

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

1KIT102.8A.18038-01 81 01-1 –АЗ

Розрахункове завдання з дисципліни
«Алгоритми та структури даних»

Пояснювальна записка
1KIT.102.8A.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 ІКІТ-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;

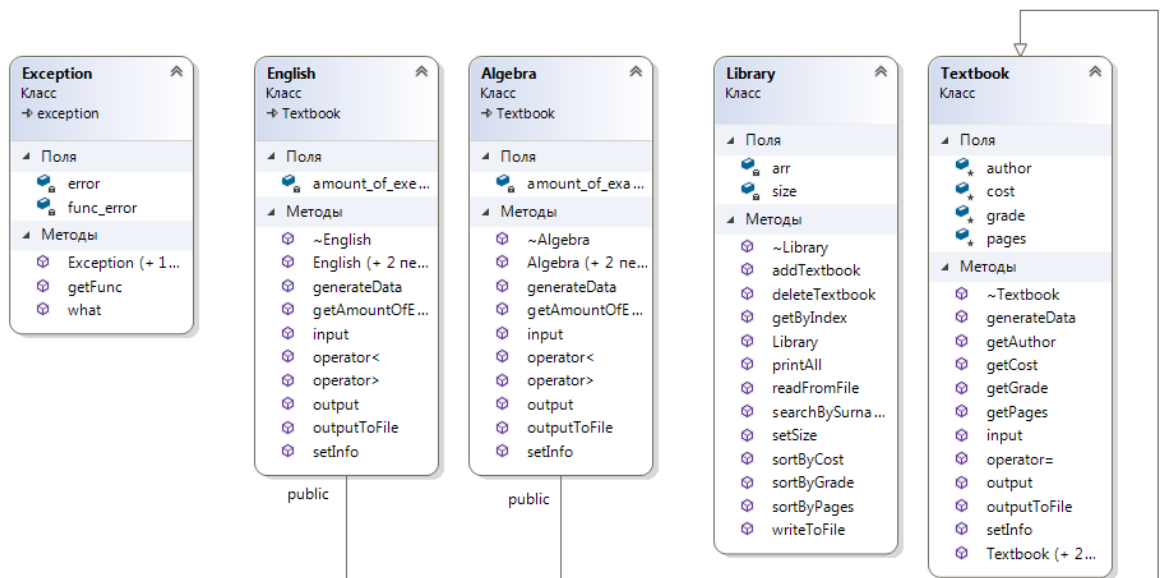
– розробити клас-тестер, основною метою якого буде перевірка коректності роботи класу-контролера.

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 присвоєння;
- ✓ 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 – Сортуювання 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

```
Welcome to our library menu. What do you want?
1 - Add a textbook
2 - Delete a textbook
3 - Search a textbook by index
4 - Search a textbook by author
5 - Sort textbooks
0 - Exit
```

Рисунок 3.1 – меню програми

```
Author is: Lone Druid
The amount of pages: 209
The grade: 8
The cost: 211
The amount of examples 712

Author is: Shadow Demon
The amount of pages: 260
The grade: 5
The cost: 147
The amount of examples 991
```

Рисунок 3.2 – Створенні дані із спадкоємцем Algebra

```

Author is: Night Stalker
The amount of pages: 168
The grade: 8
The cost: 301
The amount of exercises 574

Author is: Naga Siren
The amount of pages: 261
The grade: 11
The cost: 383
The amount of exercises 822

```

Рисунок 3.3 – Створенні дані із спадкоємцем English

```

Enter the textbook author: Daniil Kulyk
Enter the amount of pages: 300
Enter the grade: 11
Enter the cost: 400
Enter the amount of examples: 1000

```

Рисунок 3.4 – Додавання нового елементу

```

2
Enter the number of textbook you want to remove: 5

```

Рисунок 3.5 – Видалення елементу

```

Enter the name of the author of required textbook: Lone Druid
Author is : Lone Druid
The amount of pages : 209
The grade : 8
The cost : 211
The amount of examples 712

```

Рисунок 3.6 – Пошук підручника за автором

```

Sort: From large to small or From small to large
0 - < 1 -> 100 >
1 - < 100 -> 1 >
1

Sort textbooks by...
1 - By pages
2 - By grade
3 - By cost
You choose: 2

Author is : Shadow Demon
The amount of pages : 260
The grade : 5
The cost : 147
The amount of examples 991

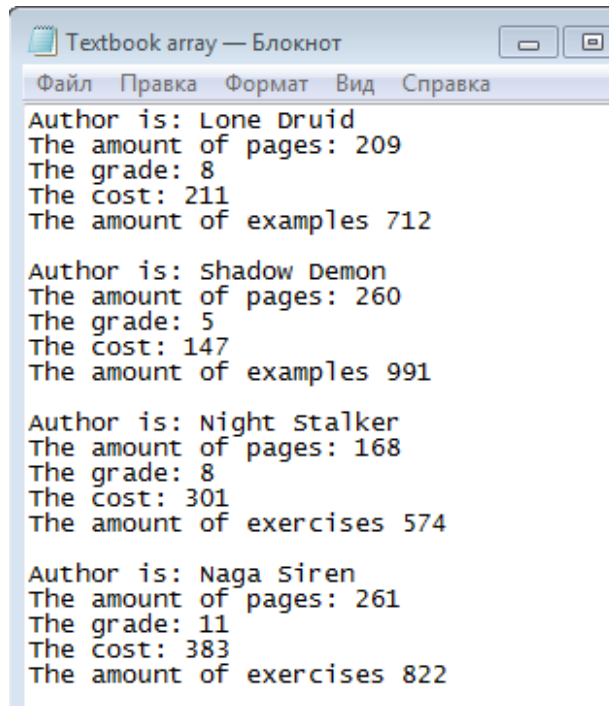
Author is : Lone Druid
The amount of pages : 209
The grade : 8
The cost : 211
The amount of examples 712

Author is : Night Stalker
The amount of pages : 168
The grade : 8
The cost : 301
The amount of exercises 574

Author is : Naga Siren
The amount of pages : 261
The grade : 11
The cost : 383
The amount of exercises 822

```

Рисунок 3.7 – Сортування за певним критерієм та напрямком



```
Textbook array — Блокнот
Файл  Правка  Формат  Вид  Справка

Author is: Lone Druid
The amount of pages: 209
The grade: 8
The cost: 211
The amount of examples 712

Author is: Shadow Demon
The amount of pages: 260
The grade: 5
The cost: 147
The amount of examples 991

Author is: Night stalker
The amount of pages: 168
The grade: 8
The cost: 301
The amount of exercises 574

Author is: Naga Siren
The amount of pages: 261
The grade: 11
The cost: 383
The amount of exercises 822
```

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();
}

```