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. 18038-01 81 01-1 –АЗ

Розробник

Виконав:

студент групи 1КІТ-102.8А

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Котенко С.М./

Перевірив:

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

Харків 2019

ЗАТВЕРДЖЕНО

1КІТ102.8А.18038-01 81 01-1 –АЗ

Розрахункове завдання з дисципліни

«Алгоритми та структури даних»

Пояснювальна записка

1КІТ.102.8А.18038-01 81 01-1 -АЗ

Листів 20

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**

- Котенко Сергій Миколайович;

- 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;

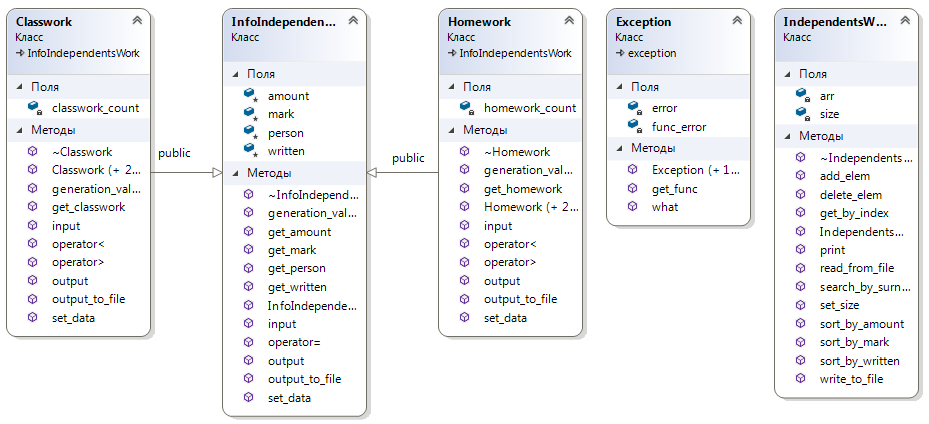
− розробити клас-тестер, основною метою якого буде перевірка коректності роботи класу-контролера.

2 ОПИС ПРОГРАМИ

**2.1 Функціональне призначення**

Програма призначена для виконання комплексних задач з курсу програмування

**2.2 Опис логічної структури**



*Діаграма класу InfoIndependentsWork*:

* ~InfoIndependentsWork - Деструктор класу;
* generation\_values – Генерація випадкових значень;
* get\_amount , get\_mark , get\_surname , get\_written - Отримання даних;
* InfoIndependentsWork - Конструктор класу;
* input – Введення нових даних;
* output – Вивід на екран;
* output\_to\_file – Вивід даних у файл;
* operator= - Перевантаження оператора присвоювання;
* set\_data - Встановлення значень .

*Діаграма класу IndependentsWork :*

* ~IndependentsWork - Деструктор класу;
* add\_elem - Додавання нового елементу;
* delete\_elem - Видалення елементу;
* IndependentsWork - Конструктор класу;
* get\_by\_index - Отримання даних за індексом;
* print - Вивід даних на екран;
* read\_from\_file – Читання даних з файлу;
* search\_by\_surname – Пошук за прізвищем студента;
* set\_size - Отримання розміру для створення масиву;
* sort\_by\_amount, sort\_by\_mark, sort\_by\_written – Сортування даних за певним критеріем;
* write\_to\_file – Запис результату у файл.

*Діаграма класу (спадкоємця) Homework* :

* ~Homework - Деструктор класу;
* generation\_values – Генерація випадкових значень;
* get\_homework - Отримання даних;
* Homework - Конструктор класу;
* input – Введення нових даних;
* operator<> - Перевантаження операторів порівняння;
* output – Вивід на екран;
* output\_to\_file – Вивід даних у файл;
* set\_data - Встановлення значень .

*Діаграма класу (спадкоємця) Classwork* :

* ~Classwork - Деструктор класу;
* generation\_values – Генерація випадкових значень;
* get\_homework - Отримання даних;
* Classwork - Конструктор класу;
* input – Введення нових даних;
* operator<> - Перевантаження операторів порівняння;
* output – Вивід на екран;
* output\_to\_file – Вивід даних у файл;
* set\_data - Встановлення значень .

*Діаграма класу Exception*:

* get\_func - Отримання даних;
* Exception - Конструктор класу;
* what – Відключення базових виключень

**3 ВАРІАНТИ ВИКОРИСТАННЯ**

**3.1 Ілюстрація роботи програми**



Рисунок 3.1 – Виправлення помилок при зчитуванні з файлу за допомогою регулярних виразів

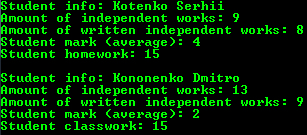


Рисунок 3.2 – Створенні данні з різними полями спадкоємців

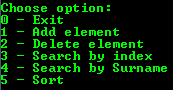


Рисунок 3.3 – Можливі опції роботи з програмою

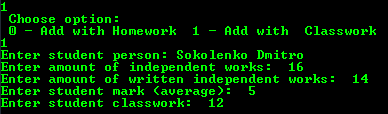


Рисунок 3.4 – Додавання нового елементу



Рисунок 3.5 – Видалення елементу

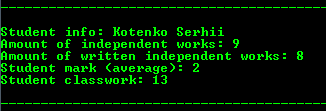


Рисунок 3.6 – Пошук за індексом

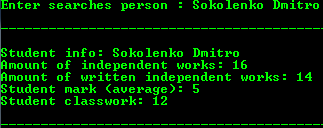


Рисунок 3.7 – Пошук певного студента

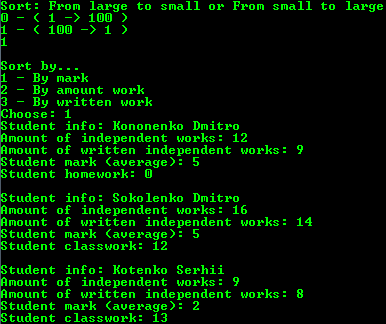


Рисунок 3.8 – Сортування за певним критерієм та напрямком

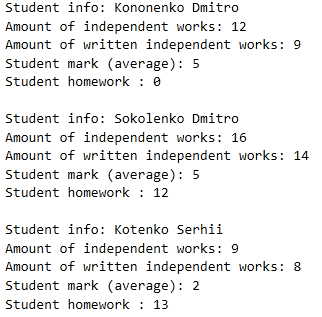


Рисунок 3.9 – Записаний результат у файл

**ВИСНОВОК**

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

Main.cpp

/\*

\* @ mainpage

\* @ author - Kotenko Sergey

\* @ date - 06.06.19

\* @ version - 1.0

\*/

#include "InfoIndependentsWork.h"

#include "IndependentsWork.h"

#include "Homework.h"

#include "Classwork.h"

#include "Exception.h"

bool sort(int a, int b) {

return a < b;

}

bool sort2(int a, int b) {

return a > b;

}

int main() {

system("color A");

try {

auto i = 0;

std::cout << "Enter size : ";

std::cin >> i;

IndependentsWork Work;

std::regex regex\_spaces("[\\s]{2,}");

std::regex regex\_firstSymbol("^[A-Z]");

system("cls");

Work.read\_from\_file(i);

system("cls");

Work.print();

InfoIndependentsWork\* new\_work;

int option = 0;

do {

std::cout << "Choose option:" << std::endl << "0 - Exit " << std::endl << "1 - Add element" << std::endl << "2 - Delete element" << std::endl << "3 - Search by index" << std::endl << "4 - Search by Surname" << std::endl << "5 - Sort" << std::endl << std::endl;

std::cin >> option;

switch (option) {

case 1: {

std::cout << " Choose option: " << std::endl << " 0 - Add with Homework " << " 1 - Add with Classwork " << std::endl;

std::cin >> option;

switch (option) {

case 0: {

new\_work = new Homework;

new\_work->input();

Work.add\_elem(new\_work);

break;

}

case 1: {

new\_work = new Classwork;

new\_work->input();

Work.add\_elem(new\_work);

break;

}

}

system("cls");

Work.print();

break;

}

case 2: {

auto j = 0;

std::cout << std::endl << "Enter index by delete element : ";

std::cin >> j;

std::cout << std::endl;

Work.delete\_elem(j);

system("cls");

Work.print();

break;

}

case 3: {

auto z = 0;

std::cout << std::endl << "Enter index : ";

std::cin >> z;

std::cout << std::endl;

system("cls");

Work.print();

Work.get\_by\_index(z);

break;

}

case 4: {

system("cls");

std::string search\_surname;

std::cout << "Enter searches person : ";

std::cin.ignore();

getline(std::cin, search\_surname);

Work.search\_by\_surname(search\_surname);

break;

}

case 5: {

system("cls");

bool(\*pointer)(int a, int b);

int s;

std::cout << "Sort: From large to small or From small to large " << std::endl;

std::cout << "0 - ( 1 -> 100 )" << std::endl;

std::cout << "1 - ( 100 -> 1 )" << std::endl;

std::cin >> s;

if (s == 0) {

pointer = sort;

}

else {

pointer = sort2;

}

std::cout << std::endl << "Sort by..." << std::endl;

std::cout << "1 - By mark" << std::endl;

std::cout << "2 - By amount work" << std::endl;

std::cout << "3 - By written work" << std::endl;

std::cout << "Choose: ";

std::cin >> s;

switch (s) {

case 1:

Work.sort\_by\_mark(pointer);

Work.print();

break;

case 2:

Work.sort\_by\_amount(pointer);

Work.print();

break;

case 3:

Work.sort\_by\_written(pointer);

Work.print();

break;

}

break;

default:

break;

}

}

} while (option != 0);

Work.write\_to\_file();

system("cls");

}

catch (Exception& exception) {

std::cout << "An error has occurred in working." << exception.what() << std::endl << " Error in this function: " << exception.get\_func() << std::endl;

}

catch (std::exception& exception) {

std::cout << "An error has occurred in working." << exception.what() << std::endl;

}

catch (...) {

std::cout << "Unknown error!" << std::endl;

}

\_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 \_CrtDumpMemoryLeaks();

system("pause");

}

InfoIndependentsWork.h

#pragma once

#include <iostream>

#include <ctime>

#include <fstream>

#include <string>

#include <cstdio>

#include <regex>

class InfoIndependentsWork {

protected:

int amount;

int written;

int mark;

std::string person;

public:

/\*\*

\* InfoIndependentsWork class constructors.

\*/

InfoIndependentsWork();

/\*\*

\* Copie-constructor.

\* Used initialization lists.

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

\*/

InfoIndependentsWork(const InfoIndependentsWork &obj);

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param amount initializes InfoIndependentsWork::amount.

\* @param written initializes InfoIndependentsWork::written.

\* @param mark initializes InfoIndependentsWork::mark.

\* @param person initializes InfoIndependentsWork::person.

\*/

InfoIndependentsWork(int amount, int written, int mark, std::string person);

/\*\*

\* Operator overload = .

\*/

InfoIndependentsWork& operator= (const InfoIndependentsWork &obj);

/\*\*

\* Overloading of the output operator at the pointer and without.

\*/

friend std::ostream& operator<< (std::ostream &out, const InfoIndependentsWork &obj);

friend std::ostream& operator<< (std::ostream &out, const InfoIndependentsWork \*obj);

/\*\*

\* Overloading of the input operator at the pointer and without.

\*/

friend std::istream& operator>> (std::istream &in, InfoIndependentsWork &obj);

friend std::istream& operator>> (std::istream &in, InfoIndependentsWork \*obj);

/\*\*

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

\*/

virtual void input() = 0;

virtual void output() = 0;

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

/\*\*

\* Virtual function to generate values.

\*/

virtual void generation\_values(std::string s);

/\*\*

\* Get copy of field.

\* Return current value.

\*/

int get\_amount();

int get\_written();

int get\_mark();

std::string get\_person();

/\*\*

\* Assignment field value.

\*/

void set\_data(int amount, int written, int mark, std::string person);

/\*\*

\* InfoIndependentsWork class destructor.

\*/

~InfoIndependentsWork();

};

InfoIndependentsWork.cpp

#include "InfoIndependentsWork.h"

InfoIndependentsWork::InfoIndependentsWork() :amount(0), written(0), mark(0) {

}

InfoIndependentsWork::InfoIndependentsWork(const InfoIndependentsWork &obj) : amount(obj.amount), written(obj.written), mark(obj.mark) {

}

InfoIndependentsWork::InfoIndependentsWork(int amount, int written, int mark, std::string person) : amount(amount), written(written), mark(mark), person(person) {

}

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

out << obj.person << ": " << obj.amount << std::endl;

out << obj.person << ": " << obj.written << std::endl;

out << obj.person << ": " << obj.mark << std::endl;

return out;

}

std::ostream& operator<< (std::ostream &out, const InfoIndependentsWork \*obj) {

out << obj->person << ": " << obj->amount << std::endl;

out << obj->person << ": " << obj->written << std::endl;

out << obj->person << ": " << obj->mark << std::endl;

return out;

}

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

in >> obj.amount;

in >> obj.written;

in >> obj.mark;

in >> obj.person;

return in;

}

std::istream& operator>> (std::istream &in, InfoIndependentsWork \*obj) {

in >> obj->amount;

in >> obj->written;

in >> obj->mark;

in >> obj->person;

return in;

}

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

amount = obj.amount;

written = obj.written;

mark = obj.mark;

return \*this;

}

InfoIndependentsWork::~InfoIndependentsWork() {

};

int InfoIndependentsWork::get\_amount() {

return InfoIndependentsWork::amount;

}

int InfoIndependentsWork::get\_written() {

return InfoIndependentsWork::written;

}

int InfoIndependentsWork::get\_mark() {

return InfoIndependentsWork::mark;

}

std::string InfoIndependentsWork::get\_person() {

return InfoIndependentsWork::person;

}

void InfoIndependentsWork::generation\_values(std::string s) {

this->amount = 8 + rand() % 8;

this->written = amount - rand() % 8;

this->mark = 1 + rand() % 5;

this->person = s;

}

void InfoIndependentsWork::set\_data(int amount, int written, int mark, std::string person) {

this->person = person;

this->amount = amount;

this->written = written;

this->mark = mark;

}

IndependentsWork.h

#pragma once

#include "InfoIndependentsWork.h"

class IndependentsWork {

private:

int size;

InfoIndependentsWork \*\*arr;

public:

/\*\*

\* IndependentsWork class constructors.

\*/

IndependentsWork();

/\*\*

\* Overloading of the output operator.

\*/

friend std::ostream& operator<< (std::ostream &out, const IndependentsWork &obj);

/\*\*

\* Overloading of the input operator.

\*/

friend std::istream& operator>> (std::istream &in, IndependentsWork &obj);

void set\_size(int size);

void print();

void add\_elem(InfoIndependentsWork\* new\_work);

void delete\_elem(int l);

void get\_by\_index(int index);

void read\_from\_file(int new\_size);

void write\_to\_file();

void search\_by\_surname(std::string search\_person);

void sort\_by\_mark(bool(\*sort)(int a, int b));

void sort\_by\_amount(bool(\*sort)(int a, int b));

void sort\_by\_written(bool(\*sort)(int a, int b));

/\*\*

\* InfoIndependentsWork class destructor.

\*/

~IndependentsWork();

};

IndependentsWork.cpp

#include "IndependentsWork.h"

#include "Homework.h"

#include "Classwork.h"

#include "Exception.h"

IndependentsWork::IndependentsWork() : size(0) {

arr = nullptr;

}

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

out << obj.size << std::endl;

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

out << obj.arr[i];

}

return out;

}

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

in >> obj.size;

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

in >> obj.arr[i];

}

return in;

}

IndependentsWork::~IndependentsWork() {

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

delete arr[i];

}

delete[] arr;

}

void IndependentsWork::set\_size(int size) {

IndependentsWork::size = size;

}

void IndependentsWork::read\_from\_file(int new\_size) {

srand(time(NULL));

std::string \*person = new std::string[new\_size];

std::regex regex\_firstSymbol("^[A-Z]");

std::regex regex\_spaces("[\\s]{2,}");

std::ifstream fin;

fin.open("StudentsSurname.txt");

if (!fin) {

throw Exception("Can't open file for reading", "read\_from\_file");

}

InfoIndependentsWork\* new\_work;

int choice;

for (int k = 0; k < new\_size; k++) {

getline(fin, person[k]);

if (!(regex\_search(person[k], regex\_firstSymbol)) || regex\_search(person[k], regex\_spaces)) {

std::cout << "Incorrect entry, writing with large letters(A - Z): " << person[k] << std::endl;

std::cin.ignore();

getline(std::cin, person[k]);

}

choice = rand() % 2;

switch (choice) {

case 0:

new\_work = new Homework;

new\_work->generation\_values(person[k]);

add\_elem(new\_work);

break;

case 1:

new\_work = new Classwork;

new\_work->generation\_values(person[k]);

add\_elem(new\_work);

break;

}

}

delete[] person;

fin.close();

}

void IndependentsWork::print() {

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

arr[i]->output();

}

}

void IndependentsWork::add\_elem(InfoIndependentsWork\* new\_work) {

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

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

mas[i] = arr[i];

}

size++;

mas[size - 1] = new\_work;

delete[] arr;

arr = mas;

}

void IndependentsWork::delete\_elem(int l) {

if (size < 2) {

throw Exception("You cant delete last element.", "delete\_elem");

}

if (l - 1 >= size) {

throw Exception("You cant enter index more then size of array.", "delete\_elem");

}

size--;

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

int j = 0;

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

mas[i] = arr[j];

j++;

}

j++;

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

mas[i] = arr[j];

j++;

}

delete arr[l - 1];

delete[]arr;

arr = mas;

}

void IndependentsWork::get\_by\_index(int index) {

if (index - 1 >= size) {

throw Exception("The index cannot be larger than the array size.", "get\_by\_index");

}

std::cout << std::endl;

std::cout << "--------------------------------------------------------------------------------" << std::endl;

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

std::cout << "--------------------------------------------------------------------------------" << std::endl;

std::cout << std::endl << std::endl;

}

void IndependentsWork::write\_to\_file() {

std::ofstream fout;

fout.open("InfoStud.txt");

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

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

}

fout.close();

}

void IndependentsWork::search\_by\_surname(std::string search\_person) {

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

if (search\_person == arr[i]->get\_person()) {

std::cout << std::endl;

std::cout << "--------------------------------------------------------------------------------" << std::endl;

arr[i]->output();

std::cout << "--------------------------------------------------------------------------------" << std::endl;

}

}

}

void IndependentsWork::sort\_by\_mark(bool(\*sort)(int a, int b)) {

InfoIndependentsWork \*temp;

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

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

if (sort(arr[i]->get\_mark(), arr[j]->get\_mark())) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void IndependentsWork::sort\_by\_amount(bool(\*sort)(int a, int b)) {

InfoIndependentsWork \*temp;

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

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

if (sort(arr[i]->get\_amount(), arr[j]->get\_amount())) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void IndependentsWork::sort\_by\_written(bool(\*sort)(int a, int b)) {

InfoIndependentsWork \*temp;

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

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

if (sort(arr[i]->get\_written(), arr[j]->get\_written())) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

Homework.h

#pragma once

#include "InfoIndependentsWork.h"

class Homework : public InfoIndependentsWork {

private:

int homework\_count;

public:

friend std::ostream& operator<< (std::ostream &out, const Homework &obj);

friend std::ostream& operator<< (std::ostream &out, const Homework \*obj);

friend std::istream& operator>> (std::istream &in, Homework &obj);

friend std::istream& operator>> (std::istream &in, Homework \*obj);

bool operator< (const Homework obj);

bool operator> (const Homework obj);

/\*\*

\* Homework class constructors.

\*/

Homework();

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param amount initializes Homework::amount.

\* @param written initializes Homework::written.

\* @param mark initializes Homework::mark.

\* @param person initializes Homework::person.

\* @param homework\_count initializes Homework::homework\_count.

\*/

Homework(int amount, int written, int mark, std::string person, int homework\_count);

/\*\*

\* Copie-constructor.

\* Used initialization lists.

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

\*/

Homework(const Homework &obj);

/\*\*

\* Get copy of field.

\* Return current value.

\*/

int get\_homework();

/\*\*

\* Virtual function of generating values.

\*/

virtual void generation\_values(std::string s) override;

/\*\*

\* 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;

/\*\*

\* Assignment field value.

\*/

void set\_data(int homework\_count, int amount, int written, int mark, std::string person);

/\*\*

\* InfoIndependentsWork class destructor.

\*/

~Homework();

};

Homework.cpp

#include "Homework.h"

Homework::Homework(int amount, int written, int mark, std::string person, int homework\_count) : homework\_count(homework\_count), InfoIndependentsWork() {

}

Homework::Homework() : homework\_count(0) {

}

Homework::Homework(const Homework &obj) : homework\_count(obj.homework\_count), InfoIndependentsWork(obj.amount, obj.written, obj.mark, obj.person) {

}

Homework::~Homework() {

}

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

out << obj.person << ": " << obj.homework\_count << std::endl;

return out;

}

std::ostream& operator<< (std::ostream &out, const Homework \*obj) {

out << obj->person << ": " << obj->homework\_count << std::endl;

return out;

}

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

in >> obj.amount;

in >> obj.written;

in >> obj.mark;

in >> obj.homework\_count;

in >> obj.person;

return in;

}

std::istream& operator>> (std::istream &in, Homework \*obj) {

in >> obj->amount;

in >> obj->written;

in >> obj->mark;

in >> obj->homework\_count;

in >> obj->person;

return in;

}

bool Homework::operator< (const Homework obj) {

return (amount < obj.amount && written < obj.written && mark < obj.mark && homework\_count < obj.homework\_count);

}

bool Homework::operator> (const Homework obj) {

return (amount > obj.amount && written > obj.written && mark > obj.mark && homework\_count > obj.homework\_count);

}

int Homework::get\_homework() {

return Homework::homework\_count;

}

void Homework::generation\_values(std::string s) {

homework\_count = rand() % 16;

amount = 8 + rand() % 8;

written = amount - rand() % 8;

mark = 1 + rand() % 5;

person = s;

}

void Homework::set\_data(int homework\_count, int amount, int written, int mark, std::string person) {

this->homework\_count = homework\_count;

this->person = person;

this->amount = amount;

this->written = written;

this->mark = mark;

}

void Homework::input() {

std::cout << "Enter student person: ";

std::cin.ignore();

getline(std::cin, person);

std::cout << "Enter amount of independent works: ";

std::cin >> amount;

std::cout << "Enter amount of written independent works: ";

std::cin >> written;

std::cout << "Enter student mark (average): ";

std::cin >> mark;

std::cout << "Enter student homework: ";

std::cin >> homework\_count;

std::cout << std::endl;

}

void Homework::output() {

std::cout << "Student info: " << person << std::endl;

std::cout << "Amount of independent works: " << amount << std::endl;

std::cout << "Amount of written independent works: " << written << std::endl;

std::cout << "Student mark (average): " << mark << std::endl;

std::cout << "Student homework: " << homework\_count << std::endl << std::endl;

}

void Homework::output\_to\_file(std::ofstream& file) {

file << "Student info: " << person << std::endl;

file << "Amount of independent works: " << amount << std::endl;

file << "Amount of written independent works: " << written << std::endl;

file << "Student mark (average): " << mark << std::endl ;

file << "Student homework : " << homework\_count << std::endl << std::endl;

}

Classwork.h

#pragma once

#include "InfoIndependentsWork.h"

class Classwork : public InfoIndependentsWork {

private:

int classwork\_count;

public:

friend std::ostream& operator<< (std::ostream &out, const Classwork &obj);

friend std::ostream& operator<< (std::ostream &out, const Classwork \*obj);

friend std::istream& operator>> (std::istream &in, Classwork &obj);

friend std::istream& operator>> (std::istream &in, Classwork \*obj);

bool operator< (const Classwork obj);

bool operator> (const Classwork obj);

/\*\*

\* Classwork class constructors.

\*/

Classwork();

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param amount initializes Classwork::amount.

\* @param written initializes Classwork::written.

\* @param mark initializes Classwork::mark.

\* @param person initializes Classwork::person.

\* @param classwork\_count initializes Classwork::classwork\_count.

\*/

Classwork(int amount, int written, int mark, std::string person, int classwork\_count);

/\*\*

\* Copie-constructor.

\* Used initialization lists.

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

\*/

Classwork(const Classwork &obj);

/\*\*

\* Get copy of field.

\* Return current value.

\*/

int get\_classwork();

/\*\*

\* Virtual function of generating values.

\*/

virtual void generation\_values(std::string s) override;

/\*\*

\* 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;

/\*\*

\* Assignment field value.

\*/

void set\_data(int classwork\_count, int amount, int written, int mark, std::string person);

/\*\*

\* Classwork class destructor.

\*/

~Classwork();

};

Classwork.cpp

#include "Classwork.h"

Classwork::Classwork(int amount, int written, int mark, std::string person, int classwork\_count) : classwork\_count(classwork\_count), InfoIndependentsWork() {

}

Classwork::Classwork() : classwork\_count(0) {

}

Classwork::Classwork(const Classwork &obj) : classwork\_count(obj.classwork\_count), InfoIndependentsWork(obj.amount, obj.written, obj.mark, obj.person) {

}

Classwork::~Classwork() {

}

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

out << obj.person << ": " << obj.classwork\_count << std::endl;

return out;

}

std::ostream& operator<< (std::ostream &out, const Classwork \*obj) {

out << obj->person << ": " << obj->classwork\_count << std::endl;

return out;

}

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

in >> obj.amount;

in >> obj.written;

in >> obj.mark;

in >> obj.classwork\_count;

in >> obj.person;

return in;

}

std::istream& operator>> (std::istream &in, Classwork \*obj) {

in >> obj->amount;

in >> obj->written;

in >> obj->mark;

in >> obj->classwork\_count;

in >> obj->person;

return in;

}

bool Classwork::operator< (const Classwork obj) {

return (amount < obj.amount && written < obj.written && mark < obj.mark);

}

bool Classwork::operator> (const Classwork obj) {

return (amount > obj.amount && written > obj.written && mark > obj.mark);

}

int Classwork::get\_classwork() {

return Classwork::classwork\_count;

}

void Classwork::generation\_values(std::string s) {

classwork\_count = rand() % 16;

amount = 8 + rand() % 8;

written = amount - rand() % 8;

mark = 1 + rand() % 5;

person = s;

}

void Classwork::set\_data(int classwork\_count, int amount, int written, int mark, std::string person) {

this->classwork\_count = classwork\_count;

this->person = person;

this->amount = amount;

this->written = written;

this->mark = mark;

}

void Classwork::input() {

std::cout << "Enter student person: ";

std::cin.ignore();

getline(std::cin, person);

std::cout << "Enter amount of independent works: ";

std::cin >> amount;

std::cout << "Enter amount of written independent works: ";

std::cin >> written;

std::cout << "Enter student mark (average): ";

std::cin >> mark;

std::cout << "Enter student classwork: ";

std::cin >> classwork\_count;

std::cout << std::endl;

}

void Classwork::output() {

std::cout << "Student info: " << person << std::endl;

std::cout << "Amount of independent works: " << amount << std::endl;

std::cout << "Amount of written independent works: " << written << std::endl;

std::cout << "Student mark (average): " << mark << std::endl;

std::cout << "Student classwork: " << classwork\_count << std::endl << std::endl;

}

void Classwork::output\_to\_file(std::ofstream& file) {

file << "Student info: " << person << std::endl;

file << "Amount of independent works: " << amount << std::endl;

file << "Amount of written independent works: " << written << std::endl;

file << "Student mark (average): " << mark << std::endl;

file << "Student homework : " << classwork\_count << std::endl << std::endl;

}

Exception.h

#pragma once

#include <exception>

#include <string>

class Exception : public std::exception {

private:

std::string error;

std::string func\_error;

public:

/\*\*

\* Exception class constructors.

\*/

Exception();

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param error initializes Exception::error.

\* @param func\_error initializes Exception::func\_error.

\*/

Exception(std::string error, std::string func\_error);

/\*\*

\* Get copy of field.

\* Return current value.

\*/

std::string get\_func();

/\*\*

\* Makes information about the error.

\*/

virtual const char\* what() const noexcept override;

};

Exception.cpp

#include"Exception.h"

Exception::Exception() : error(), func\_error() {

}

Exception::Exception(std::string error, std::string func\_error) : error(error), func\_error(func\_error) {

}

std::string Exception::get\_func() {

return func\_error;

}

const char\* Exception::what() const noexcept {

return error.c\_str();

}