|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  «МИРЭА – Российский технологический университет»  РТУ МИРЭА |

Институт комплексной безопасности и специального приборостроения

(наименование института, филиала)

Кафедра КБ-3 «Управление и моделирование систем»

(наименование кафедры)

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Языки программирования»

(наименование дисциплины)

**Тема курсового проекта**  Разработка файловой системы студентов c выполнением задания в соответствии с требуемым вариантом.

**Студент группы** Ионов Максим Сергеевич, группа БББО-07-19 \_\_\_\_\_\_\_\_\_\_

(учебная группа, фамилия, имя, отчество студента) (подпись студента)

**Руководитель курсового проекта** Мерсов Андрей Александрович \_\_\_\_\_\_\_\_\_\_\_

(должность, звание, ученая степень) (подпись руководителя)

**Рецензент (при наличии)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

(должность, звание, ученая степень) (подпись рецензента)

Работа представлена к защите «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Допущен к защите «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г

|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  «МИРЭА – Российский технологический университет»  РТУ МИРЭА |

Институт комплексной безопасности и специального приборостроения

(наименование института, филиала)

Кафедра КБ-3 «Управление и моделирование систем»

(наименование кафедры)

Утверждаю

Заведующий кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

подпись ФИО

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**ЗАДАНИЕ**

**На выполнение курсового проекта** по дисциплине

«Языки программирования»

Студент Ионов Максим Сергеевич Группа БББО-07-19

**Тема:** разработка файловой системы студентов с выполнением задания в соответствии с вариантом, а именно распределение всех студентов по группам: хорошисты, троечники, отличники за все время обучения и сортировка каждой части по алфавиту.

**Исходные данные:** информация о группе студентов из N человек, где запись о студенте содержит определенные данные.

**Перечень вопросов, подлежащих разработке и обязательного графического материала:** изучение теоретической части объектно-ориентированного программирования, основными элементами которого являются наследование, классы, объекты и методы классов, дружественные классы.

Срок представления к защите курсового проекта: до «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Задание на курсовой проект выдал \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Подпись руководителя Ф.И.О. руководителя

Задание на курсовой проект получил \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Подпись обучающегося Ф.И.О.исполнителя

Оглавление:

[ВВЕДЕНИЕ 3](#_Toc41330816)

[**Глава 1. Разработка программного обеспечения** 4](#_Toc41330817)

[**1.1 Описание поставленной задачи** 4](#_Toc41330818)

[**1.2 Анализ исходных данных и их хранение** 4](#_Toc41330819)

[**1.3 Структура файлов для хранения данных** 4](#_Toc41330820)

[**1.4 Определение структур, классов, методов и функций** 4](#_Toc41330821)

[**1.5 Реализация архитектуры** 5](#_Toc41330822)

[**1.6 Средства для считывания и вывода данных** 6](#_Toc41330823)

[**Глава 2. Класс StudentWorker** 6](#_Toc41330824)

[**Глава 3. Класс SubjectWorker** 7](#_Toc41330825)

[**Глава 4. Класс MarksWorker** 8](#_Toc41330826)

[**Листинг программы** 10](#_Toc41330827)

[**Date.h** 10](#_Toc41330828)

[**Mark.h** 10](#_Toc41330829)

[**MarksWorker.h** 11](#_Toc41330830)

[**Student.h** 11](#_Toc41330831)

[**StudentsWorker.h** 12](#_Toc41330832)

[**Subject.h** 13](#_Toc41330833)

[**SubjectWorker.h** 13](#_Toc41330834)

[**Worker.h** 14](#_Toc41330835)

[**Date.cpp** 14](#_Toc41330836)

[**Mark.cpp** 15](#_Toc41330837)

[**MarksWorker.cpp** 17](#_Toc41330838)

[**Student.cpp** 18](#_Toc41330839)

[**Subject.cpp** 24](#_Toc41330840)

[**students.cpp** 27](#_Toc41330841)

[**Процесс работы программы** 34](#_Toc41330842)

[**Итог** 37](#_Toc41330843)

[**Вывод** 37](#_Toc41330844)

ВВЕДЕНИЕ

Задачей курсовой работы является разработка файловой системы, в ко-торой распределятся студенты на отличников, хорошистов, троечников за все время обучения, и каждая часть сортируется в алфавитном порядке.

Цель курсовой работы по дисциплине «Языки программирования» со-стоит в закреплении и углублении знаний и навыков, полученных при изуче-нии дисциплины. Курсовая работа предполагает выполнение задания повы-шенной сложности по проектированию, разработке и тестированию про-граммного обеспечения, а также оформлению сопутствующей документации.

**Глава 1. Разработка программного обеспечения**

**1.1 Описание поставленной задачи**

В данной работе передо мной была поставлена задача реализации консольного приложения для работы со студентами в рамках учебной программы. Необходимо реализовать работу с предметами и разработать систему выставления оценок. Данная модель должна быть полной, с возможностью изменения каждого объекта (удаление и создание нового объекта). В ходе работы были определены основные классы и методы для решения поставленной задачи.

**1.2 Анализ исходных данных и их хранение**

В данной задаче идет работа с множеством студентов, предметов и оценок. Все данные представляют из себя строковые, числовые данные, которые можно хранить различными способами. В результате разработки модели, было решено сделать программу для работы с объектами в системе, чтобы можно было удалять и добавлять новые элементы. Хранение данных было решено реализовать в виде текстовых файлов заданного формата. Студенты и предметы связываются посредством оценок, которые выставляют преподаватели.

**1.3 Структура файлов для хранения данных**

При решении поставленной задачи были выбраны текстовые файлы как хранилище для данных. В файле Students.txt хранятся данные о студентах, по одной записи на строку, данные внутри строки разделяются пробелами. Формат строки: ID Имя Фамилия Отчество Пол(m/f) Дата Рождения Дата регистрации Факультет номер группы номер зачетки.

В файле Subjects.txt хранятся данные о предметах, записи разделяются переводами строки. Формат записи: ID Имя Описание.

Оценки хранятся в файле Marks.txt, где на каждой строке указаны ID студента, которому выставлена оценка и ID предмета. Формат строки: ID ID\_Студента ID\_предмета.

**1.4 Определение структур, классов, методов и функций**

Для работы с данными описаны классы Worker, который наследуют основные классы для работы с данными:

1. StudentWorker.h
2. SubjectWorker.h
3. MarksWorker.h

Основные функции это Load() и Save(), которые позволяют считывать и загружать файлы в программу. Каждый класс выполняет логику работы с объектами, которая указана в задании.

Каждый объект в задаче содержит метод, позволяющие получить экземпляр класса из строки из текстовых файлов. Важно следить за тем, чтобы данные внутри файла были в заданном формате, иначе возникнет проблема с целостностью данных и будет невозможно считать их программой. Также каждый объект содержит в себе метод to\_string() который преобразует все поля класса в строковый эквивалент для хранения на диске.

В конструкторах классов Worker указывается путь к файлам, из которых необходимо считать данные для работы. Функция Save() для классов производит перезапись файлов данных по тому пути, по которому они были инициализированы.

**1.5 Реализация архитектуры**

В данной программе работа с предметами, студентами и оценками разделена на три отдельных модуля, для того чтобы их можно было независимо менять друг от друга. Классы для работы со студентами выполняют свои действия не обращаясь друг к другу, а класс выставления и хранения оценок работает лишь с ID студентов и предметов. Данный подход делает данную задачу легко модифицируемой.

**1.6 Средства для считывания и вывода данных**

Работа с файлами в поставленной задаче использует библиотеку fstream. Для каждого файла в соответствующем классе открывается поток ifstream для считывания и ofstream для записи.Для считывания строк из файла используется функция getline().

Fstream (сокращение от «FileStream») — заголовочный файл из стандартной библиотеки C++, включающий набор классов, методов и функций, которые предоставляют интерфейс для чтения/записи данных из/в файл.

**Глава 2. Класс StudentWorker**

В классе StudentWorker описана логика работы со студентами. Путь, используемый для хранения адреса файла, из которого происходит считывание хранится в отдельной приватной переменной типа string и изменяется только при инициализации класса.

Данные о студентах хранятся в приватном поле stud, представляющий из себя vector, содержащий классы студентов. При считывании данных их файла каждый элемент множества инициализируется из считанной строки из файла. После выполнения функции Load() в массиве stud хранятся все данные для студентов. Это поле является приватным, так как некоторые данные могут быть конфиденциальны и обращение к ним недопустимо. Для доступа и редактирования данных о студентах реализованы методы:

Student\* getStudentById(int id) — позволяет получить ссылку на объект класса студента, по заданному id. Возвращает NULL, если студент с таким идентификатором не найден.

getStudentByRecordBook) — позволяет получить ссылку на объект класса студента, по заданному номеру зачетной книжки. Возвращает NULL, если студент с таким номером зачетной книжки не найден.

int AddStudent(Student s) — добавляет студента в программу и возвращает id студента если добавление прошло успешно. Если по какой-то причине процесс не удался, функция вернет -1.

void DeleteStudentById(int id) — удаляет студента по заданному id.

bool existId(int id) — функция проверяет наличие студента в списках и возвращает true если искомый id присутствует.

int Count() — возвращает текущее количество студентов с списке.

vector<string> studNames(char g) — возвращает список имен студентов заданного пола.

vector<string> getInfo(char g) — возвращает список полной информации о студентах заданного пола.

vector<int> getAllId(char g) — возвращает список ID студентов заданного пола.

В последних трех функциях в качестве аргумента передается пол, для выбора студентов: m – мужской, f – женский, b – оба пола.

Все функции представлены в листинге программы в файле StudentWorker.cpp.

**Глава 3. Класс SubjectWorker**

В классе SubjectWorker описана логика работы с предметами. Путь, используемый для хранения адреса файла, из которого происходит считывание хранится в отдельной приватной переменной типа string и изменяется только при инициализации класса.

Данные о предметах хранятся в приватном поле subjects, представляющий из себя vector, содержащий классы предметов. При считывании данных их файла каждый элемент множества инициализируется из считанной строки из файла. После выполнения функции Load() в массиве subjects хранятся все данные о предметах. Это поле является приватным, так как некоторые данные могут быть конфиденциальны и обращение к ним недопустимо. Для доступа и редактирования данных о предметах реализованы методы:

Subject\* getSubjectById(int id) — возвращает ссылку на экземпляр класса предмета, по если идентификатору, возвращает NULL, если предмет с таким ID не найден.

int AddSubject(Subject s) — добавляет новый предмет в список. Возвращает ID, если предмет успешно добавился, иначе возвращает -1.

void DeleteSubjectById(int id) — удаляет предмет по указанному идентификатору.

bool existId(int id) — возвращает true или false, в зависимости от того, присутствует ли предмет с идентификатором id в списке.

int Count() — возвращает текущее количество предметов в списке.

string\* subjectNames() — возвращает список названий предметов без описания.

string\* getInfo()— возвращает список названий предметов с описанием.

Все функции представлены в листинге программы в файле StudentWorker.cpp.

**Глава 4. Класс MarksWorker**

В классе MarksWorker описана логика работы с оценками в системе. Путь, используемый для хранения адреса файла, из которого происходит считывание хранится в отдельной приватной переменной типа string и изменяется только при инициализации класса.

Данные об оценках хранятся в приватном поле marks, представляющий из себя vector, содержащий классы оценок. При считывании данных их файла каждый элемент множества инициализируется из считанной строки из файла. После выполнения функции Load() в массиве marks хранятся все данные об оценках. Это поле является приватным, так как некоторые данные могут быть конфиденциальны и обращение к ним недопустимо. Для доступа и редактирования данных о предметах реализованы методы:

Mark\* getMarkById(int id) — возвращает ссылку на экземпляр класса оценки, по если идентификатору, возвращает NULL, если предмет с таким ID не найден.

int AddMark(Mark s) — добавляет новую оценку в список. Возвращает ID, если предмет успешно добавился, иначе возвращает -1.

void DeleteMarkById(int id) — удаляет из списка оценку с указанным идентификатором.

bool existId(int id) — проверяет, присутствует ли оценка с заданным id в спискею.

int Count() — возвращает текущее количество оценок в системе.

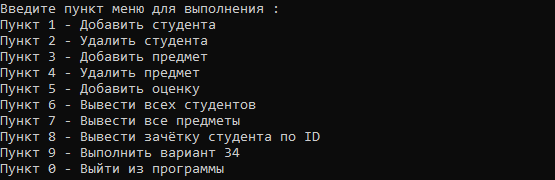
string\* getInfo() — возвращает массив строк с данным и о оценках в строчноп представлении.

vector<int>\* getMarksForStudentSubj(int studId, int SubjectId) — возвращает список оценок для указанного студента по указанному предмету. Входные параметры это ID студента и предмета соответственно.

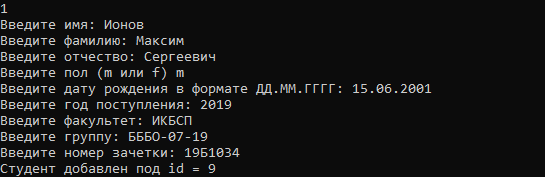
vector<int> getMarksForStudent(int studId) — возвращает список оценок для указанного студента.

**Процесс работы программы**

При запуске программы пользователю выводится меню

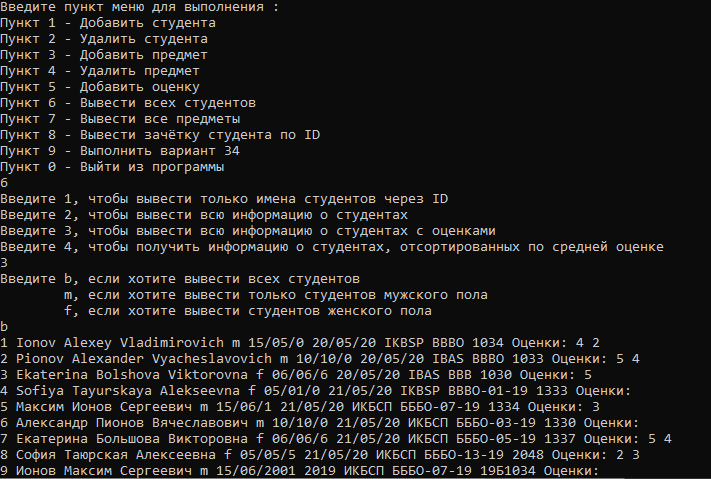


Пользователь должен выбрать пункт меню. При выборе пункта 1, пользователю предоставляется возможность ввести студентов с клавиатуры.



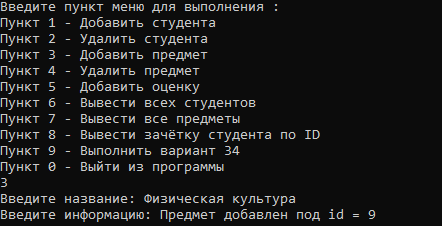
Вводятся данные о студенте: его ФИО, пол (m(male) или f(female)), дата рождения студента, год поступления, факультет, группа и номер зачётки. Студенту автоматически присваивается ID.

Запустив пункт 6, мы можем увидеть то, что студент был добавлен в файл Students.txt



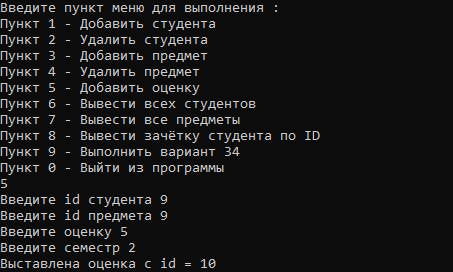
Как можно заметить, что под ID 9 у нас находится введённый пользователем студент, а также всё информация о нём. Так как оценки мы не добавляли, то их в консоли не выводит.

Оценки выставляются по предметам, поэтому нам нужно добавить предмет через пункт 3.



Пользователь должен ввести название предмета и ему автоматически выдаётся ID, в нашем случае это ID равно 9.

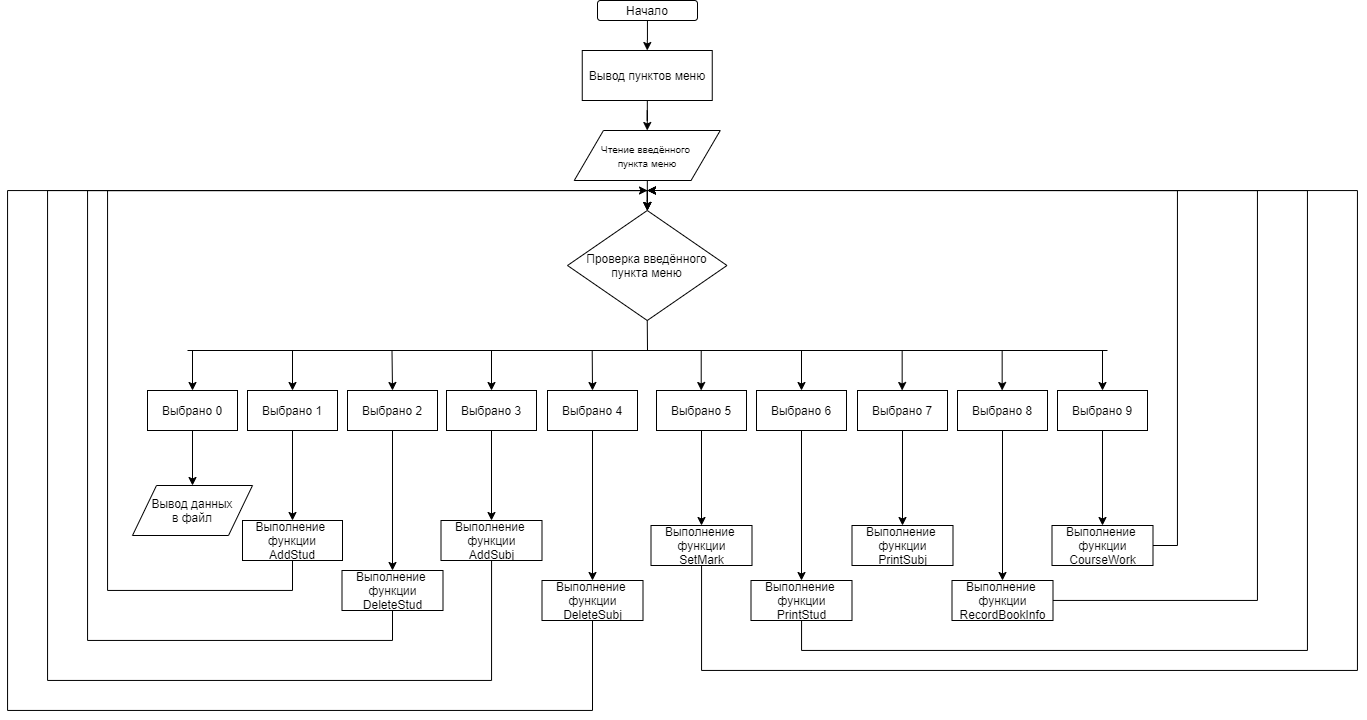
У нас есть предмет и мы можем теперь добавить оценку студенту по этому предмету. Выбираем пункт 5.



Программа запрашивает ID студента, ID предмета по которому нужно выставить оценку, оценку за этот предмет и семестр. После ввода данных создаётся оценка под собственным ID.

Выше был описан полный процесс добавления и выставления оценки с добавлением студента.

**Блок схема работы меню**



**Итог**

В файле students.cpp представлена реализация поставленной задачи в виде консольного приложения. В большинстве функций идет описание работы с пользователем. При запуске программы классы для работы со студентами, оценками и предметами инициализируются строками, где хранятся данные. Затем выводится меню в котором описываются возможные действия пользователя.

При выводе информации о студентах возможно указать пол студентов, по которому необходимо отфильтровать список (m, f, b).

Когда пользователь просит вывести всех студентов по их среднему баллу, происходит вычисление суммы всех баллов пользователя, затем эти данные сортируются и выводятся пользователю.

Интерфейс программы зациклен и каждый раз ожидает ввод от пользователя, в зависимости от того уровня меню, где он находится.

Сохранение данных в файл после модификации списков студентов, предметов или выставленных оценок происходит только при вводе соответствующей позиции в меню.

Все ошибки ввода обрабатываются и в случае неудачи предоставляют возможность ввести команду еще раз.

**Вывод**

В процессе выполнения курсовой работы была написана программа сортировки в алфавитном порядке всех студентов определенного пола на три группы: хорошистов, отличников и троечников. Программа была реализована c использованием технологии ООП.

**Листинг программы**

**Date.h**

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Date {

public:

Date();

Date(int d, int m, int y);

Date(const Date& date);

Date(string s);

int getDay() const;

int getMonth() const;

int getYear() const;

void setDay(int day);

void setMonth(int month);

void setYear(int year);

void setDate(int day, int month, int year);

string ToString();

private:

int day;

int month;

int year;

};

**Mark.h**

#pragma once

#include <string>

using namespace std;

class Mark

{

private:

int Id;

int Value;

int StudentId;

int SubjectId;

int term;

public:

Mark(int id, int v, int stdid, int subjid, int trm);

Mark(string s);

Mark(const Mark& m);

int getId();

int getValue();

int getTerm();

int getStudentId();

int getSubjectId();

void setId(int id);

void setValue(int v);

void setStudentId(int stdid);

void setSubjectId(int subjid);

void setTurm(int tem);

void setMark(int id, int v, int stdid, int subjid, int term);

string ToString();

};

**MarksWorker.h**

#pragma once

#include <string>

#include <vector>

#include <fstream>

#include "Worker.h"

#include "Mark.h"

using namespace std;

class MarksWorker : Worker

{

public:

MarksWorker(string p = "");

void Load();

void Save();

Mark\* getMarkById(int id);

int AddMark(Mark s);

void DeleteMarkById(int id);

bool existId(int id);

int Count();

string\* getInfo();

vector<int> getMarksForStudentSubj(int studId, int SubjectId, int term = 0);

vector<int> getMarksForStudent(int studId, int term = 0);

private:

string path;

vector<Mark> marks;

};

**Student.h**

#pragma once

#include <string>

#include <regex>

#include "Date.h"

using namespace std;

class Student

{

private:

int Id;

string Name;

string SecName;

string Patronymic;

char Gender;

Date DayOfBirth;

string YearStart;

string Faculty;

string Group;

string RecordBook;

public:

Student();

Student(int id, string name, string secname, string patro, char g, Date dayofbirth, string yearstart,

string faculttm, string group, string recbook);

Student(const Student& s);

Student(string s);

int getId();

string getName();

string getSecName();

string getPatronymic();

Date getDayOfBirth();

string getYearStart();

string getRecordBook();

string getFaculty();

string getGroup();

char getGender();

void setId(int id);

void setName(string name);

void setSecName(string secname);

void setPatronymic(string patro);

void setDayOfBirth(Date date);

void setYearStart(string date);

void setRecordBook(string r);

void setFaculty(string f);

void setGroup(string g);

void setGender(char c);

void setStudent(int id, string name, string secname, string patro, char g, Date dayofbirth, string yearstart,

string faculttm, string group, string recbook);

string getInfo();

string ToString();

};

**StudentsWorker.h**

#include <iostream>

#include "Student.h"

#include "Worker.h"

#include <vector>

#include <fstream>

#pragma once

class StudentsWorker : Worker

{

public:

StudentsWorker(string p = "");

void Load();

void Save();

Student\* getStudentById(int id);

Student\* getStudentByRecordBook(string rec);

int AddStudent(Student s);

void DeleteStudentById(int id);

bool existId(int id);

int Count();

vector<string> studNames(char g);

vector<string> getInfo(char g);

vector<int> getAllId(char g);

private:

string path;

vector<Student> stud;

};

**Subject.h**

#include <string>

#pragma once

using namespace std;

class Subject

{

public:

Subject(int id, string name, string info);

Subject(string s);

Subject(const Subject& s);

string getInfo();

string getName();

int getId();

void setName(string name);

void setInfo(string info);

void setId(int id);

void setSubject(int id, string name, string info);

string ToString();

private:

int Id;

string Name;

string Info;

};

**SubjectWorker.h**

#include <iostream>

#include "Subject.h"

#include "Worker.h"

#include <vector>

#include <fstream>

#pragma once

class SubjectWorker : Worker

{

public:

SubjectWorker(string p = "");

void Load();

void Save();

Subject\* getSubjectById(int id);

int AddSubject(Subject s);

void DeleteSubjectById(int id);

bool existId(int id);

int Count();

string\* subjectNames();

string\* getInfo();

vector<int> getSubjectsId();

private:

string path;

vector<Subject> subjects;

};

**Worker.h**

#pragma once

class Worker

{

public:

virtual void Load() = 0;

virtual void Save() = 0;

};

**Date.cpp**

#include "Date.h"

Date::Date() {

setDate(1, 1, 1970);

}

Date::Date(int d, int m, int y) {

setDate(d, m, y);

}

Date::Date(const Date& date) {

setDate(date.day, date.month, date.year);

}

Date::Date(string s)

{

setDate(stoi(s.substr(0, 2)), stoi(s.substr(3, 2)), stoi(s.substr(6, 4)));

}

int Date::getDay() const {

return day;

}

int Date::getMonth() const {

return month;

}

int Date::getYear() const {

return year;

}

void Date::setDay(int day) {

day = day;

}

void Date::setMonth(int month) {

month = month;

}

void Date::setYear(int year) {

year = year;

}

void Date::setDate(int d, int m, int y) {

day = d;

month = m;

year = y;

}

string Date::ToString()

{

day %= 100;

month %= 100;

string temp = "";

if (day < 9) {

temp += "0";

}

temp += to\_string(day) + "/";

if (month < 9) {

temp += "0";

}

temp += to\_string(month) + "/";

temp += to\_string(year);

return temp;

}

**Mark.cpp**

#include "Mark.h"

Mark::Mark(int id, int v, int stdid, int subjid, int trm)

{

setMark(id, v, stdid, subjid, trm);

}

Mark::Mark(string s)

{

if (s[s.size() - 1] != ' ') {

s += " ";

}

int last = 0;

int c = 0;

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

if (s[i] == ' ') {

string p = s.substr(last, i - last);

last = i + 1;

switch (c)

{

case 0:

setId(stoi(p));

break;

case 1:

setValue(stoi(p));

break;

case 2:

setStudentId(stoi(p));

break;

case 3:

setSubjectId(stoi(p));

break;

case 4:

setTurm(stoi(p));

break;

default:

break;

}

c++;

}

}

}

Mark::Mark(const Mark& m)

{

setMark(m.Id, m.Value, m.StudentId, m.SubjectId, m.term);

}

int Mark::getId()

{

return Id;

}

int Mark::getValue()

{

return Value;

}

int Mark::getTerm()

{

return term;

}

int Mark::getStudentId()

{

return StudentId;

}

int Mark::getSubjectId()

{

return SubjectId;

}

void Mark::setId(int id)

{

Id = id;

}

void Mark::setValue(int v)

{

Value = v;

}

void Mark::setStudentId(int stdid)

{

StudentId = stdid;

}

void Mark::setSubjectId(int subjid)

{

SubjectId = subjid;

}

void Mark::setTurm(int tem)

{

term = tem;

}

void Mark::setMark(int id, int v, int stdid, int subjid, int trm)

{

setId(id);

setValue(v);

setStudentId(stdid);

setSubjectId(subjid);

setTurm(trm);

}

string Mark::ToString()

{

return to\_string(Id) + " " + to\_string(Value) + " " + to\_string(StudentId) + " " + to\_string(SubjectId) + " " + to\_string(term);

}

**MarksWorker.cpp**

#include "MarksWorker.h"

MarksWorker::MarksWorker(string p)

{

path = p;

}

void MarksWorker::Load()

{

ifstream in(path);

string q;

while (getline(in, q)) {

marks.emplace\_back(Mark(q));

}

in.close();

}

void MarksWorker::Save()

{

ofstream out(path);

for (auto i : marks) {

out << i.ToString() << endl;

}

}

Mark\* MarksWorker::getMarkById(int id)

{

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

if (marks[i].getId() == id) {

return &marks[i];

}

}

return NULL;

}

int MarksWorker::AddMark(Mark s)

{

if (s.getId() == -1) {

int mx = 0;

for (auto i : marks) {

if (i.getId() > mx) {

mx = i.getId();

}

}

s.setId(mx + 1);

marks.emplace\_back(s);

return mx + 1;

}

else {

marks.emplace\_back(s);

return s.getId();

}

}

void MarksWorker::DeleteMarkById(int id)

{

if (existId(id)) {

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

if (marks[i].getId() == id) {

marks.erase(marks.begin() + i);

return;

}

}

}

}

bool MarksWorker::existId(int id)

{

for (auto i : marks) {

if (i.getId() == id) {

return true;

}

}

return false;

}

int MarksWorker::Count()

{

return marks.size();

}

string\* MarksWorker::getInfo()

{

string\* a = new string[marks.size()];

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

a[i] = marks[i].ToString();

}

return a;

}

vector<int> MarksWorker::getMarksForStudentSubj(int studid, int subjectId, int term)

{

vector<int> m;

for (auto i : marks) {

if (i.getStudentId() == studid && i.getSubjectId() == subjectId && (term == 0 || i.getTerm() == term)) {

m.emplace\_back(i.getValue());

}

}

return m;

}

vector<int> MarksWorker::getMarksForStudent(int studId, int term)

{

vector<int> m;

for (auto i : marks) {

if (i.getStudentId() == studId && (term == 0 || i.getTerm() == term)) {

m.emplace\_back(i.getValue());

}

}

return m;

}

**Student.cpp**

class Date;

#include "Student.h"

#include "Date.h"

Student::Student()

{

Name = string();

SecName = string();

Patronymic = string();

DayOfBirth = Date();

YearStart = string();

Faculty = string();

Group = string();

RecordBook = string();

}

Student::Student(int id, string name, string secname, string patro, char g, Date dayofbirth,

string yearstart, string faculttm, string group, string recbook)

{

setStudent(id, name, secname, patro, g, dayofbirth, yearstart, faculttm, group, recbook);

}

Student::Student(const Student& s)

{

setStudent(s.Id, s.Name, s.SecName, s.Patronymic, s.Gender, s.DayOfBirth, s.YearStart, s.Faculty, s.Group, s.RecordBook);

}

Student::Student(string s)

{

if (s[s.size() - 1] != ' ') {

s += " ";

}

int last = 0;

int c = 0;

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

if (s[i] == ' ') {

string p = s.substr(last, i - last);

last = i + 1;

switch (c)

{

case 0:

setId(stoi(p));

break;

case 1:

setName(p);

break;

case 2:

setSecName(p);

break;

case 3:

setPatronymic(p);

break;

case 4:

setGender(p[0]);

break;

case 5:

setDayOfBirth(Date(p));

break;

case 6:

setYearStart(p);

break;

case 7:

setFaculty(p);

break;

case 8:

setGroup(p);

break;

case 9:

setRecordBook(p);

break;

default:

break;

}

c++;

}

}

}

int Student::getId() {

return Id;

}

string Student::getName() {

return Name;

}

string Student::getSecName() {

return SecName;

}

string Student::getPatronymic() {

return Patronymic;

}

Date Student::getDayOfBirth()

{

return DayOfBirth;

}

string Student::getYearStart()

{

return YearStart;

}

string Student::getRecordBook()

{

return RecordBook;

}

string Student::getFaculty()

{

return Faculty;

}

string Student::getGroup()

{

return Group;

}

char Student::getGender()

{

return Gender;

}

void Student::setId(int id)

{

Id = id;

}

void Student::setName(string name)

{

Name = name;

}

void Student::setSecName(string secname)

{

SecName = secname;

}

void Student::setPatronymic(string patro)

{

Patronymic = patro;

}

void Student::setDayOfBirth(Date date)

{

DayOfBirth = date;

}

void Student::setYearStart(string date)

{

YearStart = date;

}

void Student::setRecordBook(string r)

{

RecordBook = r;

}

void Student::setFaculty(string f)

{

Faculty = f;

}

void Student::setGroup(string g)

{

Group = g;

}

void Student::setGender(char c)

{

Gender = c;

}

void Student::setStudent(int id, string name, string secname, string patro, char g, Date dayofbirth,

string yearstart, string faculttm, string group, string recbook)

{

setId(id);

setName(name);

setSecName(secname);

setPatronymic(patro);

setGender(g);

getDayOfBirth();

getYearStart();

setDayOfBirth(dayofbirth);

setYearStart(yearstart);

setRecordBook(recbook);

setFaculty(faculttm);

setGroup(group);

}

string Student::getInfo()

{

return Name + " " + SecName + " " + Patronymic;

}

string Student::ToString()

{

return to\_string(Id) + " " + Name +

" " + SecName + " " + Patronymic + " " + Gender +

" " + DayOfBirth.ToString() + " " +

YearStart + " " + Faculty +

" " + Group + " " + RecordBook;

}

**StudentsWorker.cpp**

#include "StudentsWorker.h"

StudentsWorker::StudentsWorker(string p)

{

path = p;

stud.clear();

}

void StudentsWorker::Load()

{

ifstream in(path);

string q;

while (getline(in, q)) {

stud.emplace\_back(Student(q));

}

in.close();

}

void StudentsWorker::Save()

{

ofstream out(path);

for (auto i : stud) {

out << i.ToString() << endl;

}

}

Student\* StudentsWorker::getStudentById(int id)

{

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

if (stud[i].getId() == id) {

return &stud[i];

}

}

return NULL;

}

Student\* StudentsWorker::getStudentByRecordBook(string record)

{

for (auto i : stud) {

if (i.getRecordBook() == record) {

return &i;

}

}

return NULL;

}

int StudentsWorker::AddStudent(Student s)

{

if (s.getId() == -1) {

int mx = 0;

for (auto i : stud) {

mx = std::max(mx, i.getId());

}

s.setId(mx + 1);

stud.emplace\_back(s);

return s.getId();

}

else {

stud.emplace\_back(s);

return s.getId();

}

}

void StudentsWorker::DeleteStudentById(int id)

{

if (existId(id)) {

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

if (stud[i].getId() == id) {

stud.erase(stud.begin() + i);

return;

}

}

}

}

int StudentsWorker::Count()

{

return stud.size();

}

vector<string> StudentsWorker::studNames(char g)

{

vector<string> a;

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

if (g == 'b') {

a.push\_back(to\_string(stud[i].getId()) + " " + stud[i].getName());

}

if (g == 'm' && stud[i].getGender() == 'm') {

a.push\_back(to\_string(stud[i].getId()) + " " + stud[i].getName());

}

if (g == 'f' && stud[i].getGender() == 'f') {

a.push\_back(to\_string(stud[i].getId()) + " " + stud[i].getName());

}

}

return a;

}

vector<string> StudentsWorker::getInfo(char g)

{

vector<string> a;

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

if (g == 'b') {

a.emplace\_back(stud[i].ToString());

}

if (g == 'm' && stud[i].getGender() == 'm') {

a.emplace\_back(stud[i].ToString());

}

if (g == 'f' && stud[i].getGender() == 'f') {

a.emplace\_back(stud[i].ToString());

}

}

return a;

}

vector<int> StudentsWorker::getAllId(char g)

{

vector<int> r;

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

if (g == 'b') {

r.emplace\_back(stud[i].getId());

}

if (g == 'm' && stud[i].getGender() == g) {

r.emplace\_back(stud[i].getId());

}

if (g == 'f' && stud[i].getGender() == g) {

r.emplace\_back(stud[i].getId());

}

}

return r;

}

bool StudentsWorker::existId(int id) {

for (auto i : stud) {

if (i.getId() == id) {

return true;

}

}

return false;

}

**Subject.cpp**

#include "Subject.h"

Subject::Subject(int id, string name, string info)

{

setSubject(id, name, info);

}

Subject::Subject(string s)

{

if (s[s.size() - 1] != ' ') {

s += " ";

}

int last = 0;

int c = 0;

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

if (s[i] == ' ') {

string p = s.substr(last, i - last);

last = i + 1;

switch (c)

{

case 0:

setId(stoi(p));

break;

case 1:

setName(p);

break;

case 2:

setInfo(p);

break;

default:

break;

}

c++;

}

}

}

Subject::Subject(const Subject& s)

{

setSubject(s.Id, s.Name, s.Info);

}

string Subject::getInfo()

{

return Info;

}

string Subject::getName()

{

return Name;

}

int Subject::getId()

{

return Id;

}

void Subject::setName(string name)

{

Name = name;

}

void Subject::setInfo(string info)

{

Info = info;

}

void Subject::setId(int id)

{

Id = id;

}

void Subject::setSubject(int id, string name, string info)

{

setId(id);

setName(name);

setInfo(info);

}

string Subject::ToString()

{

return to\_string(Id) + " " + Name + " " + Info;

}

**SubjectWorker.cpp**

#include "SubjectWorker.h"

SubjectWorker::SubjectWorker(string p)

{

path = p;

}

void SubjectWorker::Load()

{

ifstream in(path);

string q;

while (getline(in, q)) {

subjects.emplace\_back(Subject(q));

}

in.close();

}

void SubjectWorker::Save()

{

ofstream out(path);

for (auto i : subjects) {

out << i.ToString() << endl;

}

}

Subject\* SubjectWorker::getSubjectById(int id)

{

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

if (subjects[i].getId() == id) {

return &subjects[i];

}

}

return NULL;

}

int SubjectWorker::AddSubject(Subject s)

{

if (s.getId() == -1) {

int mx = 0;

for (auto i : subjects) {

if (i.getId() > mx) {

mx = i.getId();

}

}

s.setId(mx + 1);

subjects.emplace\_back(s);

return mx + 1;

}

else {

subjects.emplace\_back(s);

return s.getId();

}

}

void SubjectWorker::DeleteSubjectById(int id)

{

if (existId(id)) {

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

if (subjects[i].getId() == id) {

subjects.erase(subjects.begin() + i);

return;

}

}

}

}

bool SubjectWorker::existId(int id)

{

for (auto i : subjects) {

if (i.getId() == id) {

return true;

}

}

return false;

}

int SubjectWorker::Count()

{

return subjects.size();

}

string\* SubjectWorker::subjectNames()

{

string\* a = new string[subjects.size()];

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

a[i] = to\_string(subjects[i].getId()) + " " + subjects[i].getName();

}

return a;

}

string\* SubjectWorker::getInfo()

{

string\* a = new string[subjects.size()];

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

a[i] = subjects[i].ToString();

}

return a;

}

vector<int> SubjectWorker::getSubjectsId()

{

vector<int> a(subjects.size());

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

a[i] = subjects[i].getId();

}

return a;

}

**students.cpp**

#include <iostream>

#include <time.h>

#include <string>

#include <windows.h>;

#include "StudentsWorker.h"

#include "SubjectWorker.h"

#include "MarksWorker.h"

using namespace std;

string getRecordBook(int id, StudentsWorker\* stdw, SubjectWorker\* sw, MarksWorker\* mw)

{

string result = stdw->getStudentById(id)->getInfo() + "\n";

result += "Факультет " + stdw->getStudentById(id)->getFaculty() + "\n";

result += "Зачетная книжка " + stdw->getStudentById(id)->getRecordBook() + "\n";

result += "Год поступления " + stdw->getStudentById(id)->getYearStart() + "\n";

auto subjects = sw->getSubjectsId();

bool existenything = 0;

for (int term = 1; term <= 6; term++) {

string r = "Marks in " + to\_string(term) + " term:\n";

bool exist = 0;

for (int i = 0; i < sw->Count(); i++) {

auto marks = mw->getMarksForStudentSubj(id, subjects[i], term);

if (marks.size() > 0) {

r += sw->getSubjectById(subjects[i])->getName() + ": ";

exist = 1;

for (auto j : marks) {

r += to\_string(j) + " ";

}

r += "\n";

}

}

if (exist) {

result += r;

existenything = 1;

}

}

if (existenything) {

return result;

}

else {

return result + "У студента нет оценок";

}

}

//Проверка строки на правильность ввода (пустая строка)

bool enterString(string\* s, string text) {

cout << text << ": ";

cin >> \*s;

if (s->size() == 0) {

cout << "Ошибка ввода, повторите ввод: ";

cin >> \*s;

if (s->size() == 0) {

cout << "Ошибка ввода\n";

return false;

}

}

return true;

}

//Проверка правильности ввода даты

bool checkDate(string str) {

static std::regex rgx{ R"((0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[- /.](19|20\d\d))" };

std::smatch res;

if (std::regex\_search(str, res, rgx))

{

return true;

}

return false;

}

//Вывод меню

void menu() {

cout << "Введите пункт меню для выполнения :\n";

cout << "Пункт 1 - Добавить студента\n";

cout << "Пункт 2 - Удалить студента\n";

cout << "Пункт 3 - Добавить предмет\n";

cout << "Пункт 4 - Удалить предмет\n";

cout << "Пункт 5 - Добавить оценку\n";

cout << "Пункт 6 - Вывести всех студентов\n";

cout << "Пункт 7 - Вывести все предметы\n";

cout << "Пункт 8 - Вывести зачётку студента\n";

cout << "Пункт 9 - Выполнить вариант 34\n";

cout << "Пункт 0 - Выйти из программы\n";

}

void menuPrintStud() {

cout << "Введите 1, чтобы вывести только имена студентов через ID\n";

cout << "Введите 2, чтобы вывести всю информацию о студентах\n";

cout << "Введите 3, чтобы вывести всю информацию о студентах с оценками\n";

cout << "Введите 4, чтобы получить информацию о студентах, отсортированных по средней оценке\n";

}

void menuCourse() {

cout << "Введите M, чтобы вывести студентов без троек мужского пола\n";

cout << "Введите F, чтобы вывести студентов без троек женского пола\n";

}

void menuPrintSubj() {

cout << "Введите 1, чтобы вывести только названия предметов с их ID\n";

cout << "Введите 2, чтобы вывести всю информацию о предметах\n";

}

//Выставление оценки

void SetMark(StudentsWorker\* sw, SubjectWorker\* sbw, MarksWorker \* mw) {

int stdid, sbjid, val, id, term;

cout << "Введите id студента ";

cin >> stdid;

if (!sw->existId(stdid)) {

cout << "Нет такого студента" << endl;

return;

}

cout << "Введите id предмета ";

cin >> sbjid;

if (!sbw->existId(sbjid)) {

cout << "Нет такого предмета" << endl;

return;

}

cout << "Введите оценку ";

cin >> val;

cout << "Введите семестр ";

cin >> term;

id = mw->AddMark(Mark(-1, val, stdid, sbjid, term));

cout << "Выставлена оценка с id = " << id << endl;

}

// Процедура вывода информации о предметах

void PrintSubj(StudentsWorker \* sw, SubjectWorker \* sbw, MarksWorker\* mw) {

int comand;

string\* list;

menuPrintSubj();

cin >> comand;

switch (comand)

{

case 1:

list = sbw->subjectNames();

for (int i = 0; i < sbw->Count(); i++) {

cout << list[i] << endl;

}

break;

case 2:

list = sbw->getInfo();

for (int i = 0; i < sbw->Count(); i++) {

cout << list[i] << endl;

}

break;

default:

break;

}

}

// Добавление нового студента

void AddStud(StudentsWorker \* sw) {

string name, scname, patro, dob, dor, fac, group, rb, id;

char g;

enterString(&name, "Введите имя");

enterString(&scname, "Введите фамилию");

enterString(&patro, "Введите отчество");

cout << "Введите пол (m или f) ";

cin >> g;

if (g != 'm' && g != 'f') {

cout << "Ошибка ввода, повторите ввод (m или f): " << endl;

cin >> g;

if (g != 'm' && g != 'f') {

cout << "Ошибка ввода\n";

return;

}

}

cout << "Введите дату рождения в формате ДД.ММ.ГГГГ: ";

cin >> dob;

if (!checkDate(dob)) {

cout << "Ошибка ввода, повторите ввод: ";

cin >> dob;

if (!checkDate(dob)) {

"Ошибка ввода\n";

return;

}

}

enterString(&dor, "Введите год поступления");

enterString(&fac, "Введите факультет");

enterString(&group, "Введите группу");

enterString(&rb, "Введите номер зачетки");

id = sw->AddStudent(Student(-1, name, scname, patro, g, Date(dob), dor, fac, group, rb));

cout << "Студент добавлен под id = " << sw->getStudentByRecordBook(rb)->getId() << endl;

}

// Удаление студента

void DeleteStud(StudentsWorker \* sw) {

int id;

cout << "Введите id студента для удаления ";

cin >> id;

if (sw->existId(id)) {

sw->DeleteStudentById(id);

}

else {

cout << "Нет студента с таким id";

}

}

// Вывод информации о студентах

void PrintStud(StudentsWorker \* sw, SubjectWorker \* sbw, MarksWorker\* mw) {

vector<string> list;

char g = 'b';

vector<int> m, ids;

vector<pair<int, string> > a;

int comand;

menuPrintStud();

cin >> comand;

cout << "Введите b, если хотите вывести всех студентов\n m, если хотите вывести только студентов мужского пола\n f, если хотите вывести студентов женского пола\n";

cin >> g;

if (g != 'b' && g != 'f' && g != 'm') {

cout << "Ошибка ввода";

return;

}

switch (comand)

{

case 1:

list = sw->studNames(g);

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

cout << list[i] << endl;

}

break;

case 2:

list = sw->getInfo(g);

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

cout << list[i] << endl;

}

break;

case 3:

ids = sw->getAllId(g);

list = sw->getInfo(g);

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

cout << list[i] << " Оценки: ";

m = mw->getMarksForStudent(ids[i]);

for (auto j : m) {

cout << j << ' ';

}

cout << endl;

}

break;

case 4:

int term;

cout << "Введите номер семестра или 0, если интересуют все семестры: ";

cin >> term;

if (term < 0 || term > 6) {

cout << "Ошибка ввода";

return;

}

ids = sw->getAllId(g);

list = sw->getInfo(g);

a.resize(list.size());

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

a[i].second = sw->getStudentById(ids[i])->getInfo();

m = mw->getMarksForStudent(ids[i], term);

double sum = 0;

for (auto j : m) {

sum += j;

}

if (m.size() == 0) {

a[i].first = -1;

}

else {

a[i].first = sum / m.size();

}

}

sort(a.begin(), a.end());

reverse(a.begin(), a.end());

for (auto i : a) {

if (i.first != -1) {

cout << i.second << " Средний балл: " << i.first << endl;

}

}

break;

default:

break;

}

}

// Добавление предмета

void AddSubj(SubjectWorker \* sbw) {

string name, info;

enterString(&name, "Введите название");

enterString(&info, "Введите информацию");

int id = sbw->AddSubject(Subject(-1, name, info));

cout << "Предмет добавлен под id = " << id << endl;

}

// Удаление предмета

void DeleteSubj(SubjectWorker \* sbw) {

int id;

cout << "Введите id предмета для удаления ";

cin >> id;

if (sbw->existId(id)) {

sbw->DeleteSubjectById(id);

}

else {

cout << "Нет предмета с таким id\n";

}

}

void RecordBookInfo(StudentsWorker\* sw, SubjectWorker\* sbw, MarksWorker\* mw) {

int id;

cout << "Введите id студнета ";

cin >> id;

if (sw->existId(id)) {

cout << getRecordBook(id, sw, sbw, mw) << endl;

}

else {

cout << "Нет такого студента, попробуйте снова: ";

cin >> id;

if (sw->existId(id)) {

cout << getRecordBook(id, sw, sbw, mw) << endl;

}

else {

cout << "Ошибка ввода";

return;

}

}

}

//выполнение варианта 34

void CourseWork(StudentsWorker\* sw, SubjectWorker\* sbw, MarksWorker\* mw) {

vector<string> list;

char g = 'b';

vector<int> m, ids;

menuCourse();

cin >> g;

if (g != 'b' && g != 'f' && g != 'm' && g != 'м' && g != 'ж') {

cout << "Ошибка ввода";

return;

}

else {

ids = sw->getAllId(g);

list = sw->getInfo(g);

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

int block = 0;

m = mw->getMarksForStudent(ids[i]);

for (auto j : m)

{

if (j <= 3)

{

block++;

}

}

if (block != 0)

{

}

else {

cout << list[i] << " Оценки: ";

for (auto j : m) {

cout << j << ' ';

}

}

cout << endl;

}

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

menu();

StudentsWorker sw("Students.txt");

SubjectWorker sbw("Subjects.txt");

MarksWorker mw("Marks.txt");

sw.Load();

sbw.Load();

mw.Load();

int id, comand;

int c;

cin >> c;

while (1) {

switch (c)

{

case 1:

AddStud(&sw);

break;

case 2:

DeleteStud(&sw);

break;

case 3:

AddSubj(&sbw);

break;

case 4:

DeleteSubj(&sbw);

break;

case 5:

SetMark(&sw, &sbw, &mw);

break;

case 6:

PrintStud(&sw, &sbw, &mw);

break;

case 7:

PrintSubj(&sw, &sbw, &mw);

break;

case 8:

RecordBookInfo(&sw, &sbw, &mw);

break;

case 9:

CourseWork(&sw, &sbw, &mw);

break;

case 0:

sw.Save();

sbw.Save();

mw.Save();

return 0;

break;

default:

cout << "Неправильная команда, повторите ввод";

break;

}

menu();

cin >> c;

}

}