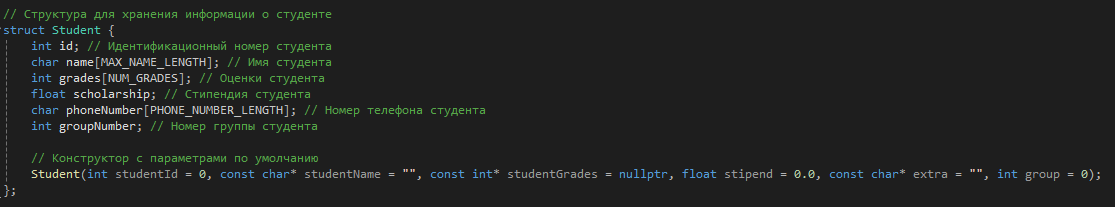


Рисунок 38 - Структура Student

Конструктор структуры Student (рис. 39) предназначен для инициализации объектов этой структуры с заданными значениями. Он принимает параметры для идентификационного номера студента, имени, массива оценок, размера стипендии, номера телефона и номера группы. Если параметры не указаны, конструктор использует значения по умолчанию: 0 для идентификационного номера и номера группы, пустую строку для имени и номера телефона, nullptr для массива оценок (что приводит к инициализации оценок нулями) и 0.0 для размера стипендии. Внутри конструктора данные копируются в соответствующие поля структуры, используя безопасные функции копирования строк и массивов.

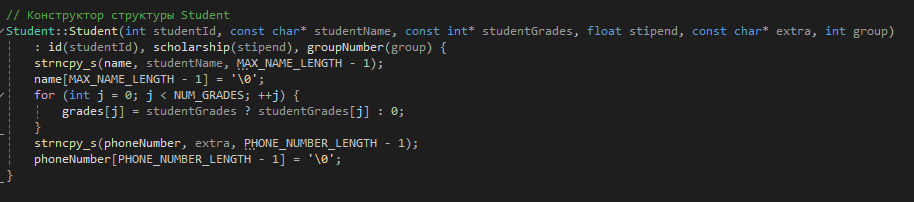


Рисунок 39 - Конструктор структуры Student

Структура StudentNode (рис. 40) представляет собой узел связного списка для студентов. Она содержит объект Student и указатель на следующий узел списка (поле StudentNode\* next). Конструктор этой структуры инициализирует объект Student и устанавливает указатель на следующий узел. Это позволяет создавать связные списки студентов, которые могут быть использованы для хранения и управления списками студентов в различных структурах данных.

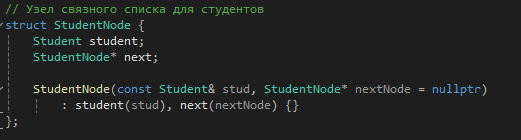


Рисунок 40 - Структура StudentNode

Структура StudentHashTable (рис. 41 - 43) предназначена для хранения студентов в виде хэш-таблицы. Она содержит массив указателей на узлы StudentNode (поле StudentNode\*\* table), размер таблицы (поле int tableSize), количество элементов в таблице (поле int numElements) и следующий идентификатор для нового студента (поле int nextId). Конструктор StudentHashTable инициализирует таблицу с заданным начальным размером и устанавливает значения по умолчанию для остальных полей. Деструктор освобождает память, занятую хэш-таблицей, удаляя все элементы. Функции добавления, удаления и поиска студентов реализуют основные операции для работы с хэш-таблицей.

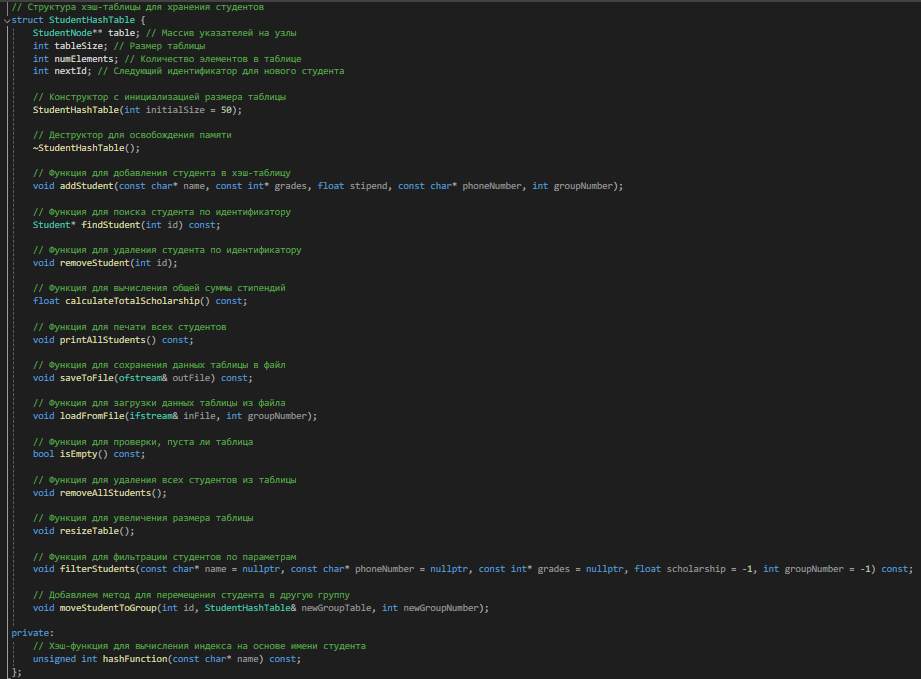


Рисунок 41 - Структура StudentHashTable

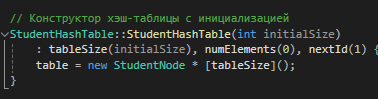


Рисунок 42 - Конструктор хэш-таблицы с инициализацией

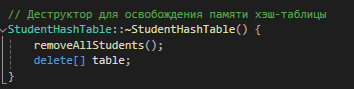


Рисунок 43 - Деструктор для освобождения памяти хэш-таблицы

### 1.6.1. Методы для управления студентами

Структура Student и связанные с ней методы предназначены для управления данными о студентах в системе. Эти методы включают функции добавления, поиска, удаления студентов и работы с хэш-таблицей, что обеспечивает эффективное управление и обработку информации.

Основной компонент для управления студентами — это структура StudentHashTable, которая реализует хэш-таблицу для хранения объектов Student. Конструктор StudentHashTable(int initialSize) инициализирует хэш-таблицу с заданным размером, а деструктор ~StudentHashTable() освобождает память, занятую таблицей. Хэш-таблица обеспечивает эффективное добавление и поиск студентов по идентификатору. Важной частью хэш-таблицы является функция unsigned int hashFunction(const char\* name) const (рис. 44), которая вычисляет хэш-значение для имени студента и определяет индекс в таблице, что позволяет быстро находить записи. В этой функции используется алгоритм, определенный в функции unsigned int hashStudentName(const char\* name) (рис. 45), который последовательно обрабатывает каждый символ имени студента для вычисления уникального хэш-значения.

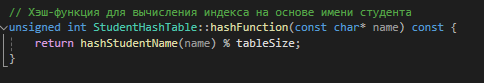


Рисунок 44 - Функция для вычисления хэш-значения имени студента

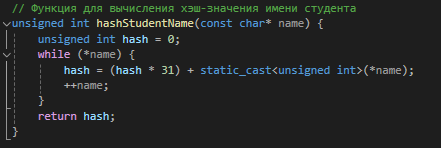


Рисунок 45- Функция для вычисления хэш-значения имени студента

Метод void addStudent(const char\* name, const int\* grades, float stipend, const char\* phoneNumber, int groupNumber) (рис. 46) добавляет нового студента в хэш-таблицу. Если количество элементов в таблице превышает 75% от её текущего размера, вызывается метод void resizeTable() (рис. 47), который увеличивает размер таблицы и пересчитывает хэш-значения для всех студентов. Этот метод сначала удваивает текущий размер таблицы, создает новый массив указателей на узлы, затем перераспределяет все существующие записи в новый массив на основе новых хэш-значений. Это обеспечивает более равномерное распределение записей и снижает вероятность коллизий, что улучшает производительность операций добавления и поиска.

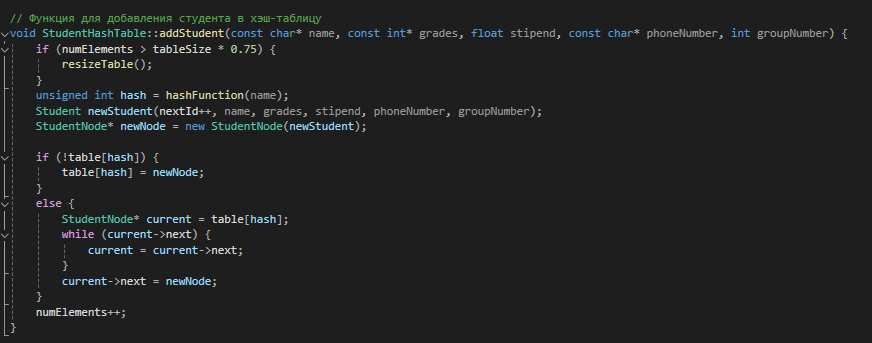


Рисунок 46 – Метод addStudent для добавления нового студента в хэш-таблицу

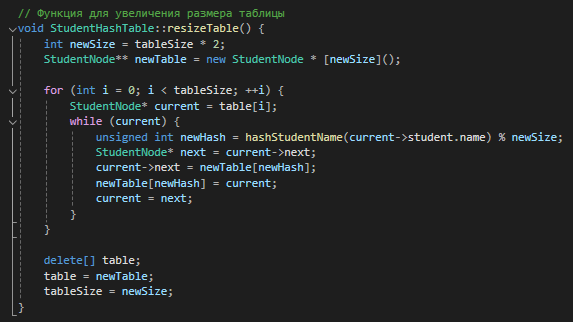


Рисунок 47 - Метод resizeTable, который увеличивает размер таблицы и пересчитывает хэш-значения для всех студентов

Метод Student\* findStudent(int id) const (рис. 48) позволяет найти студента по его идентификатору. Если студент с указанным идентификатором существует, функция возвращает указатель на объект Student, иначе возвращает nullptr.

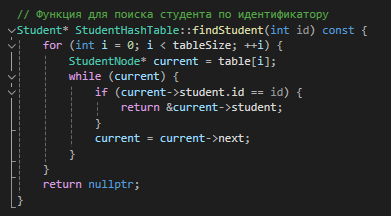


Рисунок 48 – Метод findStudent для поиска студента по идентификатору

Метод void removeStudent(int id) (рис. 49) удаляет студента из хэш-таблицы по его идентификатору. Вначале он проходит по всем индексам таблицы, начиная с 0 до tableSize - 1. Для каждого индекса проверяет наличие элементов в связанном списке. Если находит элемент с совпадающим идентификатором, то проверяет, является ли этот элемент первым в списке (то есть prev == nullptr). Если это так, то обновляет указатель таблицы на следующий элемент. В противном случае, обновляет указатель next предыдущего элемента списка на следующий элемент текущего. Удаленный элемент освобождается с помощью оператора delete. После удаления, количество элементов уменьшается на 1, и выводится сообщение о том, что студент удален. Если студент не найден, выводится соответствующее сообщение.

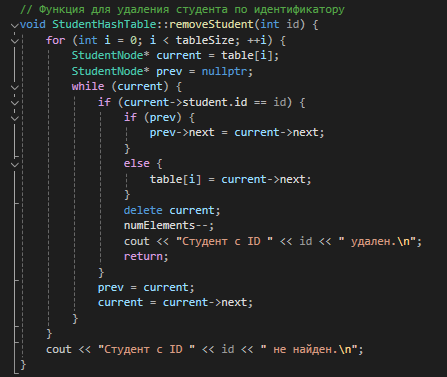


Рисунок 49 - Метод removeStudent для удаления студента по идентификатору

Метод void removeAllStudents() (рис. 50) удаляет всех студентов из хэш-таблицы. Он проходит по всем индексам таблицы, начиная с 0 до tableSize - 1, и для каждого индекса проверяет наличие элементов в связанном списке. Если элементы существуют, он проходит по всему списку, удаляя каждый элемент с помощью оператора delete. После удаления всех элементов по текущему индексу, указатель таблицы для этого индекса устанавливается в nullptr. В конце, количество элементов устанавливается в 0, а следующий идентификатор студента (nextId) сбрасывается на 1.

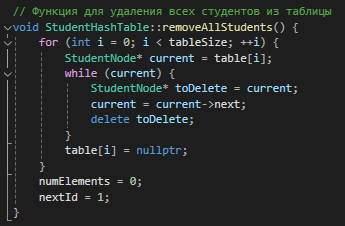


Рисунок 50 - Метод removeAllStudents для удаления всех студентов из таблицы

Метод moveStudentToGroup (рис. 51) предназначен для перемещения студента из одной группы в другую, обеспечивая корректное обновление данных в хэш-таблицах обеих групп. Метод принимает три параметра: идентификатор студента, которого необходимо переместить (int id), хэш-таблицу новой группы (StudentHashTable& newGroupTable), и номер новой группы (int newGroupNumber). Алгоритм работы метода начинается с поиска студента в текущей группе. Для этого используется метод findStudent, который находит студента по его идентификатору в текущей хэш-таблице. Если студент не найден, метод выводит соответствующее сообщение и завершает свою работу.

Если студент найден, его данные добавляются в хэш-таблицу новой группы с использованием метода addStudent. При этом передаются параметры: имя студента, оценки, размер стипендии, номер телефона и номер новой группы. После успешного добавления студента в новую группу, метод удаляет его из текущей группы, используя метод removeStudent. Это позволяет избежать дублирования данных и сохраняет целостность информации в системе. В конце метод выводит сообщение о том, что студент успешно перемещен в новую группу.

Метод isEmpty (рис. 51) проверяет, пуста ли таблица.

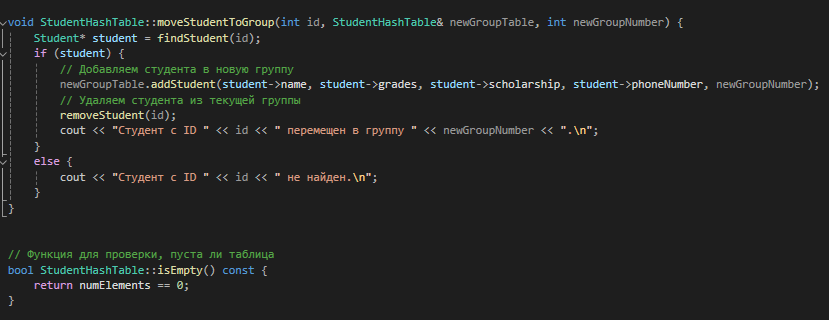


Рисунок 51 - Метод moveStudentToGroup для перемещения студента из одной группы в другую и метод isEmpty для проверки, пуста ли таблица

### 1.6.2. Методы для работы с файлами

Методы для работы с файлами в структуре StudentHashTable обеспечивают сохранение и загрузку данных о студентах. Эти функции позволяют хранить информацию о студентах в файлах и восстанавливать её при необходимости, что важно для долговременного хранения данных и обмена ими между сессиями программы.

Метод void saveToFile(ofstream& outFile) const (рис. 52) сохраняет данные всех студентов в файл. Этот метод проходит по всей хэш-таблице и для каждого индекса проверяет наличие элементов в связанном списке. Если элементы существуют, он последовательно сохраняет данные каждого студента в файл. Сначала записывается идентификатор студента, затем имя, оценки, стипендия и номер телефона. Для разделения записей используется специальный символ или строка. Метод обеспечивает последовательную запись данных, что упрощает последующую загрузку. После успешного выполнения метода, файл будет содержать полную информацию обо всех студентах, что позволяет восстановить состояние хэш-таблицы в будущем.

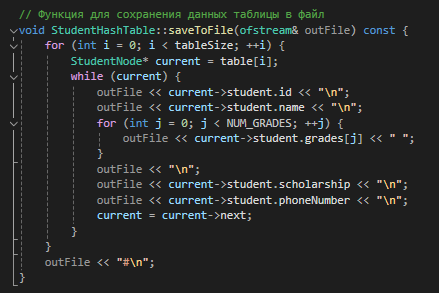


Рисунок 52 - Метод saveToFile для сохранения данных всех студентов в файл

Метод void loadFromFile(ifstream& inFile, int groupNumber) (рис. 52) загружает данные студентов из файла в хэш-таблицу. Этот метод очищает текущую хэш-таблицу, удаляя все существующие записи, и затем считывает данные из файла. Файл читается построчно, где каждая строка соответствует определенному полю студента. Сначала считывается идентификатор студента, затем имя, оценки, стипендия и номер телефона. После считывания всех полей создается новый объект Student, который добавляется в хэш-таблицу. Если файл содержит несколько записей студентов, метод последовательно обрабатывает каждую запись до конца файла. В процессе загрузки метод обновляет следующий идентификатор студента (nextId), чтобы избежать конфликтов идентификаторов при добавлении новых студентов.



Рисунок 52 - Метод loadFromFile для загрузки данных таблицы из файла

Эти методы обеспечивают надежное сохранение и восстановление данных о студентах. Они позволяют сохранять состояние хэш-таблицы между запусками программы, а также упрощают обмен данными и резервное копирование.

### 1.6.3. Метод для фильтрации студентов

Метод фильтрации студентов в структуре StudentHashTable позволяет выбирать и выводить студентов, соответствующих заданным критериям. Этот метод предоставляет гибкие средства для поиска и отбора студентов по различным параметрам, таким как имя, номер телефона, оценки, размер стипендии и номер группы.

Метод void filterStudents(const char\* name = nullptr, const char\* phoneNumber = nullptr, const int\* grades = nullptr, float scholarship = -1, int groupNumber = -1) (рис. 53) const реализует фильтрацию студентов. Этот метод проходит по всей хэш-таблице и проверяет каждого студента на соответствие заданным критериям. Если переданный параметр равен nullptr или имеет значение по умолчанию, он игнорируется в процессе фильтрации.

Фильтрация начинается с проверки имени студента. Если имя задано, и оно не совпадает с именем текущего студента, метод переходит к следующему студенту. Аналогично проверяется номер телефона, оценки, размер стипендии и номер группы. Если все заданные параметры совпадают с соответствующими полями текущего студента, его данные выводятся на экран.

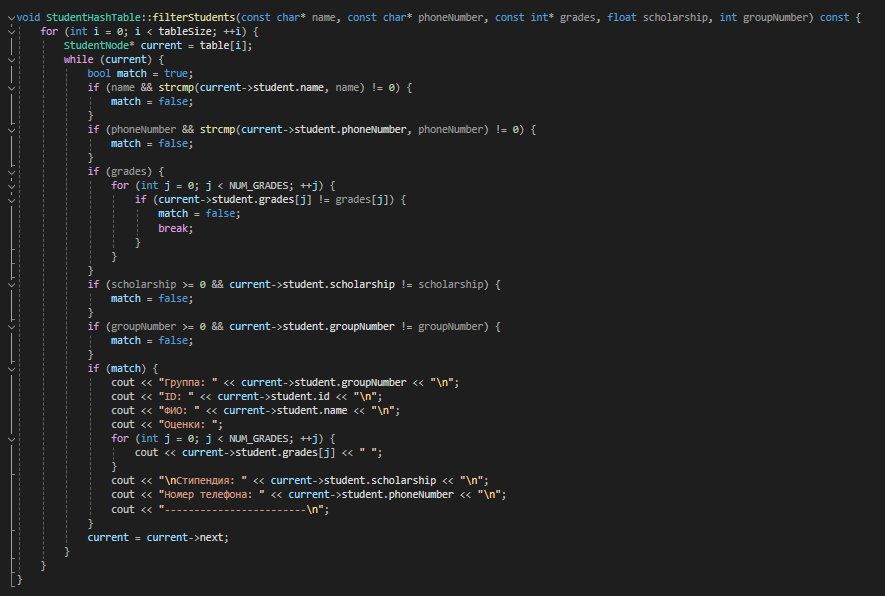


Рисунок 53 - Метод filterStudents для фильтрации студентов

## 1.7. Структура Group

Структура Group и связанные с ней методы предназначены для управления данными о группах студентов в системе. Эти методы включают функции создания, хранения, редактирования и удаления групп, что обеспечивает эффективное управление и обработку информации о группах.

Основной компонент для управления группами — это структура GroupNode (рис. 54), которая представляет собой узел, содержащий информацию о конкретной группе студентов. Поля структуры включают номер группы (int groupNumber), хэш-таблицу студентов (StudentHashTable students), а также указатели на следующий (GroupNode\* next) и предыдущий (GroupNode\* prev) узлы в двусвязном списке. Конструктор GroupNode(int num) (рис. 55) инициализирует объект GroupNode заданным номером группы и устанавливает указатели next и prev в nullptr.

Методы структуры GroupNode обеспечивают управление данными о группе. Метод void printStudents() выводит список всех студентов в группе. Он вызывает соответствующий метод хэш-таблицы студентов для печати данных. Метод float calculateTotalScholarship() const вычисляет общую сумму стипендий в группе, вызывая метод хэш-таблицы для вычисления суммарной стипендии всех студентов. Метод void saveToFile(ofstream& outFile) const сохраняет данные группы в файл, записывая номер группы и вызывая метод сохранения для хэш-таблицы студентов. Метод void loadFromFile(ifstream& inFile, int groupNumber) загружает данные группы из файла, устанавливая номер группы и вызывая метод загрузки для хэш-таблицы студентов.

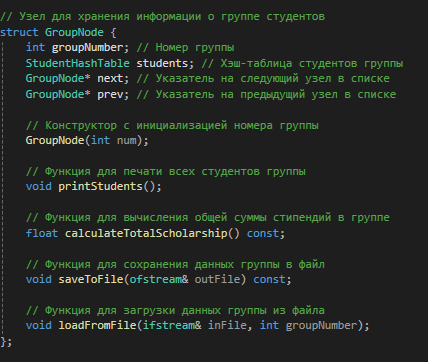


Рисунок 54 - Структура GroupNode для хранения информации о группе студентов



Рисунок 55 – Конструктор GroupNode с инициализацией номера группы

Структура GroupList (рис. 56) представляет собой двусвязный список групп, позволяя управлять множеством групп одновременно. Поля структуры включают указатели на первый (GroupNode\* head) и последний (GroupNode\* tail) узлы списка. Конструктор GroupList() (рис. 57) инициализирует пустой список групп. Методы структуры GroupList обеспечивают добавление, удаление и поиск групп, а также управление студентами в группах. Метод void addGroup(int groupNumber) добавляет новую группу в список, создавая новый узел и обновляя указатели. Метод void removeGroup(int groupNumber) удаляет группу по номеру, обновляя указатели соседних узлов и освобождая память. Метод GroupNode\* findGroup(int groupNumber) const ищет группу по номеру, возвращая указатель на соответствующий узел.

Методы для управления студентами в группах включают void addStudentToGroup(int groupNumber, const char\* studentName, const int\* grades, float stipend, const char\* phoneNumber), который добавляет студента в указанную группу, вызывая соответствующий метод хэш-таблицы. Метод void removeStudentFromGroup(int groupNumber, int studentId) удаляет студента из группы, вызывая метод удаления в хэш-таблице. Метод void removeAllStudentsFromGroup(int groupNumber) удаляет всех студентов из указанной группы, очищая хэш-таблицу.

Методы для работы с данными групп включают void printAllGroupNumbers() const и void printAllGroups() const, которые выводят номера всех групп и информацию обо всех группах и их студентах соответственно. Метод float calculateTotalScholarship() const вычисляет общую сумму стипендий во всех группах, суммируя значения из каждого узла. Метод bool isEmpty() const проверяет, пуст ли список групп. Метод void printGroupScholarshipPercentage() const вычисляет и выводит процент стипендий по каждой группе относительно общей суммы.

Для сохранения и загрузки данных о группах служат методы void saveToFile(const char\* filename) const и void loadFromFile(const char\* filename). Метод сохранения записывает данные всех групп в файл, вызывая метод сохранения для каждого узла. Метод загрузки очищает текущий список групп и загружает данные из файла, создавая новые узлы и вызывая метод загрузки для каждого узла.

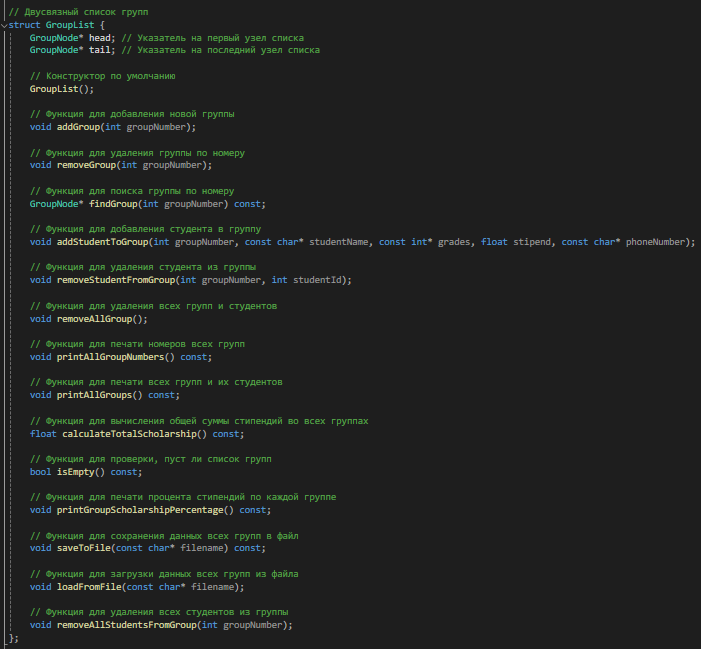


Рисунок 56 - Структура GroupList двусвязный список групп



Рисунок 57 - Конструктор GroupList

### 1.7.1. Методы для управления группами

Структура GroupList содержит методы, которые обеспечивают создание, удаление, поиск и управление данными о группах студентов. Эти методы позволяют эффективно управлять информацией о группах, добавлять и удалять группы, а также осуществлять поиск групп по их номеру.

Метод void addGroup(int groupNumber) (рис. 58) добавляет новую группу в список. Этот метод создает новый узел GroupNode, инициализируя его номером группы. Затем он обновляет указатели списка: если список пуст, новый узел становится и головным, и хвостовым элементом. Если список не пуст, новый узел добавляется в конец, и указатель next предыдущего хвостового узла обновляется на новый узел, а указатель prev нового узла устанавливается на предыдущий хвостовой узел. Новый узел становится хвостовым элементом списка.

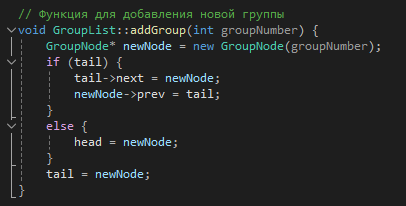


Рисунок 58 – Метод addGroup для добавления новой группы

Метод void removeGroup(int groupNumber) (рис. 59) удаляет группу по номеру. Этот метод проходит по списку, начиная с головного узла, и ищет узел с соответствующим номером группы. Если такой узел найден, метод обновляет указатели соседних узлов: если удаляемый узел является головным, указатель head обновляется на следующий узел; если удаляемый узел является хвостовым, указатель tail обновляется на предыдущий узел. Если узел находится посередине списка, указатели next и prev соседних узлов обновляются соответственно. После этого удаляемый узел освобождается с помощью оператора delete, и выводится сообщение о том, что группа удалена. Если группа не найдена, выводится соответствующее сообщение.

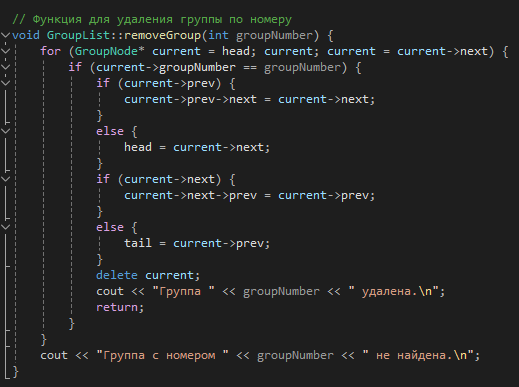


Рисунок 59 - Метод removeGroup для удаления группы по номеру

Метод GroupNode\* findGroup(int groupNumber) (рис. 60) const ищет группу по номеру. Этот метод проходит по списку, начиная с головного узла, и сравнивает номер каждой группы с искомым номером. Если узел с соответствующим номером группы найден, метод возвращает указатель на этот узел. Если поиск доходит до конца списка и группа не найдена, метод возвращает nullptr.

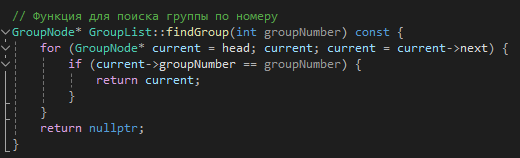


Рисунок 60 - Метод findGroup для поиска группы по номеру

Методы для управления студентами в группах включают void addStudentToGroup(int groupNumber, const char\* studentName, const int\* grades, float stipend, const char\* phoneNumber) (рис. 61), который добавляет студента в указанную группу. Этот метод сначала находит нужную группу с помощью метода findGroup. Если группа найдена, вызывается метод addStudent для хэш-таблицы студентов данной группы, добавляя нового студента. Если группа не найдена, выводится соответствующее сообщение.

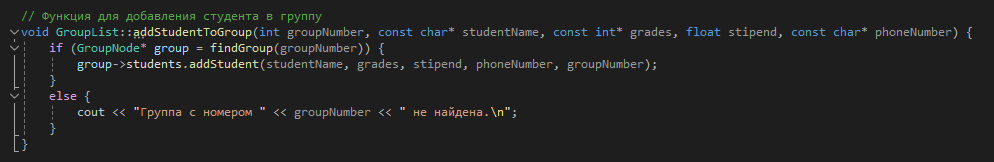


Рисунок 61 - Метод addStudentToGroup для добавления студента в указанную группу

Метод void removeStudentFromGroup(int groupNumber, int studentId) (рис. 62) удаляет студента из группы. Этот метод находит нужную группу с помощью метода findGroup. Если группа найдена, вызывается метод removeStudent для хэш-таблицы студентов данной группы, удаляя студента по его идентификатору. Если группа не найдена, выводится соответствующее сообщение.

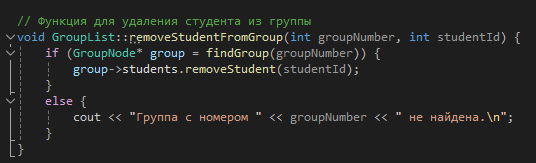


Рисунок 62 - Метод removeStudentFromGroup для удаления студента из группы

Метод void removeAllStudentsFromGroup(int groupNumber) (рис. 63) удаляет всех студентов из указанной группы. Этот метод находит нужную группу с помощью метода findGroup. Если группа найдена, вызывается метод removeAllStudents для хэш-таблицы студентов данной группы, удаляя всех студентов. Если группа не найдена, выводится соответствующее сообщение.

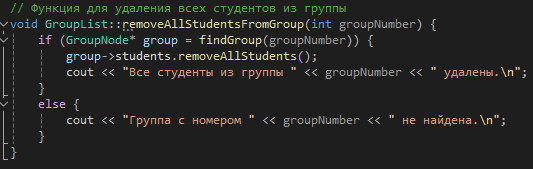


Рисунок 63 - Метод removeAllStudentsFromGroup для удаления всех студентов из указанной группы

Метод void removeAllGroup() (рис. 64) удаляет все группы и студентов. Этот метод проходит по всему списку, начиная с головного узла, и удаляет каждый узел один за другим. Для каждого узла сначала вызывается метод removeAllStudents, чтобы удалить всех студентов из текущей группы, а затем освобождается память, занимаемая узлом. Указатели head и tail обновляются на nullptr, указывая на то, что список теперь пуст. После выполнения этого метода все группы и студенты будут удалены из списка, и выводится сообщение о том, что все группы удалены.

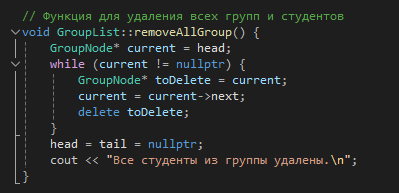


Рисунок 64 - Метод removeAllGroup для удаления всех групп и студентов

### 1.7.2. Методы для работы с файлами

Методы для работы с файлами в структуре GroupList и GroupNode обеспечивают сохранение и загрузку данных о группах и студентах. Эти функции позволяют сохранять информацию о группах в файлах и восстанавливать её при необходимости, что важно для долговременного хранения данных и обмена ими между сессиями программы.

Метод void saveToFile(const char\* filename) const (рис. 65) сохраняет данные всех групп в файл. Этот метод проходит по всему списку групп, начиная с головного узла, и для каждой группы вызывает метод saveToFile узла GroupNode. Вначале метод формирует полное имя файла, добавляя расширение .txt, если оно не указано. Затем открывает файл для записи и, если файл успешно открыт, проходит по списку групп. Для каждой группы сначала записывается её номер, а затем вызывается метод saveToFile хэш-таблицы студентов данной группы. После записи всех данных файл закрывается. Если файл не удалось открыть, выводится сообщение об ошибке.

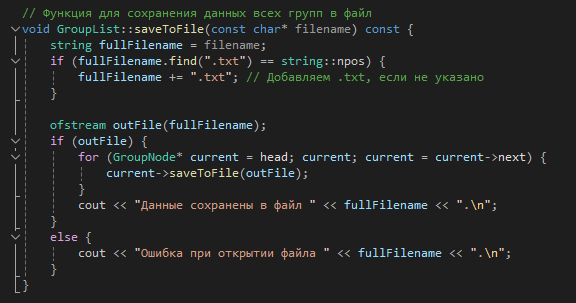


Рисунок 65 - Метод saveToFile для сохранения данных всех групп в файл

Метод void loadFromFile(const char\* filename) (рис. 66) загружает данные всех групп из файла. Этот метод сначала очищает текущий список групп, вызывая метод removeAllGroup. Затем метод формирует полное имя файла, добавляя расширение .txt, если оно не указано, и открывает файл для чтения. Если файл успешно открыт, метод последовательно считывает данные из файла. В цикле читается номер группы, создается новый узел GroupNode, и для него вызывается метод loadFromFile хэш-таблицы студентов данной группы. Новая группа добавляется в конец списка, обновляя указатели head и tail при необходимости. После загрузки всех данных файл закрывается. Если файл не удалось открыть, выводится сообщение об ошибке.

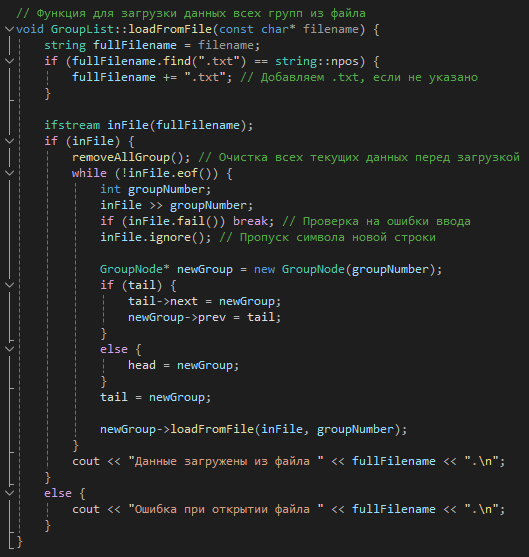


Рисунок 66 - Метод loadFromFile для загрузки данных всех групп из файла

Метод void GroupNode::saveToFile(ofstream& outFile) (рис. 67) const сохраняет данные конкретной группы в файл. Вначале записывается номер группы, затем вызывается метод saveToFile хэш-таблицы студентов, который записывает данные всех студентов данной группы в файл. Метод void GroupNode::loadFromFile(ifstream& inFile, int groupNumber) (рис. 67) загружает данные группы из файла. Вначале считывается номер группы, затем вызывается метод loadFromFile хэш-таблицы студентов, который восстанавливает данные всех студентов из файла. После этого группа добавляется в список групп.

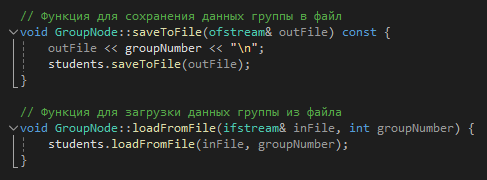


Рисунок 67 - Методы GroupNode::saveToFile и GroupNode::loadFromFile

Эти методы обеспечивают надежное сохранение и восстановление данных о группах и студентах. Они позволяют сохранять состояние списка групп между запусками программы, а также упрощают обмен данными и резервное копирование.

### 1.7.3. Методы для печати информации и вычисления данных

Структура GroupList содержит методы, которые обеспечивают вывод информации о группах и студентах на экран. Эти методы позволяют пользователю получить детализированные данные о группах, их студентах, а также об общей сумме стипендий.

Метод void printAllGroupNumbers() const (рис. 68) выводит на экран номера всех групп в списке. Этот метод проходит по всему списку, начиная с головного узла, и выводит номер каждой группы. Если список групп пуст, метод выводит сообщение о том, что список пуст.

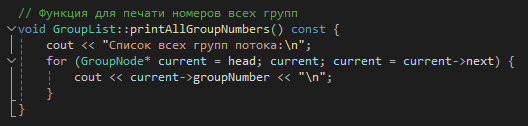


Рисунок 68 - Метод printAllGroupNumbers

Метод void printAllGroups() const (рис. 69) выводит информацию обо всех группах и их студентах. Этот метод проходит по всему списку, начиная с головного узла, и для каждой группы вызывает метод printStudents, который выводит список всех студентов в данной группе. Если список групп пуст, метод выводит сообщение о том, что список пуст.

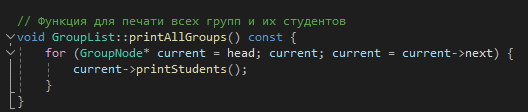


Рисунок 69 - Метод printAllGroups

Метод void GroupNode::printStudents() const (рис. 70) выводит информацию обо всех студентах конкретной группы. Этот метод вызывает соответствующий метод хэш-таблицы студентов printAllStudents, который проходит по всем элементам хэш-таблицы и выводит данные каждого студента: идентификационный номер, имя, оценки, размер стипендии и номер телефона. Если список студентов пуст, метод выводит сообщение о том, что список студентов пуст.

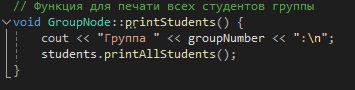


Рисунок 70 - Метод GroupNode::printStudents

Метод void printGroupScholarshipPercentage() const (рис. 71) вычисляет и выводит процент стипендий по каждой группе относительно общей суммы стипендий. Вначале метод вычисляет общую сумму стипендий всех групп, вызывая метод calculateTotalScholarship. Затем он проходит по списку групп и для каждой группы вычисляет процент от общей суммы стипендий, вызывая метод GroupNode::calculateTotalScholarship для каждой группы. Полученный процент выводится на экран вместе с номером группы. Если общая сумма стипендий равна нулю, метод выводит соответствующее сообщение, избегая деления на ноль.

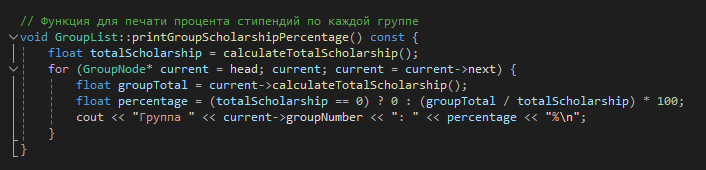


Рисунок 71 - Метод printGroupScholarshipPercentage

Метод float calculateTotalScholarship() const (рис. 72) вычисляет общую сумму стипендий во всех группах. Этот метод проходит по всему списку групп, начиная с головного узла, и для каждой группы вызывает метод GroupNode::calculateTotalScholarship хэш-таблицы студентов данной группы. Метод GroupNode::calculateTotalScholarship (рис. 73) возвращает сумму стипендий всех студентов в группе, суммируя стипендии всех студентов, хранящихся в хэш-таблице. Полученные суммы стипендий суммируются и возвращаются как итоговая общая сумма стипендий всех групп.

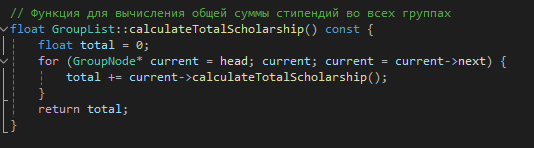


Рисунок 72 - Метод calculateTotalScholarship

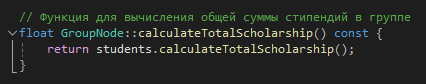


Рисунок 73 - Метод GroupNode::calculateTotalScholarship

Метод bool isEmpty() const (рис. 74) проверяет, пуст ли список групп. Этот метод возвращает true, если указатель на головной узел списка (head) равен nullptr, что указывает на пустой список. Если head указывает на какой-либо узел, метод возвращает false, что означает, что в списке есть хотя бы одна группа. Этот метод полезен для быстрого определения, содержит ли список какие-либо группы, и может использоваться перед выполнением операций, требующих наличия групп.

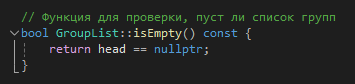


Рисунок 74 - Метод isEmpty

Эти методы обеспечивают наглядное представление данных о группах и студентах, позволяя пользователю легко получать нужную информацию и анализировать данные.

## 1.8. Функции взаимодействия с пользователем

Файл course-work-2sem.cpp содержит несколько функций, обеспечивающих взаимодействие пользователя с программой. Эти функции предоставляют интерфейс для управления данными о группах и студентах, позволяя пользователю выполнять различные операции, такие как добавление, редактирование, удаление и фильтрация записей.

Функция void safeReadString(char\* buffer, int maxLength) (рис. 29) обеспечивает безопасное считывание строки из ввода, предотвращая переполнение буфера. Она считывает строку до заданной длины и игнорирует все оставшиеся символы в потоке ввода, чтобы избежать возможных проблем с дальнейшим вводом. Функция принимает два параметра: buffer — указатель на массив символов, куда будет записана строка, и maxLength — максимальная длина строки, включая завершающий нулевой символ. После считывания строки, функция очищает оставшиеся символы в потоке ввода, чтобы предотвратить нежелательное поведение при следующем вводе.

Функция void filterMenu(GroupList& groupList) (рис. 30) предоставляет пользователю возможность фильтрации студентов по различным критериям. Пользователь может выбрать фильтрацию по ФИО, номеру телефона, оценкам, размеру стипендии или номеру группы. Функция отображает меню с соответствующими опциями и обрабатывает выбор пользователя. В зависимости от выбранного критерия, программа считывает необходимую информацию (например, имя студента или номер телефона) и вызывает соответствующий метод фильтрации для каждой группы, выводя отфильтрованные данные на экран. Функция использует цикл для повторного отображения меню до тех пор, пока пользователь не выберет опцию выхода.

Функция void editStudentMenu(GroupList& groupList, int groupNumber, int studentId) (рис. 31) позволяет пользователю редактировать данные конкретного студента. Пользователь может изменить ФИО, оценки, размер стипендии, номер телефона или переместить студента в другую группу. Функция сначала находит указанную группу и студента по их идентификаторам. Затем отображается меню с опциями редактирования, и программа обрабатывает выбор пользователя, предлагая ввести новые данные для выбранного поля. После ввода новых данных вызываются соответствующие методы для обновления информации о студенте. Если пользователь выбирает перемещение студента в другую группу, программа добавляет студента в новую группу и удаляет из старой.

Главное меню программы реализовано функцией void menu(GroupList& groupList) (рис. 32- 37). Оно предоставляет пользователю интерфейс для выполнения различных операций с группами и студентами. Пользователь может выбрать из следующих опций:

* Добавление группы: программа запрашивает номер группы и добавляет новую группу, если группы с таким номером еще не существует.
* Удаление группы: программа запрашивает номер группы и удаляет её, если группа существует.
* Удаление всех студентов группы: программа запрашивает номер группы и удаляет всех студентов из указанной группы.
* Показ процентов стипендий по группам: программа вычисляет и отображает процент стипендий по каждой группе.
* Удаление всех групп: программа удаляет все группы и их студентов.
* Добавление студента в группу: программа запрашивает данные студента и добавляет его в указанную группу.
* Удаление студента из группы: программа запрашивает номер группы и идентификатор студента, затем удаляет студента из группы.
* Редактирование данных студента: программа запрашивает номер группы и идентификатор студента, затем позволяет редактировать данные студента.
* Показ всех групп и студентов: программа отображает информацию обо всех группах и их студентах.
* Фильтрация студентов: программа вызывает функцию фильтрации студентов.
* Сохранение данных в файл: программа запрашивает имя файла и сохраняет данные всех групп в файл.
* Загрузка данных из файла: программа запрашивает имя файла и загружает данные всех групп из файла.
* Завершение работы: программа завершает выполнение.

Функция int main() (рис. 38) является точкой входа в программу. Она инициализирует список групп GroupList groupList и вызывает главное меню для взаимодействия с пользователем. В функции main устанавливается кодировка консоли для корректного ввода и вывода данных на русском языке с помощью функций SetConsoleCP(1251) и SetConsoleOutputCP(1251). Затем вызывается функция menu(groupList), которая обеспечивает основной интерфейс пользователя и предоставляет все возможности для управления данными о студентах и группах.

Эти функции обеспечивают удобное взаимодействие пользователя с программой, позволяя выполнять все необходимые операции для управления данными о студентах и группах.

# 2. Тестирование программы

## 2.1. Работа программы

Были проверены различные методы удаления и добавления студентов

Ввод:

1

3371

6

3371

Иванов Иван Иванович

5 5 4 5 4

2500

+79570727899

9

4

7

3371

1

9

Вывод:

Введите номер группы: Группа успешно добавлена

Введите номер группы: Группа с номером 3371 уже существует

Введите ФИО студента (до 50 символов): Введите 5 оценок: Введите размер стипендии: Введите номер телефона (в формате +7XXXXXXXXX): Студент успешно добавлен

Список всех групп потока:

3371

Группа 3371: 100%

Список всех групп потока:

3371

Группа 3371:

ID: 1

ФИО: Иванов Иван Иванович

Оценки: 5 5 4 5 4

Стипендия: 2500

Номер телефона: +79570727899

------------------------

Введите номер группы: Введите идентификационный номер студента: Студент с ID 1 удален.

Список всех групп потока:

3371

Группа 3371:

Список студентов пуст

Список всех групп потока:

3371

Группа 3371:

Список студентов пуст

# Заключение

В ходе выполнения курсовой работы была разработана и реализована программа для управления данными о студентах и группах в учебном потоке. Программа, написанная на языке программирования C++, предоставляет возможность добавлять и удалять группы и студентов, редактировать данные студентов, фильтровать студентов по различным критериям, а также сохранять и загружать данные из файлов.

Основная цель работы заключалась в создании гибкой и эффективной системы управления учебными данными, способной обрабатывать большое количество групп и студентов с высокой скоростью. Все поставленные задачи были успешно решены. В ходе разработки использованы динамические структуры данных, такие как хэш-таблица и двусвязный список, обеспечивающие быструю обработку данных. Реализованы основные функции программы, включая консольный ввод/вывод данных, редактирование и фильтрацию информации, а также сохранение и загрузку данных из файлов. Применение эффективных структур данных позволило минимизировать время выполнения операций добавления, удаления и поиска.

Проведенное тестирование подтвердило корректность выполнения всех предусмотренных функций. Результаты тестирования показали, что программа работает стабильно и соответствует поставленной цели.

Таким образом, разработанная система полностью удовлетворяет требованиям задания и обеспечивает надежное и эффективное управление данными о студентах и группах, что подтверждается успешными результатами тестирования.

# Список использованных источников

1. Павловская Т.А., Чаевников В.В., Юрков Н.К. Программирование на языке С++. Электронное методическое пособие. СПб.: Издательство СПбГЭТУ «ЛЭТИ». 2014.
2. Практикум по процедурному программированию на языке C++ С. А. Ивановский, А. А. Лисс, В. П. Самойленко, О. М. Шолохова. Практикум по процедурному программированию на языке C++: учеб. пособие. СПб.: Издательство СПбГЭТУ«ЛЭТИ», 2016.
3. Справочник по стандартной библиотеке C++ (STL) // Справочный сайт learn.microsoft.com URL: https://learn.microsoft.com/ru-ru/cpp/standard-library/cpp-standard-library-reference?view=msvc-170 (дата обращения 21.05.2024).
4. Документация по языку C++ // Справочный сайт learn.microsoft.com URL: https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения 22.05.2024).
5. Файловые потоки. Открытие и закрытие // Интернет платформа metanit.com URL: https://metanit.com/cpp/tutorial/8.2.php (дата обращения 23.05.2024).

# Приложение A

## Код написанной программы

Файл «student.h»:

#ifndef STUDENT\_H

#define STUDENT\_H

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

// Константы для максимальной длины имени, номера телефона и количества оценок

constexpr int MAX\_NAME\_LENGTH = 50;

constexpr int PHONE\_NUMBER\_LENGTH = 13; // Максимальная длина номера телефона, включая код страны

constexpr int NUM\_GRADES = 5;

// Функция для вычисления хэш-значения имени студента

unsigned int hashStudentName(const char\* name);

// Структура для хранения информации о студенте

struct Student {

int id; // Идентификационный номер студента

char name[MAX\_NAME\_LENGTH]; // Имя студента

int grades[NUM\_GRADES]; // Оценки студента

float scholarship; // Стипендия студента

char phoneNumber[PHONE\_NUMBER\_LENGTH]; // Номер телефона студента

int groupNumber; // Номер группы студента

// Конструктор с параметрами по умолчанию

Student(int studentId = 0, const char\* studentName = "", const int\* studentGrades = nullptr, float stipend = 0.0, const char\* extra = "", int group = 0);

};

// Узел связного списка для студентов

struct StudentNode {

Student student;

StudentNode\* next;

StudentNode(const Student& stud, StudentNode\* nextNode = nullptr)

: student(stud), next(nextNode) {}

};

// Структура хэш-таблицы для хранения студентов

struct StudentHashTable {

StudentNode\*\* table; // Массив указателей на узлы

int tableSize; // Размер таблицы

int numElements; // Количество элементов в таблице

int nextId; // Следующий идентификатор для нового студента

// Конструктор с инициализацией размера таблицы

StudentHashTable(int initialSize = 50);

// Деструктор для освобождения памяти

~StudentHashTable();

// Функция для добавления студента в хэш-таблицу

void addStudent(const char\* name, const int\* grades, float stipend, const char\* phoneNumber, int groupNumber);

// Функция для поиска студента по идентификатору

Student\* findStudent(int id) const;

// Функция для удаления студента по идентификатору

void removeStudent(int id);

// Функция для вычисления общей суммы стипендий

float calculateTotalScholarship() const;

// Функция для печати всех студентов

void printAllStudents() const;

// Функция для сохранения данных таблицы в файл

void saveToFile(ofstream& outFile) const;

// Функция для загрузки данных таблицы из файла

void loadFromFile(ifstream& inFile, int groupNumber);

// Функция для проверки, пуста ли таблица

bool isEmpty() const;

// Функция для удаления всех студентов из таблицы

void removeAllStudents();

// Функция для увеличения размера таблицы

void resizeTable();

// Функция для фильтрации студентов по параметрам

void filterStudents(const char\* name = nullptr, const char\* phoneNumber = nullptr, const int\* grades = nullptr, float scholarship = -1, int groupNumber = -1) const;

// Добавляем метод для перемещения студента в другую группу

void moveStudentToGroup(int id, StudentHashTable& newGroupTable, int newGroupNumber);

private:

// Хэш-функция для вычисления индекса на основе имени студента

unsigned int hashFunction(const char\* name) const;

};

#endif

Файл «student.cpp»:

#include "student.h"

// Функция для вычисления хэш-значения имени студента

unsigned int hashStudentName(const char\* name) {

unsigned int hash = 0;

while (\*name) {

hash = (hash \* 31) + static\_cast<unsigned int>(\*name);

++name;

}

return hash;

}

// Конструктор структуры Student

Student::Student(int studentId, const char\* studentName, const int\* studentGrades, float stipend, const char\* extra, int group)

: id(studentId), scholarship(stipend), groupNumber(group) {

strncpy\_s(name, studentName, MAX\_NAME\_LENGTH - 1);

name[MAX\_NAME\_LENGTH - 1] = '\0';

for (int j = 0; j < NUM\_GRADES; ++j) {

grades[j] = studentGrades ? studentGrades[j] : 0;

}

strncpy\_s(phoneNumber, extra, PHONE\_NUMBER\_LENGTH - 1);

phoneNumber[PHONE\_NUMBER\_LENGTH - 1] = '\0';

}

// Конструктор хэш-таблицы с инициализацией

StudentHashTable::StudentHashTable(int initialSize)

: tableSize(initialSize), numElements(0), nextId(1) {

table = new StudentNode \* [tableSize]();

}

// Деструктор для освобождения памяти хэш-таблицы

StudentHashTable::~StudentHashTable() {

removeAllStudents();

delete[] table;

}

// Хэш-функция для вычисления индекса на основе имени студента

unsigned int StudentHashTable::hashFunction(const char\* name) const {

return hashStudentName(name) % tableSize;

}

// Функция для добавления студента в хэш-таблицу

void StudentHashTable::addStudent(const char\* name, const int\* grades, float stipend, const char\* phoneNumber, int groupNumber) {

if (numElements > tableSize \* 0.75) {

resizeTable();

}

unsigned int hash = hashFunction(name);

Student newStudent(nextId++, name, grades, stipend, phoneNumber, groupNumber);

StudentNode\* newNode = new StudentNode(newStudent);

if (!table[hash]) {

table[hash] = newNode;

}

else {

StudentNode\* current = table[hash];

while (current->next) {

current = current->next;

}

current->next = newNode;

}

numElements++;

}

// Функция для увеличения размера таблицы

void StudentHashTable::resizeTable() {

int newSize = tableSize \* 2;

StudentNode\*\* newTable = new StudentNode \* [newSize]();

for (int i = 0; i < tableSize; ++i) {

StudentNode\* current = table[i];

while (current) {

unsigned int newHash = hashStudentName(current->student.name) % newSize;

StudentNode\* next = current->next;

current->next = newTable[newHash];

newTable[newHash] = current;

current = next;

}

}

delete[] table;

table = newTable;

tableSize = newSize;

}

// Функция для поиска студента по идентификатору

Student\* StudentHashTable::findStudent(int id) const {

for (int i = 0; i < tableSize; ++i) {

StudentNode\* current = table[i];

while (current) {

if (current->student.id == id) {

return &current->student;

}

current = current->next;

}

}

return nullptr;

}

// Функция для удаления студента по идентификатору

void StudentHashTable::removeStudent(int id) {

for (int i = 0; i < tableSize; ++i) {

StudentNode\* current = table[i];

StudentNode\* prev = nullptr;

while (current) {

if (current->student.id == id) {

if (prev) {

prev->next = current->next;

}

else {

table[i] = current->next;

}

delete current;

numElements--;

cout << "Студент с ID " << id << " удален.\n";

return;

}

prev = current;

current = current->next;

}

}

cout << "Студент с ID " << id << " не найден.\n";

}

// Функция для удаления всех студентов из таблицы

void StudentHashTable::removeAllStudents() {

for (int i = 0; i < tableSize; ++i) {

StudentNode\* current = table[i];

while (current) {

StudentNode\* toDelete = current;

current = current->next;

delete toDelete;

}

table[i] = nullptr;

}

numElements = 0;

nextId = 1;

}

// Функция для вычисления общей суммы стипендий

float StudentHashTable::calculateTotalScholarship() const {

float total = 0;

for (int i = 0; i < tableSize; ++i) {

StudentNode\* current = table[i];

while (current) {

total += current->student.scholarship;

current = current->next;

}

}

return total;

}

// Функция для печати всех студентов

void StudentHashTable::printAllStudents() const {

if (isEmpty()) {

cout << "Список студентов пуст\n";

return;

}

for (int i = 0; i < tableSize; ++i) {

StudentNode\* current = table[i];

while (current) {

cout << "ID: " << current->student.id << "\n";

cout << "ФИО: " << current->student.name << "\n";

cout << "Оценки: ";

for (int j = 0; j < NUM\_GRADES; ++j) {

cout << current->student.grades[j] << " ";

}

cout << "\nСтипендия: " << current->student.scholarship << "\n";

cout << "Номер телефона: " << current->student.phoneNumber << "\n";

cout << "------------------------\n";

current = current->next;

}

}

}

// Функция для сохранения данных таблицы в файл

void StudentHashTable::saveToFile(ofstream& outFile) const {

for (int i = 0; i < tableSize; ++i) {

StudentNode\* current = table[i];

while (current) {

outFile << current->student.id << "\n";

outFile << current->student.name << "\n";

for (int j = 0; j < NUM\_GRADES; ++j) {

outFile << current->student.grades[j] << " ";

}

outFile << "\n";

outFile << current->student.scholarship << "\n";

outFile << current->student.phoneNumber << "\n";

current = current->next;

}

}

outFile << "#\n";

}

void StudentHashTable::filterStudents(const char\* name, const char\* phoneNumber, const int\* grades, float scholarship, int groupNumber) const {

for (int i = 0; i < tableSize; ++i) {

StudentNode\* current = table[i];

while (current) {

bool match = true;

if (name && strcmp(current->student.name, name) != 0) {

match = false;

}

if (phoneNumber && strcmp(current->student.phoneNumber, phoneNumber) != 0) {

match = false;

}

if (grades) {

for (int j = 0; j < NUM\_GRADES; ++j) {

if (current->student.grades[j] != grades[j]) {

match = false;

break;

}

}

}

if (scholarship >= 0 && current->student.scholarship != scholarship) {

match = false;

}

if (groupNumber >= 0 && current->student.groupNumber != groupNumber) {

match = false;

}

if (match) {

cout << "Группа: " << current->student.groupNumber << "\n";

cout << "ID: " << current->student.id << "\n";

cout << "ФИО: " << current->student.name << "\n";

cout << "Оценки: ";

for (int j = 0; j < NUM\_GRADES; ++j) {

cout << current->student.grades[j] << " ";

}

cout << "\nСтипендия: " << current->student.scholarship << "\n";

cout << "Номер телефона: " << current->student.phoneNumber << "\n";

cout << "------------------------\n";

}

current = current->next;

}

}

}

void StudentHashTable::moveStudentToGroup(int id, StudentHashTable& newGroupTable, int newGroupNumber) {

Student\* student = findStudent(id);

if (student) {

// Добавляем студента в новую группу

newGroupTable.addStudent(student->name, student->grades, student->scholarship, student->phoneNumber, newGroupNumber);

// Удаляем студента из текущей группы

removeStudent(id);

cout << "Студент с ID " << id << " перемещен в группу " << newGroupNumber << ".\n";

}

else {

cout << "Студент с ID " << id << " не найден.\n";

}

}

// Функция для загрузки данных таблицы из файла

void StudentHashTable::loadFromFile(ifstream& inFile, int groupNumber) {

char line[MAX\_NAME\_LENGTH];

nextId = 1;

while (true) {

inFile.getline(line, MAX\_NAME\_LENGTH);

if (strcmp(line, "#") == 0 || inFile.eof()) {

break;

}

int id = atoi(line);

inFile.getline(line, MAX\_NAME\_LENGTH);

char studentName[MAX\_NAME\_LENGTH];

strncpy\_s(studentName, line, MAX\_NAME\_LENGTH);

int grades[NUM\_GRADES];

for (int i = 0; i < NUM\_GRADES; ++i) {

inFile >> grades[i];

}

inFile.ignore();

float stipend;

inFile >> stipend;

inFile.ignore();

inFile.getline(line, PHONE\_NUMBER\_LENGTH);

char phoneNumber[PHONE\_NUMBER\_LENGTH];

strncpy\_s(phoneNumber, line, PHONE\_NUMBER\_LENGTH);

addStudent(studentName, grades, stipend, phoneNumber, groupNumber);

if (id >= nextId) {

nextId = id + 1;

}

}

}

// Функция для проверки, пуста ли таблица

bool StudentHashTable::isEmpty() const {

return numElements == 0;

}

Файл «group.h»:

#ifndef GROUP\_H

#define GROUP\_H

#include "student.h"

// Узел для хранения информации о группе студентов

struct GroupNode {

int groupNumber; // Номер группы

StudentHashTable students; // Хэш-таблица студентов группы

GroupNode\* next; // Указатель на следующий узел в списке

GroupNode\* prev; // Указатель на предыдущий узел в списке

// Конструктор с инициализацией номера группы

GroupNode(int num);

// Функция для печати всех студентов группы

void printStudents();

// Функция для вычисления общей суммы стипендий в группе

float calculateTotalScholarship() const;

// Функция для сохранения данных группы в файл

void saveToFile(ofstream& outFile) const;

// Функция для загрузки данных группы из файла

void loadFromFile(ifstream& inFile, int groupNumber);

};

// Двусвязный список групп

struct GroupList {

GroupNode\* head; // Указатель на первый узел списка

GroupNode\* tail; // Указатель на последний узел списка

// Конструктор по умолчанию

GroupList();

// Функция для добавления новой группы

void addGroup(int groupNumber);

// Функция для удаления группы по номеру

void removeGroup(int groupNumber);

// Функция для поиска группы по номеру

GroupNode\* findGroup(int groupNumber) const;

// Функция для добавления студента в группу

void addStudentToGroup(int groupNumber, const char\* studentName, const int\* grades, float stipend, const char\* phoneNumber);

// Функция для удаления студента из группы

void removeStudentFromGroup(int groupNumber, int studentId);

// Функция для удаления всех групп и студентов

void removeAllGroup();

// Функция для печати номеров всех групп

void printAllGroupNumbers() const;

// Функция для печати всех групп и их студентов

void printAllGroups() const;

// Функция для вычисления общей суммы стипендий во всех группах

float calculateTotalScholarship() const;

// Функция для проверки, пуст ли список групп

bool isEmpty() const;

// Функция для печати процента стипендий по каждой группе

void printGroupScholarshipPercentage() const;

// Функция для сохранения данных всех групп в файл

void saveToFile(const char\* filename) const;

// Функция для загрузки данных всех групп из файла

void loadFromFile(const char\* filename);

// Функция для удаления всех студентов из группы

void removeAllStudentsFromGroup(int groupNumber);

};

#endif

Файл «group.cpp»:

#include "group.h"

// Конструктор GroupNode с инициализацией номера группы

GroupNode::GroupNode(int num) : groupNumber(num), next(nullptr), prev(nullptr) {}

// Функция для печати всех студентов группы

void GroupNode::printStudents() {

cout << "Группа " << groupNumber << ":\n";

students.printAllStudents();

}

// Функция для вычисления общей суммы стипендий в группе

float GroupNode::calculateTotalScholarship() const {

return students.calculateTotalScholarship();

}

// Функция для сохранения данных группы в файл

void GroupNode::saveToFile(ofstream& outFile) const {

outFile << groupNumber << "\n";

students.saveToFile(outFile);

}

// Функция для загрузки данных группы из файла

void GroupNode::loadFromFile(ifstream& inFile, int groupNumber) {

students.loadFromFile(inFile, groupNumber);

}

// Конструктор GroupList по умолчанию

GroupList::GroupList() : head(nullptr), tail(nullptr) {}

// Функция для добавления новой группы

void GroupList::addGroup(int groupNumber) {

GroupNode\* newNode = new GroupNode(groupNumber);

if (tail) {

tail->next = newNode;

newNode->prev = tail;

}

else {

head = newNode;

}

tail = newNode;

}

// Функция для удаления группы по номеру

void GroupList::removeGroup(int groupNumber) {

for (GroupNode\* current = head; current; current = current->next) {

if (current->groupNumber == groupNumber) {

if (current->prev) {

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

}

else {

head = current->next;

}

if (current->next) {

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

}

else {

tail = current->prev;

}

delete current;

cout << "Группа " << groupNumber << " удалена.\n";

return;

}

}

cout << "Группа с номером " << groupNumber << " не найдена.\n";

}

// Функция для поиска группы по номеру

GroupNode\* GroupList::findGroup(int groupNumber) const {

for (GroupNode\* current = head; current; current = current->next) {

if (current->groupNumber == groupNumber) {

return current;

}

}

return nullptr;

}

// Функция для добавления студента в группу

void GroupList::addStudentToGroup(int groupNumber, const char\* studentName, const int\* grades, float stipend, const char\* phoneNumber) {

if (GroupNode\* group = findGroup(groupNumber)) {

group->students.addStudent(studentName, grades, stipend, phoneNumber, groupNumber);

}

else {

cout << "Группа с номером " << groupNumber << " не найдена.\n";

}

}

// Функция для удаления студента из группы

void GroupList::removeStudentFromGroup(int groupNumber, int studentId) {

if (GroupNode\* group = findGroup(groupNumber)) {

group->students.removeStudent(studentId);

}

else {

cout << "Группа с номером " << groupNumber << " не найдена.\n";

}

}

// Функция для удаления всех студентов из группы

void GroupList::removeAllStudentsFromGroup(int groupNumber) {

if (GroupNode\* group = findGroup(groupNumber)) {

group->students.removeAllStudents();

cout << "Все студенты из группы " << groupNumber << " удалены.\n";

}

else {

cout << "Группа с номером " << groupNumber << " не найдена.\n";

}

}

// Функция для удаления всех групп и студентов

void GroupList::removeAllGroup() {

GroupNode\* current = head;

while (current != nullptr) {

GroupNode\* toDelete = current;

current = current->next;

delete toDelete;

}

head = tail = nullptr;

cout << "Все студенты из группы удалены.\n";

}

// Функция для печати номеров всех групп

void GroupList::printAllGroupNumbers() const {

cout << "Список всех групп потока:\n";

for (GroupNode\* current = head; current; current = current->next) {

cout << current->groupNumber << "\n";

}

}

// Функция для печати всех групп и их студентов

void GroupList::printAllGroups() const {

for (GroupNode\* current = head; current; current = current->next) {

current->printStudents();

}

}

// Функция для вычисления общей суммы стипендий во всех группах

float GroupList::calculateTotalScholarship() const {

float total = 0;

for (GroupNode\* current = head; current; current = current->next) {

total += current->calculateTotalScholarship();

}

return total;

}

// Функция для проверки, пуст ли список групп

bool GroupList::isEmpty() const {

return head == nullptr;

}

// Функция для печати процента стипендий по каждой группе

void GroupList::printGroupScholarshipPercentage() const {

float totalScholarship = calculateTotalScholarship();

for (GroupNode\* current = head; current; current = current->next) {

float groupTotal = current->calculateTotalScholarship();

float percentage = (totalScholarship == 0) ? 0 : (groupTotal / totalScholarship) \* 100;

cout << "Группа " << current->groupNumber << ": " << percentage << "%\n";

}

}

// Функция для сохранения данных всех групп в файл

void GroupList::saveToFile(const char\* filename) const {

string fullFilename = filename;

if (fullFilename.find(".txt") == string::npos) {

fullFilename += ".txt"; // Добавляем .txt, если не указано

}

ofstream outFile(fullFilename);

if (outFile) {

for (GroupNode\* current = head; current; current = current->next) {

current->saveToFile(outFile);

}

cout << "Данные сохранены в файл " << fullFilename << ".\n";

}

else {

cout << "Ошибка при открытии файла " << fullFilename << ".\n";

}

}

// Функция для загрузки данных всех групп из файла

void GroupList::loadFromFile(const char\* filename) {

string fullFilename = filename;

if (fullFilename.find(".txt") == string::npos) {

fullFilename += ".txt"; // Добавляем .txt, если не указано

}

ifstream inFile(fullFilename);

if (inFile) {

removeAllGroup(); // Очистка всех текущих данных перед загрузкой

while (!inFile.eof()) {

int groupNumber;

inFile >> groupNumber;

if (inFile.fail()) break; // Проверка на ошибки ввода

inFile.ignore(); // Пропуск символа новой строки

GroupNode\* newGroup = new GroupNode(groupNumber);

if (tail) {

tail->next = newGroup;

newGroup->prev = tail;

}

else {

head = newGroup;

}

tail = newGroup;

newGroup->loadFromFile(inFile, groupNumber);

}

cout << "Данные загружены из файла " << fullFilename << ".\n";

}

else {

cout << "Ошибка при открытии файла " << fullFilename << ".\n";

}

}

Файл «course-work-2sem.cpp»:

#include <iostream>

#include <windows.h>

#include "group.h"

using namespace std;

// Функция для безопасного чтения строки из ввода

void safeReadString(char\* buffer, int maxLength) {

cin.get(buffer, maxLength);

while (cin.get() != '\n') {} // Чтение и игнорирование всех оставшихся символов в строке ввода

}

void filterMenu(GroupList& groupList) {

int choice;

do {

system("cls");

cout << "\nФильтрация студентов:\n";

cout << "1. Фильтрация по ФИО\n";

cout << "2. Фильтрация по номеру телефона\n";

cout << "3. Фильтрация по оценкам\n";

cout << "4. Фильтрация по стипендии\n";

cout << "5. Фильтрация по номеру группы\n";

cout << "6. Назад\n";

cout << "Выберите пункт: ";

cin >> choice;

char studentName[MAX\_NAME\_LENGTH];

char phoneNumber[PHONE\_NUMBER\_LENGTH];

int grades[NUM\_GRADES];

float scholarship;

int groupNumber;

switch (choice) {

case 1:

cout << "Введите ФИО студента: ";

cin.ignore();

safeReadString(studentName, MAX\_NAME\_LENGTH);

for (GroupNode\* group = groupList.head; group; group = group->next) {

group->students.filterStudents(studentName);

}

system("pause");

break;

case 2:

cout << "Введите номер телефона: ";

cin.ignore();

safeReadString(phoneNumber, PHONE\_NUMBER\_LENGTH);

for (GroupNode\* group = groupList.head; group; group = group->next) {

group->students.filterStudents(nullptr, phoneNumber);

}

system("pause");

break;

case 3:

cout << "Введите " << NUM\_GRADES << " оценок: ";

for (int i = 0; i < NUM\_GRADES; ++i) {

cin >> grades[i];

}

for (GroupNode\* group = groupList.head; group; group = group->next) {

group->students.filterStudents(nullptr, nullptr, grades);

}

system("pause");

break;

case 4:

cout << "Введите размер стипендии: ";

cin >> scholarship;

for (GroupNode\* group = groupList.head; group; group = group->next) {

group->students.filterStudents(nullptr, nullptr, nullptr, scholarship);

}

system("pause");

break;

case 5:

cout << "Введите номер группы: ";

cin >> groupNumber;

for (GroupNode\* group = groupList.head; group; group = group->next) {

group->students.filterStudents(nullptr, nullptr, nullptr, -1, groupNumber);

}

system("pause");

break;

case 6:

return;

default:

cout << "Неверный ввод, попробуйте снова.\n";

break;

}

} while (choice != 6);

}

void editStudentMenu(GroupList& groupList, int groupNumber, int studentId) {

GroupNode\* group = groupList.findGroup(groupNumber);

if (!group) {

cout << "Группа с номером " << groupNumber << " не найдена.\n";

return;

}

Student\* student = group->students.findStudent(studentId);

if (!student) {

cout << "Студент с ID " << studentId << " не найден\n";

return;

}

int choice;

do {

system("cls");

cout << "Редактирование студента (ID: " << studentId << ")\n";

cout << "1. Изменить ФИО\n";

cout << "2. Изменить оценки\n";

cout << "3. Изменить стипендию\n";

cout << "4. Изменить номер телефона\n";

cout << "5. Переместить в другую группу\n";

cout << "6. Назад\n";

cout << "Выберите пункт: ";

cin >> choice;

cin.ignore(); // Пропуск символа новой строки

switch (choice) {

case 1: {

char studentName[MAX\_NAME\_LENGTH];

cout << "Введите новое ФИО: ";

safeReadString(studentName, MAX\_NAME\_LENGTH);

strncpy\_s(student->name, studentName, MAX\_NAME\_LENGTH);

cout << "ФИО изменено.\n";

system("pause");

break;

}

case 2: {

int grades[NUM\_GRADES];

cout << "Введите новые оценки (" << NUM\_GRADES << " оценок): ";

for (int i = 0; i < NUM\_GRADES; ++i) {

cin >> grades[i];

}

for (int i = 0; i < NUM\_GRADES; ++i) {

student->grades[i] = grades[i];

}

cout << "Оценки изменены.\n";

system("pause");

break;

}

case 3: {

float stipend;

cout << "Введите новый размер стипендии: ";

cin >> stipend;

student->scholarship = stipend;

cout << "Стипендия изменена.\n";

system("pause");

break;

}

case 4: {

char phoneNumber[PHONE\_NUMBER\_LENGTH];

cout << "Введите новый номер телефона: ";

safeReadString(phoneNumber, PHONE\_NUMBER\_LENGTH);

strncpy\_s(student->phoneNumber, phoneNumber, PHONE\_NUMBER\_LENGTH);

cout << "Номер телефона изменен.\n";

system("pause");

break;

}

case 5: {

int newGroupNumber;

cout << "Введите новый номер группы: ";

cin >> newGroupNumber;

cin.ignore(); // Пропуск символа новой строки

GroupNode\* newGroup = groupList.findGroup(newGroupNumber);

if (!newGroup) {

cout << "Группа с номером " << newGroupNumber << " не найдена. Создаем новую группу.\n";

groupList.addGroup(newGroupNumber);

newGroup = groupList.findGroup(newGroupNumber);

}

group->students.moveStudentToGroup(studentId, newGroup->students, newGroupNumber);

return; // Выходим из меню редактирования, так как студент перемещен

}

case 6:

return;

default:

cout << "Неверный ввод, попробуйте снова.\n";

system("pause");

break;

}

} while (choice != 6);

}

// Меню для взаимодействия с пользователем

void menu(GroupList& groupList) {

SetConsoleCP(1251); // Установка кодировки консоли для ввода

SetConsoleOutputCP(1251); // Установка кодировки консоли для вывода

int choice;

do {

system("cls");

cout << "\n\tРАБОТА С ГРУППАМИ\n";

cout << "1. Добавить группу\n";

cout << "2. Удалить группу\n";

cout << "3. Удалить всех студентов группы\n";

cout << "4. Показать проценты стипендии по группам\n";

cout << "5. Удалить все группы\n";

cout << "\n\tРАБОТА СО СТУДЕНТАМИ\n";

cout << "6. Добавить студента в группу\n";

cout << "7. Удалить студента из группы\n";

cout << "8. Редактировать данные студента\n";

cout << "9. Показать все группы и студентов\n";

cout << "10. Фильтрация студентов\n";

cout << "\n\tРАБОТА С ФАЙЛАМИ\n";

cout << "11. Сохранить данные в файл\n";

cout << "12. Загрузить данные из файла\n";

cout << "\n\tЗАВЕРШЕНИЕ РАБОТЫ\n";

cout << "13. Выйти\n";

cout << "Выберите пункт: ";

cin >> choice;

switch (choice) {

case 1: {

int groupNumber;

cout << "Введите номер группы: ";

cin >> groupNumber;

if (groupList.findGroup(groupNumber)) {

cout << "Группа с номером " << groupNumber << " уже существует\n";

}

else {

groupList.addGroup(groupNumber);

cout << "Группа успешно добавлена\n";

}

system("pause");

break;

}

case 2: {

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

system("cls");

int groupNumber;

groupList.printAllGroupNumbers();

cout << "Введите номер группы для удаления: ";

cin >> groupNumber;

groupList.removeGroup(groupNumber);

}

system("pause");

break;

}

case 3: {

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

int groupNumber;

system("cls");

groupList.printAllGroupNumbers();

cout << "Введите номер группы: ";

cin >> groupNumber;

cin.ignore(); // Пропуск символа новой строки

GroupNode\* group = groupList.findGroup(groupNumber);

if (!group) {

cout << "Группа с номером " << groupNumber << " не найдена.\n";

}

else {

group->printStudents();

if (!(group->students.isEmpty())) {

groupList.removeAllStudentsFromGroup(groupNumber);

}

}

}

system("pause");

break;

}

case 4:

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

system("cls");

groupList.printGroupScholarshipPercentage();

}

system("pause");

break;

case 5: {

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

groupList.removeAllGroup();

}

system("pause");

break;

}

case 6: {

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

int groupNumber, grades[NUM\_GRADES];

char studentName[MAX\_NAME\_LENGTH];

char phoneNumber[PHONE\_NUMBER\_LENGTH];

float stipend;

system("cls");

groupList.printAllGroupNumbers();

cout << "Введите номер группы: ";

cin >> groupNumber;

cin.ignore(); // Пропуск символа новой строки

if (!groupList.findGroup(groupNumber)) {

cout << "Группа с номером " << groupNumber << " не найдена.\n";

}

else {

cout << "Введите ФИО студента (до 50 символов): ";

safeReadString(studentName, MAX\_NAME\_LENGTH);

cout << "Введите " << NUM\_GRADES << " оценок: ";

for (int i = 0; i < NUM\_GRADES; ++i) {

cin >> grades[i];

}

cout << "Введите размер стипендии: ";

cin >> stipend;

cin.ignore();

cout << "Введите номер телефона (в формате +7XXXXXXXXX): ";

safeReadString(phoneNumber, PHONE\_NUMBER\_LENGTH);

groupList.addStudentToGroup(groupNumber, studentName, grades, stipend, phoneNumber);

cout << "Студент успешно добавлен\n";

}

}

system("pause");

break;

}

case 7: {

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

int groupNumber, studentId;

system("cls");

groupList.printAllGroupNumbers();

cout << "Введите номер группы: ";

cin >> groupNumber;

cin.ignore(); // Пропуск символа новой строки

GroupNode\* group = groupList.findGroup(groupNumber);

if (!group) {

cout << "Группа с номером " << groupNumber << " не найдена.\n";

}

else {

group->printStudents();

if (!(group->students.isEmpty())) {

cout << "Введите идентификационный номер студента: ";

cin >> studentId;

cin.ignore(); // Пропуск символа новой строки

groupList.removeStudentFromGroup(groupNumber, studentId);

}

}

}

system("pause");

break;

}

case 8: {

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

int groupNumber, studentId;

system("cls");

groupList.printAllGroupNumbers();

cout << "Введите номер группы: ";

cin >> groupNumber;

cin.ignore(); // Пропуск символа новой строки

GroupNode\* group = groupList.findGroup(groupNumber);

if (!group) {

cout << "Группа с номером " << groupNumber << " не найдена.\n";

}

else {

group->printStudents();

if (!(group->students.isEmpty())) {

cout << "Введите идентификационный номер студента: ";

cin >> studentId;

cin.ignore(); // Пропуск символа новой строки

if (!group->students.findStudent(studentId)) {

cout << "Студент с ID " << studentId << " не найден";

}

editStudentMenu(groupList, groupNumber, studentId);

}

}

}

system("pause");

break;

}

case 9:

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

system("cls");

groupList.printAllGroups();

}

system("pause");

break;

case 10: {

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

filterMenu(groupList);

}

system("pause");

break;

}

case 11: {

if (groupList.isEmpty()) {

cout << "Список групп пуст!\n";

}

else {

char filename[MAX\_NAME\_LENGTH];

cout << "Введите имя файла для сохранения: ";

cin.ignore();

safeReadString(filename, MAX\_NAME\_LENGTH);

groupList.saveToFile(filename);

}

system("pause");

break;

}

case 12: {

char filename[MAX\_NAME\_LENGTH];

cout << "Введите имя файла для загрузки: ";

cin.ignore();

safeReadString(filename, MAX\_NAME\_LENGTH);

groupList.loadFromFile(filename);

system("pause");

break;

}

case 13:

cout << "Выход из программы.\n";

break;

default:

cout << "Неверный ввод, попробуйте снова.\n";

break;

}

} while (choice != 13);

}

// Основная функция программы

int main() {

GroupList groupList;

menu(groupList);

return 0;

}

# Приложение B

Файл data.txt:

1

3371

6

3371

Иванов Иван Иванович

5 5 4 5 4

2500

+79570727899

9

4

7

3371

1

9

Файл result.txt:

Введите номер группы: Группа успешно добавлена

Введите номер группы: Группа с номером 3371 уже существует

Введите ФИО студента (до 50 символов): Введите 5 оценок: Введите размер стипендии: Введите номер телефона (в формате +7XXXXXXXXX): Студент успешно добавлен

Список всех групп потока:

3371

Группа 3371: 100%

Список всех групп потока:

3371

Группа 3371:

ID: 1

ФИО: Иванов Иван Иванович

Оценки: 5 5 4 5 4

Стипендия: 2500

Номер телефона: +79570727899

------------------------

Введите номер группы: Введите идентификационный номер студента: Студент с ID 1 удален.

Список всех групп потока:

3371

Группа 3371:

Список студентов пуст

Список всех групп потока:

3371

Группа 3371:

Список студентов пуст