

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
Макаренко Владислав Олександрович

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

Інформаційна система (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 зіштовхуються майже усі.

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

Розроблена інформаційна система має колекцію рюкзаків та методи роботи з нею. З загально та індивідуального завдання колекція має методи: пошуку німецького міського шкіряного рюкзака з велюру, пошуку замшевого рюкзаку без підкладки та пошуку синього рюкзака з найбільшим об'ємом. Також є можливість сортування колекції залежно від заданого користувачем напрямку та по вказаному критерію рюкзака. Є також методи, які дають змогу: видалити заданий користувачем рюкзак з колекцію; очистити колекцію рюкзаків; додати рюкзак до колекції; замінити або ж отримати рюкзак по індексу.

Дану інформаційну систему можна застосовувати в різних цілях. Здебільшого в галузях пов'язаних на роботі з рюкзаками, наприклад інтернет магазин та інше.

2. Постановка завдання до розробки

1.1 Загальне завдання

- 1) З розділу "Розрахункове завдання / Індивідуальні завдання", відповідно до варіанта завдання, обрати прикладну галузь;
- 2) Для прикладної галузі розробити розгалужену ієрархію класів, що описана у завданні та складається з одного базового класу та двох спадкоємців. Класи повинні мати перевантажені оператори введення-виведення даних та порівняння;
- 3) 3. 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) додавання/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 їх 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.cpp`, основною метою якого буде перевірка коректності роботи класу-контролера.

1.2 Індивідуальне завдання

– Варіант 17. "Рюкзак"

- Поля базового класу:
 - Наявність відділу для ноутбука (наприклад: так, ні)
 - Колір (наприклад: синій, зелений)
 - Об'єм, літри (наприклад: 20, 25)
 - Фірма (структура, що містить назву фірми та країну її місце знаходження)
 - Призначення (один з переліку: міський, тактичний, туристичний)
- Спадкоємець 1 - Шкіряний рюкзак. Додаткові поля:
 - Наявність підкладки (наприклад: так, ні)
 - Тип шкіри (один з переліку: анілінова, велюр, замша)
- Спадкоємець 2 - Рюкзак з тканини. Додаткові поля:
 - Чи є водонепроникним (наприклад: так, ні)
 - Тканина (один з переліку: синтетика, бавовна, брезент)
- Методи роботи з колекцією:
 1. Знайти німецьки шкіряний міський рюкзак з велюру
 2. Знайти замшеві рюкзаки без підкладки
 3. Знайти рюкзак синього кольору з найбільшим об'ємом

3. Опис вхідних та вихідних даних

3.1 Опис вхідних даних

Під час запуску програми, необхідно ввести ім'я файлу, звідки будуть взяті вхідні дані. В файлі повинні бути наступні дані: першим повинен бути символ ('L' чи 'F'), котрий позначає тип вхідного об'єкту ('L' – шкіряний рюкзак, 'F' – тканинний рюкзак), далі цифра 1 чи 0, що позначає чи наявний відділ для ноутбука (1 – так, 0 – ні), потім колір рюкзака, об'єм, призначення рюкзака (0 – міський, 1 – тактичний, 2 – туристичний), назва фірми та країну її місце знаходження, далі цифра 1 чи 0, що позначає чи наявна підкладка(для шкіряних рюкзаків) або чи є водонепроникним(для тканинних рюкзаків) (1 – так, 0 – ні), і в кінці тип шкіри (для шкіряних рюкзаків: 0 – анілін, 1 – велюр, 2 – замша) або тип тканини (для тканинних рюкзаків: 0 – синтетика, 1 – бавовна, 2 – брезент). Приклад файлу з вхідними даними дивись 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

4. 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.Run()`.

Метод `menu.Run()` викликає метод `ShowMenu()` котрий виводить на екран діалогове меню, та отримує від користувача номер дії, яку необхідно виконати з колекцією. І в залежності від номеру поверненого методом `ShowMenu()`, метод `menu.Run()` викликає відповідний метод роботи з колекцією.

Метод `addBackpackByIndex(Backpack* backpack, const int index)` додає елемент до колекції по індексу. Приймає об'єкт, який необхідно додати до колекції та індекс, куди вставити отриманий об'єкт

Схема алгоритму методу подана на рис. 3.

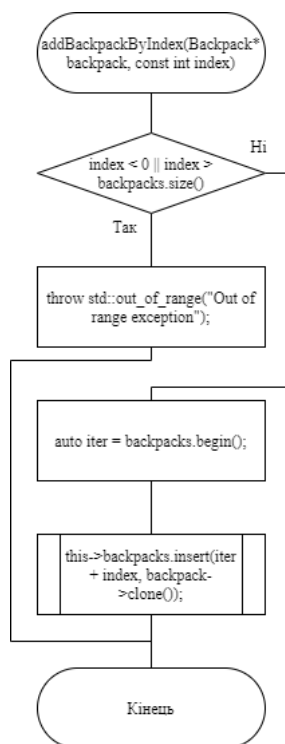


Рисунок 3 – Схема алгоритму методу `addBackpackByIndex`

Метод `DeleteElement(const int index)` видаляє елемент з колекції по індексу. Індекс, звідки необхідно видалити об'єкт.

Схема алгоритму методу подана на рис. 4.

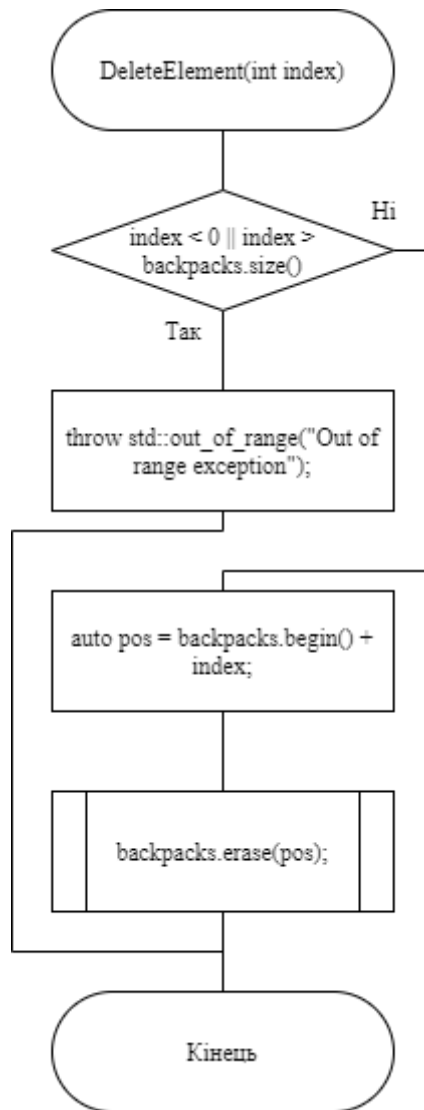


Рисунок 4 — Схема алгоритму методу `DeleteElement`

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.

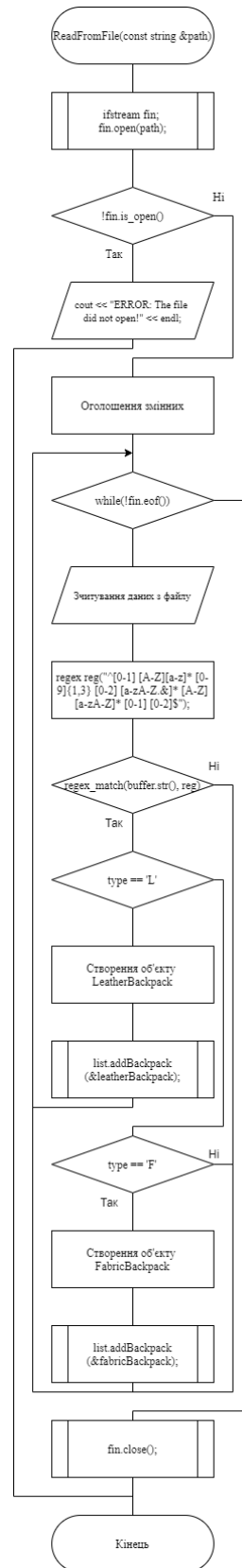


Рисунок 5 — Схема алгоритму методу `ReadFromFile`

Метод `FindGermanVelourBackpack()` виконує «метод 1» з індивідуального завдання. Метод за допомогою відповідного функтора знаходить серед колекції усі німецькі міські шкіряні рюкзаки з велюру, записує їх у вектор та повертає цей вектор зі знайденими рюкзаками. (Реалізація методу – дивись Додаток А)

Схема алгоритму методу подана на рис. 6.

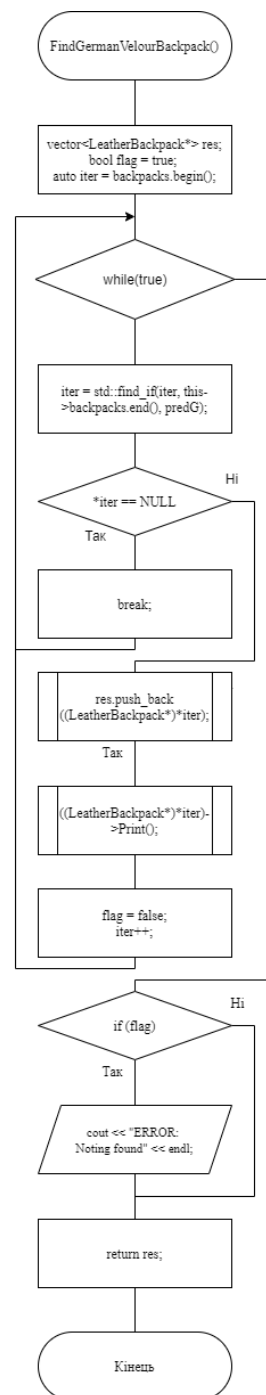


Рисунок 6 — Схема алгоритму методу `FindGermanVelourBackpack`

Метод `FindSuedeNoLiningBackpack()` виконує «метод 2» з індивідуального завдання. Метод за допомогою відповідного функтора знаходить серед колекції усі замшеві рюкзаки без підкладки, записує їх у вектор та повертає цей вектор зі знайденими рюкзаками. (Реалізація методу – дивись Додаток Б)

Схема алгоритму методу подана на рис. 7.

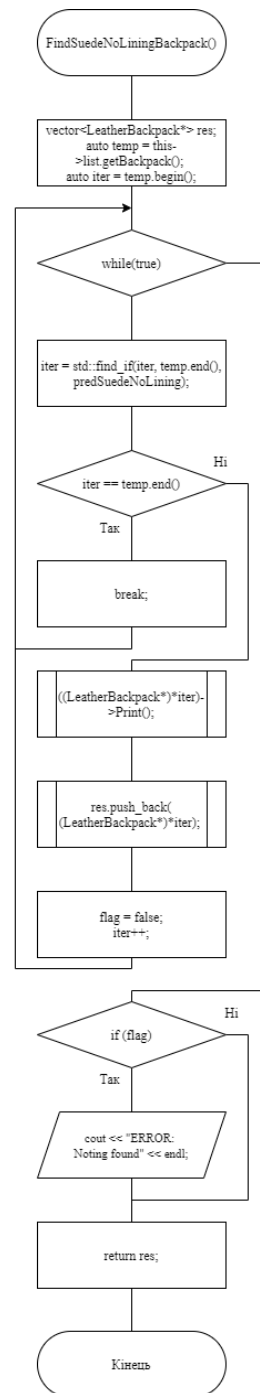


Рисунок 7 — Схема алгоритму методу `FindSuedeNoLiningBackpack`

Метод `FindBlueMaxVolumeBackpack()` виконує «метод 3» з індивідуального завдання. Метод за допомогою відповідного функтора знаходить серед колекції синій рюкзак з найбільшим об'ємом, записує його у вектор та повертає цей вектор зі знайденим рюкзаком. (Реалізація методу – дивись Додаток В)

Схема алгоритму методу подана на рис. 8.

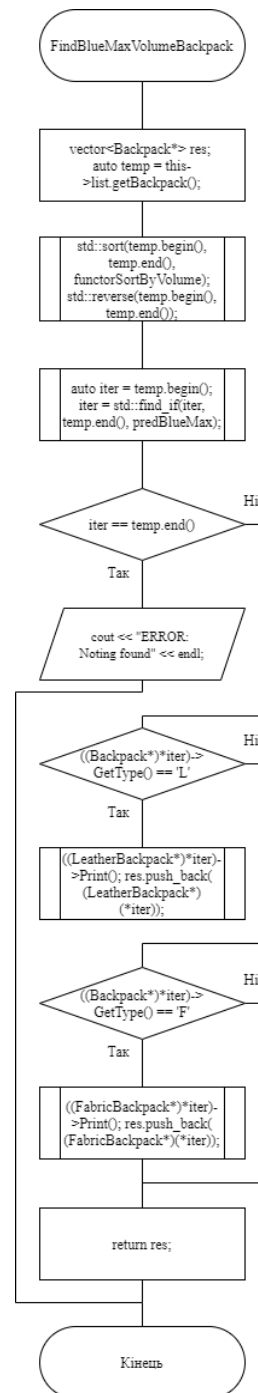


Рисунок 8 — Схема алгоритму методу `FindBlueMaxVolumeBackpack`

Метод `SortByCriterion(int criterion, bool direction)` 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.

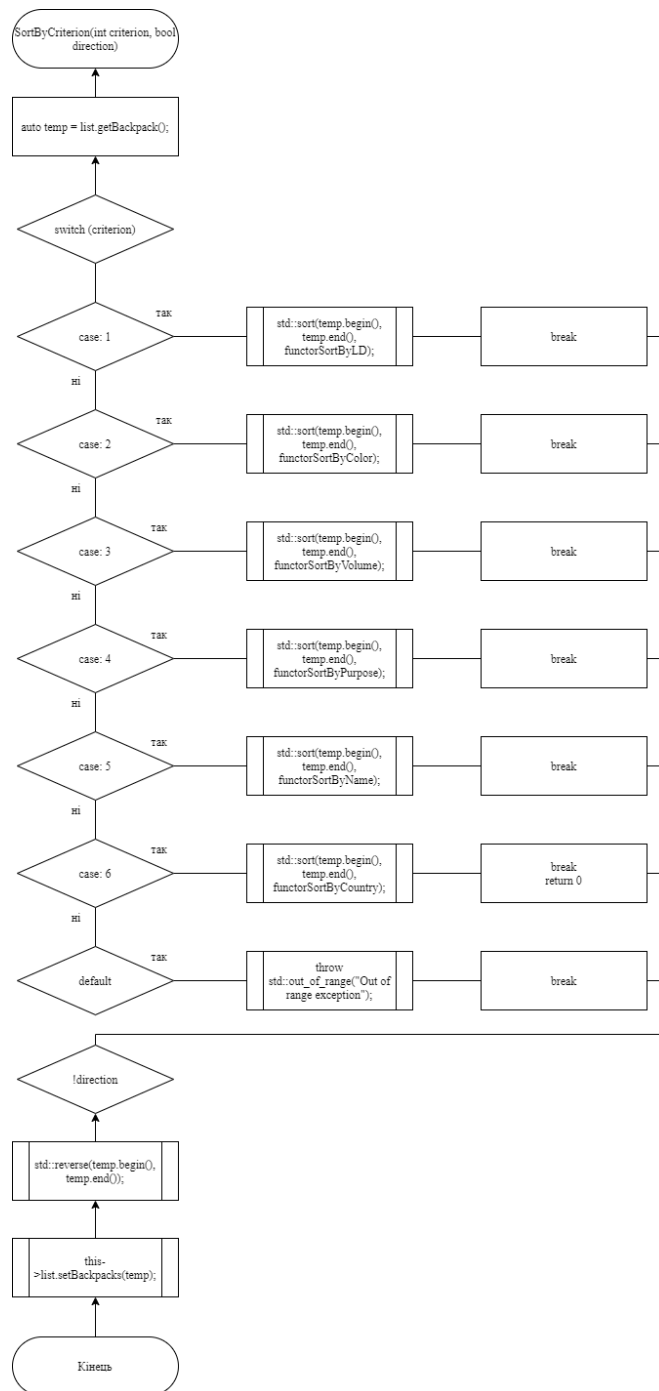


Рисунок 9 — Схема алгоритму методу `SortByCriterion`

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.

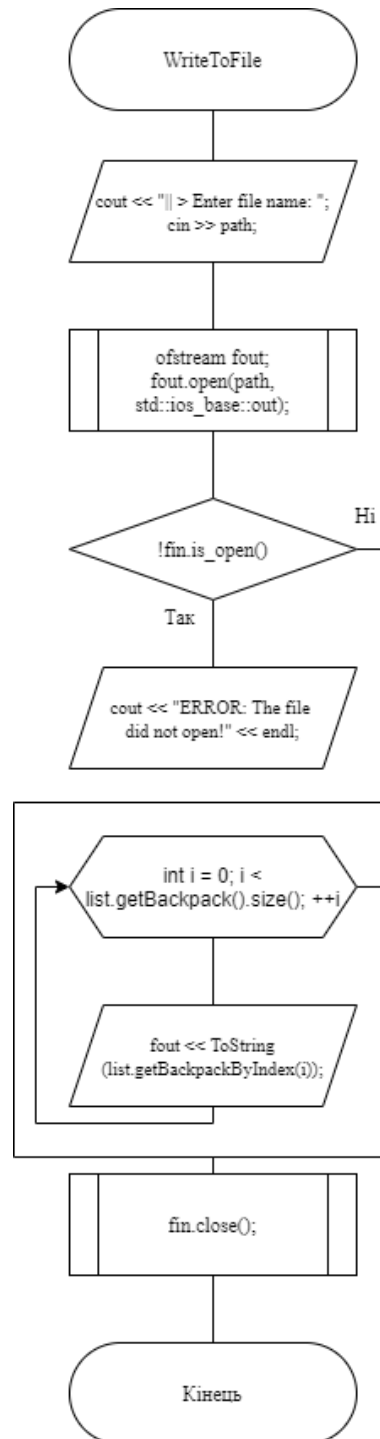


Рисунок 10 — Схема алгоритму методу `WriteToFile()`

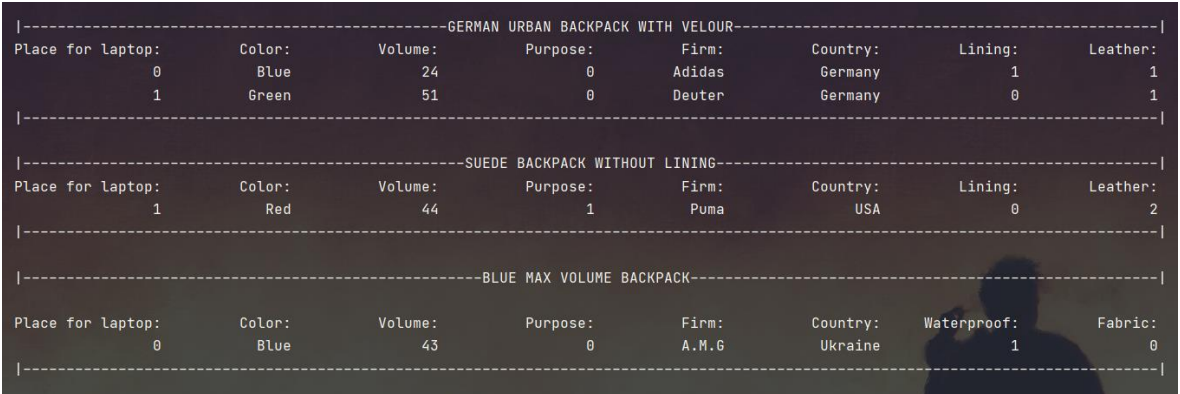
4.3 Структура проекту

```
|— Individual-work
    |— Doxyfile
    |— CMakeLists.txt
    |— Backpacks.txt
    |— doc
        |— Individual-work.docx
        |— Individual-work.pdf
        └─ assets
    |— src
        |— Backpack.cpp
        |— Backpack.h
        |— Container.cpp
        |— Container.h
        |— Controller.cpp
        |— Controller.h
        |— Menu.cpp
        |— Menu.h
        └─ main.cpp
    └─ test
        └─ ControllerTest.cpp
```

4.4 Варіанти використання

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

Крок 1 (рис. 11). Продемонструємо виконання методів пошуку



-----GERMAN URBAN BACKPACK WITH VELOUR-----							
Place for laptop:	Color:	Volume:	Purpose:	Firm:	Country:	Lining:	Leather:
0	Blue	24	0	Adidas	Germany	1	1
1	Green	51	0	Deuter	Germany	0	1
-----SUEDE BACKPACK WITHOUT LINING-----							
Place for laptop:	Color:	Volume:	Purpose:	Firm:	Country:	Lining:	Leather:
1	Red	44	1	Puma	USA	0	2
-----BLUE MAX VOLUME BACKPACK-----							
Place for laptop:	Color:	Volume:	Purpose:	Firm:	Country:	Waterproof:	Fabric:
0	Blue	43	0	A.M.G	Ukraine	1	0

Рисунок 11 — результат роботи методів пошуку

Крок 2 (див. рис. 12). Продемонструємо виконання методу сортування

```
//===== [ SORT LIST ]=====\\
|| > Select a sorting criterion:
|| 1. Space for a laptop.
|| 2. Color.
|| 3. Volume.
|| 4. Purpose.
|| 5. The name of the company.
|| 6. Country of origin.
||-----||
|| > Your choice: 3
||-----||
|| > Select sorting direction:
|| 1. From smallest to largest.
|| 0. From the largest to the smallest.
||-----||
|| > Your choice: 1
\\=====\\
```

а) Запит критерія та напрямку сортування

```
//===== [ ORIGINAL LIST ]=====\\
Place for laptop:   Color:      Volume:   Purpose:   Firm:      Country:   WP/Lining: Fabric/Leather:
0                   Pink       24       0          Kanken    British   1          1
0                   Black      33       2          Gucci     Italy     1          1
1                   Orange     38       1          Adidas    Germany  0          2
0                   Blue       39       2          Nike      America  1          0
0                   Yellow     43       0          A.M.G     Ukraine  1          0
1                   Red        44       1          Puma      America  0          2
1                   Green      51       0          Deuter    Germany  0          1
1                   Purple     55       1          Bagllet   Ukraine  0          2
\\=====\\
```

б) Результат сортування

Рисунок 12 — результат роботи методу сортування

Висновки

Виконуючи розрахункове завдання ми закріпили отримані знання з дисципліни «Програмування» та отримали практичні навички шляхом виконання типового комплексного завдання.

Додаток А. Реалізація метода *FindGermanVelourBackpack()*

```
bool predGermanUrbanVelour(Backpack* a)
{
    if (a->GetType() == 'L') {
        auto* Le = (LeatherBackpack*) a->clone();
        if (Le->getfirm().getCountry() == "Germany"
            && Le->getLeather() == VELOUR
            && Le->getpurpose() == URBAN) {
            delete Le;
            return true;
        }
        else {
            delete Le;
            return false;
        }
    }
    else {
        return false;
    }
}

vector<LeatherBackpack*> Controller::FindGermanVelourBackpack() const{
    vector<LeatherBackpack*> res;
    vector<Backpack*> tmp = this->list.getBackpack();
    bool flag = true;
    auto iter = tmp.begin();
    while (true) {
        iter = std::find_if(iter, tmp.end(), predGermanUrbanVelour);
        if (iter == tmp.end()) {
            break;
        }
        flag = false;
        res.push_back((LeatherBackpack*)*iter);
        ((LeatherBackpack*)*iter)->Print();
        cout << endl;
        iter++;
    }
    if (flag) {
        cout << "ERROR: Nothing found" << endl;
    }
    while (!tmp.empty()) {
        tmp.pop_back();
    }
    tmp.clear();
    tmp.shrink_to_fit();
    return res;
}
```

Додаток Б. Реалізація метода *FindSuedeNoLiningBackpack()*

```
bool predSuedeNoLining(Backpack* a)
{
    if (a->GetType() == 'L') {
        auto* Le = (LeatherBackpack*)a->clone();
        if (Le->getLeather() == SUEDE && !Le->getlining()) {
            delete Le;
            return true;
        }
        else {
            delete Le;
            return false;
        }
    }
    else {
        return false;
    }
}

vector<LeatherBackpack*> Controller::FindSuedeNoLiningBackpack() const{
    vector<LeatherBackpack*> res;
    vector<Backpack*> temp = this->list.getBackpack();
    bool flag = true;
    auto iter = temp.begin();
    while (*iter) {
        iter = std::find_if(iter, temp.end(), predSuedeNoLining);
        if (iter == temp.end()) {
            break;
        }
        flag = false;
        ((LeatherBackpack*)*iter)->Print();
        cout << endl;
        res.push_back((LeatherBackpack*)*iter);
        iter++;
    }
    if (flag) {
        cout << "ERROR: Nothing found" << endl;
    }
    while (!temp.empty()) {
        temp.pop_back();
    }
    temp.clear();
    temp.shrink_to_fit();
    return res;
}
```

Додаток В. Реалізація метода *FindBlueMaxVolumeBackpack()*

```
bool predBlueMax(Backpack* a)
{
    auto* Le = (LeatherBackpack*)a->clone();
    if (Le->getcolor() == "Blue") {
        delete Le;
        return true;
    }
    else {
        delete Le;
        return false;
    }
}

vector<Backpack*> Controller::FindBlueMaxVolumeBackpack() const {
    vector<Backpack*> res;
    auto temp = this->list.getBackpack();
    std::sort(temp.begin(), temp.end(), functorSortByVolume);
    std::reverse(temp.begin(), temp.end());

    auto iter = temp.begin();
    iter = std::find_if(iter, temp.end(), predBlueMax);
    if (iter == temp.end()) {
        cout << "ERROR: Nothing found" << endl;
    }
    if (((Backpack*)*iter)->GetType() == 'L') {
        PrintForm1();
        cout << endl;
        ((LeatherBackpack*)*iter)->Print();
        cout << endl;
        res.push_back((LeatherBackpack*)(*iter));
    }
    else {
        PrintForm2();
        cout << endl;
        ((FabricBackpack*)*iter)->Print();
        cout << endl;
        res.push_back((FabricBackpack*)(*iter));
    }

    while (!temp.empty()) {
        temp.pop_back();
    }
    temp.clear();
    temp.shrink_to_fit();
    return res;
}
```

Додаток Г. Реалізація метода *SortByCriterion* ()

```
void Controller::SortByCriterion(int criterion, bool direction) {
    auto temp = list.getBackpack();
    switch (criterion) {
        case 1:
            std::sort(temp.begin(), temp.end(), functorSortByLD);
            break;
        case 2:
            std::sort(temp.begin(), temp.end(), functorSortByColor);
            break;
        case 3:
            std::sort(temp.begin(), temp.end(), functorSortByVolume);
            break;
        case 4:
            std::sort(temp.begin(), temp.end(), functorSortByPurpose);
            break;
        case 5:
            std::sort(temp.begin(), temp.end(), functorSortByName);
            break;
        case 6:
            std::sort(temp.begin(), temp.end(), functorSortByCountry);
            break;
        default:
            throw std::out_of_range("Out of range exception");
            break;
    }
    if (!direction) {
        std::reverse(temp.begin(), temp.end());
    }
    this->list.setBackpacks(temp);
}
```