МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ “ХПІ”

Кафедра “Обчислювальна техніка та програмування”

Розрахункове завдання з дисципліни

«Основи програмування ч.2»

Пояснювальна записка

ЄСПД ГОСТ 19.404–79(СТЗВО – ХПІ – 30.05-2021 ССОНП)

КІТ.120А.17-01 90 01-1 -ЛЗ

Виконав:

студент групи КІТ-120А

Клименко Юрій Юрійович

Перевірив:

Давидов В'ячеслав Вадимович

Харків 2021

**Розрахункове завдання**

***Тема:*** *Розробка інформаційно-довідкової системи*

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

1. **Призначення та галузь застосування**

**Інформацíйна систéма** (англ. *Information system*) — сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів.

Необхідність накопичення великих об’ємів професійно цінної інформації і оперування ними – одна із проблем, з якою зіштовхується майже кожна наукова галузь.

Інформаційно-довідкові системи полегшують розв’язання цієї проблеми, виступаючи як засіб надійного збереження професійних знань, забезпечує зручний і швидкий пошук необхідних відомостей.

Розроблена мною інформаційна система має колекцію книг та методи роботи з нею. З загального та індивідуального завдання колекція має методи: пошук книг видавництва «Ранок», пошук детективу, що має онлайн версію та пошук книги з найбільшою кількістю сторінок. Також є можливість сортування колекції залежно від заданого користувачем напрямку та по вказаному критерію книги. Є також методи, які дають змогу: видалити задану користувачем книгу з колекції; очистити колекцію книг; додати книгу до колекції; замінити або ж отримати книгу по індексу.

Дану інформаційну систему можна застосовувати в різних цілях. Наприклад, у книжковому магазині або в бібліотеці.

1. **Постановка завдання до розробки**

**1.1 Загальне завдання**

1. З розділу ”Розрахункове завдання / Індивідуальні завдання”, відповідно до варіанта завдання(15), обрати прикладну галузь;
2. Для прикладної галузі розробити розгалужену ієрархію класів, що описана у завдані та складається з одного базового класу та двох спадкоємців. Класи повинні мати перевантажені оператори введення-виведення даних та порівняння;
3. З. Розробити клас-список List.[h/cpp], що буде включати до себе масив (STL-колекцію) вказівників до базового класу. А також базові методи роботи з списком: а) очистка списку б) відображення списку в) додання/видалення/отримання/оновлення елементу;
4. Розробити клас-контролер Controller.[h/cpp], що буде включати колекцію розроблених класів, та наступні методи роботи з цією колекцією: а) читання даних з файлу та ix запис у контейнер (STL-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 вказаними критеріями (три критерія, щоприсутні у кожному варіанті);
5. Розробити клас Menu.[h/cpp], який має відображати діалогове меню для демонстрації реалізованих функцій класу контролера;
6. Оформити схеми алгоритмів функцій класів контролера (за необхідністю), тесту-контролера та діалогового меню;
7. Оформити документацію: пояснювальну записку.

**Додаткові вимоги на оцінку «відмінно»:**

* виконати перевірку вхідних даних за допомогою регулярних виразів.
* критерій для пошуку та сортування задавати у вигляді функтора;
* розробити клас-тестер контролеру ControllerTest.срр, основною метою якого буде перевірка коректності роботи класу-контролера.

**1.2 Індивідуальне завдання**

**–** Варіант 15. "Книга"

* Поля базового класу:
* Чи є електронна версія (наприклад: так, ні)
* Назва (наприклад: Пригоди Тома Сойера)
* Кількість сторінок (наприклад: 330, 510)
* Видавництво (структура, що містить назву видавництва та версію видання)
* Палітурка (одна з переліку: тверда, м’яка)
* Спадкоємець 1 – Художня книга. Додаткові поля:
* Напрям (один з переліку: відродження, модерн, постмодерн)
* Жанр (один з переліку: роман, детектив, новела, повість)
* Спадкоємець 2 – Наукова книга. Додаткові поля:
* Сфера (одна з переліку: хімія, біологія, фізика)
* Чи є сертифікованою (наприклад: так, ні)
* Методи роботи з колекцією:

1. Знайти всі книги видавництва «Ранок».
2. Знайти детективи, що мають електронну версію.
3. Знайти книгу з найбільшою кількістю сторінок.
4. **Опис вхідних та вихідних даних**

**3.1 Опис вхідних даних**

Під час запуску програми, відкривається файл **books.txt**, звідки будуть взяті вхідні дані. В файлі повинні бути наступні дані: першим повинен бути символ (‘F’ чи ‘S’ ), котрий позначає тип вхідного об’єкту (‘F’ – художня книга, ‘S’ – наукова книга), далі цифра 1 чи 0, що позначає чи наявна цифрова версія(1 – так, 0 – ні), потім назва книги, кількість сторінок, назва видавництва та версія видання, палітурка(0 – тверда, 1 – м’яка), напрям(0 – відродження, 1 – модерн, 2 – постмодерн --- для художньої книги), жанр(0 – роман, 1 – детектив, 2 – новела, 3 – повість --- для художньої книги), сфера(0 – хімія, 1 – біологія, 2 – фізика, 3 – інформатика --- для наукової книги), чи є сертифікованою(0 – ні, 1 – так --- для наукової книги) . Приклад файлу з вхідними даними зображено на рисунку 1.

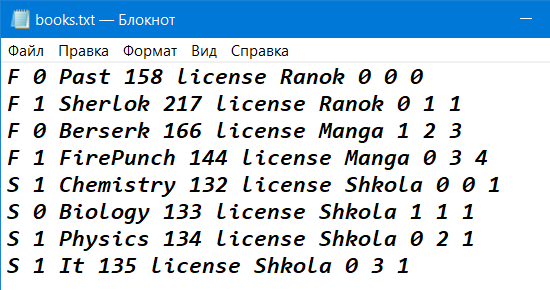


Рисунок 1 – Приклад вхідного файлу

**3.2 Опис вихідних даних**

Вихідні данні записуються у вказаний користувачем файл, в тому ж порядку, в якому були задані у вхідному файлі. Приклад файлу з вихідними даними дивись на рисунку 2.

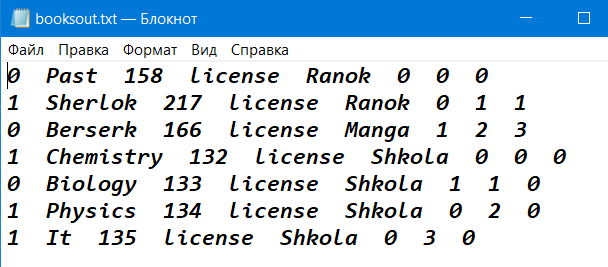


Рисунок 2 – Приклад вихідного файлу

1. **Опис складу технічних та програмних засобів**

**4.1 Функціональне призначення**

Програма виводить меню можливих дій с колекцією, та в залежності від отриманих від користувача даних виконує методи із загального та індивідуально завдань.

**4.2 Опис логічної структури програми**

*Головна функція* main() створює клас-меню Menu та викликає метод menu.User\_menu().

*Метод menu.User\_menu()*виводить на екран діалогове меню, та отримує від користувача номер дії, яку необхідно виконати з колекцією. І в залежності від отриманої цифри викликає відповідний метод роботи з колекцією.

*Метод addBook(Book \*book)* додає елемент до колекції. Приймає об’єкт, який необхідно додати до колекції. *Схема алгоритму методу* подана на рис. 3.(ст. 8)

*Метод DeleteElement(int index)* видаляє елемент з колекції по індексу. Приймає індекс, звідки необхідно видалити об’єкт. *Схема алгоритму методу* подана на рис. 4.(ст. 9)

*Метод ReadFromFile(const string &path)* зчитує колекцію із заданого файлу. Приймає строку – путь до файлу з колекцією, яку необхідно записати в STL-контейнер. *Схема алгоритму методу* подана на рис. 5.(ст. 10)

*Метод findRanokBook()* виконує «метод 1» з індивідуального завдання. Метод за допомогою відповідного предиката знаходить серед колекції усі книги видавництва «Ранок», записує їх у вектор та повертає цей вектор зі знайденими книгами.(Реалізація методу – див. Додаток А) *Схема алгоритму методу* подана на рис. 6.(ст. 11)

*Метод findOnlineDetective()* виконує «метод 2» з індивідуального завдання. Метод за допомогою відповідного предиката знаходить серед колекції усі детективи, що мають цифрову версію, записує їх у вектор та повертає цей вектор зі знайденими книгами. (Реалізація методу – див. Додаток Б) *Схема алгоритму методу* подана на рис. 7.(ст. 12)

*Метод findMaxPages()* виконує «метод 3» з індивідуального завдання. Метод за допомогою відповідного функтора знаходить серед колекції книгу з найбільшою кількістю сторінок, записує її у вектор та повертає цей вектор зі знайденим рюкзаком. (Реалізація методу – див. Додаток В) *Схема алгоритму методу* подана на рис. 8.(ст. 13)

*Метод SortByField (string field)* виконує сортування колекції за заданим критерієм та напрямом. Метод приймає критерій сортування та виконує сортування за допомогою функтора відповідного до заданого критерія. (Реалізація методу – дивись Додаток Г)  *Схема алгоритму методу* подана на рис. 9.(ст. 14, а краще **4.png** в директорії **assets**)

*Метод WriteToFile()* записує колекцію в файл. Метод виконує запит у користувача за ім’я файлу, куди буде записаний результат роботи з колекцією, та виконує запис у файл. *Схема алгоритму методу* подана на рис. 10.(ст. 15)

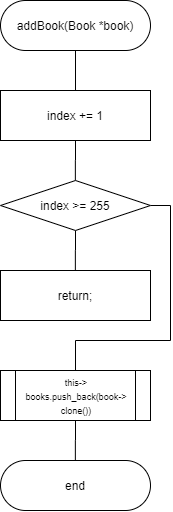
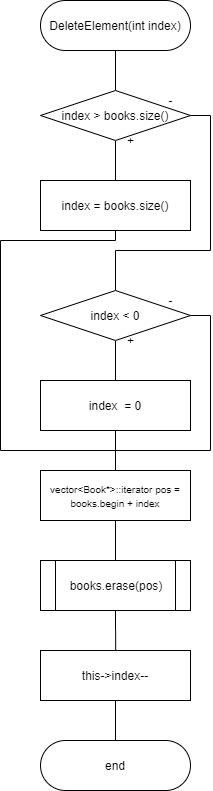
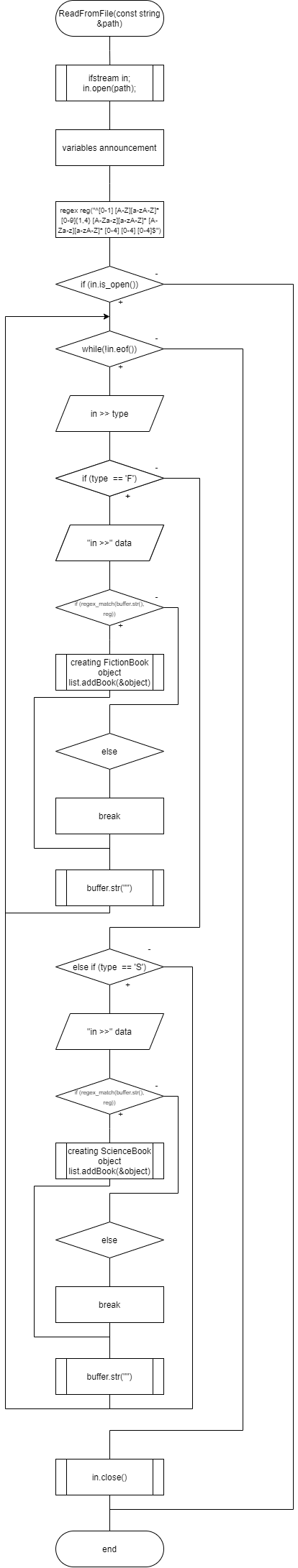


Рисунок 3 – Схема алгоритму методу addBook

  
Рисунок 4 — *Схема алгоритму методу* *DeleteElement*

  
Рисунок 5 — *Схема алгоритму методу**ReadFromFile*

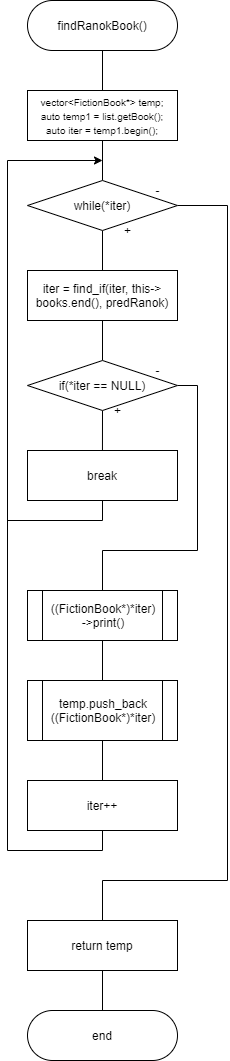


Рисунок 6 — *Схема алгоритму методу**findRanokBook*

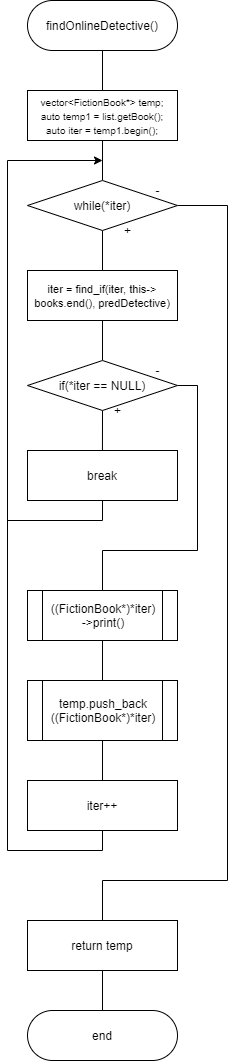


Рисунок 7 — *Схема алгоритму методу**findOnlineDetective*

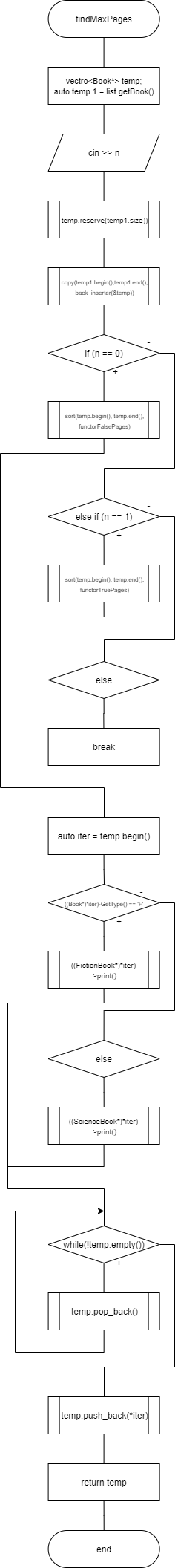


Рисунок 8 — *Схема алгоритму методу**findMaxPages*



Рисунок 9 — *Схема алгоритму методу**SortByField*

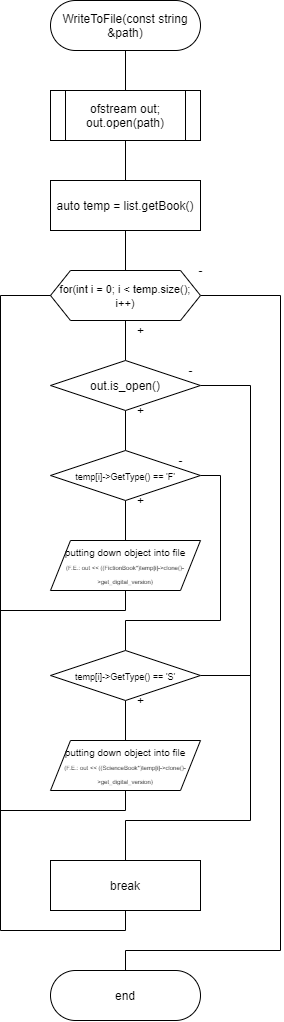


Рисунок 10 — *Схема алгоритму методу WriteToFile*

**4.3 Структура проекту**

├── project

├── Doxyfile

├── CMakeLists.txt

├── Books.txt

├── doc

├── report.docx

├── report.pdf

└── assets

├── src

├── book.cpp

├── book.h

├── list.cpp

├── list.h

├── controller.cpp

├── controller.h

├── menu.cpp

├── menu.h

└── main.cpp

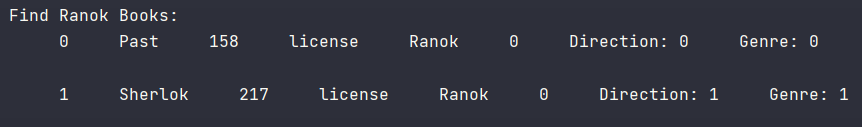
└── test

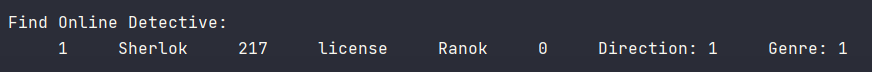
└── test.cpp

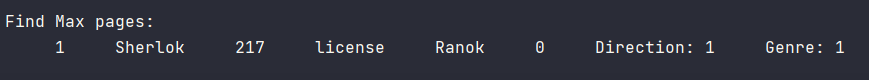
**4.4 Варіанти використання**

Для демонстрації результатів використовується IDE Clion. Нижче наводиться послідовність дій запуску програми.

*Крок 1* (рис. 11-13). Продемонструємо виконання методів пошуку





  
Рис. 11-13 — результат роботи методів пошуку

*Крок 2* (див. рис. 14). Продемонструємо виконання методу сортування.

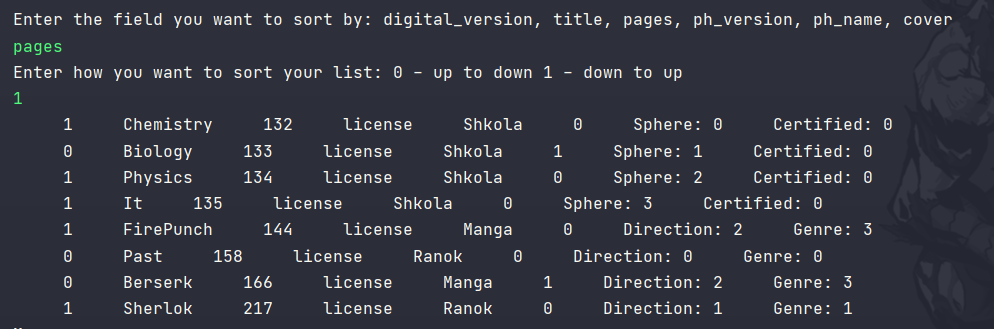


Рисунок 14 — результат роботи методу сортування

**Висновки**

Виконуючи розрахункове завдання було закріплено отримані знання з дисципліни «Програмування» та отримано практичні навички шляхом виконання типового комплексного завдання.

**Додаток А. Реалізація метода** *findRanokBook()*

bool predRanok(Book \**a*){  
 if (*a*->GetType() == 'F') {  
 auto \*Book = (FictionBook \*) *a*->clone();  
 if (Book->get\_pH().getName() == "Ranok") {  
 delete Book;  
 return true;  
 } else {  
 delete Book;  
 return false;  
 }  
 }else{  
 return false;  
 }  
}  
vector<FictionBook\*> Controller::findRanok(){  
 vector<FictionBook\*> temp;  
 auto temp1 = list.getBook();  
 auto iter = temp1.begin();  
 while(\*iter){  
 iter = find\_if(iter, temp1.end(), predRanok);  
 if(iter == temp1.end()) break;  
 ((FictionBook\*)\*iter)->print();  
 **cout**<<endl;  
 temp.push\_back((FictionBook\*)\*iter);  
 iter++;  
 }  
 return temp;  
}

**Додаток Б. Реалізація метода** *findOnlineDetective()*

bool predDetective(Book\* *a*){  
 if (*a*->GetType() == 'F') {  
 auto \*Book = (FictionBook \*) *a*->clone();  
 if (Book->get\_digital\_version() == 1  
 && Book->get\_genre() == **DETECTIVE**) {  
 delete Book;  
 return true;  
 } else {  
 delete Book;  
 return false;  
 }  
 }else{  
 return false;  
 }  
}  
  
vector<FictionBook\*> Controller::findOnlineDetective() {  
 vector<FictionBook\*> temp;  
 auto temp1 = list.getBook();  
 auto iter = temp1.begin();  
 while(\*iter){  
 iter = find\_if(iter, temp1.end(), predDetective);  
 if(iter == temp1.end()) break;  
 ((FictionBook\*)\*iter)->print();  
 **cout**<<endl;  
 temp.push\_back((FictionBook\*)\*iter);  
 iter++;  
 }  
 return temp;  
}

**Додаток В. Реалізація метода** *findMaxPages()*

bool functorTruePages(Book\* *A*, Book\* *B*){  
 bool result = false;  
  
 if (*A*->GetType() == 'F' && *B*->GetType() == 'F') {  
 auto \*temp1 = (FictionBook \*) *A*->clone();  
 auto \*temp2 = (FictionBook \*) *B*->clone();  
 if (temp1->get\_pages() < temp2->get\_pages()) {  
 result = true;  
 }  
 delete temp1;  
 delete temp2;  
 }  
 if (*A*->GetType() == 'S' && *B*->GetType() == 'S') {  
 auto \*temp3 = (ScienceBook \*) *A*->clone();  
 auto \*temp4 = (ScienceBook \*) *B*->clone();  
 if (temp3->get\_pages() < temp4->get\_pages()) {  
 result = true;  
 }  
 delete temp4;  
 delete temp3;  
 }  
 if (*A*->GetType() == 'F' && *B*->GetType() == 'S') {  
 auto \*temp3 = (FictionBook \*) *A*->clone();  
 auto \*temp1 = (ScienceBook \*) *B*->clone();  
 if (temp3->get\_pages() < temp1->get\_pages()) {  
 result = true;  
 }  
 delete temp1;  
 delete temp3;  
 }  
 if (*A*->GetType() == 'S' && *B*->GetType() == 'F') {  
 auto \*temp1 = (ScienceBook \*) *A*->clone();  
 auto \*temp3 = (FictionBook \*) *B*->clone();  
 if (temp1->get\_pages() < temp3->get\_pages()) {  
 result = true;  
 }  
 delete temp1;  
 delete temp3;  
 }  
 return result;  
}  
  
bool functorFalsePages(Book\* *A*, Book\* *B*){  
 bool result = false;  
 if (*A*->GetType() == 'F' && *B*->GetType() == 'F') {  
 auto \*temp1 = (FictionBook \*) *A*->clone();  
 auto \*temp2 = (FictionBook \*) *B*->clone();  
 if (temp1->get\_pages() > temp2->get\_pages()) {  
 result = true;  
 }  
 delete temp1;  
 delete temp2;  
 }  
 if (*A*->GetType() == 'S' && *B*->GetType() == 'S') {  
 auto \*temp3 = (ScienceBook \*) *A*->clone();  
 auto \*temp4 = (ScienceBook \*) *B*->clone();  
 if (temp3->get\_pages() > temp4->get\_pages()) {  
 result = true;  
 }  
 delete temp3;  
 delete temp4;  
 }  
 if (*A*->GetType() == 'F' && *B*->GetType() == 'S') {  
 auto \*temp3 = (FictionBook \*) *A*->clone();  
 auto \*temp1 = (ScienceBook \*) *B*->clone();  
 if (temp3->get\_pages() > temp1->get\_pages()) {  
 result = true;  
 }  
 delete temp1;  
 delete temp3;  
 }  
  
 if (*A*->GetType() == 'S' && *B*->GetType() == 'F') {  
 auto \*temp1 = (ScienceBook \*) *A*->clone();  
 auto \*temp3 = (FictionBook \*) *B*->clone();  
 if (temp1->get\_pages() > temp3->get\_pages()) {  
 result = true;  
 }  
 delete temp1;  
 delete temp3;  
 }  
  
 return result;  
}  
  
  
vector<Book\*> Controller::findMaxPages() {  
 vector<Book\*> temp;  
 auto temp1 = list.getBook();  
 temp.reserve(temp1.size());  
 copy(temp1.begin(), temp1.end(), back\_inserter(temp));  
 sort(temp.begin(), temp.end(), functorFalsePages);  
 auto iter = temp.begin();  
 if (((Book\*)\*iter)->GetType() == 'F'){  
 ((FictionBook\*)\*iter)->print();  
 **cout** << endl;  
 }else {  
 ((ScienceBook\*)\*iter)->print();  
 **cout** << endl;  
 }  
 while (!temp.empty()){  
 temp.pop\_back();  
 }  
 temp.push\_back(\*iter);  
 return temp;  
}

**Додаток Г. Реалізація метода** *SortByField(string field)*

void Controller::SortByField(string *field*) {  
 vector<Book\*> temp;  
 auto temp1 = list.getBook();  
 int n;  
 string field1 = "digital\_version";  
 string field2 = "title";  
 string field3 = "pages";  
 string field4 = "ph\_version";  
 string field5 = "ph\_name";  
 string field6 = "cover";  
 **cout** << "Enter how you want to sort your list: 0 - up to down 1 - down to up" << endl;  
 **cin** >> n;  
 temp.reserve(temp1.size());  
 copy(temp1.begin(), temp1.end(), back\_inserter((temp)));  
  
 if(*field* == field1) {  
 if (n == 0) {  
 sort(temp.begin(), temp.end(), functorFalseDV);  
 } else if (n == 1) {  
 sort(temp.begin(), temp.end(), functorTrueDV);  
 }  
 } else if (*field* == field2){  
 if (n == 0) {  
 sort(temp.begin(), temp.end(), functorFalseTitle);  
 } else if (n == 1) {  
 sort(temp.begin(), temp.end(), functorTrueTitle);  
 }  
 } else if (*field* == field3){  
 if (n == 0) {  
 sort(temp.begin(), temp.end(), functorFalsePages);  
 } else if (n == 1) {  
 sort(temp.begin(), temp.end(), functorTruePages);  
 }  
 } else if (*field* == field4){  
 if (n == 0) {  
 sort(temp.begin(), temp.end(), functorFalsePhVer);  
 } else if (n == 1) {  
 sort(temp.begin(), temp.end(), functorTruePhVer);  
 }  
 } else if (*field* == field5){  
 if (n == 0) {  
 sort(temp.begin(), temp.end(), functorFalsePhName);  
 } else if (n == 1) {  
 sort(temp.begin(), temp.end(), functorTruePhName);  
 }  
 } else if (*field* == field6){  
 if (n == 0) {  
 sort(temp.begin(), temp.end(), functorFalseCover);  
 } else if (n == 1) {  
 sort(temp.begin(), temp.end(), functorTrueCover);  
 }  
 } else {  
 **cout** << "Error";  
 }  
 for (int i = 0; i < temp.size(); ++i) {  
 temp[i]->print();  
 }  
 temp.clear();  
 temp.shrink\_to\_fit();  
}

}