Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Конструирование программ и языки программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

РАСЧЁТ И НАЧИСЛЕНИЕ СТИПЕНДИИ

БГУИР КП 1–400201.304 ПЗ

Студент: гр. 950503 Гуринович А. В.

Руководитель: старший преподаватель

Ковальчук А. М.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_\_Б. В. Никульшин

(подпись)

––"–––––––––––––––––2020г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Гуриновичу Андрею Викторовичу*

1. Тема проекта *Расчёт и начисление стипендии*

2. Срок сдачи студентом законченного проекта *15 декабря 2020 г.*

3. Исходные данные к проекту *Файл: Bachelors – содержит информацию о бакалаврах, Masters – содержит информацию о магистрантах, Doctor – содержит информацию докторантах, Settings – содержит информацию о используемых в вычислениях внутри программы константах. Контейнер: свой – Binary Tree и STL – vector.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение. Содержание. 1. Постановка задачи 2. Структура входных и выходных данных. 3. Диаграмма классов. 4. Описание классов. 5. Разработка алгоритмов. 5.1 Разработка алгоритмов по шагам. 5.1 Разработка схем алгоритмов. 6. Тестирование. Заключение. Список использованной литературы.*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. Диаграмма классов*

*2. Схема алгоритма метода* *static float calclulateBachelorScholarship(Bachelor&, float)*

*3. Схема алгоритма метода**template <typename studentType>*

*static void executeUndo(Undo<studentType>&, BinaryTree<studentType, int>&)*

6. Консультант по проекту *Ковальчук А. М*.

7. Дата выдачи задания *12 сентября 2020 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*разделы 1, 2 к 1 октября 2020 г. – 20 %;*

*разделы 3, 4 к 1 ноября 2020 г. – 30 %;*

*разделы 5, 6 ,7 к 1 декабря 2020 г. – 30 %;*

*оформление пояснительной записки и графического материала к 14 декабря 2020 г. – 20 %*

*Защита курсового проекта с 21 декабря 2020 г. по 28 декабря 2020 г.*

РУКОВОДИТЕЛЬ– –––––––––*Ковальчук А. М.*

Задание принял к исполнению –Х *Гуринович А. В.*

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ 5](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303619)

[ВВЕДЕНИЕ 6](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303620)

[1 ОБЗОР ИСТОЧНИКОВ](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303621) 8

1.1 [Постановка задачи 8](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303622)

1.2 [Анализ аналогов программного средства](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303624) 8

2 [СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303625) 9

[2.1 Структура программы](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303626) 9

[2.2 Структура входных данных](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303627) 10

2.3 [Структура промежуточных данных](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303628) 12

2.4 [Структура выходных данных](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303628) 14

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303628) 16

3.1 [Диаграмма классов](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303628) 16

3.2 [Описание классов](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303628) 16

4 АЛГОРИТМЫ25

4.1 [Алгоритмы по шагам](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303628) 25

4.2 Схемы алгоритмов 27

5 [ТЕСТИРОВАНИЕ](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303628) 28

[ЗАКЛЮЧЕНИЕ](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303628) 36

[Список использованной литературы](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303629) 37

[Приложение А Диаграмма классов](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303630) 38

[Приложение Б Схема алгоритма метода static float calclulateBachelorScholarship(Bachelor&, float)](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303631) 39

[Приложение В Схема алгоритма метода template <typename studentType> static void executeUndo(Undo<studentType>&, BinaryTree<studentType, int>&)](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303632) 40

[Приложение Г Код программы](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303633) 41

[Приложение Д Ведомость документов](file:///D:\1ФЛЕШКА\Курсовая\Варианты%20курсачей\Сахарук\Мое(О%20Господи(Чё))детище(да)(Содержание).docx#_Toc450303634) 42

**ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ**

ОС – операционная система.

ООП – Объектно-ориентированное программирование.

STL – (*Standard Template Library*) –Библиотека стандартных шаблонов.

ЭВМ – Электронно-вычислительная машина.

# **ВВЕДЕНИЕ**

Тема «Расчёт и начисления стипендии» была выбрана так как меня интересует структура университета, его система и цифровое представление. Изначально планировалось взять тему «Расписание БГУИР» и реализовать его для ОС «iOS», однако, из-за специфики разработки под данную ОС с использованием языка C++, было принято решения отказаться от этой идеи и реализовать программу со схожей темой на ОС «MacOS».

Объектно-ориентированное программирование представляет собой технологию программирования, которая базируется на классификации и абстракции объектов. Одним из наиболее популярных средств объектно-ориентированного программирования, позволяющим разрабатывать программы, эффективные по объёму кода и скорости выполнения является С++.

C++ — мощный язык, содержащий средства создания эффективных программ практически любого назначения, от низкоуровневых утилит и драйверов до сложных программных комплексов самого различного назначения. В частности:

- предсказуемое выполнение программ является важным достоинством для построения систем реального времени. Весь код, неявно генерируемый компилятором для реализации языковых возможностей (например, при преобразовании переменной к другому типу), определён в стандарте. Строго определены места программы, в которых он выполняется. Это даёт возможность замерять или рассчитывать время реакции программы на внешнее событие.

- автоматический вызов деструкторов объектов при их уничтожении, в порядке, обратном вызову конструкторов, что позволяет избежать утечек памяти.

- язык поддерживает понятие (const) константности, что позволяет компилятору, например, диагностировать ошибочные попытки изменения значения константной переменной. Объявление константности даёт программисту, читающему текст программы дополнительное представление о правильном использовании классов и функций, а также может являться подсказкой для оптимизации.

- язык имеет крупную стандартную библиотеку STL, которая включает в себя распространённые контейнеры и алгоритмы.

- С++ сочетает в себе как возможности низкоуровневых языков программирования, так и возможности высокоуровневых. Имеется возможность работы на низком уровне с памятью, адресами.

- кроссплатформенность: стандарт языка накладывает минимальные требования на ЭВМ для запуска скомпилированных программ. Доступны компиляторы для большого количества платформ, на языке C++ разрабатывают программы для самых различных платформ и систем.

- эффективность. Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы. Ни одна из языковых возможностей, приводящая к дополнительным накладным расходам, не является обязательной для использования — при необходимости язык позволяет обеспечить максимальную эффективность программы.

**1 ОБЗОР ИСТОЧНИКОВ**

* 1. **Постановка задачи**

*Программа должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню. Информация должна храниться в нескольких файлах, связанных определенным образом. Например, информация, содержащаяся в файлах: специальность, год обучения, оценки, полученные во время сессии, задолженности. Осуществлять поиск по группе, фамилии, размеру стипендии, льготах. Выдаваемая информация должна быть отсортирована по разным признакам. Другие методы зависимости от специфики задачи.*

*При реализации операции редактирования, добавления, удаления информации необходимо предусмотреть операцию отмены последних. действий. Разработать иерархию классов с использованием наследования (не меньше 3-х уровней наследования). Разработать и использовать в программе классы контейнеров и алгоритмов (свои и STL.). Производить обработку исключительных ситуаций.*

**2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ**

**2.1 Структура программы**

В программе можно выделить несколько основных элементов: блок взаимодействия пользователя с программой, блок ввода, вывода, чтения и записи информации, блок исключений, блок настроек и констант.

Блок взаимодействия пользователя с программой отвечает за выбор пользователем определённых действий, например вывод студентов с сортировкой по получаемой стипендии или вызов ввода студента.

Информация, полученная в блоке взаимодействия пользователя с программой, записывает и обрабатывается в блоке ввода, вывода, чтения и записи информации. Например, после вызова ввода студента из первого блока, происходит создание объекта класса, относящегося к блоку ввода, вывода, чтения и записи информации, он считывает информацию с клавиатуры и обрабатывает информацию.

В блоке исключений происходит обработка исключительных ситуаций, например проверка длины номера группы для ввода студента или проверка успешного открытия файла для записи или чтения.

В блоке настроек и констант содержатся static методы, которые доступны во всех точках программы, переменные. Блок используется для хранения всех констант, которые используются в программе более одного раза, например максимальную длину идентификационного номера человека, коэффициенты прибавки к стипендии для имеющих льготы учащихся.

Структурная схема представлена на рисунке 2.1.1.

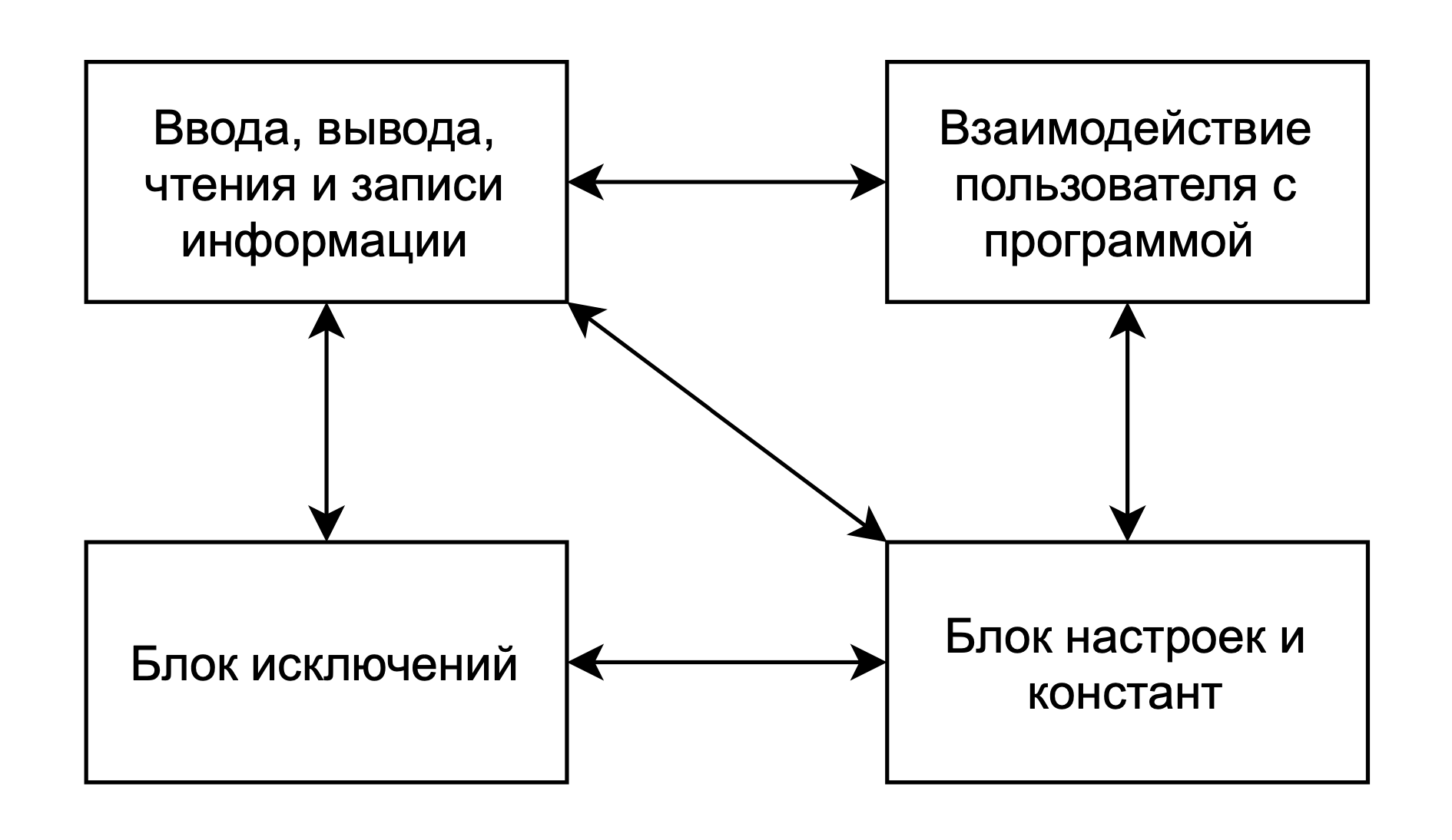


Рисунок 2.1.1 - Структурная схема

* 1. **Структура входных данных**

Информация в программе будет храниться в бинарных файлах.

Структура данных, считываемых из файла, хранящего информацию о бакалавре, представлена в таблице 2.2.1

Таблица 2.2.1 – Структура данных файла «Bachelors»

|  |  |
| --- | --- |
| идентификационный номер | 95050035 |
| имя | Andrey |
| дата рождения | 14.02.2002 |
| номер группы | 950503 |
| численное значение стипендии | 0 |
| количество оценок | 1 |
| оценка (класс Mark) | 10, false, PDaPL |
| количество льгот | 1 |
| льгота (класс Benefit) | personal |

Структура данных, считываемых из файла, хранящего информацию о константах и настройках, представлена в таблице 2.2.2

Таблица 2.2.2 – Структура данных файла «Settings»

|  |  |
| --- | --- |
| базовая стипендия бакалавра | 100 |
| базовая стипендия магистранта | 200 |
| базовая стипендия докторанта | 400 |
| влияют ли льготы на стипендию | true |
| модификатор льготы для инвалидов | 1,5 |
| модификатор социальной льготы | 2 |
| модификатор стипендии совета университета | 1,6 |
| влияют ли оценки на стипендию | true |
| модификатор оценок первого уровня | 1,1 |
| модификатор оценок второго уровня | 1,2 |
| модификатор оценок третьего уровня | 1,3 |
| влияют ли пересдачи на стипендию | true |
| влияет ли публикация на стипендию | true |
| влияет ли аккредитация издания на стипендию | true |
| влияет ли защита диссертации на стипендию | true |

Структура данных, считываемых из файла, хранящего информацию о магистрантах, представлена в таблице 2.2.3

Таблица 2.2.3 – Структура данных файла «Masters»

|  |  |  |
| --- | --- | --- |
| бакалавр (класс Bachelor) | имеется ли публикация | публикация (класс Publucation) |
| – | true | Name, Science, true |

Структура данных, считываемых из файла, хранящего информацию о докторантах, представлена в таблице 2.2.4

Таблица 2.2.4 – Структура данных файла «Masters»

|  |  |
| --- | --- |
| магистрант (класс Master) | защищена ли диссертация |
| – | true |

* 1. **Структура промежуточных данных**

В качестве промежуточных данных использовались шаблон контейнера

бинарного дерева (BinaryTree), итератор BinartyTreeIterator, контейнеры STL вектор (vector) и стек (stack).

//шаблонный класс Node

template <typename DataType, typename KeyDataType>

class Node {

public:

DataType data; //данные

KeyDataType key; //ключ, уникальный для каждого узла

Node\* left; //указатель на левую ветвь этого узла

Node\* right; //указатель на правую ветвь этого узла

Node\* parent; //указатель на родительский узел этого узла

public:

Node(DataType data, KeyDataType key); //конструктор с параметрами по умолчанию

void add(Node\* node); //добавление нового узла

Node\* find(int key); //поиск узла по ключу

Node\* getNext(KeyDataType key); //поиск узла со следующим ключом

Node\* supportDeleteFunction(Node\* root); //вспомогательный метод

void deleteKey(Node\*\* root, KeyDataType key); //удаление по ключу

void deleteAll(); //удаление всего

DataType& getData(); //геттер данных

KeyDataType getKey(); //геттер ключа

void setData(DataType data); //сеттер данных

void setKey(KeyDataType key); //сеттер ключа

//класс BinaryTree

template <typename DataType, typename KeyDataType>

class BinaryTree {

protected:

Node<DataType, KeyDataType>\* root; //корень

Iterator beginConst //иеератор указывающий на минимальный ключ

Iterator endConst //иеератор указывающий на максимальный ключ

public:

BinaryTree(Node<DataType, KeyDataType>\* root = NULL) //конструктор с параметрами по умолчанию

BinaryTree(DataType data, KeyDataType key) //конструктор с параметрами

bool empty() //пусто ли дерево

bool checkKey(KeyDataType key) //проверка существования ключа

Node<DataType, KeyDataType>\* findNodeKey(int key) //поиск узла по ключу

DataType& findKey(int); //поиск значения узла по ключу

int size() // размер

void print() //вывод информации

void add(DataType data, KeyDataType key) //добавление нового узла

void remove(KeyDataType key) //удаление узла по ключу

void clear() //удаление всех узлов

Node<DataType, KeyDataType>\* operator [] (int) //перегрузка оператора []

Iterator begin() //получение beginConst

Iterator end() //получение endConst

//класс BinaryTreeIterator

template <typename DataType, typename KeyDataType>

class BinaryTreeIterator{

private:

Node<DataType, KeyDataType>\* current;

public:

BinaryTreeIterator(Node<DataType, KeyDataType>\* current = NULL);//конструктор

DataType& operator \* (); //перегрузка оператора \*

Node<DataType, KeyDataType>\* operator & (); //перегрузка оператора &

void operator = (Node<DataType, KeyDataType>\* current); //перегрузка оператора =

void operator ++ (int); //перегрузка оператора ++

BinaryTreeIterator& operator + (int temp); //перегрузка оператора +

bool operator > (KeyDataType key); //перегрузка оператора >

bool operator >= (KeyDataType key); //перегрузка оператора >=

bool operator < (KeyDataType key); //перегрузка оператора <

bool operator <= (KeyDataType key); //перегрузка оператора <=

bool operator == (const BinaryTreeIterator& iterator); //перегрузка оператора ==

bool operator != (const BinaryTreeIterator& iterator); //перегрузка оператора !=

**2.4 Структура выходных данных**

Структура данных, записываемых в файл, хранящий информацию о бакалавре, представлена в таблице 2.4.1

Таблица 2.4.1 – Структура данных файла «Bachelors»

|  |  |
| --- | --- |
| идентификационный номер | 95050035 |
| имя | Andrey |
| дата рождения | 14.02.2002 |
| номер группы | 950503 |
| численное значение стипендии | 0 |
| количество оценок | 1 |
| оценка (класс Mark) | 10, false, PDaPL |
| количество льгот | 1 |
| льгота (класс Benefit) | personal |

Структура данных, записываемых в файл, хранящий информацию о магистрантах, представлена в таблице 2.4.2

Таблица 2.4.2 – Структура данных файла «Masters»

|  |  |  |
| --- | --- | --- |
| бакалавр (класс Bachelor) | имеется ли публикация | публикация (класс Publucation) |
| – | true | Name, Science, true |

Структура данных, записываемых в файл, хранящий информацию о докторантах, представлена в таблице 2.4.3

Таблица 2.4.3 – Структура данных файла «Masters»

|  |  |
| --- | --- |
| магистрант (класс Master) | защищена ли диссертация |
| – | true |

Структура данных, записываемых в файл, хранящий информацию о константах и настройках, представлена в таблице 2.4.4

Таблица 2.4.4 – Структура данных файла «Settings»

|  |  |
| --- | --- |
| базовая стипендия бакалавра | 100 |
| базовая стипендия магистранта | 200 |
| базовая стипендия докторанта | 400 |
| влияют ли льготы на стипендию | true |
| модификатор льготы для инвалидов | 1,5 |
| модификатор социальной льготы | 2 |
| модификатор стипендии совета университета | 1,6 |
| влияют ли оценки на стипендию | true |
| модификатор оценок первого уровня | 1,1 |
| модификатор оценок второго уровня | 1,2 |
| модификатор оценок третьего уровня | 1,3 |
| влияют ли пересдачи на стипендию | true |
| влияет ли публикация на стипендию | true |
| влияет ли аккредитация издания на стипендию | true |
| влияет ли защита диссертации на стипендию | true |

**3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

**3.1 Диаграмма классов**

Одним из важнейших пунктов в ООП является разработка логической модели системы в виде диаграммы классов. Диаграмма классов для представления статической структуры модели в терминологии классов ООП. Она отражает связи между классами и структурами.

Диаграмма классов программы представлена в приложении А.

* 1. **Описание классов**

//класс Human

protected:

string privateID; //ID человека

string name; //имя

Date birthday; //дата рождения

public:

Human(string privateID, string name, Date birthday); //конструктор с параметрами по умолчанию

void writeToFile(File&); //запись в файл

void readFromFile(File&); //чтение из файла

string getPrivateID(); //геттер ID

string getName(); //геттер имени

Date getBirthday(); //геттер даты рождения

void setPrivateID(string); //сеттер ID

void setName(string); //сеттер имени

void setBirthday(Date); //сеттер даты рождения

void editMenu(bool); //меню редактирования

void edit(int); //редактирование

static void printSpreadsheet(); //вывод заголовка таблицы

friend ostream& operator << (ostream &stream, Human human); //перегрузка <<

friend istream& operator >> (istream &stream, Human &human); //перегрузка >>

~Human(){}; //деструктор

Класс Bachelor

protected:

string group; //номер группы

float scholarship; //размер стипендии

bool showScholarship; //выводить ли стипендию при печати

vector<Mark> marks; //оценки

vector<Benefit> benefits; //льготы

public:

Bachelor(string group, float scholarship, bool showScholarship); //конструктор с параметрами по умолчанию

void writeToFile(File&); //запись в файл

void readFromFile(File&) //чтение из файла

void calculateScholarship(float base = Settings::getBachelorBase()); //расчёт стипендии

string getGroup(); //геттер

float getScholarship(); //геттер

bool getShowScholarship(); //геттер

Mark\* getMarkForID(int); //геттер

unsigned long getMarksSize(); //геттер

Benefit\* getBenefitForID(int); //геттер

unsigned long getBenefitsSize(); //геттер

void setGroup(string); //сеттер

void setScholarship(float); //сеттер

void setShowScholarship(bool); //сеттер

void addMark(Mark); //добавить оценку

void addBenefit(Benefit); //добавить льготу

void removeMarkForID(int); //удалить оценку по ID

void removeBenefitForID(int); //удалить льготу по ID

void editMenu(bool); //меню редактирования

void edit(int); //редактирование

static void printSpreadsheet(bool); //вывод заголовка таблицы

void printMarksAndBenefits(); //вывести оценки и льготы

friend ostream& operator << (ostream &stream, Bachelor bachelor);; //перегрузка <<

friend istream& operator >> (istream &stream, Bachelor &bachelor);); //перегрузка >>

~Bachelor() // деструктор

//класс Master

protected:

Publication\* publication; //указатель на публикацию

public:

Master(Publication\* publication = NULL); //конструктор с параметрами по умолчанию

Master(const Master& master); //конструктор копирования

void writeToFile(File&); //запись в файл

void readFromFile(File&); //чтение из файла

void calculateScholarship(float base = Settings::getMasterBase()); //расчёт стипендии

Publication\* getPublication(); //геттер

void setPublication(Publication\*); //сеттер

void editMenu(bool); //меню редактирования

void edit(int key); //редактирование

static void printSpreadsheet(bool); //вывод заголовка таблицы

friend ostream& operator << (ostream &stream, Master master); //перегрузка <<

friend istream& operator >> (istream &stream, Master &master); //перегрузка >>

~Master(); //дестркутор

//класс Doctor

protected:

bool dissertationComplited; //завершена ли диссертация

public:

Doctor(); //конструктор по умолчанию

void writeToFile(File&); //запись в файл

void readFromFile(File&); //чтение из файла

void calculateScholarship(float base = Settings::getDoctorBase()); // расчёт стипендии

bool getDissertationComplited(); //геттер

void setDissertationComplited(bool); //сеттер

void editMenu(bool) //меню редактирования

void edit(int key); //редактирование

static void printSpreadsheet(bool); // вывод заголовка таблицы

friend ostream& operator << (ostream &stream, Doctor doctor); //перегрузка <<

friend istream& operator >> (istream &stream, Doctor &doctor); //перегрузка >>

~Doctor(){}; // дестркутор

//класс ExceptionTypes

protected:

enum DataType type; //тип исключения

string errorMessage; //сообщение об ошибке

string userMessage; //сообщение для пользователя

public:

ExceptionTypes(enum DataType type, string string); //конструктор с параметрами по умолчанию

string getErrorTypeString(); //геттер

string getUserMessage(); //геттер

static void Terminate(); //terminate функция

~ExceptionTypes(); //деструктор

//класс ExceptionEnter

public:

template <typename numberType>

static numberType NumberData(enum DataType type); //

template <typename studentType>

static bool checkThisID(string ID, BinaryTree<studentType, int> &tree); //

static string stringData(enum DataType); //

static bool onlyLetters(string); // имеются ли в строке символы кроме букв

static bool onlyDigiths(string); //имеются ли в строке символы кроме цифр

//класс ExceptionFile

public:

static fstream openFile(enum FileType, bool); //открытие файла

//класс Date

private:

int day; //день

int month; //месяц

int year; //год

public:

Date(int day, int month, int year); //конструктор с параметрами по умолчанию

Date operator = (const Date&); //перегрузка оператора присваивания

friend ostream& operator << (ostream &outputStream, const Date date);//перегрузка <<

friend istream& operator >> (istream &inputStream, Date &Date); //перегрузка >>

~Date(); //деструктор

//класс Mark

protected:

int mark; //оценка

bool resit; //была ли пересдача

string subject;

public:

Mark(int mark, bool resit, string subject); //конструктор с параметрами по умолчанию

int getMark(); //геттер

bool getResit(); //геттер

string getSubject(); //геттер

void setMark(int); //сеттер

void setResit(bool); //сеттер

void setSubject(string); //сеттер

static void printSpreadsheet(); //вывод заголовка таблицы

friend ostream& operator << (ostream &stream, Mark mark); //перегрузка <<

friend istream& operator >> (istream &stream, Mark &mark); //перегрузка >>

~Mark() //дестркутор

//класс Benefit

protected:

enum BenefitType type; //тип льготы

public:

Benefit(BenefitType type) //конструктор с параметрами по умолчанию

enum BenefitType getType(); //геттер

float getModifier(); //вычисление модификатора

void setType(enum BenefitType);//сеттер

bool operator == (const Benefit &benefit); //перегрузка оператора присваивания

friend ostream& operator << (ostream &stream, Benefit benefit); //перегрузка <<

friend istream& operator >> (istream &stream, Benefit &benefit); //перегрузка >>

~Benefit(); //деструктор

//класс Publication

protected:

string name; //название публикации

string journalName; //название журнала

bool accreditation; //имеет ли он аккредитацию

public:

Publication(string name, string journalName, bool accreditation); //конструктор с параметрами по умолчанию

string getPublicationName(); //геттер

string getJournalName(); //геттер

bool getAccreditation(); //геттер

void setPublicationName(string); //сеттер

void setJournalName(string); //сеттер

void setAccreditation(bool); //сеттер

friend ostream& operator << (ostream &stream, Publication publication); //перегрузка <<

friend istream& operator >> (istream &stream, Publication &publication); //перегрузка >>

~Publication(); //деструктор

//класс File

private:

fstream file; //файловый поток

FileType type; //тип файла

public:

File(FileType type, bool write); //констркутор

template <typename dataType>

void read(dataType \*data);//чтение из файла

template <typename dataType>

void write(dataType data); //запись в файл

FileType getType(); //геттер

void setType(FileType); //сеттер

bool endOfFile(); //окончен ли файл

~File(); //деструктор

//класс Settings

protected:

static float bachelorBase; //аналогично файлу

static float masterBase;

static float doctorBase;

static bool benefits;

static float disabledBenefit;

static float socialBenefit;

static float councilBenefit;

static float personalBenefit;

static bool marks;

static float markFirst;

static float markSecond;

static float markThird;

static bool resit;

static bool publication;

static bool journalAccreditation;

static bool dissertationComplited;

public:

static void getSettingsFromFile(); //почитать настройки из файла

static void setSettingsToFile(); //записать настройки в файл

static void setTheDefaultSettings(); //установить настройки по умолчанию

static void menu(); //вызов меню

static void editSettings(); //вызов меню редактирования настроек

static void printSettings(); //вывод настроек

static int getLength(enum DataType); //получение максимальной длины строки для данного типа данных

static float getBachelorBase();

static float getMasterBase();

static float getDoctorBase();

static bool getBenefits();

static float getBenefitByType(enum BenefitType);

static bool getMarks();

static float getMarkFirst();

static float getMarkSecond();

static float getMarkThird();

static bool getResit();

static bool getPublication();

static bool getJournalAccreditation();

static bool getDissertationComplited();

//класс Menu

BinaryTree<Bachelor, int> bachelors; //бакалавры

BinaryTree<Master, int> masters; //магистранты

BinaryTree<Doctor, int> doctors; //докторанты

stack<Undo<Bachelor>> bachelorsUndoStack; //отмены действий бакалавров

stack<Undo<Master>> mastersUndoStack; //отмены действий магистрантов

stack<Undo<Doctor>> doctorsUndoStack; //отмены действий докторантов

int chooseStudentType(); //меню выбора типа студента

bool noStudents(); //пусты ли деревья

template <typename studentType>

bool noStudentsIn(BinaryTree<studentType, int>&); //пусты ли деревья

bool checkPrivateID(string privateID, bool printMessage = true); //проверка ID

template <typename studentType>

BinaryTree<studentType, int>sortBinaryTree(BinaryTree<studentType, int>); //сортировка

template <typename studentType>

studentType createStudent(); //создание студента

template <typename studentType>

void editStudent(BinaryTree<studentType, int>&); //редактирование студентов

template <typename studentType>

bool editStudentByID(BinaryTree<studentType, int>&, string); //редактирование студента по ID

template <typename studentType>

void removeStudent(BinaryTree<studentType, int>&); //удаление студентов

template <typename studentType>

bool removeStudentByID(BinaryTree<studentType, int>&, string); //удаление студента по ID

template <typename studentType>

void printStudent(BinaryTree<studentType, int>); //печать студентов

template <typename studentType>

void searchStudent(BinaryTree<studentType, int>&); //поиск студентов

template <typename studentType>

void undoStudent(stack<Undo<studentType>> &stack, BinaryTree<studentType, int>& tree); //выполнение отмены действия

template <typename studentType>

void addUndo(stack<Undo<studentType>>&, Undo<studentType>); //добавление отмены действия в стек

template <typename studentType>

void writeStudent(File&, BinaryTree<studentType, int>&); //запись в файл

void readStudents(); //чтение из файла

template <typename studentType>

void readStudent(File&, BinaryTree<studentType, int>&); //чтение из файла

public:

Menu() //конструктор

void call(); //вызов меню

void enter(); //ввод студентов

void edit(); //редактирование студентов

void remove(); //удаление студентов

void print(); //вывод студентов

void search(); //поиск студентов

void undo(); //отмена последних действий

void settings(); //вызов меню настроек

void write(); //запись студентов

~Menu(); //деструктор

//класс Undo

private:

enum UndoType type; //тип отмены действия

studentTypeUndo \*student; //указатель на студента

int privateID; //ID

public:

Undo(enum UndoType type, studentTypeUndo \*student, int privateID); //конструктор с параметрами по умолчанию

void execute(BinaryTree<studentTypeUndo, int>&); //выполнить отмену действия

enum UndoType getType(); //геттер

studentTypeUndo\* getStudent(); //геттер

int getPrivateID(); //геттер

void setType(enum UndoType); //сеттер

void setStudent(studentTypeUndo\*); //сеттер

void setPrivateID(int); //сеттер

//класс Algorithms

public:

static void writeBachelorToFile(Bachelor&, File&); //запись бакалавра в файл

static fstream openFile(enum FileType, bool); //безопасное открытие файла для записи или чтения

static float calclulateBachelorScholarship(Bachelor&, float); //расчёт стипендии для бакалавра

template <typename studentType>

static void executeUndo(Undo<studentType>&, BinaryTree<studentType, int>&); //исполнение отмены действия для всех трёх типов студентов

**4 АЛГОРИТМЫ**

**4.1 Алгоритмы по шагам**

Алгоритм по шагам методаstatic void writeBachelorToFile(Bachelor bachelor, File &file) – запись данных бакалавра в файл.

1. Начало.

2. Входные данные:

privateID – в идентификационный номер;

name – имя;

birthday – дата рождения;

group – номер группы;

scholarship ­– численное значение стипендии;

vector<Mark> marks – оценки;

vector< Benefit > benefits – льготы;

file – объект класса File, для данной функции всегда проинициализирован для файла «Bachelors».

3. Вызов метода класса Human writeToFile(File& file), который записывает в файл file, все данные из класса-родителя.

4. Запись group в файл file.

5. Запись scholarship в файл file.

6. Создаём переменную marksSize и инициализируем её методом marks size(), который возвращает количество элементов вектора.

7. Запись marksSize в файл file.

8. Запись всех элементов вектора marks в файл file.

2. Создаём переменную benefitsSize и инициализируем её методом benefits size(), который возвращает количество элементов вектора.

7. Запись benefitsSize в файл file.

8. Запись всех элементов вектора benefits в файл file.

12. Конец.

Алгоритм по шагам метода static fstream openFile(enum FileType, bool) – открытие файла для чтения или записи

1. Начало.

2.Входные данные:

type – тип фала;

write – открывать ли файл в режиме записи.

3. Создание переменной fileName для хранения имени будущего фала.

4. Оператор switch принимающий type. Так как type является enum (перечисляемым типом), у switch есть только четыре варианта исполнения (как и количество возможных значений enum).

5. Если значение type == settings, то filename = «Settings».

6. Если значение type == bachelors, то filename = «Bachelors».

7. Если значение type == masters, то filename = «Masters».

8. Если значение type == doctors, то filename = «Doctors».

9. Создание переменной file.

10. Если write == true, переходим к пункту 11, если иначе – к 12.

11. Вызываем метод file.open с параметрами «fileName, ios::trunc | ios::binary | ios::out». Переходим к пункту 13.

12. Вызываем метод file.open с параметрами «fileName, ios::binary | ios::in».

13. Если открыть файл не удалось (метод file. is\_open вернул false), создаём переменную tempFile, инициализируем её с параметрами «fileName, ios::binary», иначе переходим к пункту 18.

14. Закрываем файл, выполнив tempFile.close.

15. Если type == settings, переходим к пункту 16, иначе переходим к пункту 17.

16. Вызываем метод из пространства имён Settings setTheDefaultSettings, он сбрасывает настройки к настройкам по умолчанию и записывает их в файл.

17. Вызываем метод file.open с параметрами «fileName, ios::binary | ios::in».

18. Возвращаем file как результат выполнения функции.

19. Конец.

**4.2 Схемы алгоритмов**

Схема алгоритма метода static float calclulateBachelorScholarship(Bachelor&, float) приведена в приложении Б. – расчёт стипендии бакалавра.

Схема алгоритма метода template template <typename studentType> static void executeUndo(Undo<studentType>&, BinaryTree<studentType, int>&) приведена в приложении В. – нахождение следующего по идентификационному номеру узла бинарного дерева.

**5 ТЕСТИРОВАНИЕ**

На этапе тестирование проверяется правильность выполнения основных действий, совершаемых в ходе программы.

Выбираем создание нового бакалавра (рис. 5.1).

****

Рисунок 5.1

Создаём трёх бакалавров (рис. 5.2, 5.3).

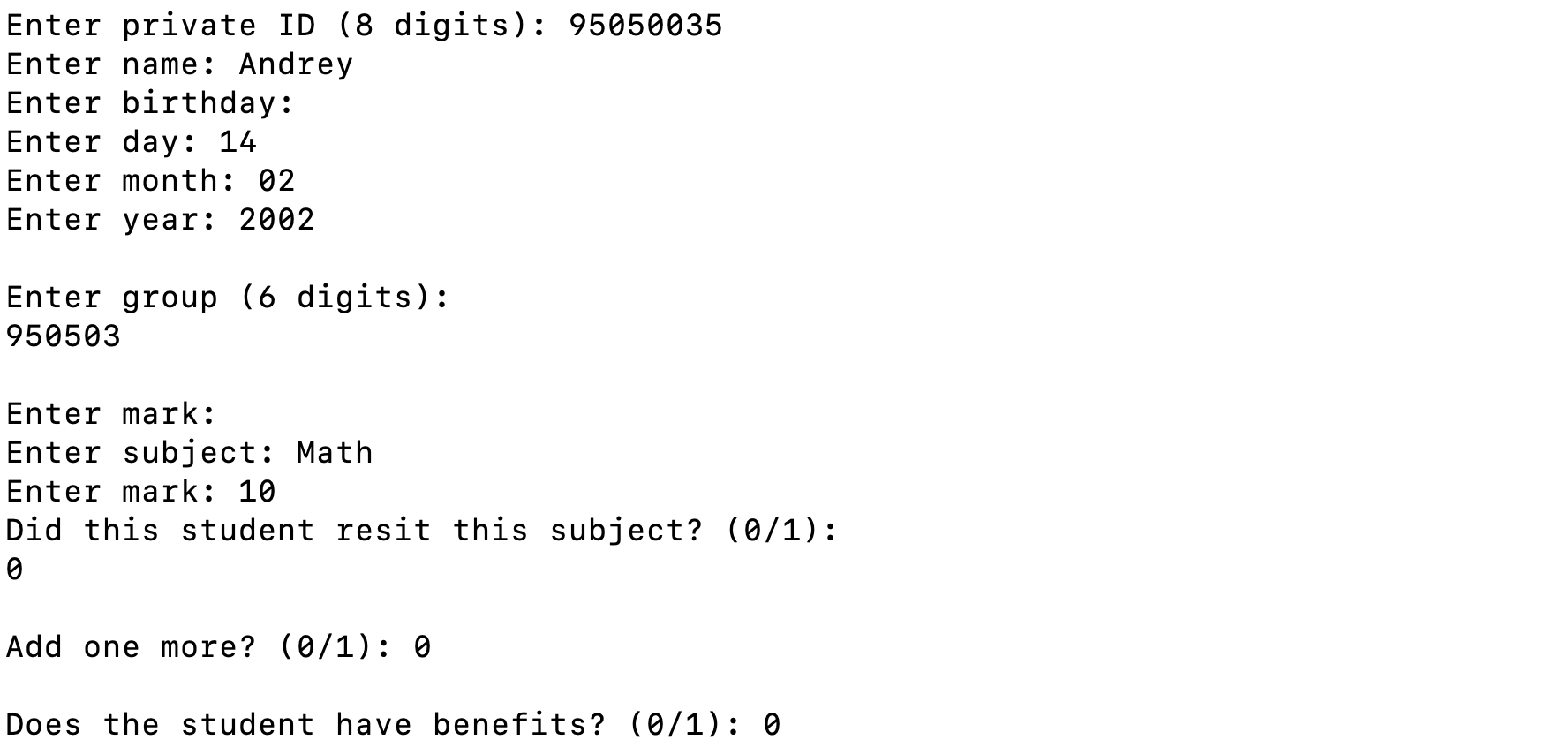
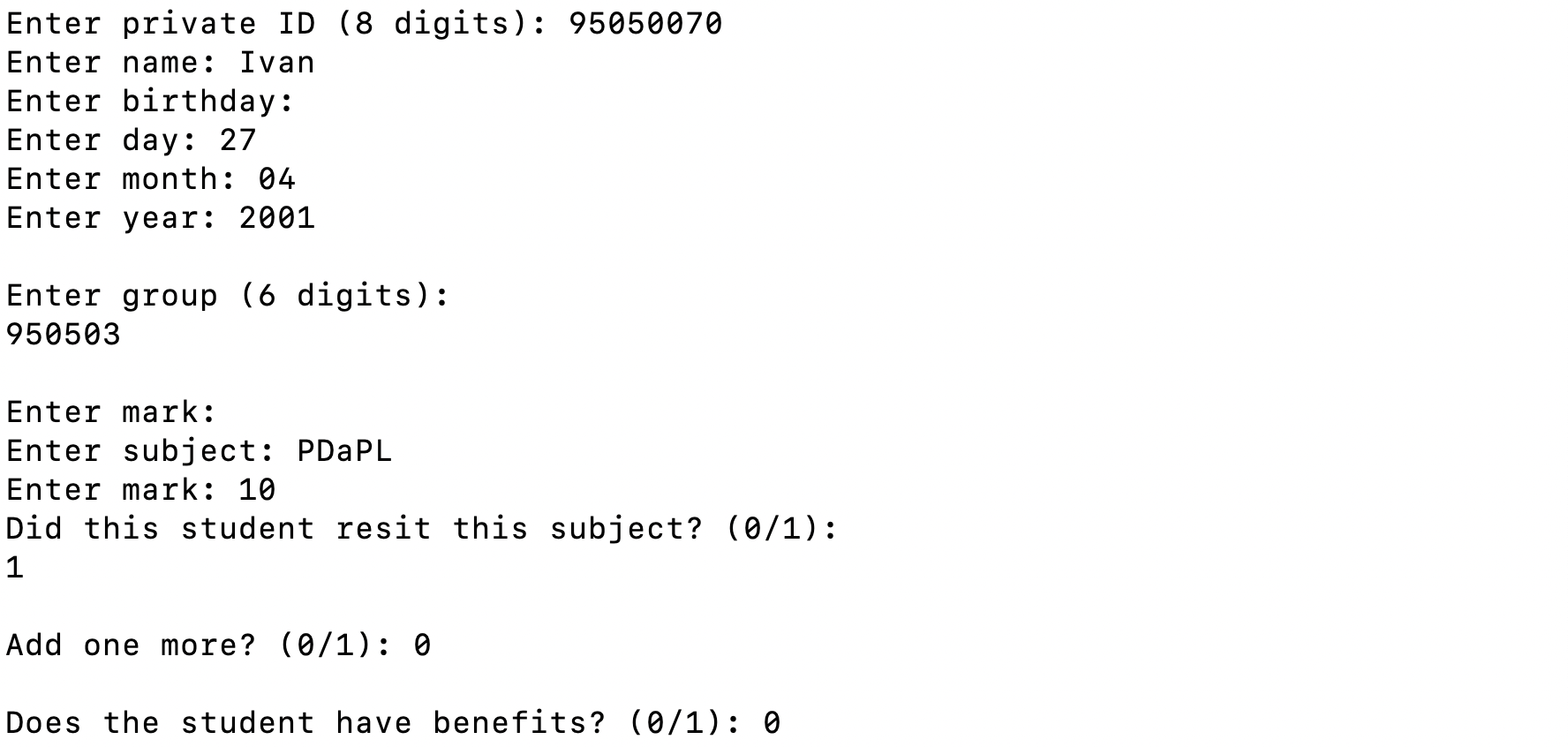
****

Рисунок 5.2



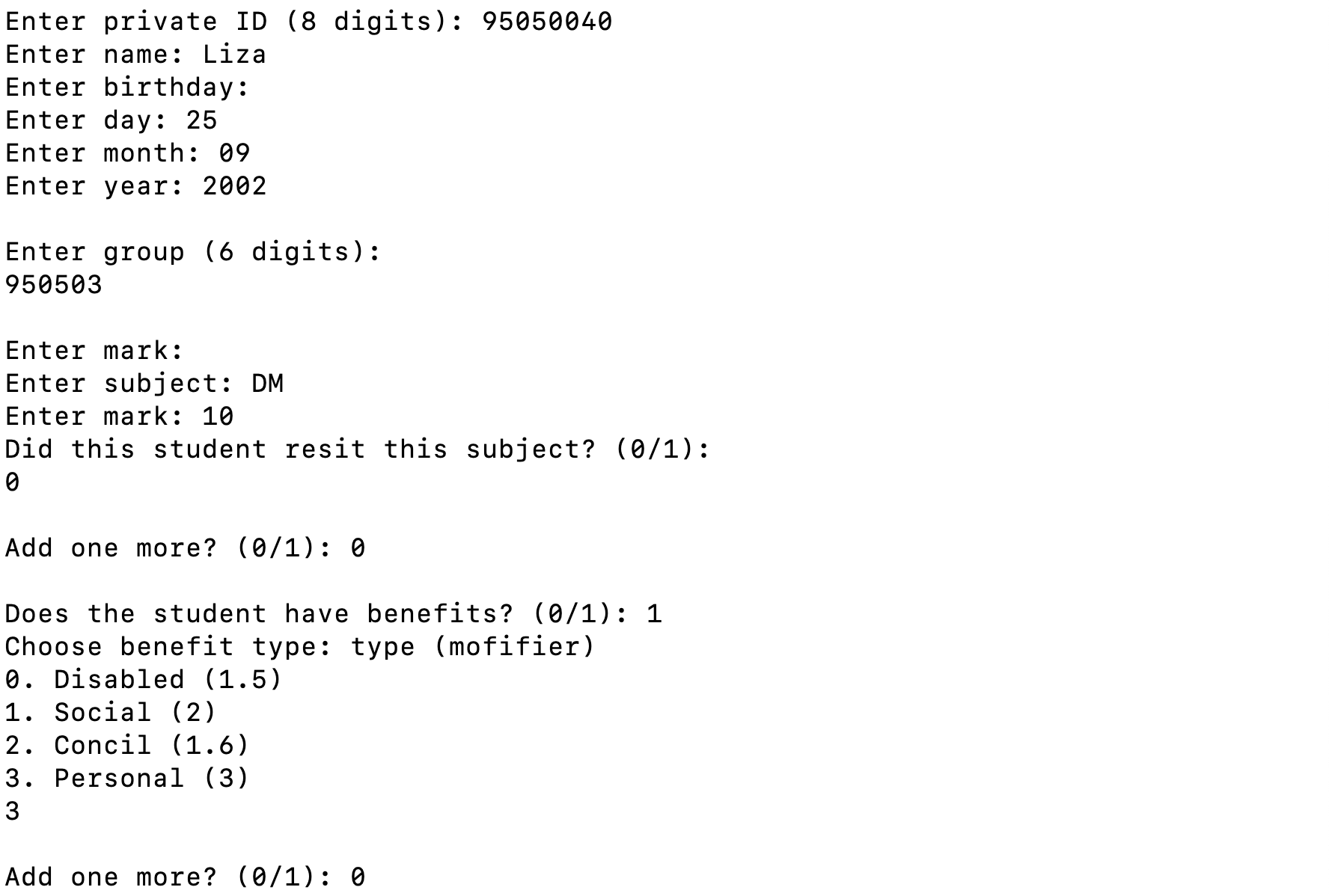


Рисунок 5.3

Просмотрим всех студентов, отсортированных по размеру стипендии (рис. 5.4). Видно, что “Ivan” не имеет стипендии, так как получил пересдачу по “PDaPL”, а “Liza” получила самую большую стипендию, так как у неё есть льгота “Personal”.

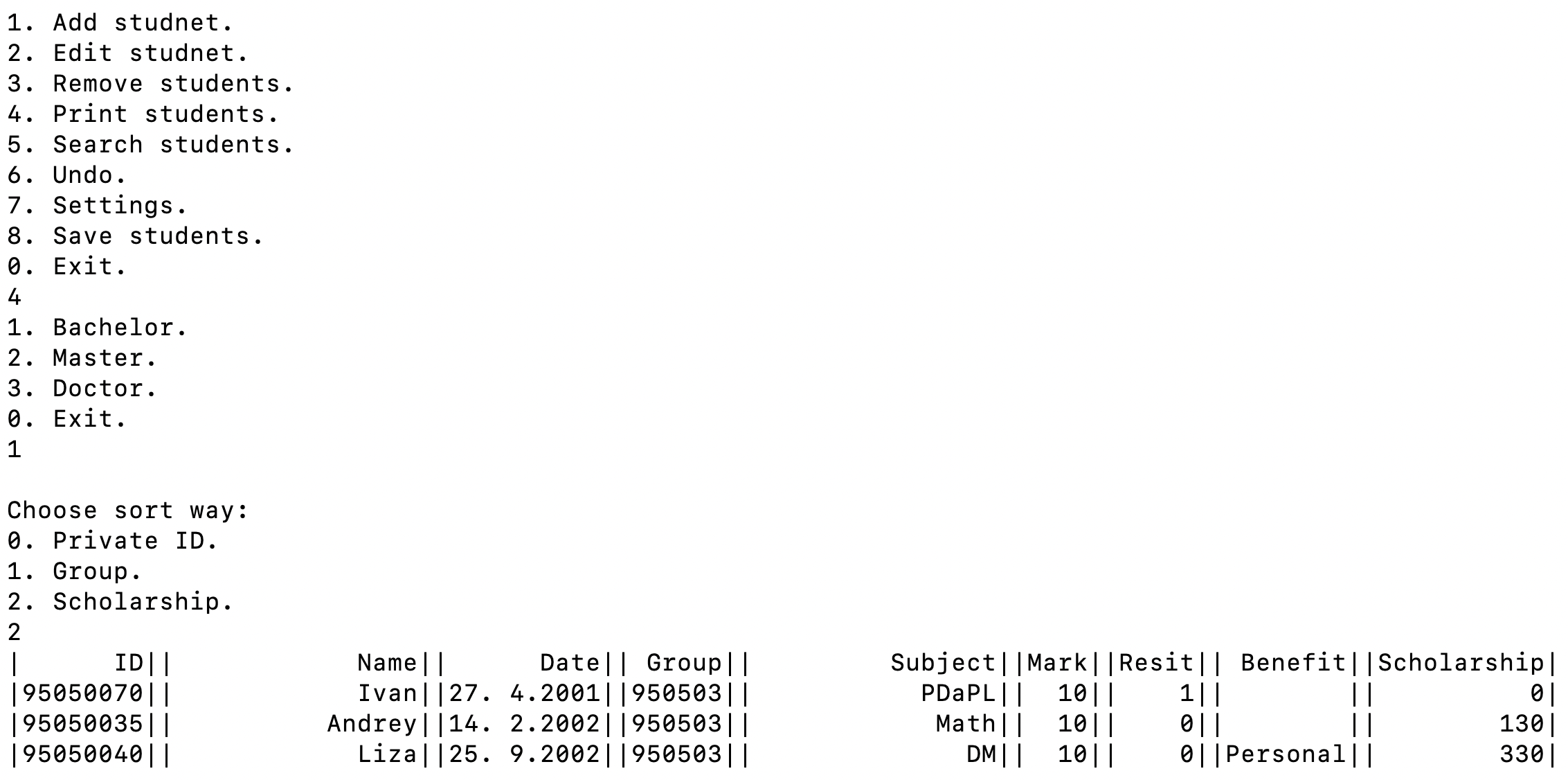
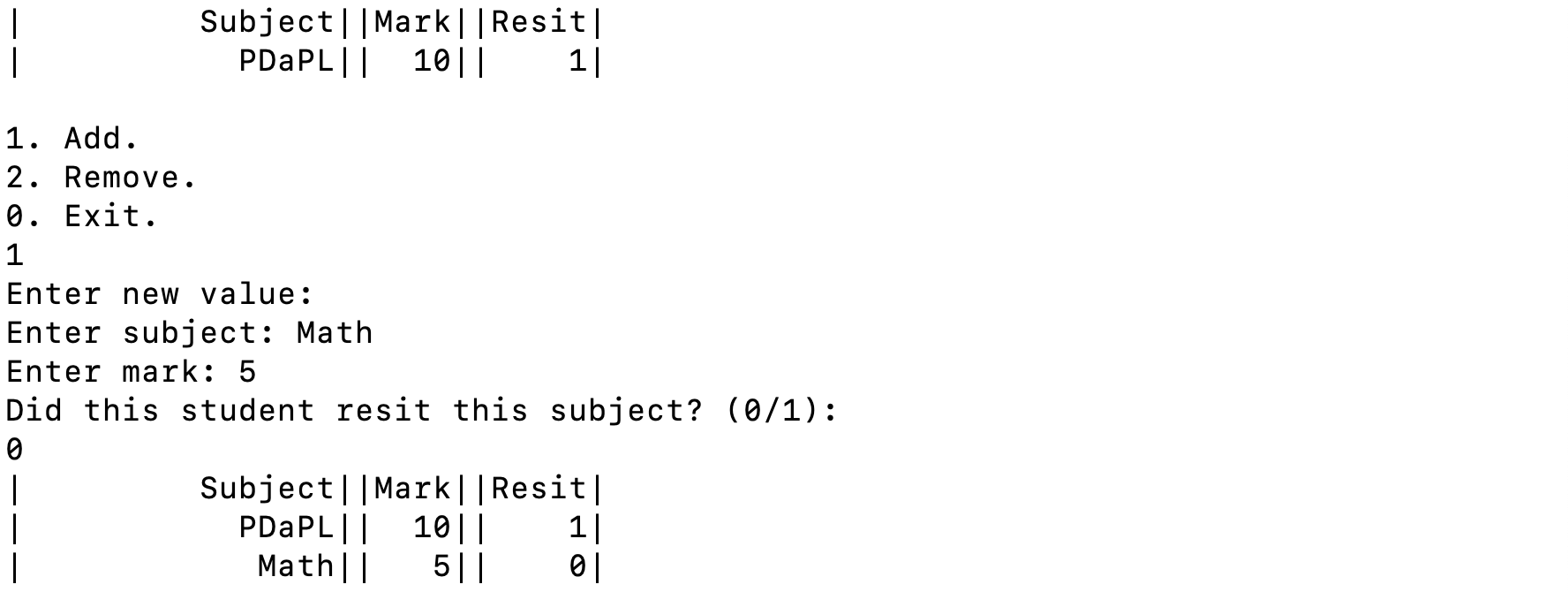


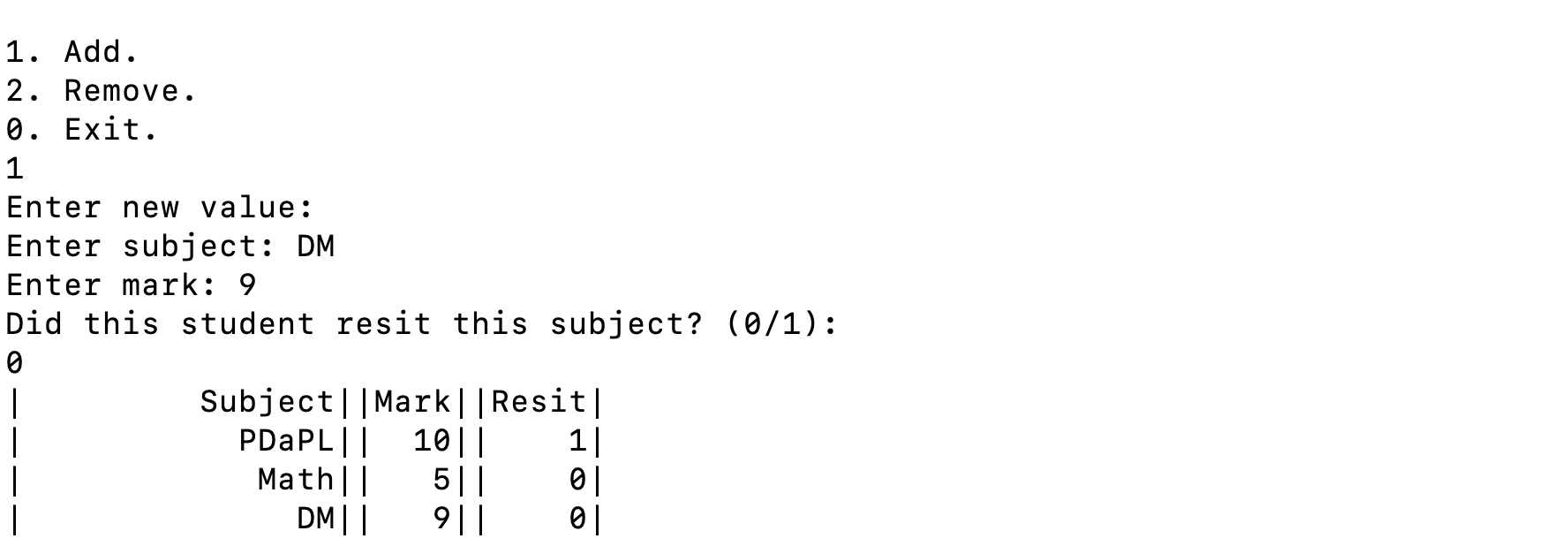
Рисунок 5.4

Теперь отредактируем “Ivan”, добавим две новые оценки и удалим оценку с пересдачей (рис. 5.5, 5.6).



Рисунок 5.5





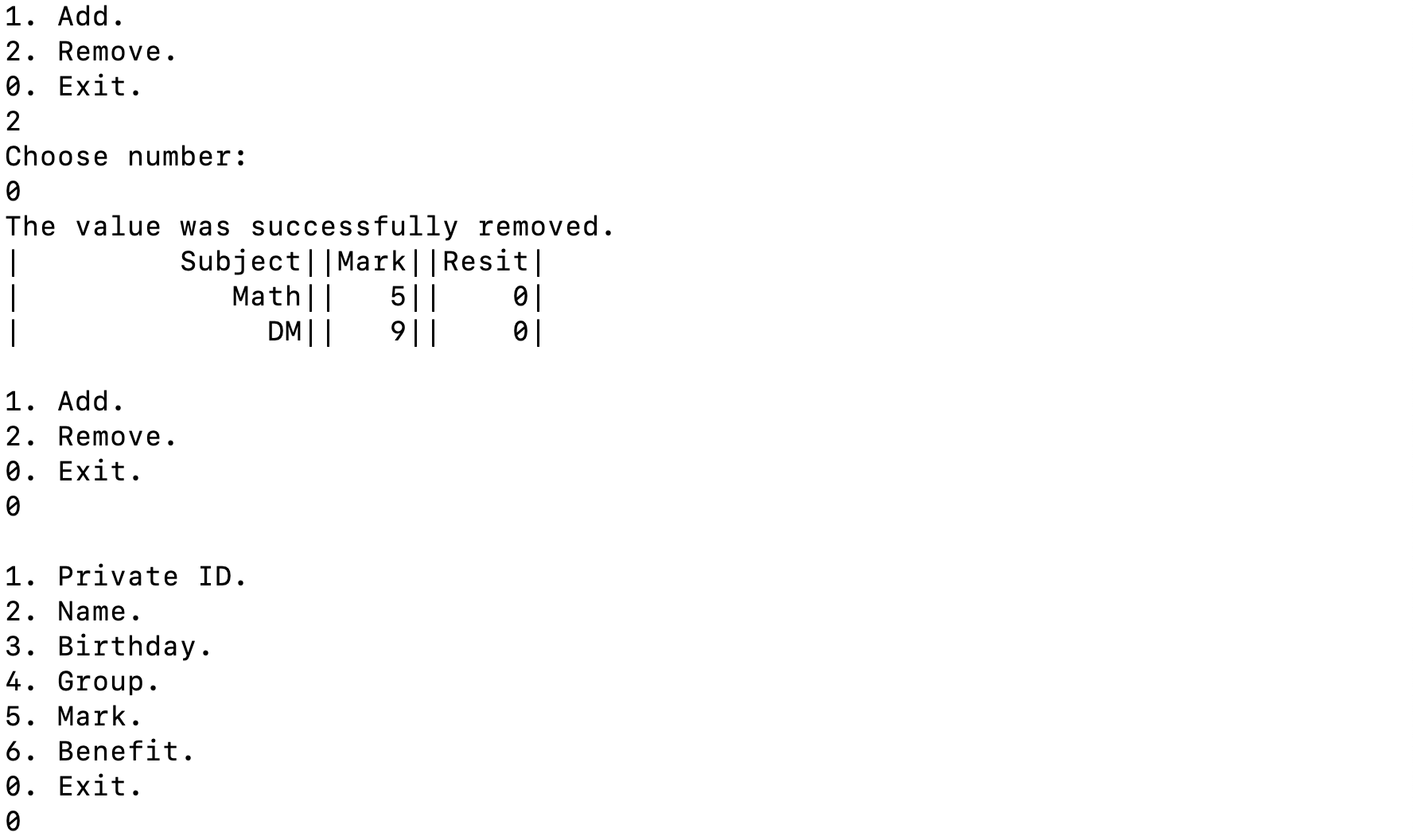
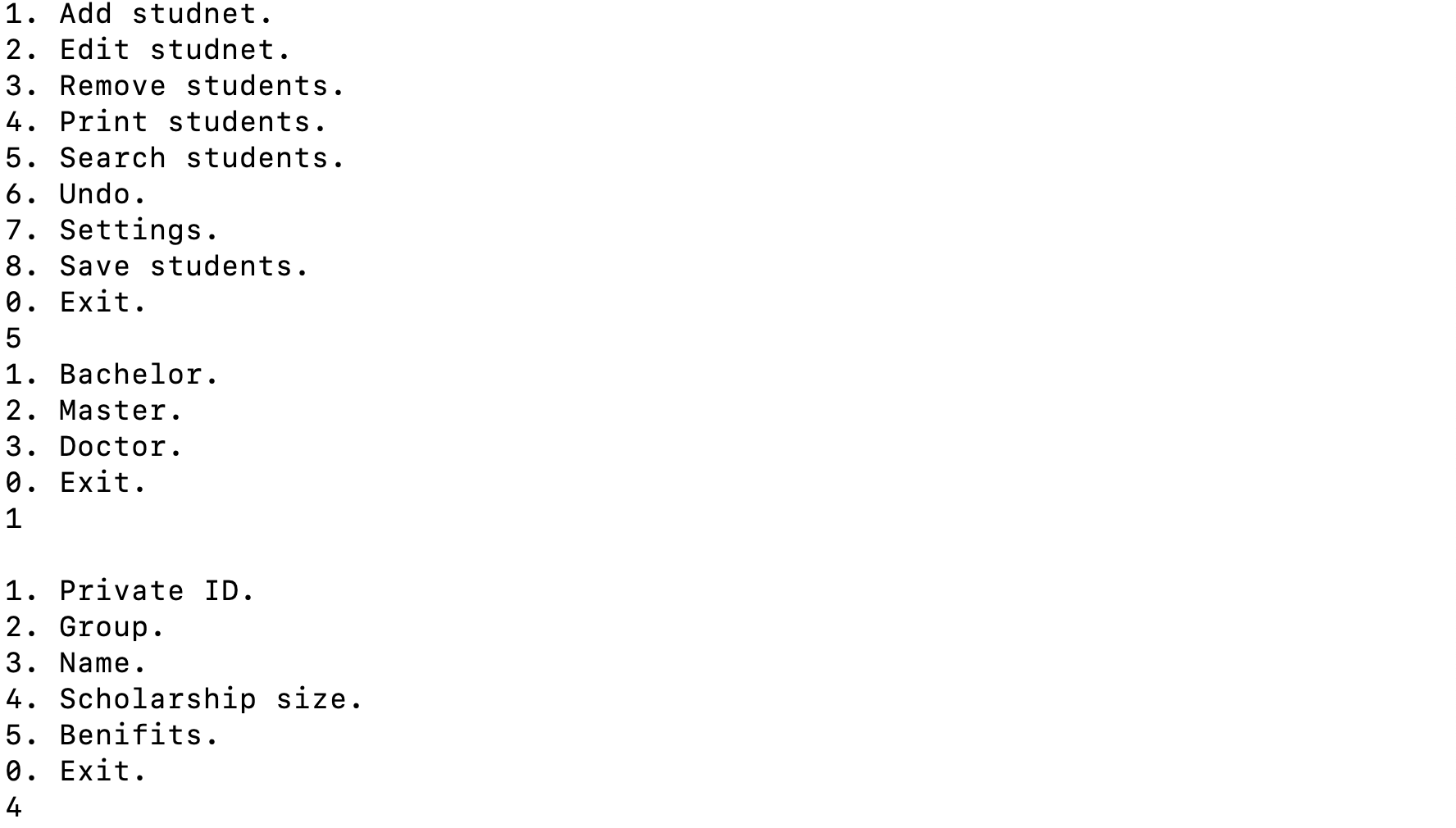


Рисунок 5.6

Найдём всех студентов с размером стипендии до 150 (рис. 5.7)



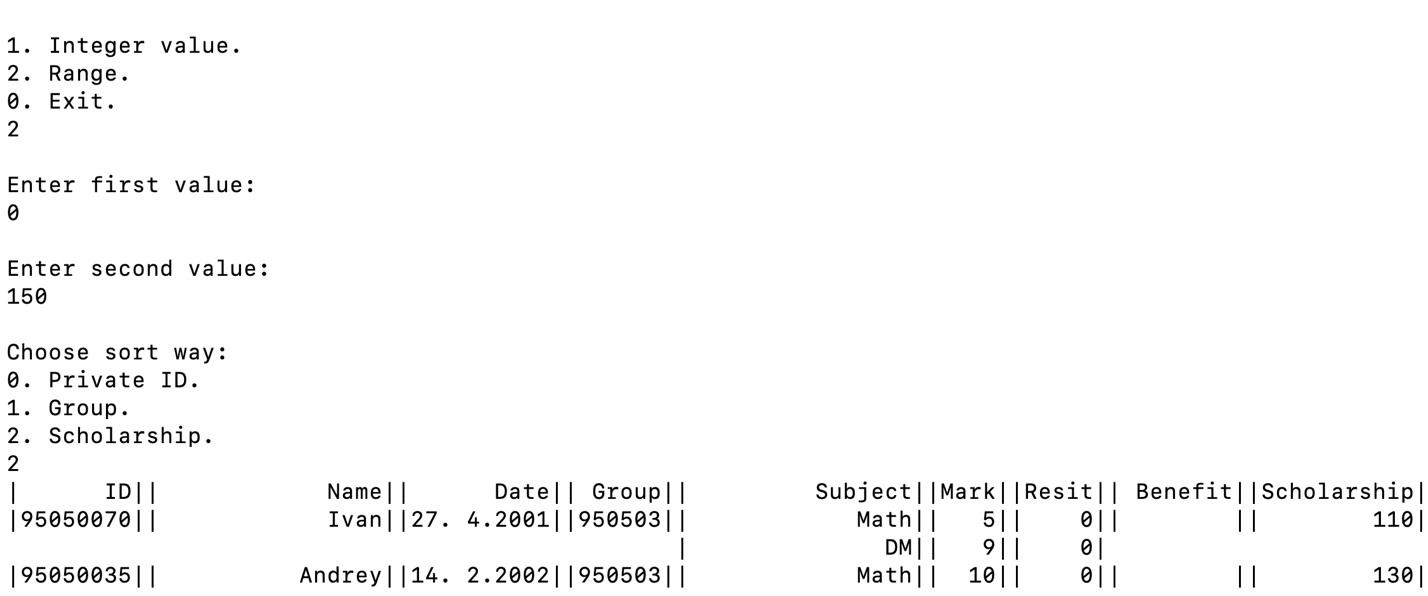


Рисунок 5.7

Теперь выполним отмену действий, чтобы отменить редактирование “Ivan” (рис. 5.8).



Рисунок 5.8

Теперь зайдём в настройки и отменим влияние пересдач на стипендию (рис 5.9, 5.10).

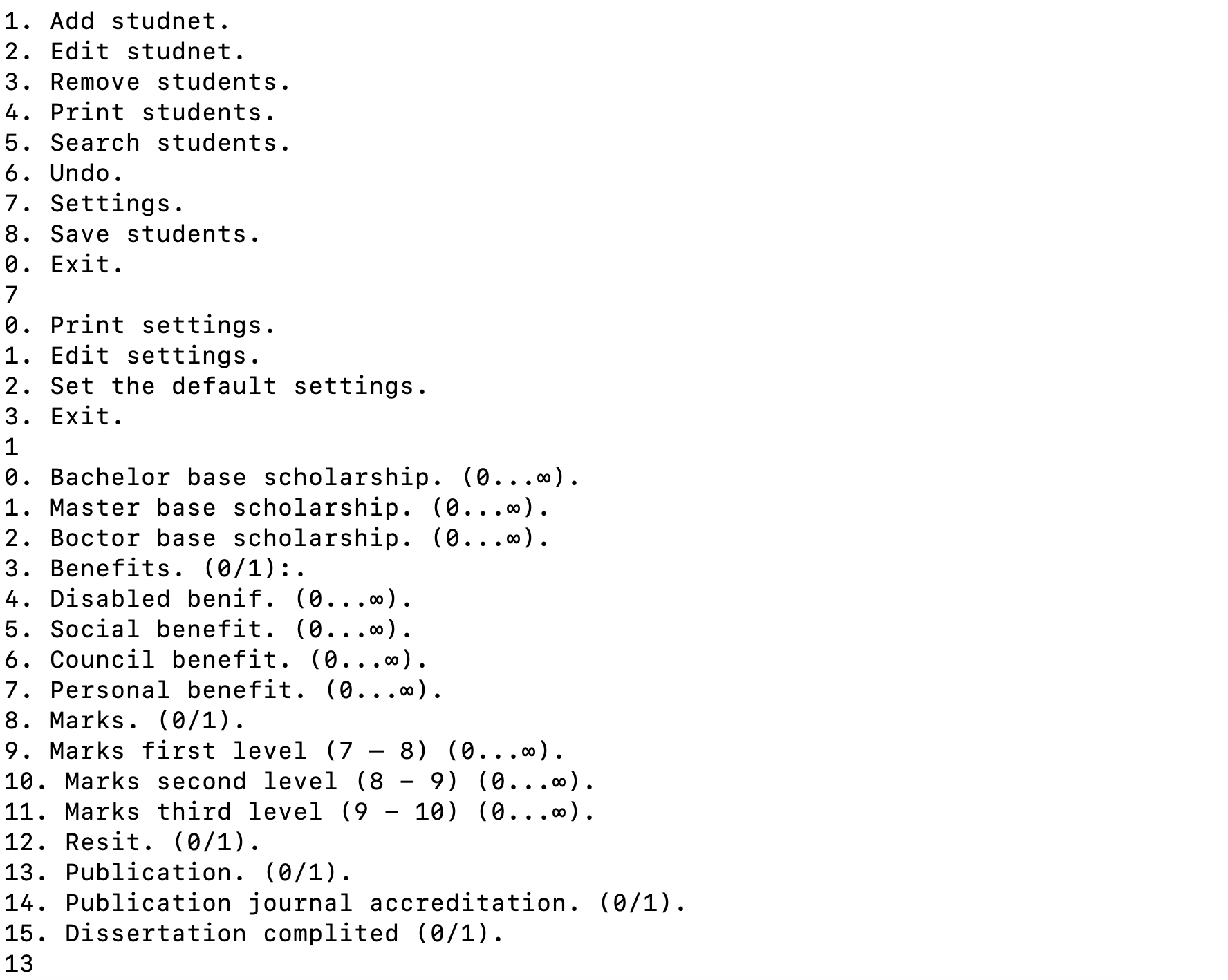


Рисунок 5.9



Рисунок 5.10

Выведем всех студентов по номеру студенческого билета (рис. 5.11). Видно, что, несмотря на то, что у «Ivan» есть пересдача, теперь он получает стипендию.



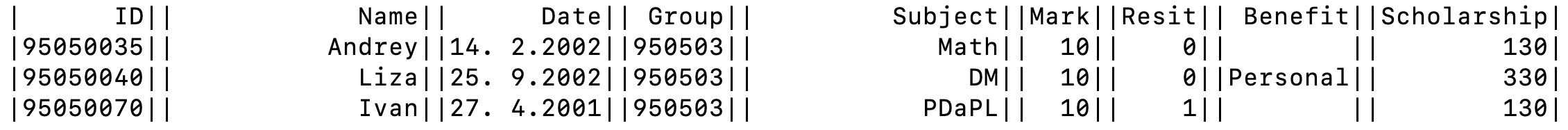


Рисунок 5.11

Теперь сохраним результат работы и выйдем из программы (рис. 5.12).

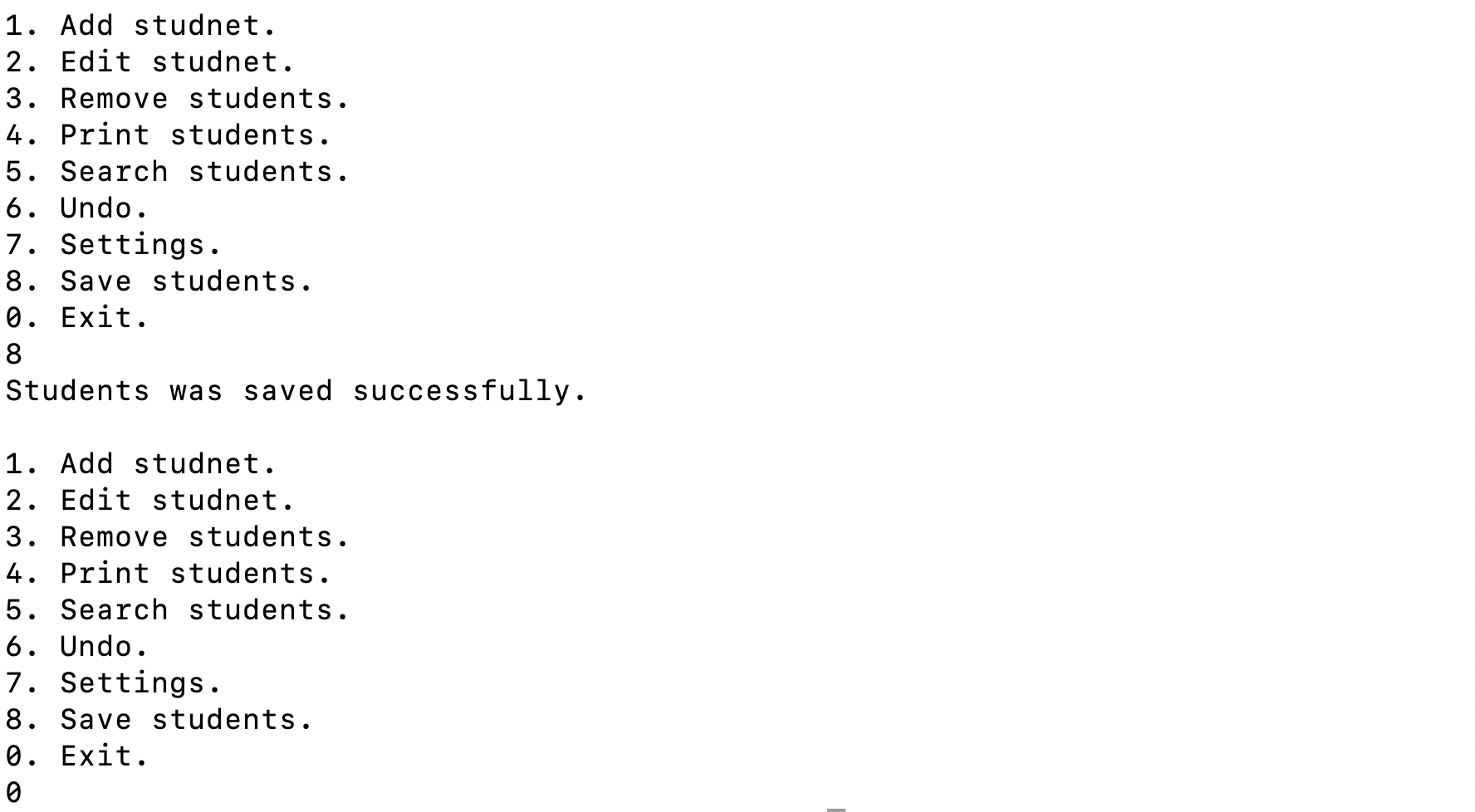


Рисунок 5.12

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения данного курсового проекта были применены на практике знания, полученные в ходе изучения курса КПиЯП, а также в ходе самостоятельного обучения. Благодаря этому была реализована программа «Расчёт и начисление стипендии», проведено тестирование данного программного продукта и устранение ошибок. Также проверена работоспособность программного продукта в различных операционных системах.

Были реализованы функции ввода, удаления и редактирования данных, поиск студентов по идентификационному номеру, группе, имени, размеру стипендии, льготам; сортировка по идентификационному номеру, номеру группы, стипендии.

Данный проект может быть усовершенствован в следующих направлениях:

- добавление графического интерфейса;

- хранение, вывод и сортировка всех типов студентов одновременно, без разделения на бакалавров, магистрантов и докторантов.

- добавление возможности локализации текста программы на другие языки без изменения программного кода.

- добавление возможности сортировки студентов по другим признакам.

- добавление возможности поиска студентов по совпадению сразу нескольких параметров.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

[1] Герберт, Ш. Самоучитель C++/Ш. Герберт. Санкт-Петербург 2003г.

[2] Дейтел, Х.М. Как программировать на С++ / Х.М. Дейтел, П.Д. Дейтел; пер. с англ. – М. : Бином, 2007. – 1152 с..

[3] Страуструп, Б. Язык программирования С++ / Б. Страуструп; специальное издание. Пер. с англ. – СПб. : BHV, 2008. – 1098 с.

[4] Элджер, Дж. C++: библиотека программиста / Дж. Элджер. – СПб. : Питер, 2001. – с.

[5] Объектно-ориентированное программирование в С++/ Роберт Лафоре г. пер. с англ. – Санкт-Петербург, 2019. – 1152 с.

**Приложение А**

*(обязательное)*

Диаграмма классов

**Приложение Б**

*(обязательное)*

Схема алгоритма метода static float calclulateBachelorScholarship(Bachelor&, float)

**Приложение В**

*(обязательное)*

Схема алгоритма метода template <typename studentType>

static void executeUndo(Undo<studentType>&, BinaryTree<studentType, int>&)

**Приложение Г**

*(обязательное)*

Код программы

**Приложение Д**

*(обязательное)*

Ведомость документов