**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №1**

**по дисциплине «Программирование»**

Тема: Типы данных, определяемые пользователем

|  |  |  |
| --- | --- | --- |
| Студент гр. |  | Полуянов В. Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Изучение и организация структур. Получение практических навыков работы со структурами. Определение преимуществ и недостатков использования структур.

**Основные теоретические положения.**

Структура – группа связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого строго однотипны. В языке программирования C++ структура является видом класса и обладает всеми его свойствами.

Для определения структуры применяется ключевое слово *struct*. Каждая входящая в структуру переменная называется членом (полем, элементом) структуры и описывается типом данных и именем. Поля структуры могут быть любого типа данных.

На структуры во многом похожи объединения. Объединения также хранят набор элементов, но в отличие от структуры все элементы объединения имеют нулевое смещение. То есть разные элементы занимают в памяти один и тот же участок.

Объединения объявляются ключевым словом *union*. В памяти члены объединения располагаются по одному адресу. Объединения не могут хранить одновременно несколько различных значений, они позволяют интерпретировать несколькими различными способами содержимое одной и той же области памяти.

Перечисления служат для определения пользовательских типов данных, значения которых принадлежат одному из элементов в некотором списке именованных целочисленных констант. Перечисление задается ключевым словом *enum*.

Для упрощения чтения программы, можно задать некоторому типу данных новое имя с помощью ключевого слова *typedef*. Объявления *typedef* можно использовать для создания более коротких или более понятных имен для типов, уже определенных в языке C/C++ или объявленных пользователем.

**Постановка задачи.**

Необходимо создать массив структур, содержащий информацию о студентах: ФИО, пол, номер группы, номер в списке группы, оценки за прошедшую сессию.

Написать функции, реализующие операции со структурами:

1. Создание новой записи о студенте.
2. Внесение изменений в уже имеющуюся запись.
3. Вывод всех данных о студентах.
4. Вывод информации обо всех студентах группы N.
5. Вывод самых успешных студентов с наивысшим по рейтингу средним баллом за прошедшую сессию.
6. Вывод количества студентов мужского и женского пола.
7. Вывод данных о студентах, которые не получают стипендию; учатся только на «хорошо» и «отлично»; учатся только на «отлично».
8. Вывод данных о студентах, имеющих номер в списке – k.

**Выполнение работы.**

Код программы представлен в приложении А.

1. При запуске программы считываются данные из *database.txt*. Если файл пустой, то выводится уведомление об этом. Затем пользователю выводится подсказка команды, выводящей меню доступных функций. Ожидается ввод команды с клавиатуры. (рис. 1).



Рисунок 1. Главное меню

1. Следующий шаг зависит от введенной команды, если пользователь ввёл:
   1. “0”, то выполнение программы завершается.
   2. “1”, то открывается меню создания новой записи (рис. 2).

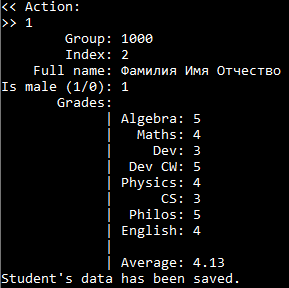


Рисунок 2. Задача 1

* 1. “2”, то ожидается ввод уникального ID студента (номер студенческого билета). После ввода открывается меню редактирования записи (рис. 3).

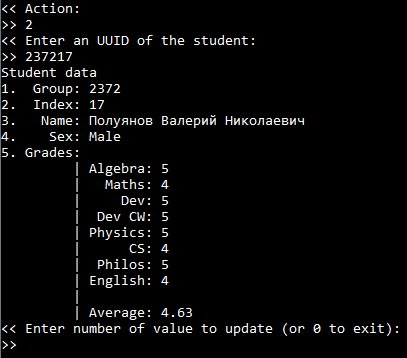


Рисунок 3. Задача 2

* 1. “3”, то выводятся все имеющиеся записи (рис. 4).

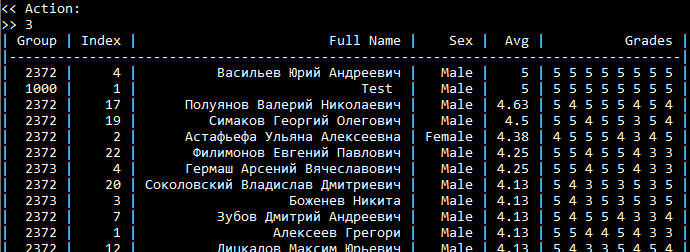


Рисунок 4. Задача 3

* 1. “4”, то ожидается ввод номера группы, после чего выводятся записи с номером группы, соответствующим введенному (рис. 5).

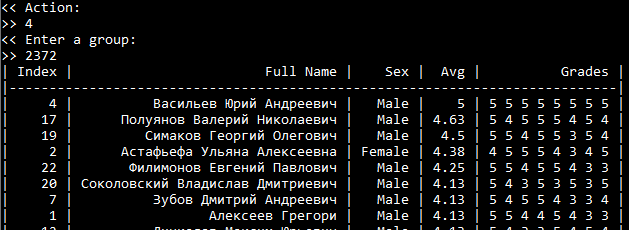


Рисунок 5. Задача 4

* 1. “5”, то выводится таблица, отсортированная по значениям среднего балла (рис. 6).

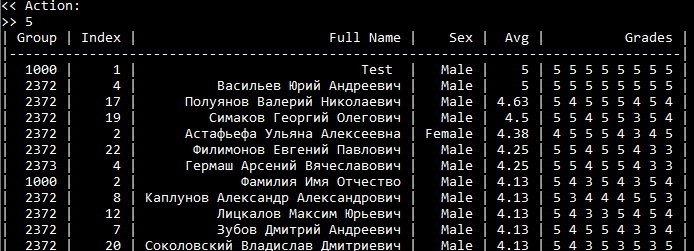


Рисунок 6. Задача 5

* 1. “6”, то выводится количество студентов мужского и женского полов соответственно (рис. 7).

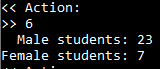


Рисунок 7. Задача 6

* 1. “7”, то выводится четыре таблицы в соответствии с размером стипендии (или её отсутствием) у студентов (рис. 8).

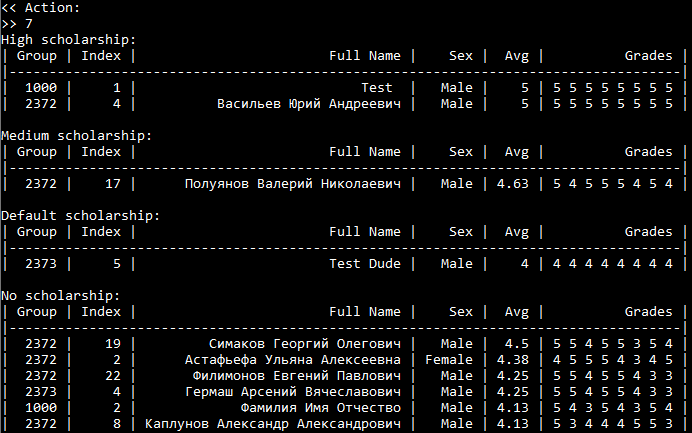


Рисунок 8. Задача 7

* 1. “4”, то ожидается ввод номера студента в группе, после чего выводятся все записи с соответствующим номером (рис. 9).

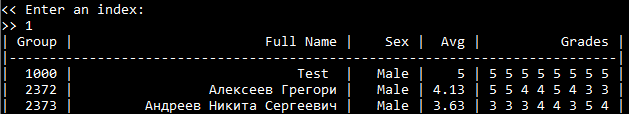


Рисунок 9. Задача 8

* 1. “h”, то выводится меню доступных команд (рис. 10).

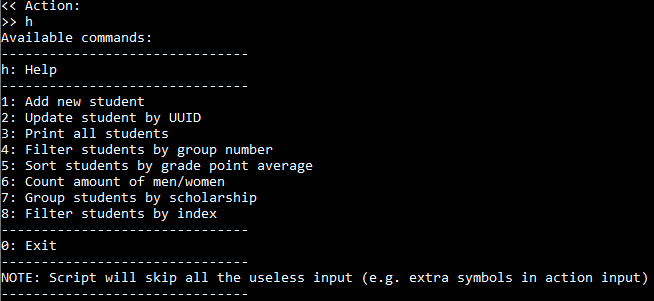


Рисунок 10. Меню

* 1. Любой символ или набор символов, отличный от существующий команд, то будет выведено сообщение об ошибке (рис. 11).



Рисунок 11. Неизвестная команда

**Выводы.**

В ходе работы были изучены структур, получены практические навыки работы с ними. Также были определены преимущества и недостатки использования структур.

Приложение А

рабочий код

#include **<iostream>**#include **<iomanip>**#include **<fstream>**#include **<Windows.h>**#include **<vector>**#include **<cmath>  
  
const** std::string lessonsInGradeBook[] = {  
 **"Algebra"**, **"Maths"**, **"Dev"**, **"Dev CW"**, **"Physics"**, **"CS"**, **"Philos"**, **"English"**};  
  
**short** getCursorPositionY() {  
 CONSOLE\_SCREEN\_BUFFER\_INFO screenBufferInfo = {};  
 GetConsoleScreenBufferInfo(GetStdHandle(**STD\_OUTPUT\_HANDLE**), &screenBufferInfo);  
 **return** screenBufferInfo.dwCursorPosition.Y;  
}  
  
**void** clearAfterCursor(**short** x = 0, **short** y = 0) {  
 COORD destCoord = {x, y};  
 HANDLE console = GetStdHandle(**STD\_OUTPUT\_HANDLE**);  
 CONSOLE\_SCREEN\_BUFFER\_INFO screen;  
 DWORD written;  
  
 GetConsoleScreenBufferInfo(console, &screen);  
 FillConsoleOutputCharacterA(  
 console, **' '**, screen.dwSize.X \* screen.dwSize.Y, destCoord, &written  
 );  
 SetConsoleCursorPosition(console, destCoord);  
}  
  
**struct** Student {  
 **unsigned short** group{};  
 **unsigned short** index{};  
 std::string fullName;  
 **bool** isMale{};  
 **unsigned short** grades[8]{}; *// 3 exams and 5 differentiated tests* **float** average{};  
};  
  
Student stringToStudent(std::string data) {  
 Student student;  
  
 student.group = std::stoi(data.substr(0, 4));  
 student.index = std::stoi(data.substr(5, 2));  
  
 data = data.substr(student.index < 10 ? 7 : 8);  
 student.fullName = **""**;  
 **for** (**int** i = 0; i < data.length(); ++i) {  
 **if** (data[i] == **','**) {  
 data = data.substr(i+1);  
 **break**;  
 }  
 student.fullName += data[i];  
 }  
  
 student.isMale = std::stoi(data.substr(0, 1));  
  
 **for** (**int** i = 0; i < 8; ++i) student.grades[i] = std::stoi(data.substr(2\*(i+1), 1));  
  
 student.average = std::stof(data.substr(18));  
  
 **return** student;  
}  
  
**unsigned** getUuid(**unsigned short** group, **unsigned short** index) {  
 **return** group \* 100 + index;  
}  
  
**float** getAverage(**const unsigned short** (&grades)[8]) {  
 **float** sum = 0;  
 **for** (**auto** grade : grades) sum += **float**(grade);  
 **return** std::round((sum / 8 \* 100)) / 100;  
}  
  
**bool** isUnique(std::vector<Student> &students, **unsigned short** group, **unsigned short** index) {  
 *// Not a std::any\_of because it needs to create uuids vector* **for** (**auto** &check : students) *// NOLINT(readability-use-anyofallof)* **if** (getUuid(check.group, check.index) == getUuid(group, index)) **return false**;  
 **return true**;  
}  
  
**void** getStudents(std::vector<Student> &students) {  
 std::ifstream database(**"database.txt"**);  
 **if** (!database.is\_open()) **return**;  
 std::string line;  
 **while** (!database.eof()) {  
 std::getline(database, line);  
 **if** (!line.empty()) students.push\_back(stringToStudent(line));  
 }  
}  
  
**int** saveStudents(std::vector<Student> &students) {  
 std::ofstream database(**"database.txt"**, std::ios::trunc);  
 **if** (!database.is\_open()) **return** 2;  
 **for** (**const auto** &student : students) {  
 database << student.group << **','**;  
 database << student.index << **','**;  
 database << student.fullName << **','**;  
 database << student.isMale << **','**;  
 **for** (**auto** grade : student.grades) database << grade << **','**;  
 database << student.average << std::endl;  
 }  
 database.close();  
}  
  
**int** addStudent(std::vector<Student> &students) {  
 */\*  
 \* return 0: No errors  
 \* return 1: TypeError in input  
 \* return 2: File read/write error  
 \* return 4: Student already exists  
 \*/* Student student;  
  
 *// Init values* std::cout << std::setw(15) << **"Group: "**;  
 **if** ((std::cin >> student.group).fail() || student.group < 1000 || student.group >= 10000) **return** 1;  
  
 std::cout << std::setw(15) << **"Index: "**;  
 **if** ((std::cin >> student.index).fail() || student.index < 1 || student.index >= 100) **return** 1;  
  
 **if** (!isUnique(students, student.group, student.index)) **return** 4;  
  
 std::cout << std::setw(15) << **"Full name: "**;  
 std::getline(std::cin.ignore(), student.fullName);  
  
 std::cout << std::setw(15) << **"Is male (1/0): "**;  
 **if** ((std::cin >> student.isMale).fail()) **return** 1;  
  
 std::cout << std::setw(15) << **"Grades:\n"**;  
 **for** (**int** i = 0; i < 8; ++i) {  
 std::cout << std::setw(14) << **'|'** << std::setw(8) << lessonsInGradeBook[i] << **": "**;  
 **if** ((std::cin >> student.grades[i]).fail() || student.grades[i] < 0 || student.grades[i] > 5) **return** 1;  
 **if** (student.grades[i] <= 2) {  
 std::cout << student.fullName << **" will be expelled. Saving to DB aborted"**;  
 **return** 2;  
 }  
 }  
  
 student.average = getAverage(student.grades);  
 std::cout << std::setw(15) << **"|\n"**;  
 std::cout << std::setw(14) << **'|'** << std::setw(10) << **"Average: "** << student.average << std::endl;  
  
 *// Push back and return* students.push\_back(student);  
 **return** 0;  
}  
  
**int** updateStudent(std::vector<Student> &students, **unsigned** uuid) {  
 */\*  
 \* return 0: No errors  
 \* return 1: Wrong input  
 \* return 2: File read/write error  
 \* return 3: Student not found  
 \* return 4: Student already exists  
 \*/  
  
 // Input error handler* **if** (uuid < 100000 || uuid > 999999) **return** 1;  
  
 *// Get student by UUID* **bool** isFound = **false**;  
 Student student;  
 **unsigned short** studentInd;  
 **for** (**int** i = 0; i < students.size(); ++i) {  
 **if** ((getUuid(students[i].group, students[i].index)) == uuid) {  
 isFound = **true**;  
 student = students[i];  
 studentInd = i;  
 }  
 }  
 **if** (!isFound) **return** 3;  
  
 *// Update value while not interrupted by user* **unsigned short** userInput = 0;  
 **short** startCursorPosY = getCursorPositionY();  
 **do** {  
 *// Print student info* std::cout << **"Student data"** << (userInput == 0 ? **""** : **"(updated)"**) << std::endl;  
 std::cout << **"1. "** << std::setw(8) << **"Group: "** << student.group << std::endl;  
 std::cout << **"2. "** << std::setw(8) << **"Index: "** << student.index << std::endl;  
 std::cout << **"3. "** << std::setw(8) << **"Name: "** << student.fullName << std::endl;  
 std::cout << **"4. "** << std::setw(8) << **"Sex: "** << (student.isMale ? **"Male"** : **"Female"**) << std::endl;  
 std::cout << **"5. "** << std::setw(8) << **"Grades: "** << std::endl;  
 **for** (**int** i = 0; i < 8; ++i) {  
 std::cout << std::setw(10) << **'|'** << std::setw(8);  
 std::cout << lessonsInGradeBook[i] << **": "** << student.grades[i] << std::endl;  
 }  
 std::cout << std::setw(11) << **"|\n"**;  
 std::cout << std::setw(10) << **'|'** << std::setw(10) << **"Average: "** << student.average << std::endl;  
  
 *// Update value* std::cout << **"<< Enter number of value to update (or 0 to exit):\n>> "**;  
 std::cin >> userInput;  
 **switch** (userInput) {  
 **case** 0: **break**;  
 **case** 1: {  
 std::cout << **"Group: "**;  
 **if** ((std::cin >> student.group).fail() || student.group < 1000 || student.group >= 10000) **return** 1;  
 **if** (!isUnique(students, student.group, student.index)) **return** 4;  
 **break**;  
 }  
 **case** 2: {  
 std::cout << **"Index: "**;  
 **if** ((std::cin >> student.index).fail() || student.index < 1 || student.index >= 100) **return** 1;  
 **if** (!isUnique(students, student.group, student.index)) **return** 4;  
 **break**;  
 }  
 **case** 3: {  
 std::cout << **"Full name (really?): "**;  
 std::getline(std::cin.ignore(), student.fullName);  
 **break**;  
 }  
 **case** 4: {  
 std::cout << **"Is male (1/0) (ayo dude's from Turkey): "**;  
 **if** ((std::cin >> student.isMale).fail()) **return** 1;  
 **break**;  
 }  
 **case** 5: {  
 *// Clear all after the start and write the grade book data* clearAfterCursor(0, startCursorPosY);  
 std::cout << **"Student grades data"** << std::endl;  
 **for** (**int** i = 0; i < 8; ++i) {  
 std::cout << i+1 << **'.'** << std::setw(8);  
 std::cout << lessonsInGradeBook[i] << **": "** << student.grades[i] << std::endl;  
 }  
 std::cout << std::setw(12) << **"Average: "** << student.average << std::endl;  
  
 *// Get input* **unsigned short** gradeInd;  
 std::cout << **"<< Enter number of value to update (or 0 to exit):\n>> "**;  
 **if** ((std::cin >> gradeInd).fail() || gradeInd > 8) **return** 1;  
 gradeInd--;  
 std::cout << lessonsInGradeBook[gradeInd] << **": "**;  
 **if** ((std::cin >> student.grades[gradeInd]).fail()) **return** 1; *// Not an uint* **if** (student.grades[gradeInd] < 0 || student.grades[gradeInd] > 5) **return** 1; *// Out of range* **if** (student.grades[gradeInd] <= 2) { *// Expelled* std::cout << student.fullName << **" will be expelled. Saving to DB aborted"**;  
 **return** 2;  
 }  
 student.average = getAverage(student.grades);  
 **break**;  
 }  
 **default**: **return** 1; *// Not an uint or not in range* }  
  
 *// Clear all after the start and replace cursor to the start line* **if** (userInput != 0) clearAfterCursor(0, startCursorPosY);  
  
 } **while** (userInput != 0);  
  
 *// Save the update and return* students[studentInd] = student;  
 **return** 0;  
}  
  
**void** printAllStudents(std::vector<Student> &students) {  
 *// Header* std::cout << **"| Group | Index | Full Name | Sex | Avg | Grades |\n"**;  
 std::cout << **'|'** << std::setw(86) << std::setfill(**'-'**) << **"|\n"** << std::setfill(**' '**);  
  
 *// Empty table check* **if** (students.empty()) {  
 std::cout << **'|'** << std::setw(45) << **"No data."** << std::setw(41) << **"|\n"**;  
 **return**;  
 }  
  
 *// Table* **for** (**auto** &student : students) {  
 std::cout << **'|'** << std::setw(6) << student.group << **' '**;  
 std::cout << **'|'** << std::setw(6) << student.index << **' '**;  
 std::cout << **'|'** << std::setw(33) << student.fullName << **' '**;  
 std::cout << **'|'** << std::setw(7) << (student.isMale ? **"Male"** : **"Female"**) << **' '**;  
 std::cout << **'|'** << std::setw(5) << getAverage(student.grades) << **" |"**;  
 **for** (**auto** grade : student.grades) std::cout << **' '** << grade;  
 std::cout << **" |\n"**;  
 }  
}  
  
**void** printStudentsByGroup(std::vector<Student> &students, **unsigned short** group) {  
 *// Header* std::cout << **"| Index | Full Name | Sex | Avg | Grades |\n"**;  
 std::cout << **'|'** << std::setw(78) << std::setfill(**'-'**) << **"|\n"** << std::setfill(**' '**);  
  
 *// Wrong group -> table empty* **if** (group < 1000 || group >= 10000) {  
 std::cout << **'|'** << std::setw(41) << **"No data."** << std::setw(37) << **"|\n"**;  
 **return**;  
 }  
  
 *// Table* **bool** isPrinted = **false**;  
 **for** (**auto** &student : students) {  
 **if** (student.group == group) {  
 std::cout << **'|'** << std::setw(6) << student.index << **' '**;  
 std::cout << **'|'** << std::setw(33) << student.fullName << **' '**;  
 std::cout << **'|'** << std::setw(7) << (student.isMale ? **"Male"** : **"Female"**) << **' '**;  
 std::cout << **'|'** << std::setw(5) << student.average << **" |"**;  
 **for** (**auto** grade : student.grades) std::cout << **' '** << grade;  
 std::cout << **" |\n"**;  
 isPrinted = **true**;  
 }  
 }  
 **if** (!isPrinted) *// Empty table* std::cout << **'|'** << std::setw(41) << **"No data."** << std::setw(37) << **"|\n"**;  
}  
  
**bool** compareGrades(**const** Student &st1, **const** Student &st2) {  
 **return** (st1.average > st2.average);  
}  
  
**void** printNumberByGender(std::vector<Student> &students) {  
 **unsigned short** maleAmount = 0, femaleAmount = 0;  
 **for** (**auto** &student : students) {  
 **if** (student.isMale) maleAmount++;  
 **else** femaleAmount++;  
 }  
 std::cout << **" Male students: "** << maleAmount << std::endl;  
 std::cout << **"Female students: "** << femaleAmount << std::endl;  
}  
  
**void** printByScholarship(std::vector<Student> &students) {  
 std::vector<Student> noScholarship, lowScholarship, midScholarship, highScholarship;  
  
 *// Filter by '3' | '4' | '4-5' | '5'* **for** (**auto** &student : students) {  
 **bool** any3 = **false**, any4 = **false**, any5 = **false**;  
 **for** (**auto** grade : student.grades) {  
 **if** (grade == 3) {  
 any3 = **true**;  
 **break**; *// If '3' in grades then no scholarship* }  
 **if** (grade == 4) any4 = **true**;  
 **else** any5 = **true**;  
 }  
  
 **if** (any3) {  
 noScholarship.push\_back(student); *// '3' in grades* } **else if** (any5) {  
 **if** (any4) midScholarship.push\_back(student); *// '4' or '5'* **else** highScholarship.push\_back(student); *// All '5'* } **else** {  
 lowScholarship.push\_back(student); *// All '4'* }  
 }  
  
 *// Print data* std::cout << **"High scholarship:\n"**;  
 printAllStudents(highScholarship);  
 std::cout << **"\nMedium scholarship:\n"**;  
 printAllStudents(midScholarship);  
 std::cout << **"\nDefault scholarship:\n"**;  
 printAllStudents(lowScholarship);  
 std::cout << **"\nNo scholarship:\n"**;  
 printAllStudents(noScholarship);  
}  
  
**void** printStudentsByIndex(std::vector<Student> &students, **unsigned short** index) {  
 *// Header* std::cout << **"| Group | Full Name | Sex | Avg | Grades |\n"**;  
 std::cout << **'|'** << std::setw(78) << std::setfill(**'-'**) << **"|\n"** << std::setfill(**' '**);  
  
 *// Wrong group -> table empty* **if** (index < 0 || index >= 100) {  
 std::cout << **'|'** << std::setw(42) << **"No data."** << std::setw(36) << **"|\n"**;  
 **return**;  
 }  
  
 *// Table* **bool** isPrinted = **false**;  
 **for** (**auto** &student : students) {  
 **if** (student.index == index) {  
 std::cout << **'|'** << std::setw(6) << student.group << **' '**;  
 std::cout << **'|'** << std::setw(33) << student.fullName << **' '**;  
 std::cout << **'|'** << std::setw(7) << (student.isMale ? **"Male"** : **"Female"**) << **' '**;  
 std::cout << **'|'** << std::setw(5) << student.average << **" |"**;  
 **for** (**auto** grade : student.grades) std::cout << **' '** << grade;  
 std::cout << **" |\n"**;  
 isPrinted = **true**;  
 }  
 }  
 **if** (!isPrinted) *// Empty table* std::cout << **'|'** << std::setw(42) << **"No data."** << std::setw(36) << **"|\n"**;  
}  
  
**int** main() {  
 setlocale(**LC\_ALL**, **"ru"**);  
  
 *// Get students from the DB* std::vector<Student> students;  
 getStudents(students);  
 **if** (students.empty())  
 std::cout << **"Database is empty. You've to add students via '1' command.\n"**;  
  
 *// Main loop* std::cout << **"Enter 'h' to get list of commands\n"**;  
 **while** (**true**) {  
  
 *// Get command from the user* **char** userAction;  
 std::cout << **"<< Action:\n>> "**;  
 std::cin.sync();  
 std::cin >> userAction;  
  
 *// Exit* **if** (userAction == **'0'**) **break**;  
  
 **switch** (userAction) {  
  
 *// Add new student* **case '1'**: {  
 **int** response = addStudent(students);  
 **if** (response != 0) {  
 std::cout << **"Student's data wasn't saved. Something went wrong.\n"**;  
 **if** (response == 1) std::cout << **"ValueError: Wrong input\n"**;  
 **else if** (response == 2) std::cout << **"FileError: Cannot open the \"database.txt\"\n"**;  
 **else if** (response == 4) std::cout << **"ExistsError: Student already exists\n"**;  
 std::cin.clear();  
 } **else** {  
 std::cout << **"Student's data has been saved.\n"**;  
 }  
 **break**;  
 }  
  
 *// Update student with specified UUID* **case '2'**: {  
 **unsigned** uuid;  
 std::cout << **"<< Enter an UUID of the student:\n>> "**;  
 std::cin >> uuid;  
 **int** response = updateStudent(students, uuid);  
 **if** (response != 0) {  
 std::cout << **"Student's data wasn't updated. Something went wrong.\n"**;  
 **if** (response == 1) std::cout << **"ValueError: Wrong input\n"**;  
 **else if** (response == 2) std::cout << **"FileError: Cannot open the \"database.txt\"\n"**;  
 **else if** (response == 3) std::cout << **"NotFoundError: There's no student with such UUID\n"**;  
 **else if** (response == 4) std::cout << **"ExistsError: Student already exists\n"**;  
 } **else** {  
 std::cout << **"Student's data has been updated.\n"**;  
 }  
 **break**;  
 }  
  
 *// Print all students* **case '3'**: {  
 printAllStudents(students);  
 **break**;  
 }  
  
 *// Print students filtered by group* **case '4'**: {  
 **unsigned short** group;  
 std::cout << **"<< Enter a group:\n>> "**;  
 **if** ((std::cin >> group).fail()) {  
 std::cout << **"ValueError: Wrong input\n"**;  
 **break**;  
 }  
 printStudentsByGroup(students, group);  
 **break**;  
 }  
  
 *// Print students sorted by average grade* **case '5'**: {  
 std::sort(students.begin(), students.end(), compareGrades);  
 printAllStudents(students);  
 **break**;  
 }  
  
 *// Print amount of students by each gender* **case '6'**: {  
 printNumberByGender(students);  
 **break**;  
 }  
  
 *// Print students grouped by scholarship* **case '7'**: {  
 printByScholarship(students);  
 **break**;  
 }  
  
 *// Print students filtered by index* **case '8'**: {  
 **unsigned short** index;  
 std::cout << **"<< Enter an index:\n>> "**;  
 **if** ((std::cin >> index).fail()) {  
 std::cout << **"ValueError: Wrong input\n"**;  
 **break**;  
 }  
 printStudentsByIndex(students, index);  
 **break**;  
 }  
  
 *// Help* **case 'h'**: {  
 std::cout << **"Available commands:\n"**;  
 std::cout << std::setw(32) << std::setfill(**'-'**) << **'\n'**;  
 std::cout << **"h: Help\n"**;  
 std::cout << std::setw(32) << std::setfill(**'-'**) << **'\n'**;  
 std::cout << **"1: Add new student\n"**;  
 std::cout << **"2: Update student by UUID\n"**;  
 std::cout << **"3: Print all students\n"**;  
 std::cout << **"4: Filter students by group number\n"**;  
 std::cout << **"5: Sort students by grade point average\n"**;  
 std::cout << **"6: Count amount of men/women\n"**;  
 std::cout << **"7: Group students by scholarship\n"**;  
 std::cout << **"8: Filter students by index\n"**;  
 std::cout << std::setw(32) << std::setfill(**'-'**) << **'\n'**;  
 std::cout << **"0: Exit\n"**;  
 std::cout << std::setw(32) << std::setfill(**'-'**) << **'\n'**;  
 std::cout << **"NOTE: Script will skip all the useless input (e.g. extra symbols in action input)\n"**;  
 std::cout << std::setw(32) << std::setfill(**'-'**) << **'\n'**;  
 std::cout << std::setfill(**' '**);  
 **break**;  
 }  
  
 *// Unknown command error* **default**: std::cout << **"RuntimeError: unknown command\n"**;  
 }  
  
 *// Update the DB* saveStudents(students);  
 }  
  
 **return** 0;  
}