**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 |  | Ларионов И.М |
| Преподаватель |  | Писарев А.С. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |
| --- |
| Студент Ларионов И.М. |
| 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:   * 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; * Вывести в алфавитном порядке фамилии N (вводится с клавиатуры) первых по среднему баллу студентов потока   Общие требования к программам:   * Программа должна поддерживать систему меню, пункты которых соответствуют выполнению функций, предусмотренных общей частью задания. * Предлагаемые структуры данных должны учитывать изначальную неопределенность возможного количества групп и студентов в группах, а также обеспечивать максимальную скорость процессов обработки данных, предусмотренных заданием. * Тексты программ должны содержать комментарии, объясняющие назначение основных функций, типов и объектов данных, функциональных блоков и т.п. * Представляемые тексты программ должны обеспечивать возможность их компиляции и построения в среде MS Visual Studio |

Дата выдачи задания: 21.04.2024

|  |  |
| --- | --- |
| Дата сдачи реферата: 00.00.2000 | |
| Дата защиты реферата:  00.00.2000 |

|  |  |  |
| --- | --- | --- |
| Студент |  | Ларионов И.М. |
| Преподаватель |  | Писарев А.С. |

**Аннотация**

В рамках данной курсовой работы была разработана программа на C++ для управления данными студентов учебного потока. Программа предоставляет функции для консольного и файлового ввода/вывода данных, редактирования информации о студентах и группах, а также сортировки групп по определённым критериям. Программа эффективно управляет данными студентов, включая вычисление соотношения стипендии к общему числу баллов студентов в каждой группе.

**Summary**

As part of this course project, a C++ program was developed to manage student data within an academic stream. The program provides functionalities for console and file-based input/output of data, editing student and group information, as well as sorting groups based on specified criteria. The program efficiently handles student data, including calculating the ratio of scholarships to the total number of student points within each group.

Заголовок

[Введение 6](#_Toc177813463)

[1. Описание блоков программы 7](#_Toc177813464)

[1.1. Подключаемые заголовочные файлы основного файла 7](#_Toc177813465)

[1.2. Структура Student 8](#_Toc177813466)

[1.3. Реализация двусвязного списка 8](#_Toc177813467)

[1.4. Методы управления студентами 13](#_Toc177813468)

[1.5. Функции для преобразования 17](#_Toc177813469)

[1.6. Функции для ввода/вывода данных студентов через файлы 17](#_Toc177813470)

[1.7. Основное меню и подменю программы 20](#_Toc177813471)

[1.8. Функция main для управления программой 22](#_Toc177813472)

[2. Тестирование программы: 28](#_Toc177813473)

[Заключение 31](#_Toc177813474)

[Приложение A 33](#_Toc177813475)

[**Код написанной программы** 33](#_Toc177813476)

[Приложение B 54](#_Toc177813477)

**Введение**

Целью данной курсовой работы является разработка программного обеспечения для управления данными студентов учебного потока. Программа должна предоставлять удобный и эффективный способ хранения, редактирования и обработки информации о студентах и группах. Основные задачи работы включают:

Основные задачи, поставленные в работе, включают:

* Создание структуры данных для хранения информации о студентах и группах.
* Реализация функций ввода и вывода данных как в консоль, так и в файл.
* Разработка механизмов для редактирования информации о студентах и группах, включая добавление и удаление записей.
* Обеспечение возможности фильтрации и сортировки данных по заданным критериям.

Для решения поставленных задач использованы следующие методы:

* Применение двусвязного списка для эффективного хранения и поиска информации о студентах.
* Реализация алгоритмов ввода, вывода, редактирования и фильтрации данных с использованием указанных структур данных.

В результате работы было создано программное обеспечение, позволяющее эффективно управлять данными о студентах и выполнять различные операции по их обработке.

Описание блоков программы

* 1. Подключаемые заголовочные файлы основного файла

Файл Course\_worke.cpp является основным модулем программы, который реализует пользовательский интерфейс и предоставляет ключевые функции для работы с данными о студентах и группах. В начале файла подключаются необходимые заголовочные файлы, а также объявляется использование стандартного пространства имен.

Заголовочные файлы:

* #include <iostream>: для работы с вводом/выводом данных через консоль.
* #include <string>: для работы с классом строк.
* #include <fstream>: для работы с файловым вводом/выводом.

Пространство имен:

* using namespace std;: Использование стандартного пространства имен для упрощения синтаксиса.

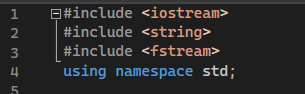


Рисунок 1 – Подключение библиотек и пространства имен

* 1. Структура Student

Структура Student(рис.2) предназначена для хранения и управления информацией о студенте. Она включает в себя основные данные о студенте, а также методы для их редактирования и получения.

Поля:

* ID: Уникальный идентификатор студента.
* group: Номер группы, в которой учится студент.
* stipend: Размер стипендии студента.
* student\_course: Номер курса, на котором учится студент.
* marks[5]: Массив из пяти оценок студента.
* name: Имя студента, представлено в виде строки.

Конструктор по умолчанию: Инициализирует все числовые поля нулями, строку name — пустой строкой, а массив оценок заполняет нулями.

Конструктор с параметрами: Позволяет инициализировать объект студента с заданными значениями для всех полей. Поле name обрезается до 50 символов, если строка превышает это значение. Массив оценок инициализируется переданным массивом marksArray.

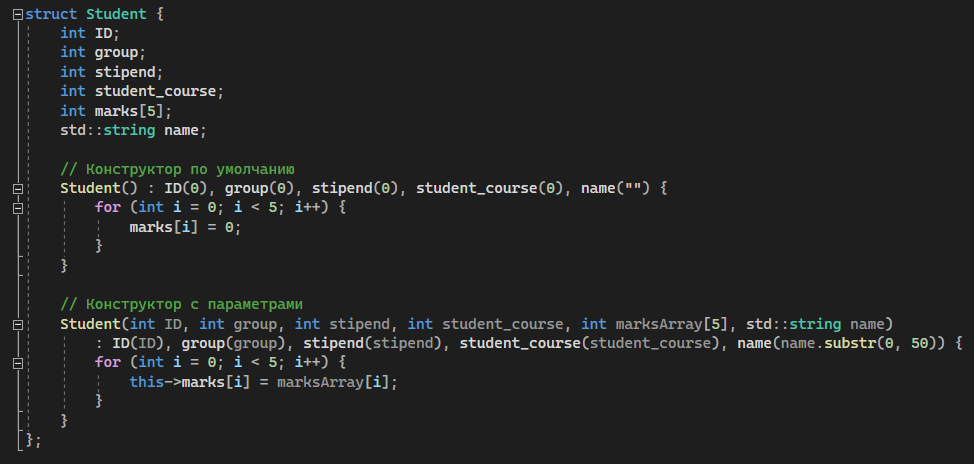


Рисунок 2 – Структура Student

* 1. **Реализация двусвязного списка**

Структуры Node и List используются для создания списка, а также для управления его элементами.

Структура Node:

* T data: Хранит данные узла (элемент списка).
* Node next\*: Указатель на следующий узел в списке.
* Node prev\*: Указатель на предыдущий узел в списке.

Структура List:

* size: Количество элементов в списке.
* currentIndex: Индекс текущего элемента, на который указывает current.
* current: Указатель на текущий узел списка.
* head: Указатель на первый узел списка.
* tail: Указатель на последний узел списка.

Основные методы:

* toFirst: Устанавливает указатель current на первый элемент списка (head), индекс текущего элемента — на 0.
* toLast: Устанавливает указатель current на последний элемент списка (tail) и обновляет currentIndex.
* toNext: Перемещает указатель на следующий узел списка, обновляя индекс.
* toPrev: Перемещает указатель на предыдущий узел списка, уменьшая индекс.
* currentTo: Перемещает указатель current на элемент с указанным индексом, оптимизируя движение (двигается от начала или от конца в зависимости от ближайшей точки).
* removeCurrent: Удаляет текущий узел (на который указывает current), обновляя связи между соседними элементами. Если удаляется первый или последний элемент, обновляются указатели head или tail.
* insertBeforeCurrent: Вставляет новый узел перед текущим элементом.
* getCurrent: Возвращает указатель на узел, находящийся по указанному индексу.
* add: Добавляет новый элемент в конец списка, обновляя указатель на последний узел (tail).
* insert: Вставляет элемент в указанную позицию списка. Если позиция равна количеству элементов, элемент добавляется в конец.
* removeAt: Удаляет элемент по указанному индексу.
* elementAt: Возвращает данные элемента, находящегося по указанному индексу.
* count: Возвращает количество элементов в списке.
* clear: Очищает список, удаляя все элементы по одному.

Эта структура реализует стандартные операции над двусвязным списком, такие как добавление, удаление, вставка и перемещение по элементам, что делает её удобной для управления динамическими наборами данных.

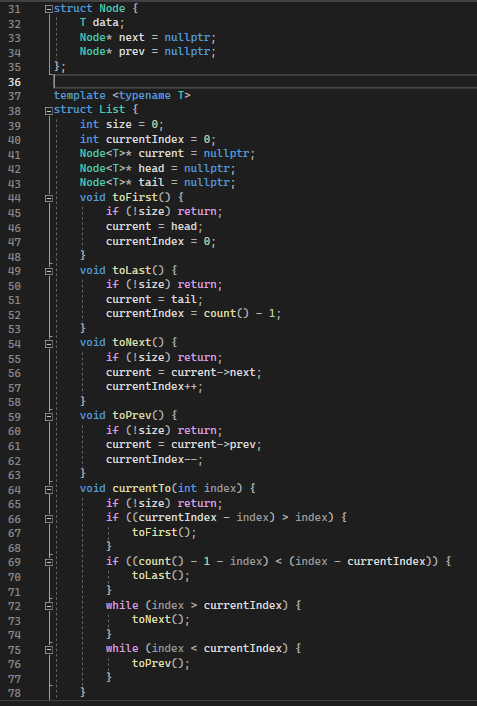


Рисунок 3 – Реализация двусвязного списка



Рисунок 4 – Реализация двусвязного списка

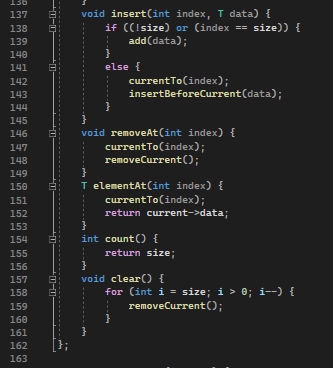


Рисунок 5 – Реализация двусвязного списка

* 1. Методы управления студентами

1. Student addStudent(int& ID)

Этот метод создает объект Student, запрашивая у пользователя информацию о студенте, и возвращает его.

* Уникальный ID присваивается студенту и автоматически увеличивается.
* Запрашивается имя студента, его группа, оценки (массив из 5 оценок), стипендия и курс.
* Возвращает заполненный объект Student.

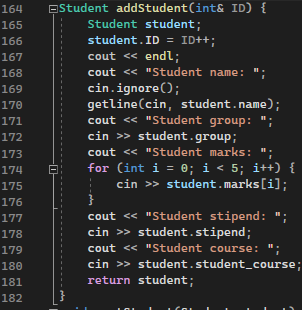


Рисунок 6 – Метод addStudent

2. void coutStudent(Student student)

Выводит в консоль информацию о студенте:

* ID, имя, номер группы.
* Оценки студента (массив из 5 оценок).
* Стипендия и курс.

Используется для форматированного вывода информации о конкретном студенте.

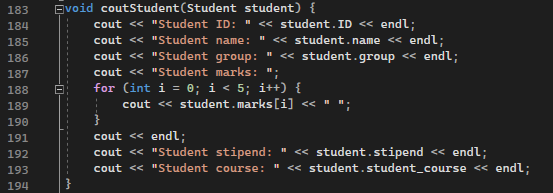


Рисунок 7 – Метод coutStudent

3. void removeGroup(List<Student>& studentList, int group)

Удаляет всех студентов из списка studentList, принадлежащих указанной группе.

* Проходит по всем элементам списка.
* Если студент принадлежит заданной группе, он удаляется.
* После удаления итератор смещается, чтобы продолжить обход списка корректно.

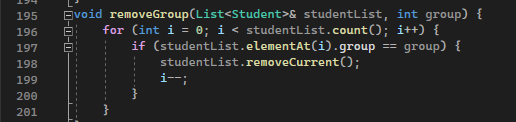


Рисунок 8 – Метод removeGroup

4. void removeStudent(List<Student>& studentList, int ID)

* Удаляет студента из списка по его ID.
* Пробегает по всему списку, пока не находит студента с заданным ID.
* При совпадении удаляет студента и завершает выполнение.

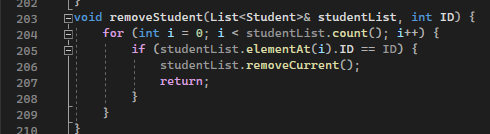


Рисунок 9 – Метод removeStudent

5. void studentsSortedByAverage(List<Student> list)

Сортирует список студентов по средней оценке, а при одинаковых средних — по имени.

* Копирует исходный список в новый outputList.
* Сортирует его пузырьковой сортировкой по средним оценкам (среднее арифметическое оценок из массива).
* Если средние оценки равны, сортирует студентов по алфавиту.
* Выводит отсортированный список с именем, группой и средней оценкой.

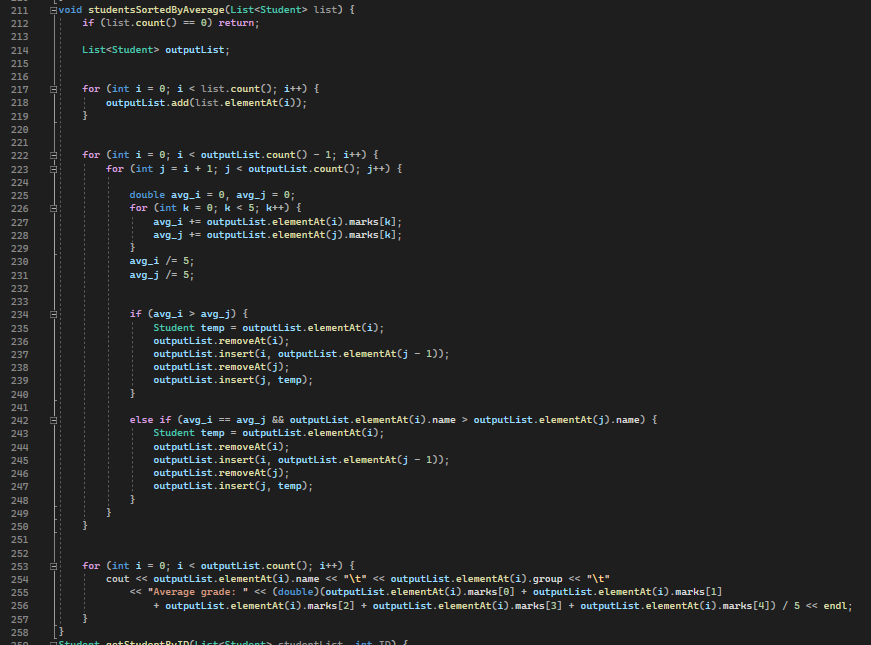


Рисунок 10 – Метод studentSortedByAverage

6. Student getStudentByID(List<Student> studentList, int ID)

Возвращает студента по заданному ID.

* Пробегает по списку и ищет студента с указанным ID.
* Если такой студент найден, он возвращается.
* Если студент с таким ID не найден, возвращается пустой объект Student.

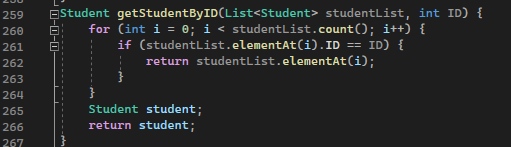


Рисунок 11 – Метод getStudentByID

* 1. Функции для преобразования

int toint(string str)

Преобразует строку в целое число.

* Последовательно проходит по строке с конца и собирает число, умножая каждую цифру на 10 в степени её позиции.
* Возвращает результат в виде целого числа.

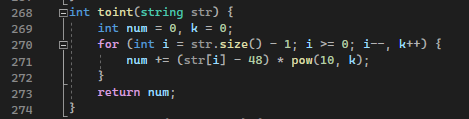


Рисунок 12 – Функция toint

int\* tointarr(string str)

Преобразует строку с числами в массив целых чисел (до 5 элементов).

* Проходит по строке и преобразует символы в числа.
* Заполняет массив чисел до 5 элементов и возвращает указатель на массив.
* Эти функции обеспечивают ввод, вывод, сортировку, удаление и поиск студентов, а также некоторые утилиты для преобразования строк в числовые типы

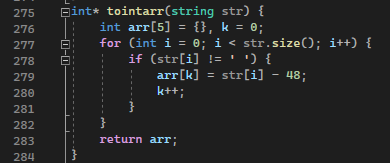


Рисунок 13 – Функция tointarr

* 1. Функции для ввода/вывода данных студентов через файлы

Этот блок программы включает две функции: fileInput для чтения данных о студентах из файла и добавления их в список, и fileOutput для записи данных студентов в файл. Оба метода взаимодействуют с файловыми потоками для работы с текстовыми файлами.

**1. void fileInput(List<Student>& list, int& ID)**

Функция читает данные о студентах из файла input.txt и добавляет их в список list.

* **Параметры**:
  + list — список студентов, куда добавляются данные.
  + ID — ссылка на идентификатор, который автоматически увеличивается для каждого студента.
* **Описание работы**:
  + Открывается файл input.txt для чтения.
  + Строки из файла читаются последовательно. Каждая строка соответствует одному из полей студента:
    1. Имя студента.
    2. Номер группы.
    3. Оценки (массив из 5 элементов).
    4. Стипендия.
    5. Курс.
  + Когда все 5 полей заполнены, создается объект Student, и он добавляется в список.
  + После этого счетчик строк сбрасывается для чтения данных следующего студента.
  + После завершения чтения файл закрывается.

**2. void fileOutput(List<Student> list)**

Функция записывает данные всех студентов из списка в файл

output.txt.

* **Параметры**:
  + list — список студентов, данные которых нужно записать в файл.
* **Описание работы**:
  + Открывается файл output.txt для записи.
  + Для каждого студента из списка последовательно записываются:
    - Имя.
    - Номер группы.
    - Оценки (5 значений).
    - Стипендия.
    - Курс.
    - ID студента.
  + Данные каждого студента выводятся в формате, удобном для последующего чтения.
  + После записи всех студентов файл закрывается.

**Основное назначение этих функций:**

* **fileInput** — загружает студентов из файла и добавляет их в список для дальнейшей обработки.
* **fileOutput** — сохраняет текущий список студентов в файл для долгосрочного хранения или передачи.

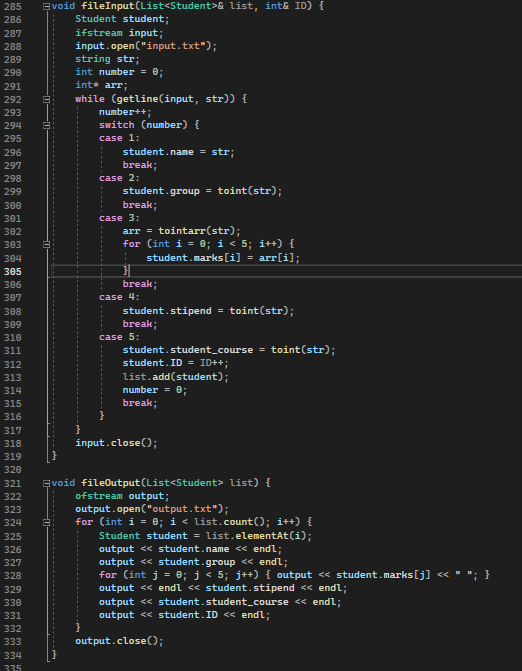


Рисунок 14 – Функции fileInput и filleOutput

* 1. Основное меню и подменю программы

Этот блок программы включает функции, отвечающие за вывод текстовых меню для взаимодействия с пользователем. Меню предоставляет доступ к различным функциям для работы с данными о студентах и группах.

**1. void menu()**

Функция выводит основное меню программы, предоставляя пользователю список доступных действий.

* **Пункты меню**:
  + 0 Terminate program: Завершить выполнение программы.
  + 1 Add student: Добавить нового студента.
  + 2 Remove student by ID: Удалить студента по его идентификатору.
  + 3 Edit student by ID: Изменить информацию о студенте по его идентификатору.
  + 4 Show student by ID: Показать информацию о студенте по идентификатору.
  + 5 Show students by parameter: Показать студентов по определённым критериям.
  + 6 Show all students: Показать список всех студентов.
  + 7 Add group of students: Добавить группу студентов.
  + 8 Remove group of students: Удалить группу студентов.
  + 9 Change group number: Изменить номер группы студентов.
  + 10 Show group of students: Показать информацию о группе студентов.
  + 11 Show students average marks: Показать средние оценки студентов.
  + 12 File input: Загрузить данные студентов из файла.
  + 13 File output: Сохранить данные студентов в файл.

**2. void submenu()**

Функция выводит подменю, используемое для выбора параметра, по которому будет выполнен поиск или фильтрация студентов.

* **Пункты подменю**:
  + 0 Cancel: Отмена действия.
  + 1 Name: Поиск по имени.
  + 2 Student course: Поиск по курсу студента.
  + 3 Group: Поиск по группе.
  + 4 Marks: Поиск по оценкам.
  + 5 Stipend: Поиск по стипендии.

**3. void submenu2()**

Функция выводит дополнительное подменю для действий, связанных с редактированием информации о студентах.

* **Пункты подменю**:
  + 0 Cancel: Отмена действия.
  + 1 Name: Изменение имени студента.
  + 2 Student course: Изменение курса студента.
  + 3 Stipend: Изменение стипендии студента.

**Основное назначение этих функций:**

* **menu()** — предоставляет пользователю полный список действий для работы с данными о студентах.
* **submenu()** — предлагает выбрать критерии для поиска или фильтрации студентов.
* **submenu2()** — используется для выбора параметров при редактировании информации о студентах.

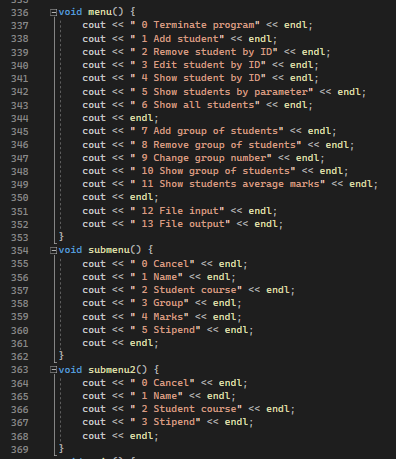


Рисунок 15 – Функции menu, submenu, submenu2

* 1. Функция main для управления программой

Эта функция реализует основной цикл программы, который предоставляет пользователю интерфейс для управления списком студентов. Пользователь может добавлять, удалять, редактировать студентов, сортировать их по параметрам, а также работать с файлами для ввода/вывода данных.

Объявление переменных

* List<Student> studentList;: Создаёт список студентов для хранения всех данных о студентах.
* int ID = 1;: Переменная для хранения уникальных идентификаторов студентов, которые автоматически увеличиваются.
* int choice = 0; и int choice2 = 0;: Переменные для хранения выбора пользователя из основного меню и подменю.
* int input = 0;: Переменная для ввода данных от пользователя.

**Цикл while(true)**

Программа работает в бесконечном цикле, предлагая пользователю действия через меню, пока не будет выполнен выход (команда 0).

Логика обработки команд (switch-case)

В зависимости от выбора пользователя выполняется соответствующее действие. Команды делятся на:

* 0 Terminate program: Завершение программы.
* 1 Add student: Добавление нового студента через функцию addStudent(ID).
* 2 Remove student by ID: Удаление студента по его уникальному идентификатору через функцию removeStudent().
* 3 Edit student by ID: Редактирование данных студента по его ID. Пользователь выбирает, какое поле нужно изменить, после чего данные обновляются.
* Подменю редактирования выводится через функцию submenu(), где можно выбрать редактируемое поле: имя, курс, группа, оценки или стипендия.
* 4 Show student by ID: Отображение информации о студенте по его идентификатору через функцию coutStudent().
* 5 Show students by parameter: Вывод студентов по заданному параметру. Можно выбрать: имя, курс или стипендию через подменю submenu2().
* 6 Show all students: Показ всех студентов с выводом их данных.
* 7 Add group of students: Добавление группы студентов. Вводится количество студентов, затем для каждого добавляется информация.
* 8 Remove group of students: Удаление всех студентов определённой группы через функцию removeGroup().
* 9 Change group number: Изменение номера группы для всех студентов в этой группе.
* 10 Show group of students: Показ всех студентов, принадлежащих к определённой группе.
* 11 Show students average marks: Вывод списка студентов, отсортированных по средним оценкам.
* 12 File input: Загрузка списка студентов из файла через функцию fileInput().
* 13 File output: Сохранение данных о студентах в файл через функцию fileOutput().



Рисунок 16 – Функция main



Рисунок 17 – Функция main



Рисунок 18 – Функция main

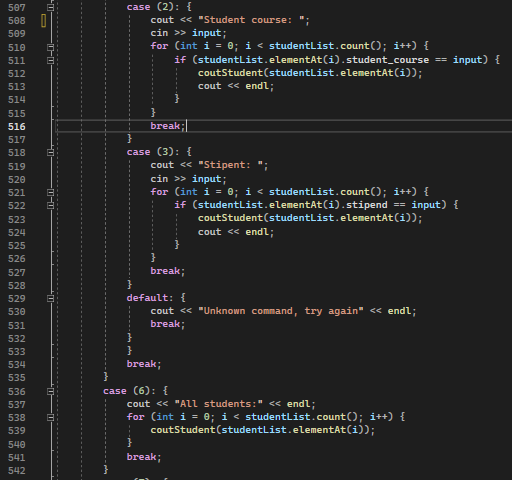


Рисунок 19 – Функция main

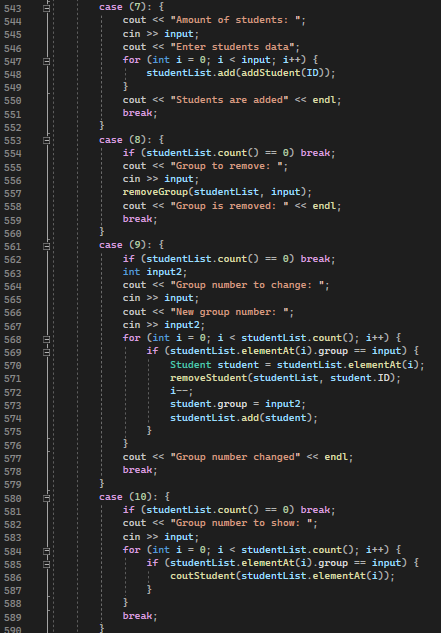


Рисунок 20 – Функция main

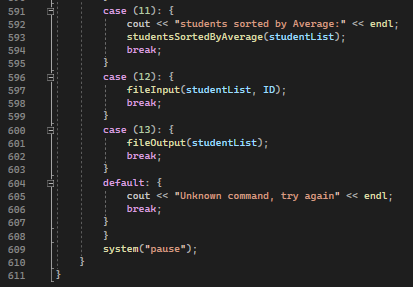


Рисунок 21 – Функция main

Тестирование программы:

Ввод данных студента (через меню)

Выбор пункта "1:

Add student": Вводим информацию о студенте:

Student name: Иванов Иван Иванович

Student group: 3371

Student marks: 5 4 3 5 4

Student stipend: 5000

Student course: 5

Вывод:

Student is added

Показ всех студентов (через меню)

Выбор пункта "6: Show all students":

Student ID: 1

Student name: Иванов Иван Иванович

Student group: 3371

Student marks: 5 4 3 5 4

Student stipend: 5000

Student course: 5

Удаление студента по ID (через меню)

Выбор пункта "2: Remove student by ID":

Вводим ID студента:

Enter student ID: 1

Вывод:

Student is removed

Редактирование студента по ID (через меню)

Выбор пункта "3: Edit student by ID":

Вводим ID студента:

Копировать код

ID of the student you want to edit: 1

Подменю редактирования (например, изменим имя студента):

What do you want to edit?

1 Name

2 Student course

3 Group

4 Marks

5 Stipend

Choice: 1

Enter new data: Петров Петр Петрович

Вывод:

Student name changed

Показ студентов по параметру (через меню)

Выбор пункта "5: Show students by parameter":

Например, по имени:

Choice: 1

Name: Петров Петр Петрович

Вывод:

Student ID: 1

Student name: Петров Петр Петрович

Student group: 3371

Student marks: 5 4 3 5 4

Student stipend: 5000

Student course: 5

Добавление группы студентов (через меню)

Выбор пункта "7: Add group of students":

Вводим количество студентов (например, 2):

Amount of students: 2

Вводим данные для двух студентов:

Student 1:

Student name: Сидоров Сергей Сергеевич

Student group: 3371

Student marks: 4 4 4 5 3

Student stipend: 6000

Student course: 4

Student 2:

Student name: Кузнецов Алексей Андреевич

Student group: 3371

Student marks: 5 5 5 4 4

Student stipend: 7000

Student course: 3

Вывод:

Students are added

Удаление группы студентов (через меню)

Выбор пункта "8: Remove group of students":

Вводим номер группы:

Group to remove: 3371

Вывод:

Group is removed

Сортировка студентов по средним оценкам (через меню)

Выбор пункта "11: Show students average marks":

Вывод студентов отсортированных по средним оценкам:

Петров Петр Петрович 3371 Average grade: 4.2

Иванов Иван Иванович 3371 Average grade: 4.4

Заключение

В ходе выполнения курсовой работы была создана и реализована программа для управления данными о студентах и учебных группах. Программа, написанная на языке C++, предоставляет функции добавления и удаления студентов и групп, редактирования данных студентов, фильтрации по различным критериям, а также возможности сохранения и загрузки данных из файлов.

Основной целью было разработать гибкую и эффективную систему для управления учебной информацией, способную обрабатывать неограниченное количество данных с высокой производительностью. Все задачи были успешно выполнены. В рамках работы была реализована динамическая структура данных, такая как двусвязный список. Программа поддерживает основные функции, включая консольный ввод/вывод, редактирование, фильтрацию данных, а также работу с файлами. Благодаря использованию эффективных структур данных, удалось минимизировать время выполнения операций по добавлению, удалению и поиску.

Тестирование подтвердило корректность работы всех функций программы. Результаты тестирования показали стабильность программы и её соответствие поставленным требованиям.

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

Список использованных источников

1. Н. И. Куракина. Основы программирования на С++: учебно-методическое пособие. СПб.: Издательство СПбГЭТУ «ЛЭТИ». 2017.
2. Практикум по процедурному программированию на языке C++ С. А. Ивановский, А. А. Лисс, В. П. Самойленко, О. М. Шолохова. СПб.: Издательство СПбГЭТУ«ЛЭТИ», 2016.
3. Павловская Т.А., Чаевников В.В., Юрков Н.К. Программирование на языке С++. Электронное методическое пособие. СПб.: Издательство СПбГЭТУ «ЛЭТИ». 2014.

# Приложение A

## **Код написанной программы**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

struct Student {

int ID;

int group;

int stipend;

int student\_course;

int marks[5];

std::string name;

// Конструктор по умолчанию

Student() : ID(0), group(0), stipend(0), student\_course(0), name("") {

for (int i = 0; i < 5; i++) {

marks[i] = 0;

}

}

// Конструктор с параметрами

Student(int ID, int group, int stipend, int student\_course, int marksArray[5], std::string name)

: ID(ID), group(group), stipend(stipend), student\_course(student\_course), name(name.substr(0, 50)) {

for (int i = 0; i < 5; i++) {

this->marks[i] = marksArray[i];

}

}

};

template <typename T>

struct Node {

T data;

Node\* next = nullptr;

Node\* prev = nullptr;

};

template <typename T>

struct List {

int size = 0;

int currentIndex = 0;

Node<T>\* current = nullptr;

Node<T>\* head = nullptr;

Node<T>\* tail = nullptr;

void toFirst() {

if (!size) return;

current = head;

currentIndex = 0;

}

void toLast() {

if (!size) return;

current = tail;

currentIndex = count() - 1;

}

void toNext() {

if (!size) return;

current = current->next;

currentIndex++;

}

void toPrev() {

if (!size) return;

current = current->prev;

currentIndex--;

}

void currentTo(int index) {

if (!size) return;

if ((currentIndex - index) > index) {

toFirst();

}

if ((count() - 1 - index) < (index - currentIndex)) {

toLast();

}

while (index > currentIndex) {

toNext();

}

while (index < currentIndex) {

toPrev();

}

}

void removeCurrent() {

if (!size) return;

Node<T>\* delNode = current;

if ((delNode != head) and (delNode != tail)) {

delNode->prev->next = delNode->next;

delNode->next->prev = delNode->prev;

current = current->next;

}

if (delNode == head) {

head = delNode->next;

if (delNode->next != nullptr) {

delNode->next->prev = nullptr;

current = current->next;

}

}

if (delNode == tail) {

tail = delNode->prev;

if (delNode->prev != nullptr) {

delNode->prev->next = nullptr;

toPrev();

}

}

delete delNode;

size--;

if (!size) current = nullptr;

}

void insertBeforeCurrent(T data) {

if (!size) return;

Node<T>\* newNode = new Node<T>;

size++;

currentIndex++;

newNode->data = data;

newNode->next = current;

newNode->prev = current->prev;

if (current->prev != nullptr) current->prev->next = newNode;

current->prev = newNode;

if (current == head) { head = newNode; }

}

Node<T>\* getCurrent(int index) {

currentTo(index);

return current;

}

void add(T data) {

Node<T>\* newNode = new Node<T>;

newNode->data = data;

if (!size) {

currentIndex = 0;

current = newNode;

head = newNode;

tail = newNode;

}

else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

size++;

}

void insert(int index, T data) {

if ((!size) or (index == size)) {

add(data);

}

else {

currentTo(index);

insertBeforeCurrent(data);

}

}

void removeAt(int index) {

currentTo(index);

removeCurrent();

}

T elementAt(int index) {

currentTo(index);

return current->data;

}

int count() {

return size;

}

void clear() {

for (int i = size; i > 0; i--) {

removeCurrent();

}

}

};

Student addStudent(int& ID) {

Student student;

student.ID = ID++;

cout << endl;

cout << "Student name: ";

cin.ignore();

getline(cin, student.name);

cout << "Student group: ";

cin >> student.group;

cout << "Student marks: ";

for (int i = 0; i < 5; i++) {

cin >> student.marks[i];

}

cout << "Student stipend: ";

cin >> student.stipend;

cout << "Student course: ";

cin >> student.student\_course;

return student;

}

void coutStudent(Student student) {

cout << "Student ID: " << student.ID << endl;

cout << "Student name: " << student.name << endl;

cout << "Student group: " << student.group << endl;

cout << "Student marks: ";

for (int i = 0; i < 5; i++) {

cout << student.marks[i] << " ";

}

cout << endl;

cout << "Student stipend: " << student.stipend << endl;

cout << "Student course: " << student.student\_course << endl;

}

void removeGroup(List<Student>& studentList, int group) {

for (int i = 0; i < studentList.count(); i++) {

if (studentList.elementAt(i).group == group) {

studentList.removeCurrent();

i--;

}

}

}

void removeStudent(List<Student>& studentList, int ID) {

for (int i = 0; i < studentList.count(); i++) {

if (studentList.elementAt(i).ID == ID) {

studentList.removeCurrent();

return;

}

}

}

void studentsSortedByAverage(List<Student> list) {

if (list.count() == 0) return;

List<Student> outputList;

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

outputList.add(list.elementAt(i));

}

for (int i = 0; i < outputList.count() - 1; i++) {

for (int j = i + 1; j < outputList.count(); j++) {

double avg\_i = 0, avg\_j = 0;

for (int k = 0; k < 5; k++) {

avg\_i += outputList.elementAt(i).marks[k];

avg\_j += outputList.elementAt(j).marks[k];

}

avg\_i /= 5;

avg\_j /= 5;

if (avg\_i > avg\_j) {

Student temp = outputList.elementAt(i);

outputList.removeAt(i);

outputList.insert(i, outputList.elementAt(j - 1));

outputList.removeAt(j);

outputList.insert(j, temp);

}

else if (avg\_i == avg\_j && outputList.elementAt(i).name > outputList.elementAt(j).name) {

Student temp = outputList.elementAt(i);

outputList.removeAt(i);

outputList.insert(i, outputList.elementAt(j - 1));

outputList.removeAt(j);

outputList.insert(j, temp);

}

}

}

for (int i = 0; i < outputList.count(); i++) {

cout << outputList.elementAt(i).name << "\t" << outputList.elementAt(i).group << "\t"

<< "Average grade: " << (double)(outputList.elementAt(i).marks[0] + outputList.elementAt(i).marks[1]

+ outputList.elementAt(i).marks[2] + outputList.elementAt(i).marks[3] + outputList.elementAt(i).marks[4]) / 5 << endl;

}

}

Student getStudentByID(List<Student> studentList, int ID) {

for (int i = 0; i < studentList.count(); i++) {

if (studentList.elementAt(i).ID == ID) {

return studentList.elementAt(i);

}

}

Student student;

return student;

}

int toint(string str) {

int num = 0, k = 0;

for (int i = str.size() - 1; i >= 0; i--, k++) {

num += (str[i] - 48) \* pow(10, k);

}

return num;

}

int\* tointarr(string str) {

int arr[5] = {}, k = 0;

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

if (str[i] != ' ') {

arr[k] = str[i] - 48;

k++;

}

}

return arr;

}

void fileInput(List<Student>& list, int& ID) {

Student student;

ifstream input;

input.open("input.txt");

string str;

int number = 0;

int\* arr;

while (getline(input, str)) {

number++;

switch (number) {

case 1:

student.name = str;

break;

case 2:

student.group = toint(str);

break;

case 3:

arr = tointarr(str);

for (int i = 0; i < 5; i++) {

student.marks[i] = arr[i];

}

break;

case 4:

student.stipend = toint(str);

break;

case 5:

student.student\_course = toint(str);

student.ID = ID++;

list.add(student);

number = 0;

break;

}

}

input.close();

}

void fileOutput(List<Student> list) {

ofstream output;

output.open("output.txt");

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

Student student = list.elementAt(i);

output << student.name << endl;

output << student.group << endl;

for (int j = 0; j < 5; j++) { output << student.marks[j] << " "; }

output << endl << student.stipend << endl;

output << student.student\_course << endl;

output << student.ID << endl;

}

output.close();

}

void menu() {

cout << " 0 Terminate program" << endl;

cout << " 1 Add student" << endl;

cout << " 2 Remove student by ID" << endl;

cout << " 3 Edit student by ID" << endl;

cout << " 4 Show student by ID" << endl;

cout << " 5 Show students by parameter" << endl;

cout << " 6 Show all students" << endl;

cout << endl;

cout << " 7 Add group of students" << endl;

cout << " 8 Remove group of students" << endl;

cout << " 9 Change group number" << endl;

cout << " 10 Show group of students" << endl;

cout << " 11 Show students average marks" << endl;

cout << endl;

cout << " 12 File input" << endl;

cout << " 13 File output" << endl;

}

void submenu() {

cout << " 0 Cancel" << endl;

cout << " 1 Name" << endl;

cout << " 2 Student course" << endl;

cout << " 3 Group" << endl;

cout << " 4 Marks" << endl;

cout << " 5 Stipend" << endl;

cout << endl;

}

void submenu2() {

cout << " 0 Cancel" << endl;

cout << " 1 Name" << endl;

cout << " 2 Student course" << endl;

cout << " 3 Stipend" << endl;

cout << endl;

}

void main() {

List<Student> studentList;

int ID = 1;

int choice = 0;

int choice2 = 0;

int input = 0;

while (true) {

system("cls");

menu();

cout << "Choice: ";

cin >> choice;

switch (choice)

{

case (0): {

return;

}

case (1): {

cout << "Enter student data";

studentList.add(addStudent(ID));

cout << "Student is added" << endl;

break;

}

case (2): {

if (studentList.count() == 0) break;

cout << "Enter student ID: ";

cin >> input;

removeStudent(studentList, input);

cout << "Student is removed" << endl;

break;

}

case (3): {

if (studentList.count() == 0) break;

cout << "ID of the student you want to edit: ";

cin >> input;

cout << "What do you want to edit?" << endl;

submenu();

cout << "Choice: ";

cin >> choice2;

switch (choice2)

{

case (0): {

break;

}

case (1): {

Student student = getStudentByID(studentList, input);

if (student.ID == 0) break;

removeStudent(studentList, input);

string input2 = "";

cout << "Enter new data: ";

cin >> input2;

student.name = input2;

studentList.add(student);

break;

}

case (2): {

Student student = getStudentByID(studentList, input);

if (student.ID == 0) break;

removeStudent(studentList, input);

int input2 = 0;

cout << "Enter new data: ";

cin >> input2;

student.student\_course = input2;

studentList.add(student);

break;

}

case (3): {

Student student = getStudentByID(studentList, input);

if (student.ID == 0) break;

removeStudent(studentList, input);

int input2 = 0;

cout << "Enter new data: ";

cin >> input2;

student.group = input2;

studentList.add(student);

break;

}

case (4): {

Student student = getStudentByID(studentList, input);

if (student.ID == 0) break;

removeStudent(studentList, input);

int input2 = 0;

cout << "Enter new data: ";

for (int i = 0; i < 5; i++) {

cin >> input2;

student.marks[i] = input2;

}

studentList.add(student);

break;

}

case (5): {

Student student = getStudentByID(studentList, input);

if (student.ID == 0) break;

removeStudent(studentList, input);

int input2 = 0;

cout << "Enter new data: ";

cin >> input2;

student.stipend = input2;

studentList.add(student);

break;

}

default: {

cout << "Unknown command, try again" << endl;

break;

}

}

break;

}

case (4): {

if (studentList.count() == 0) break;

cout << "Enter student ID: ";

cin >> input;

coutStudent(getStudentByID(studentList, input));

break;

}

case (5): {

if (studentList.count() == 0) break;

cout << "What parametr?" << endl;

submenu2();

cout << "Choice: ";

cin >> choice2;

switch (choice2)

{

case (0): {

break;

}

case (1): {

cout << "Name: ";

string input2 = "";

cin >> input2;

for (int i = 0; i < studentList.count(); i++) {

if (studentList.elementAt(i).name == input2) {

coutStudent(studentList.elementAt(i));

cout << endl;

}

}

break;

}

case (2): {

cout << "Student course: ";

cin >> input;

for (int i = 0; i < studentList.count(); i++) {

if (studentList.elementAt(i).student\_course == input) {

coutStudent(studentList.elementAt(i));

cout << endl;

}

}

break;

}

case (3): {

cout << "Stipent: ";

cin >> input;

for (int i = 0; i < studentList.count(); i++) {

if (studentList.elementAt(i).stipend == input) {

coutStudent(studentList.elementAt(i));

cout << endl;

}

}

break;

}

default: {

cout << "Unknown command, try again" << endl;

break;

}

}

break;

}

case (6): {

cout << "All students:" << endl;

for (int i = 0; i < studentList.count(); i++) {

coutStudent(studentList.elementAt(i));

}

break;

}

case (7): {

cout << "Amount of students: ";

cin >> input;

cout << "Enter students data";

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

studentList.add(addStudent(ID));

}

cout << "Students are added" << endl;

break;

}

case (8): {

if (studentList.count() == 0) break;

cout << "Group to remove: ";

cin >> input;

removeGroup(studentList, input);

cout << "Group is removed: " << endl;

break;

}

case (9): {

if (studentList.count() == 0) break;

int input2;

cout << "Group number to change: ";

cin >> input;

cout << "New group number: ";

cin >> input2;

for (int i = 0; i < studentList.count(); i++) {

if (studentList.elementAt(i).group == input) {

Student student = studentList.elementAt(i);

removeStudent(studentList, student.ID);

i--;

student.group = input2;

studentList.add(student);

}

}

cout << "Group number changed" << endl;

break;

}

case (10): {

if (studentList.count() == 0) break;

cout << "Group number to show: ";

cin >> input;

for (int i = 0; i < studentList.count(); i++) {

if (studentList.elementAt(i).group == input) {

coutStudent(studentList.elementAt(i));

}

}

break;

}

case (11): {

cout << "students sorted by Average:" << endl;

studentsSortedByAverage(studentList);

break;

}

case (12): {

fileInput(studentList, ID);

break;

}

case (13): {

fileOutput(studentList);

break;

}

default: {

cout << "Unknown command, try again" << endl;

break;

}

}

system("pause");

}

}

# Приложение B

Файл: input.txt

Andrew

3371

4 5 5 5 3

2250

3

Maksim

3372

3 3 3 3 3

2550

3

Marat

3372

4 3 3 2 3

10000

3

Igor

3372

5 5 5 5 5

10000

3

12 File input

9 Change group number

Group number to change: 3372

New group number: 3312

Group number changed

2 Remove student by ID

Enter student ID: 2

Student is removed

13 File output

Файл output.txt

Andrew

3371

4 5 5 5 3

2250

3

1

Marat

3312

4 3 3 2 3

10000

3

3

Igor

3312

5 5 5 5 5

10000

3

4