МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ “ХПІ”

Кафедра “Обчислювальна техніка та програмування”

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

1КІТ.102.8А. 18037-01 81 01-1 –АЗ

Розробник

Виконав:

студент групи 1КІТ-102.8А

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Кононенко Д. О../

Перевірив:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Старший викладач. Молчанов Г.І./

Харків 2019

ЗАТВЕРДЖЕНО

1КІТ102.8А.18037-01 81 01-1 –АЗ

Розрахункове завдання з дисципліни

«Алгоритми та структури даних»

Пояснювальна записка

1КІТ.102.8А.18037-01 81 01-1 -АЗ

Листів

Харків 2019

РОЗРАХУНКОВОГО ЗАВДАННЯ З ДИСЦИПЛІНИ

«ПРОГРАМУВАННЯ»

*Тема роботи.* Розробка інформаційно-довідкової системи.

*Мета роботи.* Закріпити отримані знання з дисципліни «Програмування» шляхом виконання типового комплексного завдання.

1 ВИМОГИ

**1.1 Розробник**

-Кононенко Дмитро Олексійович;

- Студент групи КІТ 102.8(а);

- 07-06-2019р..

**1.2 Загальне завдання**

*Завдання до роботи:*

Кожний студент отримує індивідуальне завдання. Варіант завдання обирається за номером прізвища студента у журналі групи. При виконанні завдання з розробки інформаційно-довідкової системи необхідно виконати наступне:

1) з табл. 1, відповідно до варіанта завдання, обрати прикладну галузь;

2) дослідити літературу стосовно прикладної галузі. За результатами аналізу літератури оформити перший, аналітичний розділ пояснювальної записки обсягом 2–3 сторінки;

3) для прикладної галузі розробити розгалужену ієрархію класів, яка складається з не менш ніж трьох класів, один з яких є «батьком» для інших (класів-спадкоємців). Класи повинні мати перевантажені оператори введення-виведення даних та порівняння;

4) розробити клас-контролер, що буде включати колекцію розроблених класів, та наступні методи роботи з цією колекцією:

а) читання даних з файлу та їх запис у контейнер;

б) запис даних з контейнера у файл;

в) сортування елементів у контейнері за вказаними критеріями: поле та напрям сортування, які задаються користувачем з клавіатури;

г) пошук елементів за вказаним критерієм (див. «Завдання для обходу колекції» в табл. 1);

5) розробити клас, який має відображати діалогове меню для демонстрації реалізованих функцій класу контролера;

6) оформити схеми алгоритмів функцій класів контролера та діалогового меню;

7) оформити документацію: пояснювальну записку (див. розділ 2 даних методичних вказівок).

*Увага.* Текст програми та результати роботи програми мають бути подані в додатках.

*Вимоги:*

− усі класи повинні мати конструктори та деструктори;

− якщо функція не змінює поля класу, вона має бути декларована як константна;

− рядки повинні бути типу string;

− при перевантаженні функції треба використовувати ключове слово override;

− програмний код усіх класів має бути 100 % doxygenдокументований;

− у звіті текст програми слід оформляти стилем Courier new 8 пт, інтервал – одиничний; довжина рядка не повинна перевищувати 80 символів.

*Додаткові вимоги на оцінку «добре»:*

− виконання основного завдання та додаткових наступних вимог:

− додати обробку помилок; при цьому функція, що генерує виключення, при її декларуванні повинна мати ключове слово throw;

− виконати перевірку вхідних даних за допомогою регулярних виразів.

*Додаткові вимоги на оцінку «відмінно»:*

− виконати завдання відповідно до вимог на оцінку «добре» та додаткові наступні вимоги:

− критерій для пошуку та сортування задавати у вигляді функтора;

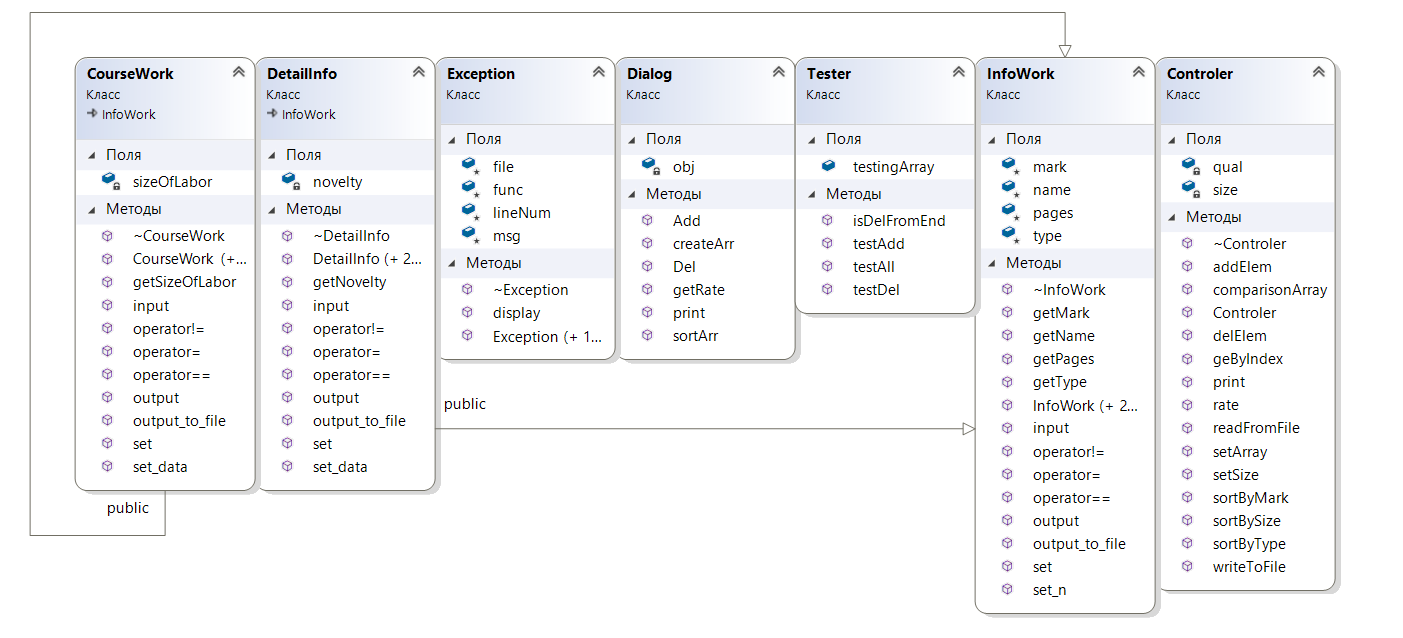
− розробити клас-тестер, основною метою якого буде перевірка коректності роботи класу-контролера.

2 ОПИС ПРОГРАМИ

**2.1 Функціональне призначення**

Програма призначена для виконання комплексних задач з курсу програмування

**2.2 Опис логічної структури**



*Діаграма класу InfoWork*:

1. ~InfoWork - Деструктор класу;
2. set – Генерація випадкових значень;
3. getName , getPages , getType , getMark - Отримання даних(полів класу);
4. InfoWork - Конструктор класу;
5. input – Введення нових даних;
6. output – Вивід на екран;
7. output\_to\_file – Вивід даних у файл;
8. operator= - Перевантаження оператора присвоювання;
9. operator<< - Оператор виведення;
10. operator>> - Оператор вводу даних;
11. operator!= -оператор порівняння;
12. operator== - оператор поріняння;
13. set\_n - Встановлення значень всіх значень одним сетером .

*Діаграма класу Controler :*

1. ~ *Controler* - Деструктор класу;
2. addElem - Додавання нового елементу;
3. delElem - Видалення елементу;
4. *Controler* - Конструктор класу;
5. rate – функція яка вираховує відсоток магістрів;
6. getByIndex - Отримання даних за індексом;
7. print - Вивід даних на екран;
8. readFromFile – Читання даних з файлу;
9. setSize - Отримання розміру для створення масиву;
10. setArray – встовлює новий масив
11. sortBySize, sortByMark, sortByType – Сортування даних за певним критеріем;
12. comprationArray – порівняння двох масивів;
13. writeToFile – Запис результату у файл.

*Діаграма класу (спадкоємця) Coursework* :

1. ~ *Coursework* - Деструктор класу;
2. set – Генерація випадкових значень;
3. getSizeOfLabor - Отримання даних;
4. *Coursework* - Конструктор класу;
5. input – Введення нових даних;
6. operator!= || == - Перевантаження операторів порівняння;
7. output – Вивід на екран;
8. outputToFile – Вивід даних у файл;
9. set\_n - Встановлення значень одним сетером .

*Діаграма класу (спадкоємця) DetailInfo* :

1. ~ *DetailInfo* - Деструктор класу;
2. set – Генерація випадкових значень;
3. - Отримання даних;
4. *DetailInfo* - Конструктор класу;
5. input – Введення нових даних;
6. operator!= || == - Перевантаження операторів порівняння;
7. output – Вивід на екран;
8. outputToFile – Вивід даних у файл;
9. set\_n - Встановлення всіх значень одним сетером .

*Діаграма класу Exception*:

1. Exception - Конструктор класу;
2. display – Виведення помилки на екран.

**Варіанти використання:**

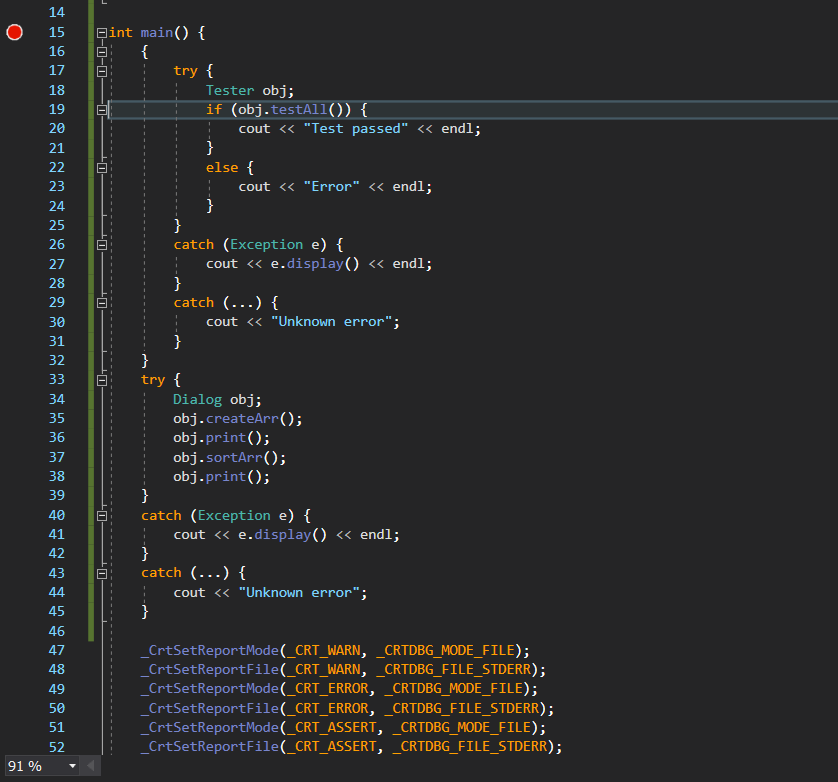


Рисунок 3.1 — функція main()

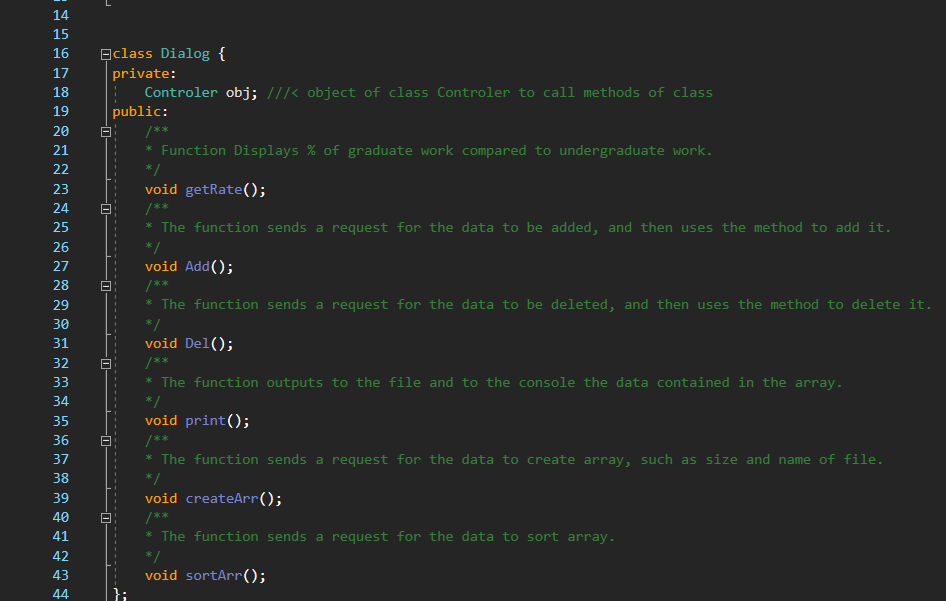


Рисунок 3.2 - клас діалог

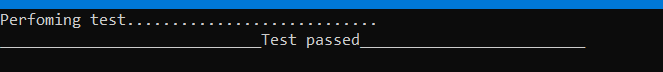


Рисунок 3.3 — робота тестового класу

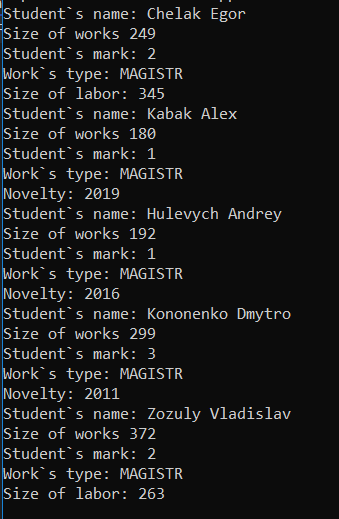
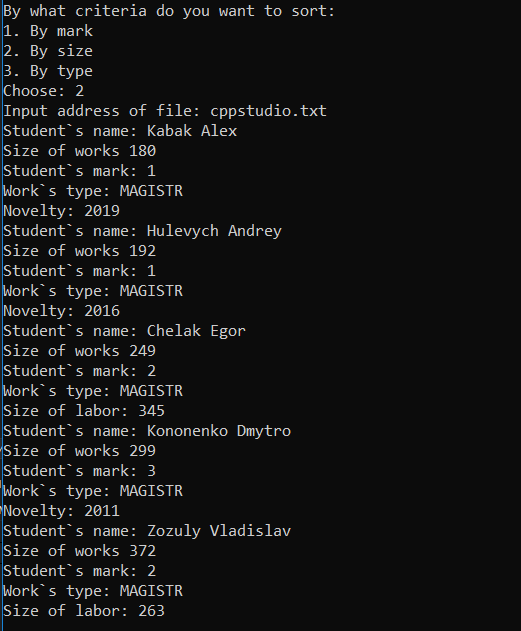


Рисунок 3.4 — робота функції виводу в класі “діалог”.

 Рисунок 3.6 — результат роботи сортування за розміром

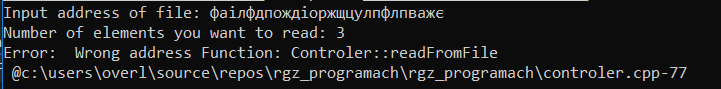


Рисунок 3.7 — робота класу Exception

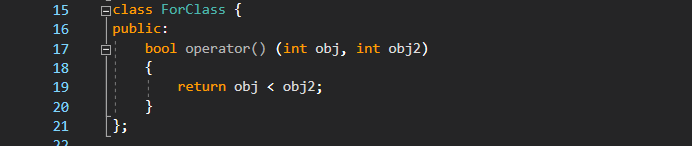


Рисунок 3.8 — функтор використаний в сортуванні

**ВИСНОВОК**

В ході виконання поставленої задачі були закріплені отримані знання з дисципліни «Програмування» шляхом виконання типового комплексного завдання.

Main.cpp

/\*\*

\* @file Main.h

\* Contain main function.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#include "Controler.h"

#include "Exception.h"

#include "InfoWork.h"

#include "Dialog.h"

#include "Tester.h"

int main() {

{

try {

Tester obj;

cout << "Perfoming test............................" << endl;

if (obj.testAll()) {

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Test passed\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

else {

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Error\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

}

catch (Exception e) {

cout << e.display() << endl;

}

catch (...) {

cout << "Unknown error";

}

try {

Dialog obj;

obj.createArr();

obj.print();

obj.sortArr();

obj.print();

}

catch (Exception e) {

cout << e.display() << endl;

}

catch (...) {

cout << "Unknown error";

}

}

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDERR);

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDERR);

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDERR);

\_CrtDumpMemoryLeaks();

return 0;

}

InfoWork.h

/\*\*

\* @file InfoWork.h

\* Declaration of InfoWork class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#pragma once

#include <cstddef>

#include <iostream>

#include <string>

#include <regex>

#include <fstream>

using namespace std;

class InfoWork {

protected:

int pages; ///< size of student`s work

int mark; ///< mark which student got

int type; ///< type of work such as (Bacalavr or Magistr)

string name; ///< student`s name

public:

/\*\*

\* Default constructor

\* Used initialization lists.

\*/

InfoWork();

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param size initializes InfoWork::pages.

\* @param points initializes InfoWork::mark.

\* @param type initializes InfoWork::type.

\* @param creator initializes InfoWork::name.

\*/

InfoWork(int size, int points, int type, string creator);

/\*\*

\* Copie-constructor.

\* Used initialization lists.

\* @param obj: its fields initialize fields current object.

\*/

InfoWork(const InfoWork &obj);

/\*\*

\* Destructor.

\*/

~InfoWork();

/\*\*

\* Set the value of the variable InfoWork::pages.

\* Set the value of the variable InfoWork::mark.

\* Set the value of the variable InfoWork::type.

\* Set the value of the variable InfoWork::name.

\* @param pages is assigned the InfoWork::pages field.

\* @param mark is assigned the InfoWork::mark field.

\* @param type is assigned the InfoWork::type field.

\* @param name is assigned the InfoWork::name field.

\*/

void set\_n(int pages, int mark, int type, string name);

/\*\*

\* Virtual function to generate values.

\*/

virtual void set(string creator);

/\*\*

\* Get copy of field InfoWork::pages.

\* @return current value InfoWork::pages.

\*/

int getPages();

/\*\*

\* Get copy of field InfoWork::mark.

\* @return current value InfoWork::mark.

\*/

int getMark();

/\*\*

\* Get copy of field InfoWork::type.

\* @return current value InfoWork::type.

\*/

int getType();

/\*\*

\* Get copy of field InfoWork::name.

\* @return current value InfoWork::name.

\*/

string getName();

/\*\*

\* Virtual data entry and output functions and file recording.

\*/

virtual void input() = 0;

virtual void output() = 0;

virtual void output\_to\_file(ofstream& file) = 0;

/\*\*

\* Overloaded comparison operator.

\* @param obj its fields compare with fields current object.

\* @return result of comparison.

\*/

bool operator== (const InfoWork &obj);

/\*\*

\* Overloaded comparison operator.

\* @param obj: its fields compare with fields current object.

\* @return result of comparison.

\*/

bool operator!= (const InfoWork &obj);

InfoWork& operator= (const InfoWork &obj);

/\*\*

\* Overloaded output operator.

\* @param out - reference to output stream.

\* @param obj - reference to InfoWork object.

\* @return reference to output stream.

\*/

friend ostream& operator<< (ostream &out, const InfoWork &obj);

/\*\*

\* Overloaded input operator.

\* @param in - reference to input stream.

\* @param obj - reference to InfoWork object.

\* @return reference to input stream.

\*/

friend istream& operator>> (istream &in, InfoWork &obj);

};

InfoWork.cpp

/\*\*

\* @file InfoWork.h

\* Declaration of InfoWork class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#pragma once

#include <cstddef>

#include <iostream>

#include <string>

#include <regex>

#include <fstream>

using namespace std;

class InfoWork {

protected:

int pages; ///< size of student`s work

int mark; ///< mark which student got

int type; ///< type of work such as (Bacalavr or Magistr)

string name; ///< student`s name

public:

/\*\*

\* Default constructor

\* Used initialization lists.

\*/

InfoWork();

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param size initializes InfoWork::pages.

\* @param points initializes InfoWork::mark.

\* @param type initializes InfoWork::type.

\* @param creator initializes InfoWork::name.

\*/

InfoWork(int size, int points, int type, string creator);

/\*\*

\* Copie-constructor.

\* Used initialization lists.

\* @param obj: its fields initialize fields current object.

\*/

InfoWork(const InfoWork &obj);

/\*\*

\* Destructor.

\*/

~InfoWork();

/\*\*

\* Set the value of the variable InfoWork::pages.

\* Set the value of the variable InfoWork::mark.

\* Set the value of the variable InfoWork::type.

\* Set the value of the variable InfoWork::name.

\* @param pages is assigned the InfoWork::pages field.

\* @param mark is assigned the InfoWork::mark field.

\* @param type is assigned the InfoWork::type field.

\* @param name is assigned the InfoWork::name field.

\*/

void set\_n(int pages, int mark, int type, string name);

/\*\*

\* Virtual function to generate values.

\*/

virtual void set(string creator);

/\*\*

\* Get copy of field InfoWork::pages.

\* @return current value InfoWork::pages.

\*/

int getPages();

/\*\*

\* Get copy of field InfoWork::mark.

\* @return current value InfoWork::mark.

\*/

int getMark();

/\*\*

\* Get copy of field InfoWork::type.

\* @return current value InfoWork::type.

\*/

int getType();

/\*\*

\* Get copy of field InfoWork::name.

\* @return current value InfoWork::name.

\*/

string getName();

/\*\*

\* Virtual data entry and output functions and file recording.

\*/

virtual void input() = 0;

virtual void output() = 0;

virtual void output\_to\_file(ofstream& file) = 0;

/\*\*

\* Overloaded comparison operator.

\* @param obj its fields compare with fields current object.

\* @return result of comparison.

\*/

bool operator== (const InfoWork &obj);

/\*\*

\* Overloaded comparison operator.

\* @param obj: its fields compare with fields current object.

\* @return result of comparison.

\*/

bool operator!= (const InfoWork &obj);

InfoWork& operator= (const InfoWork &obj);

/\*\*

\* Overloaded output operator.

\* @param out - reference to output stream.

\* @param obj - reference to InfoWork object.

\* @return reference to output stream.

\*/

friend ostream& operator<< (ostream &out, const InfoWork &obj);

/\*\*

\* Overloaded input operator.

\* @param in - reference to input stream.

\* @param obj - reference to InfoWork object.

\* @return reference to input stream.

\*/

friend istream& operator>> (istream &in, InfoWork &obj);

};

Controler.h

/\*\*

\* @file Controler.h

\* Declaration of Controler class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#pragma once

#include "InfoWork.h"

/\*\*

\* Declaration of functor.

\*/

class ForClass {

public:

bool operator() (int obj, int obj2)

{

return obj < obj2;

}

};

class Controler {

private:

int size; ///< size of array

InfoWork \*\*qual; ///< array

public:

/\*\*

\* Default constructor

\* Used initialization lists.

\*/

Controler();

/\*\*

\* The function determines the % of master's works compared to bachelor's works.

\*/

float rate();

/\*\*

\* Function which set size of array.

\*/

void setSize(int size);

/\*\*

\* Function which print array information in console.

\*/

void print();

/\*\*

\* Function which add element to array.

\* @param newWork: new obj to be add.

\*/

void addElem(InfoWork\* newWork);

/\*\*

\* Function which delete element from array.

\* @param index: index of element to delete.

\*/

void delElem(int l);

/\*\* method of comparing two arrays

\* @param ArrayToTest - pointer to the comparable array.

\* @param otherSize - the size of the array.

\* @return the status of pointers to array objects

\*/

bool comparisonArray(InfoWork \*\*ArrayToTest, size\_t otherSize) const;

/\*\*

\* @param newSize assigns the InfoWork::size field.

\* @param newArray the transferred array is executed

\* copy the objects of the Class class into the current array.

\*/

void setArray(size\_t newSize, InfoWork\*\* newArray);

/\*\*

\* Function which print element by index.

\* @param index: index of element to print.

\*/

void geByIndex(int index);

/\*\*

\* Function which take information from file.

\* @param Size: number of elements to scun.

\* @param fName: name of the file where program will take information.

\*/

void readFromFile(int Size,string fName);

/\*\*

\* Function which print array information in file.

\* @param fName: name of the file where program will print

\*/

void writeToFile(string fName);

/\*\*

\* Function which sort array by mark of work.

\*/

void sortByMark();

/\*\*

\* Function which sort array by size of work.

\*/

void sortBySize();

/\*\*

\* Function which sort array by type of work.

\*/

void sortByType();

/\*\*

\* Destructor.

\*/

~Controler();

};

Controler.cpp

/\*\*

\* @file Controler.cpp

\* Implementation of all functions of Controler class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#include "InfoWork.h"

#include "CourseWork.h"

#include "DetailInfo.h"

#include "Controler.h"

#include "Exception.h"

void Controler::setArray(size\_t newSize, InfoWork \*\*newArray)

{

size = newSize;

if (qual != nullptr) {

for (int i = 0; i < size; i++)

delete qual[i];

delete[] qual;

}

qual = new InfoWork\*[size];

for (int i = 0; i < size; i++)

qual[i] = newArray[i];

delete[] newArray;

newArray = nullptr;

}

bool Controler::comparisonArray(InfoWork \*\*ArrayToTest, size\_t otherSize) const

{

if (size != otherSize)

return false;

for (int i = 0; i < size; i++)

if (\*\*(qual + i) != \*\*(ArrayToTest + i))

return false;

return true;

}

float Controler::rate() {

float counter = 0;

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

int type = qual[i]->getType();

if (type == 2) {

counter++;

}

}

counter = counter \* 100 / this->size;

return counter;

}

Controler::Controler() : size(0) {

qual = nullptr;

}

Controler::~Controler() {

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

delete qual[i];

}

delete[] qual;

}

void Controler::setSize(int size) {

Controler::size = size;

}

void Controler::readFromFile(int newSize,string fName) {

srand(time(NULL));

string \*names = new string[newSize];

regex regex\_repeat("^[a-z].\*|.\*\\s{2,}.\*");

ifstream fin;

fin.open(fName);

if (!fin.is\_open()) {

throw Exception(" Wrong address", \_\_FILE\_\_, \_\_LINE\_\_, \_\_FUNCTION\_\_);

}

InfoWork\* array;

int choice;

int k =0;

while(k < newSize) {

getline(fin, names[k]);

if (regex\_search(names[k], regex\_repeat)){

cout << "Incorrect entry, writing with upper case: " << names[k] << std::endl;

cin.ignore();

getline(cin, names[k]);

}

choice = rand() % 2;

switch (choice) {

case 0:

array = new CourseWork;

array->set(names[k]);

addElem(array);

break;

case 1:

array = new DetailInfo;

array->set(names[k]);

addElem(array);

break;

}

k++;

}

delete[] names;

fin.close();

}

void Controler::print() {

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

qual[i]->output();

}

}

void Controler::addElem(InfoWork\* new\_work) {

InfoWork \*\*mas = new InfoWork\*[size + 1];

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

mas[i] = qual[i];

}

size++;

mas[size - 1] = new\_work;

delete[] qual;

qual = mas;

}

void Controler::delElem(int index) {

if (index < 0 || index > size) {

throw Exception(" Wrong index", \_\_FILE\_\_, \_\_LINE\_\_, \_\_FUNCTION\_\_);

}

size--;

InfoWork\*\* mas = new InfoWork\*[size];

int j = 0;

for (int i = 0; i < index - 1; i++) {

mas[i] = qual[j];

j++;

}

j++;

for (int i = index - 1; i < size; i++) {

mas[i] = qual[j];

j++;

}

delete qual[index - 1];

delete[]qual;

qual = mas;

}

void Controler::geByIndex(int index) {

if (index < 0 || index > size) {

throw Exception(" Wrong index", \_\_FILE\_\_, \_\_LINE\_\_, \_\_FUNCTION\_\_);

}

cout << endl;

cout << "Searched element: ";

qual[index - 1]->output();

cout << endl << endl;

}

void Controler::writeToFile(string fName) {

std::ofstream fout;

fout.open(fName);

if (!fout.is\_open()) {

throw Exception(" Wrong address", \_\_FILE\_\_, \_\_LINE\_\_, \_\_FUNCTION\_\_);

}

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

qual[i]->output\_to\_file(fout);

}

fout.close();

}

void Controler::sortByMark() {

ForClass obj;

InfoWork\* temp;

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

for (int j = 0; j < size; j++) {

if (obj(qual[i]->getMark(), qual[j]->getMark())) {

temp = qual[i];

qual[i] = qual[j];

qual[j] = temp;

}

}

}

}

void Controler::sortBySize() {

ForClass obj;

InfoWork\* temp;

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

for (int j = 0; j < size; j++) {

if (obj(qual[i]->getPages(), qual[j]->getPages())) {

temp = qual[i];

qual[i] = qual[j];

qual[j] = temp;

}

}

}

}

void Controler::sortByType() {

ForClass obj;

InfoWork\* temp;

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

for (int j = 0; j < size; j++) {

if (obj(qual[i]->getType(), qual[j]->getType())) {

temp = qual[i];

qual[i] = qual[j];

qual[j] = temp;

}

}

}

}

CourseWork.h

/\*\*

\* @file CourseWork.h

\* Declaration of CourseWork class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#pragma once

#include "InfoWork.h"

class CourseWork : public InfoWork

{

private:

int sizeOfLabor; ///< size of labor which student did

public:

/\*\*

\* Default constructor

\* Used initialization lists.

\*/

CourseWork();

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param size initializes CourseWork::pages.

\* @param points initializes CourseWork::mark.

\* @param type initializes CourseWork::type.

\* @param creator initializes CourseWork::name.

@param creator initializes CourseWork::sizeOfLabor.

\*/

CourseWork(int pages,int mark,int type,string name,int sizeOfLabor);

/\*\*

\* Copie-constructor.

\* Used initialization lists.

\* @param obj: its fields initialize fields current object.

\*/

CourseWork(const CourseWork &obj);

/\*\*

\* Destructor.

\*/

~CourseWork();

/\*\*

\* Overloaded comparison operator.

\* @param obj its fields compare with fields current object.

\* @return result of comparison.

\*/

bool operator== (const CourseWork &obj);

/\*\*

\* Overloaded comparison operator.

\* @param obj its fields compare with fields current object.

\* @return result of comparison.

\*/

bool operator!= (const CourseWork &obj);

/\*\*

\* Overloaded assignment operator.

\* @param obj: its fields initialize fields current object.

\* @return pointer to current object.

\*/

CourseWork& operator= (const CourseWork &obj);

/\*\*

\* Overloaded output operator.

\* @param out - reference to output stream.

\* @param obj - reference to CourseWork object.

\* @return reference to output stream.

\*/

friend ostream& operator<< (ostream &out, const CourseWork &obj);

/\*\*

\* Overloaded input operator.

\* @param in - reference to input stream.

\* @param obj - reference to CourseWork object.

\* @return reference to input stream.

\*/

friend istream& operator>> (istream &in, CourseWork &obj);

/\*\*

\* Get copy of field DetailInfo::novelty.

\* @return current value DetailInfo::novelty.

\*/

int getSizeOfLabor();

/\*\*

\* Virtual function to generate values.

\*/

virtual void set(string s) override;

/\*\*

\* Set the value of the variable DetailInfo::pages.

\* Set the value of the variable DetailInfo::mark.

\* Set the value of the variable DetailInfo::type.

\* Set the value of the variable DetailInfo::name.

\* Set the value of the variable DetailInfo::novelty.

\* @param pages is assigned the DetailInfo::pages field.

\* @param mark is assigned the DetailInfo::mark field.

\* @param type is assigned the DetailInfo::type field.

\* @param name is assigned the DetailInfo::name field.

\* @param sizeOfLabor is assigned the DetailInfo::sizeOfLabor field.

\*/

void set\_data(int pages, int mark, int type, string name, int sizeOfLabor);

/\*\*

\* Virtual data entry and output functions and file recording.

\*/

virtual void input() override;

virtual void output() override;

virtual void output\_to\_file(std::ofstream& file) override;

};

CourseWork.cpp

/\*\*

\* @file CourseWork.cpp

\* Implementation of all functions of CourseWork class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#include "CourseWork.h"

CourseWork::CourseWork() : sizeOfLabor(0), InfoWork() {}

CourseWork::CourseWork(const CourseWork &obj) : InfoWork(obj.pages, obj.mark, obj.type, obj.name), sizeOfLabor(obj.sizeOfLabor){};

CourseWork::CourseWork(int pages, int mark, int type, string name, int sizeOfLabor) : sizeOfLabor(sizeOfLabor),InfoWork(pages,mark,type,name) {}

CourseWork::~CourseWork(){}

void CourseWork::input() {

cout << "Input student`s name: ";

cin.ignore();

getline(cin, name);

cout << "Input pages: ";

cin >> pages;

cout << "Input type: ";

cin >> type;

cout << "Input student`s mark: ";

cin >> mark;

cout << "Input size of labor: ";

cin >> sizeOfLabor;

cout << endl;

}

void CourseWork::output() {

cout << "Student`s name: " << name << endl;

cout << "Size of works " << pages << endl;

cout << "Student`s mark: " << mark << endl;

cout << "Work`s type:";

if (type == 1) {

cout << " BACALAVR" <<endl;

}

else {

cout << " MAGISTR" << endl;

}

cout << "Size of labor: " << sizeOfLabor << endl;

}

void CourseWork::set(string name) {

mark = rand() % 5 + 1;

pages = rand() % 336 + 100;

type = rand() % 2 + 1;

sizeOfLabor = rand() % 300 + 100;

this->name = name;

}

void CourseWork::set\_data(int a, int b, int c, string name, int sizeOfLabor) {

mark = a;

pages = b;

type = c;

this->name = name;

this->sizeOfLabor = sizeOfLabor;

}

int CourseWork::getSizeOfLabor() {

return sizeOfLabor;

}

void CourseWork::output\_to\_file(ofstream& fout) {

fout << "Student`s name: " << name << endl;

fout << "Size of works " << pages << endl;

fout << "Student`s mark: " << mark << endl;

fout << "Work`s type:";

if (type == 1) {

fout << " BACALAVR" << endl;

}

else {

fout << " MAGISTR" << endl;

fout << "Size of labor: " << sizeOfLabor << endl;

}

}

bool CourseWork::operator== (const CourseWork &obj) {

return (pages == obj.pages && type == obj.type && mark == obj.mark && name == obj.name);

}

bool CourseWork::operator!= (const CourseWork &obj) {

return (pages != obj.pages && type != obj.type && mark != obj.mark && name != obj.name);

}

std::istream& operator>> (std::istream &in, CourseWork &obj) {

in >> obj.pages;

in >> obj.mark;

in >> obj.type;

in >> obj.name;

return in;

}

std::ostream& operator<< (std::ostream &out, const CourseWork &obj) {

out << "Name: " << obj.name << " Mark: " << obj.mark << endl;

out << "Size: " << obj.pages;

if (obj.type == 1) {

out << " BACALAVR";

}

else {

out << " MAGISTR";

}

return out;

}

CourseWork& CourseWork::operator= (const CourseWork &obj) {

pages = obj.pages;

mark = obj.mark;

type = obj.type;

name = obj.name;

sizeOfLabor = sizeOfLabor;

return \*this;

}

Exception.h

/\*\*

\* @file Exception.h

\* Declaration of Exception class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#pragma once

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <fstream>

#include <sstream>

using namespace std;

class Exception

{

public:

/\*\*

\* Destructor.

\*/

~Exception() {};

/\*\*

\* Default constructor

\* Used initialization lists.

\*/

Exception() : msg(), file(), lineNum(), func() {};

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param msg initializes Exception::msg.

\* @param file initializes Exception::file.

\* @param lineNum initializes Exception::lineNum.

\* @param func initializes Exception::func.

\*/

Exception(string pMsg, string pFile, int nLine, string funcName) : msg(pMsg), file(pFile), lineNum(nLine), func(funcName) {}

/\*\*

\* For output error.

\*/

virtual string display() {

ostringstream out;

out << "Error: " << msg

<< " Function: " << func

<< endl;

out << " @" << file << "-" << lineNum << endl;

return out.str();

}

protected:

string msg;///< message about error

string file;///< file where error occurred

int lineNum;///< line in file where error occurred

string func;///< func where error occurred

};

Dialog.cpp

/\*\*

\* @file Dialog.cpp

\* Implementation of all functions of Dialog class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#include "Dialog.h"

void Dialog::getRate() {

cout << "The percentage of graduate work compared to undergraduate work: ";

cout << obj.rate();

}

void Dialog::print(){

string address;

cout << "Input address of file: ";

cin >> address;

obj.print();

obj.writeToFile(address);

}

void Dialog::createArr() {

int size;

string address;

cout << "Input address of file: ";

cin >> address;

cout << "Number of elements you want to read: ";

cin >> size;

obj.readFromFile(size, address);

}

void Dialog::sortArr() {

int choose;

cout << "By what criteria do you want to sort: " << endl;

cout << "1. By mark" << endl;

cout << "2. By size" << endl;

cout << "3. By type" << endl;

cout << "Choose: ";

cin >> choose;

switch (choose) {

case 1:

obj.sortByMark();

break;

case 2:

obj.sortBySize();

break;

case 3:

obj.sortByType();

break;

}

}

void Dialog::Del() {

int index;

cout << "Input index of elemen which you want to delete: ";

cin >> index;

obj.delElem(index);

}

void Dialog::Add() {

int choose;

InfoWork \*temp;

cout << "Input type of new element: ";

cout << "1. CourseWork" << endl;

cout << "2. DetailInfo" << endl;

cin >> choose;

switch (choose) {

case 1:

temp = new CourseWork;

temp->input();

obj.addElem(temp);

delete temp;

break;

case 2:

temp = new DetailInfo;

temp->input();

obj.addElem(temp);

delete temp;

break;

}

}

Dialog.h

/\*\*

\* @file Dialog.h

\* Declaration of Dialog class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#pragma once

#include "Controler.h"

#include "InfoWork.h"

#include "CourseWork.h"

#include "DetailInfo.h"

class Dialog {

private:

Controler obj; ///< object of class Controler to call methods of class

public:

/\*\*

\* Function Displays % of graduate work compared to undergraduate work.

\*/

void getRate();

/\*\*

\* The function sends a request for the data to be added, and then uses the method to add it.

\*/

void Add();

/\*\*

\* The function sends a request for the data to be deleted, and then uses the method to delete it.

\*/

void Del();

/\*\*

\* The function outputs to the file and to the console the data contained in the array.

\*/

void print();

/\*\*

\* The function sends a request for the data to create array, such as size and name of file.

\*/

void createArr();

/\*\*

\* The function sends a request for the data to sort array.

\*/

void sortArr();

};

Tester.cpp

#include "Tester.h"

#include "CourseWork.h"

#include "DetailInfo.h"

bool Tester::testAdd()

{

int expectedSize = 3;

InfoWork\*\* expectedArray = new InfoWork\*[expectedSize];

InfoWork\* tempWork;

string titleStr;

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

tempWork = new CourseWork;

titleStr = to\_string(i);

tempWork->set\_n(i,i,i,titleStr);

expectedArray[i] = tempWork;

}

int testSize = expectedSize -1;

InfoWork\*\* testArray = new InfoWork\*[testSize];

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

tempWork = new CourseWork;

titleStr = to\_string(i);

tempWork->set\_n(i,i,i,titleStr);

testArray[i] = tempWork;

}

testingArray.setArray(testSize, testArray);

InfoWork\* newWork = new CourseWork;

titleStr = to\_string(testSize);

newWork->set\_n(testSize, testSize, testSize, titleStr);

testingArray.addElem(newWork);

bool result;

if (testingArray.comparisonArray(expectedArray, expectedSize))

result = true;

else

result = false;

for (int i = 0; i < expectedSize; i++)

delete expectedArray[i];

delete[] expectedArray;

return result;

}

bool Tester::isDelFromEnd()

{

int expectedSize = 2;

InfoWork\*\* expectedArray = new InfoWork\*[expectedSize];

string titleStr;

InfoWork\* tempWork;

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

tempWork = new CourseWork;

titleStr = to\_string(i);

tempWork->set\_n(i, i, i, titleStr);

expectedArray[i] = tempWork;

}

int testSize = expectedSize + 1;

InfoWork\*\* testArray = new InfoWork\*[testSize];

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

tempWork = new CourseWork;

titleStr = to\_string(i);

tempWork->set\_n(i, i, i, titleStr);

testArray[i] = tempWork;

}

testingArray.setArray(testSize, testArray);

testingArray.delElem(testSize);

bool endRemove = testingArray.comparisonArray(expectedArray, expectedSize);

for (int i = 0; i < expectedSize; i++)

delete expectedArray[i];

delete[] expectedArray;

return endRemove;

}

bool Tester::testDel()

{

if (isDelFromEnd())

return true;

else

return false;

}

bool Tester::testAll()

{

return testAdd() && testDel();

}

Tester.h

#pragma once

#include "InfoWork.h"

#include "Controler.h"

class Tester {

public:

Controler testingArray; ///< array for testing

/\*\*

\* Test method for Controler::addElem.

\* @return the status of the function Controler::addElem.

\*/

bool testAdd();

/\*\*

\* Test method for Controler::delEleme () function at

\* remove the item from the end.

\* @return the status of the Controler::delEleme () function at

\* remove the item from the end.

\*/

bool isDelFromEnd();

/\*\*

\* Test method for Controler::delElem.

\* @return the status of the function Controler::delElem.

\*/

bool testDel();

public:

/\*\*

\* The method of calling all the functions of the class Tester.

\* @return the status of all functions of the Tester class.

\*/

bool testAll();

};

DetailInfo.h

/\*\*

\* @file DetailInfo.h

\* Declaration of DetailInfo class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#pragma once

#include "InfoWork.h"

class DetailInfo : public InfoWork

{

private:

int novelty;///< novelty of student`s work presented in years

public:

/\*\*

\* Overloaded comparison operator.

\* @param obj its fields compare with fields current object.

\* @return result of comparison.

\*/

bool operator== (const DetailInfo &obj);

/\*\*

\* Overloaded comparison operator.

\* @param obj: its fields compare with fields current object.

\* @return result of comparison.

\*/

bool operator!= (const DetailInfo &obj);

/\*\*

\* Overloaded assignment operator.

\* @param obj: its fields initialize fields current object.

\* @return pointer to current object.

\*/

DetailInfo& operator= (const DetailInfo &obj);

/\*\*

\* Overloaded output operator.

\* @param out - reference to output stream.

\* @param obj - reference to DetailInfo object.

\* @return reference to output stream.

\*/

friend ostream& operator<< (ostream &out, const DetailInfo &obj);

/\*\*

\* Overloaded input operator.

\* @param in - reference to input stream.

\* @param obj - reference to DetailInfo object.

\* @return reference to input stream.

\*/

friend istream& operator>> (istream &in, DetailInfo &obj);

/\*\*

\* Get copy of field DetailInfo::novelty.

\* @return current value DetailInfo::novelty.

\*/

int getNovelty();

/\*\*

\* Set the value of the variable DetailInfo::pages.

\* Set the value of the variable DetailInfo::mark.

\* Set the value of the variable DetailInfo::type.

\* Set the value of the variable DetailInfo::name.

\* Set the value of the variable DetailInfo::novelty.

\* @param pages is assigned the DetailInfo::pages field.

\* @param mark is assigned the DetailInfo::mark field.

\* @param type is assigned the DetailInfo::type field.

\* @param name is assigned the DetailInfo::name field.

\* @param novelty is assigned the DetailInfo::novelty field.

\*/

void set\_data(int pages, int mark, int type,string name, int novelty);

/\*\*

\* Virtual function to generate values.

\*/

virtual void set(string x);

/\*\*

\* Virtual data entry and output functions and file recording.

\*/

virtual void input() override;

virtual void output() override;

virtual void output\_to\_file(std::ofstream& file) override;

/\*\*

\* Default constructor

\* Used initialization lists.

\*/

DetailInfo();

/\*\*

\* Copie-constructor.

\* Used initialization lists.

\* @param obj: its fields initialize fields current object.

\*/

DetailInfo(const DetailInfo &obj);

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param size initializes InfoWork::pages.

\* @param points initializes InfoWork::mark.

\* @param type initializes InfoWork::type.

\* @param creator initializes InfoWork::name.

\* @param creator initializes CourseWork::novelty.

\*/

DetailInfo(int pages, int mark, int type, string name, int novelty);

/\*\*

\* Destructor.

\*/

~DetailInfo();

};

DetailInfo.cpp

/\*\*

\* @file DetailInfo.cpp

\* Implementation of all functions of DetailInfo class.

\* @author Kononenko Dmytro

\* @version 1.0

\* @date 2019.06.06

\*/

#include "DetailInfo.h"

DetailInfo::~DetailInfo(){}

DetailInfo::DetailInfo() : novelty(0), InfoWork(){}

DetailInfo::DetailInfo(const DetailInfo &obj) : InfoWork(obj.pages, obj.mark, obj.type, obj.name), novelty(obj.novelty) {};

DetailInfo::DetailInfo(int pages, int mark, int type, string name, int novelty) : novelty(novelty), InfoWork(pages, mark, type, name) {}

void DetailInfo::output\_to\_file(std::ofstream& fout) {

fout << "Student`s name: " << name << endl;

fout << "Size of works " << pages << endl;

fout << "Student`s mark: " << mark << endl;

fout << "Work`s type:";

if (type == 1) {

fout << " BACALAVR" << endl;

}

else {

fout << " MAGISTR" << endl;

}

fout << "Novelty: " << novelty << endl;

}

void DetailInfo::input() {

cout << "Input student`s name: ";

cin.ignore();

getline(cin, name);

cout << "Input pages: ";

cin >> pages;

cout << "Input type: ";

cin >> type;

cout << "Input student`s mark: ";

cin >> mark;

cout << "Input novelty: ";

cin >> novelty;

cout << endl;

}

void DetailInfo::output() {

cout << "Student`s name: " << name << endl;

cout << "Size of works " << pages << endl;

cout << "Student`s mark: " << mark << endl;

cout << "Work`s type:";

if (type == 1) {

cout << " BACALAVR" <<endl;

}

else {

cout << " MAGISTR" << endl;

}

cout << "Novelty: " << novelty << endl;

}

void DetailInfo::set\_data(int a, int b, int c, string name, int novelty) {

mark = a;

pages = b;

type = c;

this->name = name;

this->novelty = novelty;

}

int DetailInfo::getNovelty() {

return novelty;

}

void DetailInfo::set(string name) {

mark = rand() % 5 + 1;

pages = rand() % 336 + 100;

type = rand() % 2 + 1;

novelty = rand() % 10 + 2010;

this->name = name;

}

bool DetailInfo::operator== (const DetailInfo &obj) {

return (pages == obj.pages && type == obj.type && mark == obj.mark && name == obj.name);

}

bool DetailInfo::operator!= (const DetailInfo &obj) {

return (pages != obj.pages && type != obj.type && mark != obj.mark && name != obj.name);

}

std::istream& operator>> (std::istream &in, DetailInfo &obj) {

in >> obj.pages;

in >> obj.mark;

in >> obj.type;

in >> obj.name;

return in;

}

std::ostream& operator<< (std::ostream &out, const DetailInfo &obj) {

out << "Name: " << obj.name << " Mark: " << obj.mark << endl;

out << "Size: " << obj.pages;

if (obj.type == 1) {

out << " BACALAVR";

}

else {

out << " MAGISTR";

}

return out;

}

DetailInfo& DetailInfo::operator= (const DetailInfo &obj) {

pages = obj.pages;

mark = obj.mark;

type = obj.type;

name = obj.name;

novelty = novelty;

return \*this;

}