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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Кулик Д.І./

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**

- Кулик Данііл Ігорович;

- Студент групи 1КІТ-102.8(а);

- 09-06-2019р..

**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 % doxygen документований;

− 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;

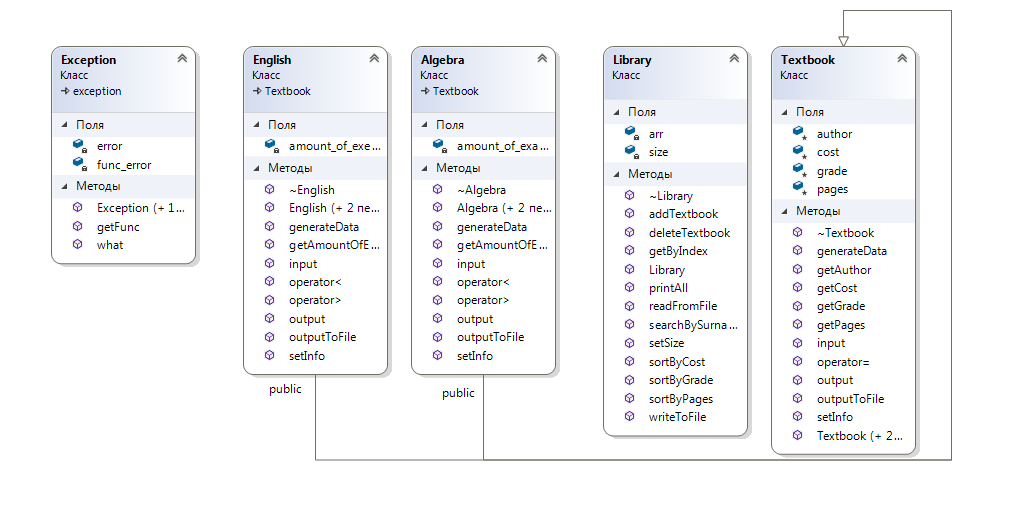
− розробити клас-тестер, основною метою якого буде перевірка коректності роботи класу-контролера.

2 ОПИС ПРОГРАМИ

**2.1 Функціональне призначення**

Програма призначена для виконання комплексних задач з курсу програмування

**2.2 Опис логічної структури**



*Діаграма класу Textbook*:

* ~Textbook - Деструктор класу;
* generateData– Генерація випадкових значень;
* getPages , getGrade, getCost , getAuthor - Отримання даних;
* Textbook- Конструктор класу;
* input – Введення нових даних;
* output – Вивід на екран;
* outputToFile– Вивід даних у файл;
* operator= - Перевантаження оператора присвоювання;
* setInfo - Встановлення значень .

*Діаграма класу Library :*

* ~Library - Деструктор класу;
* addTextbook- Додавання нового елементу;
* deleteTextbook - Видалення елементу;
* Library - Конструктор класу;
* getByIndex - Отримання даних за індексом;
* printAll - Вивід даних на екран;
* readFromFile – Читання даних з файлу;
* searchBySurname – Пошук за прізвищем студента;
* setSize - Отримання розміру для створення масиву;
* sortByPages, sortByGrade, sortByCost – Сортування даних за певним критерієм;
* writeToFile – Запис результату у файл.

*Діаграма класу (спадкоємця) Algebra* :

* ~Algebra - Деструктор класу;
* generateData – Генерація випадкових значень;
* getAmountOfExamples - Отримання даних;
* Algebra- Конструктор класу;
* input – Введення нових даних;
* operator<> - Перевантаження операторів порівняння;
* output – Вивід на екран;
* outputToFile – Вивід даних у файл;
* setInfo - Встановлення значень .

*Діаграма класу (спадкоємця) English* :

* ~English - Деструктор класу;
* generateData – Генерація випадкових значень;
* getAmountOfExamples - Отримання даних;
* English - Конструктор класу;
* input – Введення нових даних;
* operator<> - Перевантаження операторів порівняння;
* output – Вивід на екран;
* outputToFile – Вивід даних у файл;
* setInfo - Встановлення значень .

*Діаграма класу Exception*:

* getFunc- Отримання даних;
* Exception - Конструктор класу;
* what – Відключення базових виключень

**3 ВАРІАНТИ ВИКОРИСТАННЯ**

**3.1 Ілюстрація роботи програми**

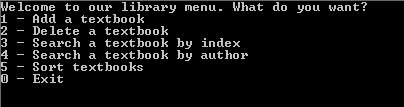


Рисунок 3.1 – меню програми

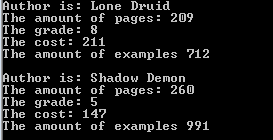


Рисунок 3.2 – Створенні дані із спадкоємцем Algebra

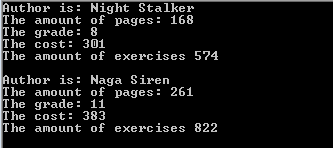


Рисунок 3.3 – Створенні дані із спадкоємцем English

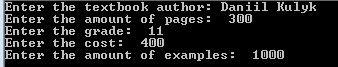


Рисунок 3.4 – Додавання нового елементу



Рисунок 3.5 – Видалення елементу

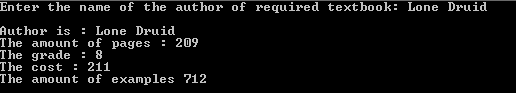


Рисунок 3.6 – Пошук підручника за автором

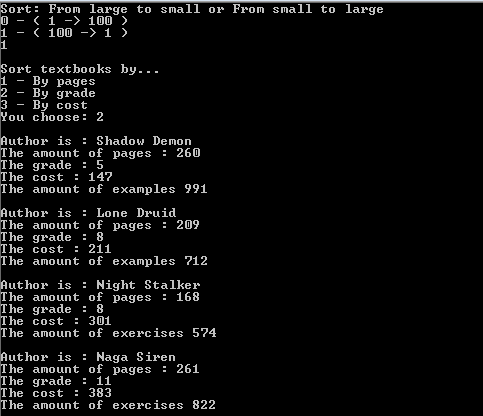


Рисунок 3.7 – Сортування за певним критерієм та напрямком

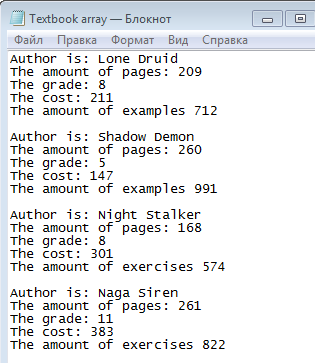


Рисунок 3.8 – Записаний результат у файл

**ВИСНОВОК**

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

Main.cpp

/\*

\* @ mainpage

\* @ author - Kulyk Daniil

\* @ date - 09.06.19

\* @ version - 1.0

\*/

#include "Library.h"

#include "Algebra.h"

#include "English.h"

#include "Exception.h"

bool sort(int a, int b) {

return a < b;

}

bool sort2(int a, int b) {

return a > b;

}

int main() {

try {

auto i = 0;

cout << "Enter the size : ";

cin >> i;

Library textbook;

regex regex\_space("[\\s]{2,}");

regex regex\_upperRegister("^[A-Z]");

system("cls");

textbook.readFromFile(i);

system("cls");

textbook.printAll();

Textbook\* new\_textbook;

int option = 0;

do {

cout << "Welcome to our library menu. What do you want? " << endl << "1 - Add a textbook" << endl << "2 - Delete a textbook" << endl

<< "3 - Search a textbook by index" << endl << "4 - Search a textbook by author" << endl << "5 - Sort textbooks" << endl << "0 - Exit " << endl << endl;

cin >> option;

switch (option) {

case 1: {

cout << " Choose option: " << endl << " 0 - Add Algebra " << " 1 - Add English " << endl;

cin >> option;

switch (option) {

case 0: {

new\_textbook = new Algebra;

new\_textbook->input();

textbook.addTextbook(new\_textbook);

break;

}

case 1: {

new\_textbook = new English;

new\_textbook->input();

textbook.addTextbook(new\_textbook);

break;

}

}

system("cls");

textbook.printAll();

break;

}

case 2: {

auto num = 0;

cout << endl << "Enter the number of textbook you want to remove: ";

cin >> num;

cout << endl;

textbook.deleteTextbook(num);

system("cls");

textbook.printAll();

break;

}

case 3: {

auto index = 0;

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

cin >> index;

cout << endl;

system("cls");

textbook.printAll();

textbook.getByIndex(index);

break;

}

case 4: {

system("cls");

string search\_surname;

cout << "Enter the name of required author: ";

cin.ignore();

getline(cin, search\_surname);

textbook.searchBySurname(search\_surname);

break;

}

case 5: {

system("cls");

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

int s;

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

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

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

cin >> s;

if (s == 0) {

pointer = sort;

}

else {

pointer = sort2;

}

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

cout << "1 - By pages" << endl;

cout << "2 - By grade" << endl;

cout << "3 - By cost" << endl;

cout << "You choose: ";

cin >> s;

switch (s) {

case 1:

textbook.sortByPages(pointer);

textbook.printAll();

break;

case 2:

textbook.sortByGrade(pointer);

textbook.printAll();

break;

case 3:

textbook.sortByCost(pointer);

textbook.printAll();

break;

}

break;

default:

break;

}

}

} while (option != 0);

textbook.writeToFile();

system("cls");

}

catch (Exception& exception) {

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

}

catch (std::exception& exception) {

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

}

catch (...) {

cout << "Unknown error!" << 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");

}

Textbook.h

#pragma once

#include <iostream>

#include <ctime>

#include <fstream>

#include <string>

#include <cstdio>

#include <regex>

using std::cout;

using std::cin;

using std::endl;

using std::regex;

using std::string;

using std::getline;

using std::istream;

using std::ostream;

using std::ofstream;

using std::ifstream;

class Textbook {

protected:

int pages;

int grade;

int cost;

string author;

public:

/\*\*

\* InfoIndependentsWork class constructors.

\*/

Textbook();

/\*\*

\* Copie-constructor.

\* Used initialization lists.

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

\*/

Textbook(const Textbook &obj);

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param pages initializes Textbook::pages.

\* @param grade initializes Textbook::grade.

\* @param cost initializes Textbook::cost.

\* @param author initializes Textbook::author.

\*/

Textbook(int pages, int grade, int cost, string author);

/\*\*

\* Operator overload = .

\*/

Textbook& operator= (const Textbook &obj);

/\*\*

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

\*/

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

friend ostream& operator<< (ostream &out, const Textbook \*obj);

/\*\*

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

\*/

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

friend istream& operator>> (istream &in, Textbook \*obj);

/\*\*

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

\*/

virtual void input() = 0;

virtual void output() = 0;

virtual void outputToFile(ofstream& file) = 0;

/\*\*

\* Virtual function to generate values.

\*/

virtual void generateData(string s);

/\*\*

\* Get copy of field.

\* Return current value.

\*/

int getPages();

int getGrade();

int getCost();

string getAuthor();

/\*\*

\* Assignment field value.

\*/

void setInfo(int pages, int grade, int cost, string author);

/\*\*

\* Textbookk class destructor.

\*/

~Textbook();

};

Textbook.cpp

#include "Textbook.h"

Textbook::Textbook() :pages(0), grade(0), cost(0) {

}

Textbook::Textbook(const Textbook &obj) : pages(obj.pages),grade(obj.grade), cost(obj.cost) {

}

Textbook::Textbook(int pages, int grade, int cost, string author) : pages(pages), grade(grade), cost(cost), author(author) {

}

ostream& operator<< (ostream &out, const Textbook &obj) {

out << obj.author << ": " << obj.pages << endl;

out << obj.author << ": " << obj.grade << endl;

out << obj.author << ": " << obj.cost << endl;

return out;

}

ostream& operator<< (ostream &out, const Textbook \*obj) {

out << obj->author << ": " << obj->pages << endl;

out << obj->author << ": " << obj->grade << endl;

out << obj->author << ": " << obj->cost << endl;

return out;

}

istream& operator>> (istream &in, Textbook &obj) {

in >> obj.pages;

in >> obj.grade;

in >> obj.cost;

in >> obj.author;

return in;

}

istream& operator>> (istream &in, Textbook \*obj) {

in >> obj->pages;

in >> obj->grade;

in >> obj->cost;

in >> obj->author;

return in;

}

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

pages = obj.pages;

grade = obj.grade;

cost = obj.cost;

return \*this;

}

Textbook::~Textbook() {

};

int Textbook::getPages() {

return Textbook::pages;

}

int Textbook::getGrade() {

return Textbook::grade;

}

int Textbook::getCost() {

return Textbook::cost;

}

string Textbook::getAuthor() {

return Textbook::author;

}

void Textbook::generateData(string s) {

this->pages = rand() % 220 + 100;

this->grade = rand() % 11 + 1;

this->cost = rand() % 300 + 100;

this->author = s;

}

void Textbook::setInfo(int pages, int grade, int cost, string author) {

this->author = author;

this->pages = pages;

this->grade = grade;

this->cost = cost;

}

Library.h

#pragma once

#include "Textbook.h"

class Library {

private:

int size;

Textbook \*\*arr;

public:

/\*\*

\* Library class constructors.

\*/

Library();

/\*\*

\* Library class destructor.

\*/

~Library();

/\*\*

\* Overloading of the output operator.

\*/

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

/\*\*

\* Overloading of the input operator.

\*/

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

void setSize(int size);

void printAll();

void addTextbook(Textbook\* new\_textbook);

void deleteTextbook(int ind);

void getByIndex(int index);

void readFromFile(int new\_size);

void writeToFile();

void searchBySurname(string search\_surname);

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

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

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

};

Library.cpp

#include "Library.h"

#include "Algebra.h"

#include "English.h"

#include "Exception.h"

Library::Library() : size(0) {

arr = nullptr;

}

ostream& operator<< (ostream &out, const Library &obj) {

out << obj.size << endl;

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

out << obj.arr[i];

}

return out;

}

istream& operator>> (istream &in, Library &obj) {

in >> obj.size;

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

in >> obj.arr[i];

}

return in;

}

Library::~Library() {

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

delete arr[i];

}

delete[] arr;

}

void Library::setSize(int size) {

Library::size = size;

}

void Library::readFromFile(int new\_size) {

srand(time(NULL));

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

regex regex\_upperRegister("^[A-Z]");

regex regex\_space("[\\s]{2,}");

ifstream fin;

fin.open("Textbook authors.txt");

if (!fin) {

throw Exception("Can't open file for reading", "readFromFile");

}

Textbook\* new\_textbook;

int choice;

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

getline(fin, person[i]);

if (!(regex\_search(person[i], regex\_upperRegister)) || regex\_search(person[i], regex\_space)) {

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

cin.ignore();

getline(cin, person[i]);

}

choice = rand() % 2;

switch (choice) {

case 0:

new\_textbook = new Algebra;

new\_textbook->generateData(person[i]);

addTextbook(new\_textbook);

break;

case 1:

new\_textbook = new English;

new\_textbook->generateData(person[i]);

addTextbook(new\_textbook);

break;

}

}

delete[] person;

fin.close();

}

void Library::printAll() {

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

arr[i]->output();

}

}

void Library::addTextbook(Textbook\* new\_textbook) {

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

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

mas[i] = arr[i];

}

size++;

mas[size - 1] = new\_textbook;

delete[] arr;

arr = mas;

}

void Library::deleteTextbook(int ind) {

if (size < 2) {

throw Exception("You can't delete last element.", "deleteTextbook");

}

if (ind - 1 >= size) {

throw Exception("You can't enter the index more then the size of array.", "deleteTextbook");

}

size--;

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

int j = 0;

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

mas[i] = arr[j];

j++;

}

j++;

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

mas[i] = arr[j];

j++;

}

delete arr[ind - 1];

delete[]arr;

arr = mas;

}

void Library::getByIndex(int index) {

if (index - 1 >= size) {

throw Exception("The index cannot be larger than the array size.", "getByIndex");

}

cout << endl;

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

cout << "--------------------------------------------------------------------------------" << endl;

cout << endl << endl;

}

void Library::writeToFile() {

ofstream fout;

fout.open("Textbook array.txt");

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

arr[i]->outputToFile(fout);

}

fout.close();

}

void Library::searchBySurname(string search\_surname) {

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

if (search\_surname == arr[i]->getAuthor()) {

cout << endl;

arr[i]->output();

cout << "--------------------------------------------------------------------------------" << endl;

}

}

}

void Library::sortByPages(bool(\*sort)(int a, int b)) {

Textbook \*temp;

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

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

if (sort(arr[i]->getPages(), arr[j]->getPages())) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void Library::sortByGrade(bool(\*sort)(int a, int b)) {

Textbook \*temp;

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

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

if (sort(arr[i]->getGrade(), arr[j]->getGrade())) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void Library::sortByCost(bool(\*sort)(int a, int b)) {

Textbook \*temp;

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

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

if (sort(arr[i]->getCost(), arr[j]->getCost())) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

Algebra.h

#pragma once

#include "Textbook.h"

class Algebra : public Textbook {

private:

int amount\_of\_examples;

public:

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

friend ostream& operator<< (ostream &out, const Algebra \*obj);

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

friend istream& operator>> (istream &in, Algebra \*obj);

bool operator< (const Algebra obj);

bool operator> (const Algebra obj);

/\*\*

\* Algebra class constructors.

\*/

Algebra();

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param pages initializes Algebra::pages.

\* @param grade initializes Algebra::grade.

\* @param cost initializes Algebra::cost.

\* @param author initializes Algebra::author.

\* @param amount\_of\_examples initializes Algebra::amount of examples.

\*/

Algebra(int pages, int grade, int cost, string author, int amount\_of\_examples);

/\*\*

\* Copie-constructor.

\* Used initialization lists.

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

\*/

Algebra(const Algebra &obj);

/\*\*

\* Get copy of field.

\* Return current value.

\*/

/\*\*

\* Algebra class destructor.

\*/

~Algebra();

int getAmountOfExamples();

/\*\*

\* Virtual function of generating values.

\*/

virtual void generateData(string s) override;

/\*\*

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

\*/

virtual void input() override;

virtual void output() override;

virtual void outputToFile(ofstream& file) override;

/\*\*

\* Assignment field value.

\*/

void setInfo(int amount\_of\_examples, int pages, int grade, int mark, string author);

};

Algebra.cpp

#include "Algebra.h"

Algebra::Algebra(int pages, int grade, int cost, string author, int amount\_of\_examples) : amount\_of\_examples(amount\_of\_examples), Textbook() {

}

Algebra::Algebra() : amount\_of\_examples(0) {

}

Algebra::Algebra(const Algebra &obj) : amount\_of\_examples(obj.amount\_of\_examples), Textbook(obj.pages, obj.grade, obj.cost, obj.author) {

}

Algebra::~Algebra() {

}

ostream& operator<< (ostream &out, const Algebra &obj) {

out << obj.author << ": " << obj.amount\_of\_examples << endl;

return out;

}

ostream& operator<< (ostream &out, const Algebra \*obj) {

out << obj->author << ": " << obj->amount\_of\_examples << endl;

return out;

}

istream& operator>> (istream &in, Algebra &obj) {

in >> obj.pages;

in >> obj.grade;

in >> obj.cost;

in >> obj.amount\_of\_examples;

in >> obj.author;

return in;

}

istream& operator>> (istream &in, Algebra \*obj) {

in >> obj->pages;

in >> obj->grade;

in >> obj->cost;

in >> obj->amount\_of\_examples;

in >> obj->author;

return in;

}

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

return (pages < obj.pages && grade < obj.grade && cost < obj.cost && amount\_of\_examples < obj.amount\_of\_examples);

}

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

return (pages > obj.pages && grade > obj.grade && cost > obj.cost && amount\_of\_examples > obj.amount\_of\_examples);

}

int Algebra::getAmountOfExamples() {

return Algebra::amount\_of\_examples;

}

void Algebra::generateData(string s) {

amount\_of\_examples = rand() % 400 + 600;

this->pages = rand() % 220 + 100;

this->grade = rand() % 11 + 1;

this->cost = rand() % 300 + 100;

this->author = s;

}

void Algebra::setInfo(int amount\_of\_examples, int pages, int grade, int cost, string author) {

this->amount\_of\_examples = amount\_of\_examples;

this->author = author;

this->pages = pages;

this->grade = grade;

this->cost = cost;

}

void Algebra::input() {

cout << "Enter the textbook author: ";

cin.ignore();

getline(cin, author);

cout << "Enter the amount of pages: ";

cin >> pages;

cout << "Enter the grade: ";

cin >> grade;

cout << "Enter the cost: ";

cin >> cost;

cout << "Enter the amount of examples: ";

cin >> amount\_of\_examples;

cout << endl;

}

void Algebra::output() {

cout << "Author is: " << author << endl;

cout << "The amount of pages: " << pages << endl;

cout << "The grade: " << grade << endl;

cout << "The cost: " << cost << endl;

cout << "The amount of examples " << amount\_of\_examples << endl << endl;

}

void Algebra::outputToFile(ofstream& file) {

file << "Author is: " << author << endl;

file << "The amount of pages: " << pages << endl;

file << "The grade: " << grade << endl;

file << "The cost: " << cost << endl;

file << "The amount of examples " << amount\_of\_examples << endl << endl;

}  
  
  
English.h

#pragma once

#include "Textbook.h"

class English : public Textbook {

private:

int amount\_of\_exercises;

public:

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

friend ostream& operator<< (ostream &out, const English \*obj);

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

friend istream& operator>> (istream &in, English \*obj);

bool operator< (const English obj);

bool operator> (const English obj);

/\*\*

\* English class constructors.

\*/

English();

/\*\*

\* Constructor with parameters.

\* Used initialization lists.

\* @param pages initializes English::pages.

\* @param grade initializes English::grade.

\* @param cost initializes English::cost.

\* @param author initializes English::author.

\* @param amount\_of\_examples initializes English::amount of examples.

\*/

English(int pages, int grade, int cost, string author, int amount\_of\_exercises);

/\*\*

\* Copie-constructor.

\* Used initialization lists.

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

\*/

English(const English &obj);

/\*\*

\* Get copy of field.

\* Return current value.

\*/

/\*\*

\* English class destructor.

\*/

~English();

int getAmountOfExercises();

/\*\*

\* Virtual function of generating values.

\*/

virtual void generateData(string s) override;

/\*\*

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

\*/

virtual void input() override;

virtual void output() override;

virtual void outputToFile(ofstream& file) override;

/\*\*

\* Assignment field value.

\*/

void setInfo(int amount\_of\_exercises, int pages, int grade, int mark, string author);

};

English.cpp

#include "English.h"

English::English(int pages, int grade, int cost, string author, int amount\_of\_exercises) : amount\_of\_exercises(amount\_of\_exercises), Textbook() {

}

English::English() : amount\_of\_exercises(0) {

}

English::English(const English &obj) : amount\_of\_exercises(obj.amount\_of\_exercises), Textbook(obj.pages, obj.grade, obj.cost, obj.author) {

}

English::~English() {

}

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

out << obj.author << ": " << obj.amount\_of\_exercises << std::endl;

return out;

}

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

out << obj->author << ": " << obj->amount\_of\_exercises << std::endl;

return out;

}

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

in >> obj.pages;

in >> obj.grade;

in >> obj.cost;

in >> obj.amount\_of\_exercises;

in >> obj.author;

return in;

}

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

in >> obj->pages;

in >> obj->grade;

in >> obj->cost;

in >> obj->amount\_of\_exercises;

in >> obj->author;

return in;

}

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

return (pages < obj.pages && grade < obj.grade && cost < obj.cost && amount\_of\_exercises < obj.amount\_of\_exercises);

}

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

return (pages > obj.pages && grade > obj.grade && cost > obj.cost && amount\_of\_exercises > obj.amount\_of\_exercises);

}

int English::getAmountOfExercises() {

return English::amount\_of\_exercises;

}

void English::generateData(string s) {

amount\_of\_exercises = rand() % 200 + 150;

pages = 200 + 100;

grade = rand() % 11 + 1;

cost = rand() % 300 + 100;

author = s;

}

void English::setInfo(int amount\_of\_exercises, int pages, int grade, int cost, string author) {

this->amount\_of\_exercises = amount\_of\_exercises;

this->author = author;

this->pages = pages;

this->grade = grade;

this->cost = cost;

}

void English::input() {

cout << "Enter the textbook author: ";

cin.ignore();

getline(cin, author);

cout << "Enter the amount of pages: ";

cin >> pages;

cout << "Enter the grade: ";

cin >> grade;

cout << "Enter the cost: ";

cin >> cost;

cout << "Enter the amount of exercises: ";

cin >> amount\_of\_exercises;

cout << endl;

}

void English::output() {

cout << "Author is: " << author << endl;

cout << "The amount of pages: " << pages << endl;

cout << "The grade: " << grade << endl;

cout << "The cost: " << cost << endl;

cout << "The amount of exercises " << amount\_of\_exercises << endl << endl;

}

void English::outputToFile(ofstream& file) {

file << "Author is: " << author << endl;

file << "The amount of pages: " << pages << endl;

file << "The grade: " << grade << endl;

file << "The cost: " << cost << endl;

file << "The amount of exercises " << amount\_of\_exercises << endl << endl;

}  
  
Exception.h

#pragma once

#include <exception>

#include <string>

using std::string;

using std::exception;

class Exception : public exception {

private:

string error;

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(string error, string func\_error);

/\*\*

\* Get copy of field.

\* Return current value.

\*/

string getFunc();

/\*\*

\* Makes information about the error.

\*/

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

};

};

Exception.cpp

#include"Exception.h"

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

}

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

}

string Exception::getFunc() {

return func\_error;

}

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

return error.c\_str();

}