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;

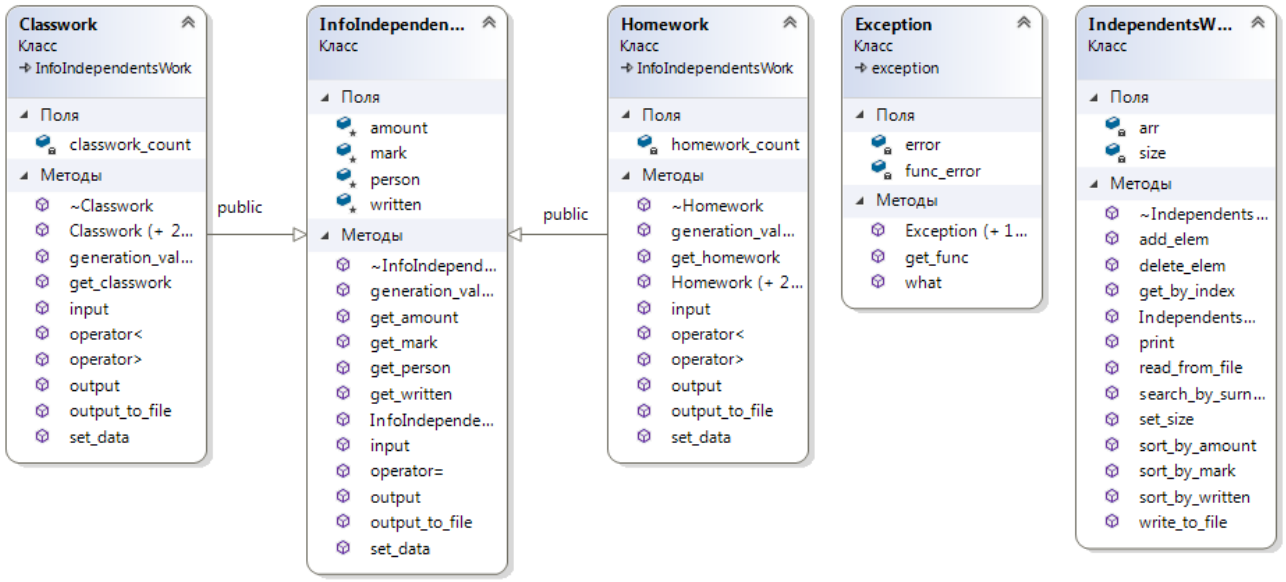
− розробити клас-тестер, основною метою якого буде перевірка коректності роботи класу-контролера.

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

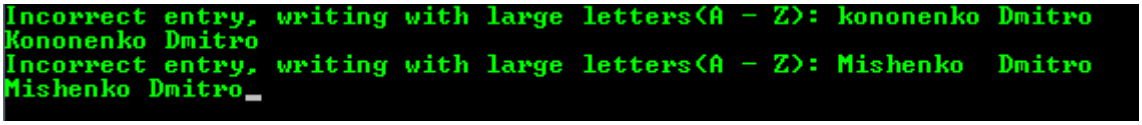


Рисунок 3.1 – Виправлення помилок при зчитуванні з файлу за допомогою регулярних виразів

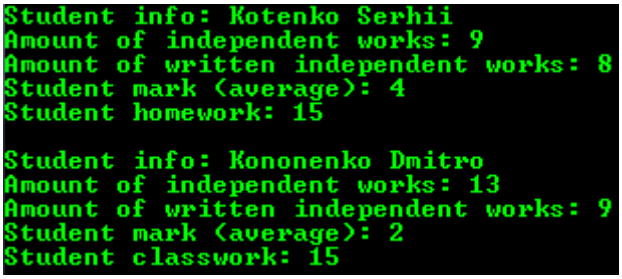


Рисунок 3.2 – Створенні данні з різними полями спадкоємців

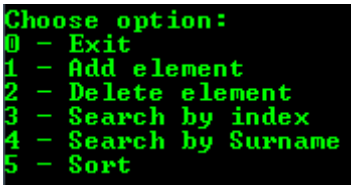


Рисунок 3.3 – Можливі опції роботи з програмою

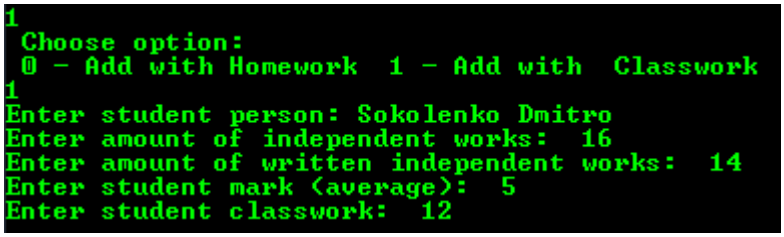


Рисунок 3.4 – Додавання нового елементу

Рисунок 3.5 – Видалення елементу

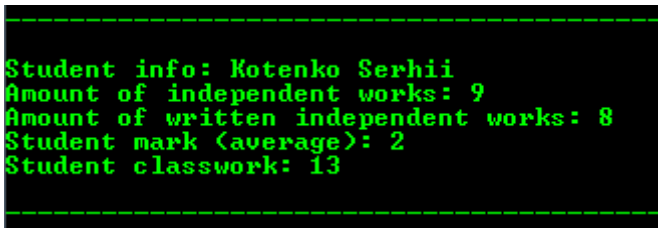


Рисунок 3.6 – Пошук за індексом

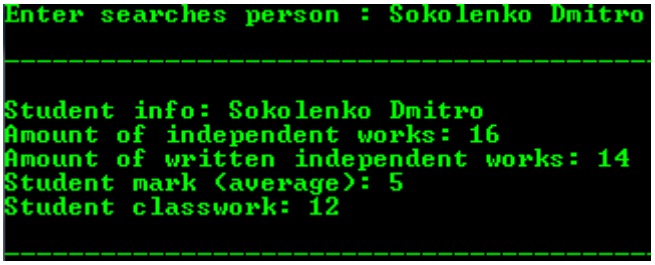


Рисунок 3.7 – Пошук певного студента

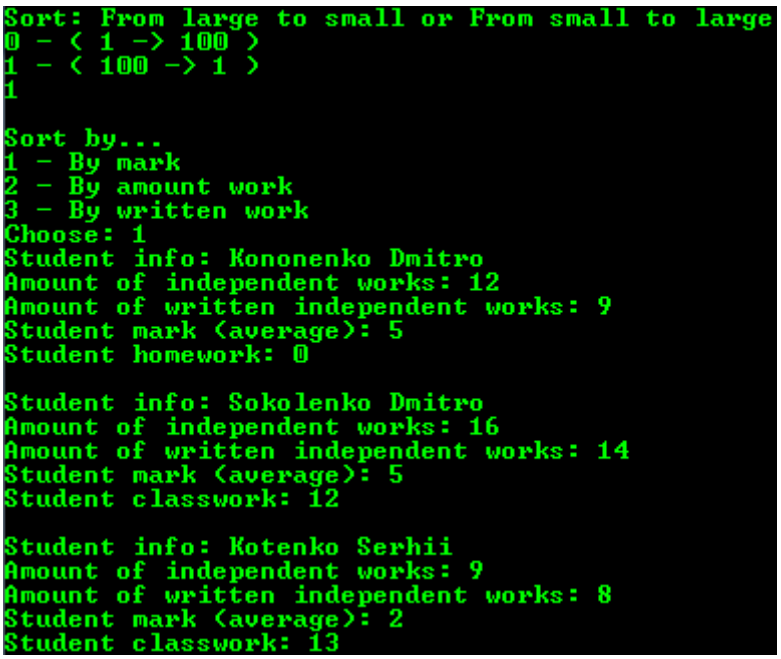


Рисунок 3.8 – Сортування за певним критерієм та напрямком

```
Student info: Kononenko Dmitro
Amount of independent works: 12
Amount of written independent works: 9
Student mark (average): 5
Student homework : 0

Student info: Sokolenko Dmitro
Amount of independent works: 16
Amount of written independent works: 14
Student mark (average): 5
Student homework : 12

Student info: Kotenko Serhii
Amount of independent works: 9
Amount of written independent works: 8
Student mark (average): 2
Student homework : 13
```

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

```cpp
#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

```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();
                                                        }
```