Зенин Вадим. Вариант - 26

Задача об инвентаризации по книгам. После нового года в библиотеке университета обнаружилась пропажа каталога. После поиска и наказания, виноватых ректор дал указание восстановить каталог силами студентов. Фонд библиотека представляет собой прямоугольное помещение, в котором находится M рядов по N шкафов по K книг в каждом шкафу. Требуется создать многопоточное приложение, составляющее каталог. При решении использовать метод «портфель задач», причем в качестве отдельного потока задается внесение в каталог записи об отдельной книге.

Примечание. Каталог — это список книг, упорядоченный (отсортированный) по названию книги (в данном случае в качестве названия можно взять и целое число — не обязательно использовать ASCII строку символов). Каждая строка каталога содержит идентифицирующее значение (номер или строку), местоположение книги, включающее номер ряда, номер шкафа, номер книги в шкафу. Перед запуском потоков по составлению каталогов необходимо случайным образом расположить книги в шкафах, используя генератор случайных чисел. Также нужно преварительно 17 вывести список замещения книг по шкафам, чтобы в конце сопоставить его с тем, как они оказались описанными в каталоге.

Отчет сразу на 8

Пользуясь предыдущим опытом, хочу сказать, что отчет написан сразу на 8, поскольку семинарист разрешил.

- В отчете должен быть приведен сценарий, описывающий одновременное поведение представленных в условии задания сущностей в терминах предметной области. То есть, описан сценарий, задающий ролевое поведение субъектов и объектов задачи (интерпретация условия с большей степенью детализации происходящего), а не то, как это будет реализовано в программе.

В данном задании речь идет о том, что у нас есть библиотека, которая представляет собой трехмерную матрицу, если можно так выразиться. В данной библиотеке хранятся в шкафах каким-то образом книги и надо с помощью

"портфеля задач" сделать каталог, отсортированный по имена книг (я решил именами книг сделать просто цифры, поскольку разрешено). Те данную задачу можно интерпритировать так, что у нас есть некоторое количество школьников(тредов) и книг, каждый из них берет по книге и заносит в каталог, разбивая большую задачу на маленькие подзадачки (внесение книги в каталог.)

- Описана модель параллельных вычислений, используемая при разработке многопоточной программы.
- Описаны входные данные программы, включающие вариативные диапазоны, возможные при многократных запусках.
- В отчете подробно описан обобщенный алгоритм, используемый при реализации программы исходного словесного сценария. В котором показано, как на программу отображается каждый из субъектов предметной области.

- Модель параллельных вычислений, используемая в данной программе - "Портфель задач". С помощью нее я составляю очередь из задач (добавления книги в каталог), доступ к которой единовременно есть только у 1 треда, чтобы исключить одновременного доступа к 1 книге несколькими тредами были использованы мьютексы.

```
// Глобальные переменные
pthread_mutex_t task_mutex;        // Мьютекс для синхронизации очереди задач
pthread_mutex_t catalog_mutex;     // Мьютекс для синхронизации каталога
pthread_mutex_t output_mutex;      // Мьютекс для синхронизации вывода
std::queue<Book> task_queue;       // Очередь задач
std::vector<Book> catalog;         // Каталог книг

// Структура для передачи данных в потоки
struct ThreadData {
    int thread_id;
};
```

```
// Функция обработки задач (работа потоков)
void* processTask(void* arg) {
    ThreadData* data = (ThreadData*)arg; // Получаем данные потока
    while (true) {
        // Захватываем мьютекс, чтобы безопасно работать с очередью задач
        pthread_mutex_lock(&task_mutex);
        if (task_queue.empty()) { // Если очередь пуста, завершаем работу
            pthread_mutex_unlock(&task_mutex);
            break;
        }
        Book task = task_queue.front(); // Извлекаем задачу из очереди
        task_queue.pop();
        pthread_mutex_unlock(&task_mutex); // Освобождаем мьютекс

        // Добавляем книгу в каталог, используя мьютекс для синхронизации
        pthread_mutex_lock(&catalog_mutex);
        catalog.push_back(task);
        pthread_mutex_unlock(&catalog_mutex);

        // Выводим информацию о выполнении задачи, синхронизируя доступ к консоли
        pthread_mutex_lock(&output_mutex);
        std::cout << "Студент " << data->thread_id << " успешно добавил книгу "
                  << task.getBookInfo() << " в каталог" << std::endl;
        pthread_mutex_unlock(&output_mutex);
    }
    return nullptr; // Завершаем поток
}
```

```
// Формирование структуры библиотеки
auto library :vector<vector<vector<Book>>> = Book::getBooksInCloset( books);
std::cout << "Начальное состояние библиотеки:" << std::endl;
Book::printLibrary( library);

// Заполнение очереди задач
for (const auto& book : books) {
    task_queue.push(book);
}

// Инициализация мьютексов
pthread_mutex_init(&task_mutex, nullptr);
pthread_mutex_init(&catalog_mutex, nullptr);
pthread_mutex_init(&output_mutex, nullptr);

// Создание потоков
const int NUM_THREADS = 10;
pthread_t threads[NUM_THREADS];
ThreadData thread_data[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; ++i) {
    thread_data[i].thread_id = i + 1;
    pthread_create(&threads[i], nullptr, processTask, &thread_data[i]);
}

// Ожидание завершения потоков
for (int i = 0; i < NUM_THREADS; ++i) {
    pthread_join(threads[i], nullptr);
}

 // Удаление мьютексов
 pthread_mutex_destroy(&task_mutex);
 pthread_mutex_destroy(&catalog_mutex);
 pthread_mutex_destroy(&output_mutex);
```

Здесь можно увидеть реализацию "Портфеля задач" с комментариями в моем ИДЗ.
Когда поток заполняет каталог, в консоль выводится какой тред и какую книгу туда добавил.

- Входные данные включают в себя очередь из книг (класс Book), которые необходимо добавить в каталог. Объект книга состоит из номеров: ряда, номера шкафа, номера в самом шкафу и имени книги (число - int). Данные могут как создаваться рандомно, так и вводиться пользователем вручную или считывается из файла.

- Для используемых генераторов случайных чисел описаны их диапазоны и то, как интерпретируются данные этих генераторов.
- В программу добавлена генерация случайных данных в допустимых диапазонах.

- Генерируемые книги получают рандомные значения номеров ряда и номера шкафов в диапазоне от 0 до 52. Номера в шкафу от 0 до 15. И имя в диапазоне от 43264 ($= 52^2 * 16$). Данные числа выбраны случайно, поскольку диапазон данных(размеры библиотеки и шкафов) может быть любым.
- Вот реализация создания "Портфеля задач"

```
// Генерация портфеля задач (список книг)
std::vector<Book> Book::generateTaskBackpack(int numbers_of_books) {
    std::vector<Book> books;
    std::random_device rd;
    std::mt19937 gen( sd: rd());
    std::uniform_int_distribution<> row_dist( a: 0,  b: 52);   // Ряды от 0 до 52
    std::uniform_int_distribution<> col_dist( a: 0,  b: 52);   // Шкафы от 0 до 52
    std::uniform_int_distribution<> num_dist( a: 0,  b: 15);  // Книги в шкафу от 0 до 15
    std::uniform_int_distribution<> name_dist( a: 0,  b: 43264); // Идентификаторы книг

    // Заполняем портфель задач, те книги, которые нужно найти.
    for (int i = 0; i < numbers_of_books; ++i) {
        books.emplace_back(row_dist( [&] gen), col_dist( [&] gen),  number: num_dist( [&] gen), name_dist( [&] gen));
    }
    return books;
}
```

- Вывод программы должен быть информативным, отражая все ключевые протекающие в ней события в терминах предметной области. Наблюдатель на основе вывода программы должен понимать, что в ней происходит в каждый момент времени ее работы.

После создания библиотеки я вывожу книги стоящие в ней в определенных столбцах и рядах шкафов

```
// Печать структуры библиотеки
void Book::printLibrary(const std::vector<std::vector<std::vector<Book>>> books) {
    for (size_t row = 0; row < books.size(); ++row) {
        for (size_t col = 0; col < books[row].size(); ++col) {
            for (const auto& book : books[row][col]) {
                std::cout << "Ряд " << row+1  << ", Шкаф " << col+1
                          << ", Порядок книги в шкафу " << book.number_ + 1 << " (Название: " << book.name_ << ")" << std::endl;
            }
        }
    }
}
```

Чтобы после выполнения работ можно были посмотреть, какой поток с какими книгами работал, все ли книги в каталоге, действительно ли он отсортирован и не сбились ли данные о книгах.

```
    // Вывод каталога
    std::cout << "\nИтоговый каталог:" << std::endl;
    for (const auto& book : catalog) {
        std::cout << book.getBookInfo() << std::endl;
    }
```

```
// Получение информации о книге
std::string Book::getBookInfo() const {
    std::ostringstream info;
    info << "Книга " << """ << name_ << """ << " (Ряд " << row_+1 << ", Шкаф " << col_+1 << ", Позиция " << number_+1 << ")";
    return info.str();
}
```

- Реализован ввод исходных данных из командной строки при запуске программы вместо ввода параметров с консоли во время выполнения программы.
- В программу, наряду с выводом в консоль, добавлен вывод результатов в файл. Имя файла для вывода данных задается в командной строке как один из ее параметров.

Чтобы реализовать ввод данных таким образом, были использованы флаги для запуска программы с определенными параметрами.

```
// Функция для ввода книг с консоли
std::vector<Book> Book::inputBooksFromConsole() {
    int num_books;
    std::cout << "Введите количество книг: ";
    std::cin >> num_books; // Ввод количества книг
    std::cout << "Введите данные для книги: " << std::endl;

    std::vector<Book> books; // Локальный вектор для хранения книг
    for (int i = 0; i < num_books; ++i) {
        int row, col, number, name;
        if(row < 0 || col < 0 || number < 0 ){
            throw std::runtime_error("Ошибка: Некорректные данные, введите положительные числа.");
        }
        std::cin >> row >> col >> number >> name;
        books.push_back( x: {row, col, number, name}); // Добавление книги в вектор
    }

    return books; // Возвращаем введенный список книг
}
```

```cpp
// Функция для чтения книг из файла
std::vector<Book> Book::readBooksFromFile(const std::string& filename) {
    std::ifstream file(filename); // Открываем файл для чтения

    if (!file.is_open()) {
        throw std::runtime_error("Не удалось открыть файл: " + filename); // Ошибка, если файл не открыт
    }

    std::vector<Book> books; // Локальный вектор для хранения книг
    int row, col, number, name;

    // Чтение данных из файла построчно
    while (file >> row >> col >> number >> name) {
        if(row < 0 || col < 0 || number < 0 ){
            throw std::runtime_error("Ошибка: Некорректные данные в файле.");
        }
        books.push_back( x: {row, col, number, name}); // Добавление книги в вектор
    }

    return books; // Возвращаем список книг, прочитанных из файла
}
```

Выше представлены 2 функции, которые обеспечивают ввод данных с консоли или с файла. Они создают вектор из полученных книг.

- Результаты работы программы должны выводиться на экран и записываться в файл.

Рассмотрим работу программы на примере тестовых файлов состоящих из 10 и 500 книг.
Вот данные тестового файла, содержащего 10 элементов.(Не буду вставлять данные файлика с 500 элементов по понятным причинам)

```
0 0 1 145
0 1 2 230
0 1 3 110
1 0 1 215
1 0 2 190
1 1 1 275
2 0 1 120
2 0 2 340
2 1 1 85
2 1 2 310
```

Скомпилируем и запустим программу с помощью флага -file

```
vadimzenin@MacBook-Pro-Vadim ИДЗ - 4. 28 вариант % g++ main.cpp -o main -std=c++20
vadimzenin@MacBook-Pro-Vadim ИДЗ - 4. 28 вариант % ./main -file input10.txt
```

Наблюдаем успешный результат работы программы, сохраненный в файлик "catalog_output.txt"

```
Начальное состояние библиотеки:
Ряд 1, Шкаф 1, Порядок книги в шкафу 2 (Название: 145)
Ряд 1, Шкаф 2, Порядок книги в шкафу 3 (Название: 230)
Ряд 1, Шкаф 2, Порядок книги в шкафу 4 (Название: 110)
Ряд 2, Шкаф 1, Порядок книги в шкафу 2 (Название: 215)
Ряд 2, Шкаф 1, Порядок книги в шкафу 3 (Название: 190)
Ряд 2, Шкаф 2, Порядок книги в шкафу 2 (Название: 275)
Ряд 3, Шкаф 1, Порядок книги в шкафу 2 (Название: 120)
Ряд 3, Шкаф 1, Порядок книги в шкафу 3 (Название: 340)
Ряд 3, Шкаф 2, Порядок книги в шкафу 2 (Название: 85)
Ряд 3, Шкаф 2, Порядок книги в шкафу 3 (Название: 310)
Студент 1 успешно добавил книгу '145' (Ряд 1, Шкаф 1, Позиция 2) в каталог
Студент 3 успешно добавил книгу '110' (Ряд 1, Шкаф 2, Позиция 4) в каталог
Студент 3 успешно добавил книгу '120' (Ряд 3, Шкаф 1, Позиция 2) в каталог
Студент 1 успешно добавил книгу '275' (Ряд 2, Шкаф 2, Позиция 2) в каталог
Студент 7 успешно добавил книгу '310' (Ряд 3, Шкаф 2, Позиция 3) в каталог
Студент 6 успешно добавил книгу '340' (Ряд 3, Шкаф 1, Позиция 3) в каталог
Студент 4 успешно добавил книгу '215' (Ряд 2, Шкаф 1, Позиция 2) в каталог
Студент 3 успешно добавил книгу '85' (Ряд 3, Шкаф 2, Позиция 2) в каталог
Студент 2 успешно добавил книгу '230' (Ряд 1, Шкаф 2, Позиция 3) в каталог
Студент 5 успешно добавил книгу '190' (Ряд 2, Шкаф 1, Позиция 3) в каталог

Итоговый каталог:
'85' (Ряд 3, Шкаф 2, Позиция 2)
'110' (Ряд 1, Шкаф 2, Позиция 4)
'120' (Ряд 3, Шкаф 1, Позиция 2)
'145' (Ряд 1, Шкаф 1, Позиция 2)
'190' (Ряд 2, Шкаф 1, Позиция 3)
'215' (Ряд 2, Шкаф 1, Позиция 2)
'230' (Ряд 1, Шкаф 2, Позиция 3)
'275' (Ряд 2, Шкаф 2, Позиция 2)
'310' (Ряд 3, Шкаф 2, Позиция 3)
'340' (Ряд 3, Шкаф 1, Позиция 3)

Каталог сохранен в файл catalog_output.txt.
```

Рассмотрим, что записалось в файлик.

```
'85' (Ряд 3, Шкаф 2, Позиция 2)
'110' (Ряд 1, Шкаф 2, Позиция 4)
'120' (Ряд 3, Шкаф 1, Позиция 2)
'145' (Ряд 1, Шкаф 1, Позиция 2)
'190' (Ряд 2, Шкаф 1, Позиция 3)
'215' (Ряд 2, Шкаф 1, Позиция 2)
'230' (Ряд 1, Шкаф 2, Позиция 3)
'275' (Ряд 2, Шкаф 2, Позиция 2)
'310' (Ряд 3, Шкаф 2, Позиция 3)
'340' (Ряд 3, Шкаф 1, Позиция 3)
```

Можно заметить, что все данные корректные и программа работает правильно.

Рассмотрим работу программы с файликом на 500 книг

```
'72' (Ряд 22, Шкаф 51, Позиция 1)
'90' (Ряд 40, Шкаф 15, Позиция 6)
'103' (Ряд 48, Шкаф 38, Позиция 8)
'361' (Ряд 49, Шкаф 46, Позиция 8)
'417' (Ряд 51, Шкаф 27, Позиция 4)
'501' (Ряд 12, Шкаф 47, Позиция 4)
'589' (Ряд 5, Шкаф 12, Позиция 13)
'745' (Ряд 48, Шкаф 17, Позиция 14)
'783' (Ряд 35, Шкаф 53, Позиция 16)
'868' (Ряд 47, Шкаф 29, Позиция 14)
'917' (Ряд 43, Шкаф 28, Позиция 14)
'978' (Ряд 53, Шкаф 26, Позиция 4)
'988' (Ряд 43, Шкаф 43, Позиция 9)
'1013' (Ряд 34, Шкаф 5, Позиция 6)
'1058' (Ряд 14, Шкаф 12, Позиция 3)
'1085' (Ряд 9, Шкаф 12, Позиция 14)
'1111' (Ряд 45, Шкаф 29, Позиция 3)
'1116' (Ряд 10, Шкаф 31, Позиция 7)
'1223' (Ряд 44, Шкаф 10, Позиция 9)
'1355' (Ряд 7, Шкаф 26, Позиция 12)
'1406' (Ряд 3, Шкаф 40, Позиция 7)
'1498' (Ряд 1, Шкаф 11, Позиция 16)
'1668' (Ряд 53, Шкаф 8, Позиция 6)
'1689' (Ряд 29, Шкаф 28, Позиция 2)
'1813' (Ряд 41, Шкаф 24, Позиция 6)
'1836' (Ряд 17, Шкаф 11, Позиция 12)
'1887' (Ряд 12, Шкаф 24, Позиция 3)
'1890' (Ряд 2, Шкаф 25, Позиция 15)
'1949' (Ряд 26, Шкаф 15, Позиция 11)
```

Заметим аналогичную корректную работу и запись в файлик.

Рассмотрим работу с вводом данных из консоли.

Аналогично запускаем программу.

```
vadimzenin@MacBook-Pro-Vadim ИДЗ - 4. 28 вариант % g++ main.cpp -o main -std=c++20
vadimzenin@MacBook-Pro-Vadim ИДЗ - 4. 28 вариант % ./main -console
Введите количество книг: 10
```

Вводим данные.

```
Введите данные для книги:
0 0 1 145
0 1 2 230
0 1 3 110
1 0 1 215
1 0 2 190
1 1 1 275
2 0 1 120
2 0 2 340
2 1 1 85
2 1 2 310
```

Получаем следующие результаты:

```
Начальное состояние библиотеки:
Ряд 1, Шкаф 1, Порядок книги в шкафу 2 (Название: 145)
Ряд 1, Шкаф 2, Порядок книги в шкафу 3 (Название: 230)
Ряд 1, Шкаф 2, Порядок книги в шкафу 4 (Название: 110)
Ряд 2, Шкаф 1, Порядок книги в шкафу 2 (Название: 215)
Ряд 2, Шкаф 1, Порядок книги в шкафу 3 (Название: 190)
Ряд 2, Шкаф 2, Порядок книги в шкафу 2 (Название: 275)
Ряд 3, Шкаф 1, Порядок книги в шкафу 2 (Название: 120)
Ряд 3, Шкаф 1, Порядок книги в шкафу 3 (Название: 340)
Ряд 3, Шкаф 2, Порядок книги в шкафу 2 (Название: 85)
Ряд 3, Шкаф 2, Порядок книги в шкафу 3 (Название: 310)
Студент 1 успешно добавил книгу '145' (Ряд 1, Шкаф 1, Позиция 2) в каталог
Студент 1 успешно добавил книгу '215' (Ряд 2, Шкаф 1, Позиция 2) в каталог
Студент 2 успешно добавил книгу '230' (Ряд 1, Шкаф 2, Позиция 3) в каталог
Студент 3 успешно добавил книгу '110' (Ряд 1, Шкаф 2, Позиция 4) в каталог
Студент 3 успешно добавил книгу '120' (Ряд 3, Шкаф 1, Позиция 2) в каталог
Студент 4 успешно добавил книгу '340' (Ряд 3, Шкаф 1, Позиция 3) в каталог
Студент 4 успешно добавил книгу '310' (Ряд 3, Шкаф 2, Позиция 3) в каталог
Студент 2 успешно добавил книгу '275' (Ряд 2, Шкаф 2, Позиция 2) в каталог
Студент 1 успешно добавил книгу '190' (Ряд 2, Шкаф 1, Позиция 3) в каталог
Студент 3 успешно добавил книгу '85' (Ряд 3, Шкаф 2, Позиция 2) в каталог

Итоговый каталог:
'85' (Ряд 3, Шкаф 2, Позиция 2)
'110' (Ряд 1, Шкаф 2, Позиция 4)
'120' (Ряд 3, Шкаф 1, Позиция 2)
'145' (Ряд 1, Шкаф 1, Позиция 2)
'190' (Ряд 2, Шкаф 1, Позиция 3)
'215' (Ряд 2, Шкаф 1, Позиция 2)
'230' (Ряд 1, Шкаф 2, Позиция 3)
'275' (Ряд 2, Шкаф 2, Позиция 2)
'310' (Ряд 3, Шкаф 2, Позиция 3)
'340' (Ряд 3, Шкаф 1, Позиция 3)
```

Заметим прекрасную работу программы на всех входных данных. Входные данные input10/100/500.txt и catalog_output100/500.txt можно найти в архиве.

Данным образом реализовано использование флагов в моей программе

```cpp
int main(int argc, char* argv[]) {
    // Проверка наличия флагов и аргументов командной строки
    if (argc < 2) {
        std::cerr << "Ошибка: Не указан режим работы. Используйте --generate, --console или --file <filename>.\n";
        return 1;
    }
    std::vector<Book> books; // Вектор для хранения исходных данных о книгах
    try {
        // Генерация случайных данных
        if (strcmp(argv[1], "-generate") == 0) {
            int num_books = 52; // По умолчанию генерируется 52 книг
            if (argc == 3) {
                num_books = std::stoi(argv[2]); // Количество книг можно передать как аргумент
            }

            books = Book::generateTaskBackpack(num_books); // Генерация книг
        }
        // Ввод данных с консоли
        else if (strcmp(argv[1], "-console") == 0) {
            books = Book::inputBooksFromConsole(); // Ввод книг с консоли
        }
        // Чтение данных из файла
        else if (strcmp(argv[1], "-file") == 0) {

            if (argc < 3) {
                throw std::runtime_error("Ошибка: Укажите имя файла после флага -file.");
            }

            books = Book::readBooksFromFile(argv[2]); // Чтение книг из указанного файла
        }
        // Если флаг не распознан
        else {
            throw std::runtime_error("Ошибка: Неизвестный флаг. Используйте -generate <numbers of book>, -console или -file <file name>.");
        }
```

Таким образом можно заметить, что учтены все требования для оценок 4-8, реализовано многопоточное консольное приложения, использующие "портфель задач" для составления каталога библиотеки.

Отчет на 9.

Еще одно решение представлено в файлике yetAnotherMain.cpp. В данной программе использовались условные переменные совместно с мьютексами. Проведем сравнительный анализ их работы:

```
vadimzenin@MacBook-Pro-Vadim ИДЗ - 4. 28 вариант % g++ yetAnotherMain.cpp -o main2 -std=c++20
vadimzenin@MacBook-Pro-Vadim ИДЗ - 4. 28 вариант % ./main2 -file input10.txt
Начальное состояние библиотеки:
Ряд 1, Шкаф 1, Порядок книги в шкафу 2 (Название: 145)
Ряд 1, Шкаф 2, Порядок книги в шкафу 3 (Название: 230)
Ряд 1, Шкаф 2, Порядок книги в шкафу 4 (Название: 110)
Ряд 2, Шкаф 1, Порядок книги в шкафу 2 (Название: 215)
Ряд 2, Шкаф 1, Порядок книги в шкафу 3 (Название: 190)
Ряд 2, Шкаф 2, Порядок книги в шкафу 2 (Название: 275)
Ряд 3, Шкаф 1, Порядок книги в шкафу 2 (Название: 120)
Ряд 3, Шкаф 1, Порядок книги в шкафу 3 (Название: 340)
Ряд 3, Шкаф 2, Порядок книги в шкафу 2 (Название: 85)
Ряд 3, Шкаф 2, Порядок книги в шкафу 3 (Название: 310)
Студент 10 успешно добавил книгу '145' (Ряд 1, Шкаф 1, Позиция 2) в каталог
Студент 9 успешно добавил книгу '340' (Ряд 3, Шкаф 1, Позиция 3) в каталог
Студент 8 успешно добавил книгу '120' (Ряд 3, Шкаф 1, Позиция 2) в каталог
Студент 5 успешно добавил книгу '230' (Ряд 1, Шкаф 2, Позиция 3) в каталог
Студент 3 успешно добавил книгу '190' (Ряд 2, Шкаф 1, Позиция 3) в каталог
Студент 4 успешно добавил книгу '275' (Ряд 2, Шкаф 2, Позиция 2) в каталог
Студент 7 успешно добавил книгу '85' (Ряд 3, Шкаф 2, Позиция 2) в каталог
Студент 2 успешно добавил книгу '110' (Ряд 1, Шкаф 2, Позиция 4) в каталог
Студент 6 успешно добавил книгу '215' (Ряд 2, Шкаф 1, Позиция 2) в каталог
Студент 10 успешно добавил книгу '310' (Ряд 3, Шкаф 2, Позиция 3) в каталог

Итоговый каталог:
'85' (Ряд 3, Шкаф 2, Позиция 2)
'110' (Ряд 1, Шкаф 2, Позиция 4)
'120' (Ряд 3, Шкаф 1, Позиция 2)
'145' (Ряд 1, Шкаф 1, Позиция 2)
'190' (Ряд 2, Шкаф 1, Позиция 3)
'215' (Ряд 2, Шкаф 1, Позиция 2)
'230' (Ряд 1, Шкаф 2, Позиция 3)
'275' (Ряд 2, Шкаф 2, Позиция 2)
'310' (Ряд 3, Шкаф 2, Позиция 3)
'340' (Ряд 3, Шкаф 1, Позиция 3)

Каталог сохранен в файл catalog_output.txt.
```

```
'85' (Ряд 3, Шкаф 2, Позиция 2)
'110' (Ряд 1, Шкаф 2, Позиция 4)
'120' (Ряд 3, Шкаф 1, Позиция 2)
'145' (Ряд 1, Шкаф 1, Позиция 2)
'190' (Ряд 2, Шкаф 1, Позиция 3)
'215' (Ряд 2, Шкаф 1, Позиция 2)
'230' (Ряд 1, Шкаф 2, Позиция 3)
'275' (Ряд 2, Шкаф 2, Позиция 2)
'310' (Ряд 3, Шкаф 2, Позиция 3)
'340' (Ряд 3, Шкаф 1, Позиция 3)
```

Заметим, что данные корректно записались в файлик и повторяют решение, сделанное программой на 8.
Следовательно можно сделать вывод о том, что программ аналогично прошлой работает корректно и полностью реализует нужный функционал.

Отчет на 10.

Программа представлена в файлике main10.cpp. Заметим, что работает она аналогично.

'85' (Ряд 3, Шкаф 2, Позиция 2)
'110' (Ряд 1, Шкаф 2, Позиция 4)
'120' (Ряд 3, Шкаф 1, Позиция 2)
'145' (Ряд 1, Шкаф 1, Позиция 2)
'190' (Ряд 2, Шкаф 1, Позиция 3)
'215' (Ряд 2, Шкаф 1, Позиция 2)
'230' (Ряд 1, Шкаф 2, Позиция 3)
'275' (Ряд 2, Шкаф 2, Позиция 2)
'310' (Ряд 3, Шкаф 2, Позиция 3)
'340' (Ряд 3, Шкаф 1, Позиция 3)