

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение высшего образования  
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА (ПРОЕКТ)  
ЗАЩИЩЕНА С ОЦЕНКОЙ  
РУКОВОДИТЕЛЬ

доц., к.т.н		А.В. Туманова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОЙ РАБОТЕ

РАЗРАБОТКА ПРОГРАММЫ  
«КАТАЛОГ ТОВАРОВ»

по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4233К		С.Р. Павлов
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

## СОДЕРЖАНИЕ

1. Постановка задачи .....	4
1.1 Общие описание задачи .....	4
1.2 Вариант задания.....	4
2. Описание структур данных .....	4
2.1 Связанный список.....	4
2.2 Класс каталога.....	6
3. Описание программы и созданных функций.....	7
3.1 Общая структура программы .....	7
3.2 Описание функций интерфейса .....	8
3.2.1 Управляющие функции .....	8
3.2.2 Функции обработки пути базы данных.....	10
3.2.3 Функции-действия .....	12
3.3 Методы класса каталога.....	18
4. Описание пользовательского интерфейса.....	20
4.1 Открытие базы данных .....	20
4.2 Главное меню.....	20
4.3 Вывод списка товаров .....	21
4.4 Под-меню редактирования товара .....	21
4.5 Меню поиска товара.....	22
4.6 Под-меню сортировки.....	22
5. Результаты тестирования программы.....	23
5.1 Юнит-тестирование классов.....	23
5.2 Итоговое тестирование приложения .....	25
5.2.1 Тестирование на открытие/чтения базы данных.....	26
5.2.2 Тестирование на некорректный ввод в выводе товаров.....	27
5.2.3 Тестирование добавления товара .....	28
5.2.4 Тестирование удаление товара .....	29
5.2.5 Тестирование поиска товара/продавца .....	30
5.2.6 Тестирование редактирования товара .....	30
5.2.7 Тестирование сортировки списка товаров.....	31
ЗАКЛЮЧЕНИЕ.....	32
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	33

ПРИЛОЖЕНИЕ А .....	34
ПРИЛОЖЕНИЕ Б .....	57
ПРИЛОЖЕНИЕ В.....	69

## **1. Постановка задачи**

### **1.1 Общие описание задачи**

Задачей курсовой работы является разработка программы для заданной предметной области – “*Каталог товаров*” на языке программирования C++, с использованием заданных структур данных, которая позволяет вводить информацию, хранить ее в файле (в формате таблицы .csv), осуществлять поиск и вывод нужной информации различными способами, производить модификацию и сортировку данных.

### **1.2 Вариант задания**

Тип хранимой информации и задание на поиск определяются в соответствии с номером варианта:

Вариант 18:

Предметная область – «Каталог товаров». Данные о товаре хранятся в структуре с именем PRICE, содержащей следующие поля:

- название товара;
- название магазина, в котором продаётся товар;
- стоимость товара в рублях.

Задание на поиск: найти информацию о товарах, продающихся в магазине, название которого введено с клавиатуры.

## **2. Описание структур данных**

### **2.1 Связанный список**

В качестве подходящей структуры данных, для хранения информации о товарах, был выбран и реализован линейный двусвязный список. Основными преимуществами, перед использованием классических массивов или векторов стали:

1. Эффективное частое добавление/удаление товаров.
2. Возможность быстро пере-связывать указатели на элементы, а не присваивать их, например для такой операции как сортировка.
3. Простой контроль памяти и размера, при частых изменениях размера списка.

Линейный двусвязный список – реализован в виде, шаблонного класса-контейнера, и схож с аналогичным ему в стандартной библиотеки (`std::list<T>`). Реализация списка, определена в файле *list.h*.

Далее будет приведен код в упрощенном виде (*без полного определения структур и функций*), полное описание и исходный код списка, находится в [ПРИЛОЖЕНИЕ Б].

#### Шаблонный класс списка:

```
template <typename T>
class List
{
private:
    // Структура узла
    struct LNode
    {
        T Item; // Элемент
        LNode* Next; // Указатель на след. узел
        LNode* Prev; // Указатель на пред. узел
    };

    // Определение класса итератора, для доступа к элементам.
    class Iterator { /*.....*/ }

    size_t listSize; // Размер списка
    LNode* listHead; // Указатель на голову списка
    LNode* listTail; // Указатель на хвост списка

    /* скрытые функции сортировки.....*/

public:
    /* Класс итератора, для списка...*/
    /* Конструкторы, деструкторы...*/
    /* Функции-геттеры, для получения размеров структуры.....*/
    /* Функции определяющие основные операции...*/
    /* Функции для получение итераторов .....*/
    /* Отладочные функции .....*/
};
```

#### Основные операции, для работы со списком:

1. Добавить/удалить в конец
2. Добавить/удалить в начало
3. Добавить/удалить по индексу
4. Оператор индексирования
5. Сортировка слиянием, принимающая в качестве аргумента функцию-сравнения
6. Очистка
7. Проверка пустоты, и получение размера списка.
8. Методы для получения итераторов указывающих на начало и конец списка.

## 2.2 Класс каталога

Согласно заданию, данные о товаре хранятся в структуре `catalog::PRICE`:

```
struct PRICE
{
    std::wstring product_name; // Название товара
    std::wstring seller_name;  // Название продавца
    double price;              // Цена в рублях
};
```

Класс каталога – представляет собой, простую реализацию каталога товаров, с которым можно работать, как с целостной сущностью. Класс хранит строку - путь к базе данных, и линейный двусвязный список товаров, описанный ранее.

Каталог осуществляет хранение, модификацию и загрузку в память базы данных каталога, в качестве формата базы данных используется .csv таблица (*Comma-Separated Values* — значения, разделённые запятыми).

```
class Catalog
{
private:
    std::wstring db_path; // Относительный путь к файлу базы данных
    List<PRICE> items_list; // Список товаров
public:
    /* Методы ...*/
};
```

Полная реализация и исходные коды находятся в [ПРИЛОЖЕНИЕ А] (листинги *catalog.h* и *catalog.cpp*).

Основные операции, реализуемые Классом каталога:

1. Работа с базой данных
2. Поиск и получение товара
3. Добавление товара
4. Удаление товара
5. Сортировка товаров

Все строки используемые в реализации, имеют не стандартный однобайтовый тип `char`, а `wchar` (двухбайтовые символы), или же UNICODE-16 символы, представленные стандартным классом ‘wide-string’ `std::wstring`, и Си-строками типа `wchar_t*` соответственно.

### **3. Описание программы и созданных функций**

#### **3.1 Общая структура программы**

Программа реализована на языке C++ в виде консольного приложения, она разделена на несколько файлов, классов и структур.

#### **Организация и структура программы:**

1. Консольный пользовательский интерфейс реализован в файле *main.cpp*, где определено все множество функций, для работы с пользователем, а именно все возможные функции вывода/печати, удаление, добавления, сортировки и другие. В этом файле, так же определена главная функция *main()*, а так же функции для обработки пользовательского ввода, и вывода. И так же, в этом файле производится обработка параметров запуска программы.
2. Класс каталога реализован в файле *catalog.cpp*, он содержит, реализацию всех операций над каталогом, и получить/изменить и как-то взаимодействовать с каталогом товаров можно только через него.
3. Шаблонный класс списка, реализован в файле *list.h*
4. Программа юнит-тестирования, реализована в подпроекте основного решения в файле *coursework\_tests.cpp*, для тестирования корректности реализованных структур, классов и методов.

## Структурная иерархия программы:



### 3.2 Описание функций интерфейса

Далее приводится описание функций, определенных в файле *main.cpp* – консольного интерфейса программы.

#### 3.2.1 Управляющие функции

- Функция `main()`

Прототип: `int main(int argc, char* argv[]);`

Является точкой входа, для проекта. Принимает входные параметры запуска программы. С помощью функций *WinAPI*, устанавливает язык локализации и кодировку консоли на поддержку русских символов. Затем вызывает функцию `run_interface(argc, argv)` – запуска интерфейса.



- **Функция `run_interface()`**

Прототип: `void run_interface(int argc, char* argv[]);`

Главная функция обработчик консольного интерфейса.

В первую очередь, по переданным в нее параметрам запуска программы, вызывает функцию `get_db_path()` задача которой установить путь к базе данных, либо через переданные аргументы в параметрах запуска программы, или же через пользовательский ввод с клавиатуры.

Затем с помощью функции `open_catalog()` произойдет попытка открытия, или создания новой базы данных, по полученному выше пути.

Далее следует цикл обработки пользовательского ввода, на экран (консоль) пользователя на каждой итерации цикла, выводится “меню управления каталогом”, каждый раз пользователь вводит номер желаемого действия, этот цикл продолжает выполняться, пока не будет передана команда 0 (выход). Все возможные номера команд, для удобства и структурности представлены в перечислимом типе `enum MENU_ACTIONS`. После обработки пользовательского ввода (команды) с помощью функции `get_action()`, вызывается функция `process_action()` для выполнения указанной команды.

В конце функции, после выхода из цикла обработки, будет вызвана функция `save_catalog()` – для сохранения модифицированной базы данных на диск.

Пример выполнения:

```
-----| Меню управления каталогом |-----
[1] Вывести список товаров
[2] Вывести список продавцов
[3] Вывести список продавцов, конкретного товара
[4] Вывести список товаров, конкретного продавца

[5] Добавить - товар
[6] Удалить - товар (конкретную позицию)
[7] Удалить - продавца (и все его товары)
[8] Удалить - товар у всех (удалит определенный товар у всех
продавцов)

[9] Поиск товара
[10] Поиск продавца

[11] Редактировать - товар

[12] Отсортировать список
```

```
[13] Сохранить базу данных
[0] Выход

Выбор действия >>
```

- **Функция process\_action()**

Прототип: `int process_action(int action);`

Вызывает функции (действия) согласно предоставленному номеру команды. Представляет собой большой блок *switch – case*.

### 3.2.2 Функции обработки пути базы данных

- **Функция get\_db\_path()**

Прототип: `void get_db_path(int argc, char* argv[]);`

Функция устанавливает путь к базе данных, проверяя его корректность, либо из параметров запуска программы, либо из пользовательского ввода.

Если в параметрах запуска программы, был передан путь к файлу базы данных, программа сначала попытается обработать его, иначе будет вызвано сообщение и приглашение к вводу, пути вручную.

Функция так же проверяет, что файл указанной базы данных, должен иметь расширение .csv.

В случае не удачи, будет запрашивать снова ввести правильный путь.

Пример #1 выполнения:

```
[DataBase] Открытие базы данных [C:\Users\user\Desktop\Курсовая
финал\final\coursework_final\coursework_final\hello.csv]
[DataBase] Загружена существующая база данных : [hello.csv]
```

Пример #2 выполнения:

```
[DataBase] Введите путь до базы данных : hello.csv
[DataBase] Открытие базы данных [C:\Users\user\Desktop\Курсовая
финал\final\coursework_final\coursework_final\hello.csv]
[DataBase] Загружена существующая база данных : [hello.csv]
```

- **Функция open\_catalog()**

Прототип: `void open_catalog(void);`

С помощью методов класса каталога, открывает или создает базу данных, по заданному ранее пути, обрабатывая исключительные ситуации и ошибки.

### 3.2.3 Функции-действия

- **Вывести список всех товаров - `action_print_all_goods()`**

Выводит весь список товаров на экран. Так же проверяет размер списка товаров, если список пуст – выводит сообщение.

Пример выполнения:

```
| Вывод списка всех товаров |
-----<Товар #0>-----
Название : стол
Продавец : оби
Цена : 10,000
-----<Товар #1>-----
Название : abs
Продавец : qwerty
Цена :
0,240
-----<Товар #2>-----
Название : горшок для цветов
Продавец : оби
Цена : 725,500
```

- **Вывести список продавцов - `action_print_all_sellers()`**

Выводит всех продавцов на экран. Так же проверяет размер списка товаров, если список пуст – выводит сообщение.

Пример выполнения:

```
| Вывод списка всех продавцов |
1. оби
2. qwerty
3. петрович
```

- **Вывести список продавцов, конкретного товара - `action_print_all_good_sellers()`**

Выводит список продавцов, конкретного товара на экран.

Для определения конкретного товара, вызывает `get_user_string()` функцию – для ввода названия товара с клавиатуры.

Пример выполнения:

```
| Вывод списка продавцов, определенного товара |
Введите название товара: стул
Продавцы товара (стул):
1. | петрович      | 1000,000
2. | оби           | 3333,000
```

- **Вывести список товаров, конкретного продавца-  
action\_print\_all\_seller\_goods()**

Выводит список товаров, конкретного продавца на экран.

Для определения конкретного продавца, вызывает `get_user_string()` функцию – для ввода названия товара с клавиатуры.

Пример выполнения:

```
| Вывод списка товаров, определенного продавца |
Введите имя продавца: оби
Товары продавца (оби):
1. | стол          | 10,000
2. | горшок для цветов | 725,500
3. | стул          | 3333,000
```

- **Добавление товара - action\_add\_good()**

Добавляет товар по проводя проверки на корректность ввода, и наличие в списке. Для добавление товара требуется три значения: название товара, имя продавца и цена, для получения этих значений вызывается функция `get_user_string()` – которая помимо ввода, проверяет строки на корректность. Так же функция производит проверку корректности цены. Если какое-либо значения введено неверно, пользователю будет предложено ввести его снова. Перед добавлением товара с помощью метода класса каталога `.check_itemExists()` – производится проверка на уже существующий товар, по заданным значениям. Затем если все величины введены корректно, и уже не существует в списке, происходит его добавление через метод класса каталога `.add_item()`.

Пример выполнения:

```
Пример выполнения:
| Добавление товара |
Название товара : Гитара
Продавец : МузТорг
Цена : 28500
```

Товар (гитара) – добавлен!

- **Удалить - товар (конкретную позицию) - `action_del_good()`**

Удаляет определенный товар, у определенного продавца. Для удаления функция вызывает `get_user_string()` – для получения названия товара и имени продавца, проводя проверки на корректность введенных строк. Так же проверяет наличие этого товара в списке, затем вызывает метод класса каталога `del_item()` – для удаления товара из списка, иначе выводит сообщение об ошибке.

Пример выполнения:

```
| Удаление товара |  
Название товара : диван  
Имя продавца : петрович  
Товар (диван) у продавца (петрович) удален!
```

- **Удалить - продавца (и все его товары) - `action_del_seller()`**

Функция удаляет все товары у указанного продавца. Вызывает функцию `get_user_string()` – для получения имени продавца. Затем с помощью метода класса каталога `del_seller()` – производит удаление всех товаров, которые продает данный продавец. В случае если таких товаров или продавца не существует – выводит сообщение об ошибке.

Пример выполнения:

```
| Удаление продавца |  
Имя продавца : оби  
Продавец (оби) – удален
```

- **Удалить - товар у всех (удалит определенный товар у всех продавцов) - `action_del_same_good()`**

Функция удаляет указанный товар у всех продавцов. Вызывает функцию `get_user_string()` – для получения названия товара. Затем с помощью метода класса каталога `del_product()` – производит удаление данного товара у всех продавцов. В случае если такого товара, нет ни одного продавца – выводит сообщение об ошибке.

Пример выполнения:

```
| Удаление опеределенного товара, у всех продавцов |
Название товара : стул
Товар (стул) - удален
```

- **Поиск товара - action\_find\_good()**

Функция ищет товар, по указанной подстроке. Выводит на экран позиции найденного товара у всех продавцов.

Сначала функция проверяет, пуст ли список товаров, в этом случае выводит соответствующее сообщение. Затем с помощью функции `get_user_string()` – получает подстроку с названием искомого товара, проводя проверки на корректность строки. Далее, с помощью итератора на список товаров, производится обход списка, и сравнение названий товаров с полученной подстрокой, и выводятся на экран позиции товара. Если в ходе поиска ничего не было найдено – выводится сообщение об ошибке.

Пример выполнения:

```
| Поиск товара |
Название товара :
стул
-----[Товары подходящие описанию]-----
-
1. | стул          | петрович      | 1000,000000
2. | стул          | оби           | 3333,000000
```

- **Поиск продавца - action\_find\_seller()**

Функция выводит все товары, по указанному имени продавца. Выводит на экран позиции найденного товара у данного продавца.

Сначала функция проверяет, пуст ли список товаров, в этом случае выводит соответствующее сообщение. Затем с помощью функции `get_user_string()` – получает подстроку с именем продавца, проводя проверки на корректность строки. Далее, с помощью итератора на список товаров, производится обход списка, и сравнение имен продавцов с полученной подстрокой, и выводятся на экран позиции товара. Если в ходе поиска ничего не было найдено – выводится сообщение об ошибке.

Пример выполнения:

```
| Поиск продавца |
Имя продавца : петро

-----[Продавцы подходящие описанию]-----
1. | стул          | петрович      | 1000,000000
2. | диван          | петрович      | 10000,000000
```

- **Редактирование товара - action\_edit\_good()**

Функция производит редактирования товара, для конкретной позиции (товар : продавец). В позиции товара можно изменить любое поле структуры PRICE, а именно название товара, имя продавца и цену. Так же производится множество проверок, которые защищают список от повторений одного и того же товара.

Сначала функция вызывает `get_user_string()` – для получения названия товара, и имени продавца, для определения изменяемого товара, и проверяет существует ли данный товар (по введённым названию;продавцу) в списке. Если товара не существует – выводится сообщение об ошибке, а также возможность пользователю выйти из меню редактирования, или начать ввод заново.

После того, как редактируемый товар, успешно определен, пользователю предлагается выполнить одни из трех возможных модификаций товара: 1. Изменить название товара 2. Изменить продавца товара 3. Изменить цену

Далее будет выполнена проверка на уникальность измененной позиции. Если новое название товара, или новое имя продавца, изменит товар таким образом, что такая позиция уже существует в списке товаров (случиться повторение товаров), то будет вызвано сообщение об ошибке. Если проверка на уникальность пройдена, то позиция будет успешно изменена.

Редактирование и обращение к структуре PRICE (элементу списка каталога), производится с помощью метода класса каталога `get_item()`.

Пример выполнения:

```
| Изменение товара |
Название товара : стул
Имя продавца : оби

[1] Изменить название
[2] Изменить продавца
[3] Изменить цена
```



```
Выбор действия >> 1
Новое название товара : стул для кухни
Название товара - изменено!
```

Пример выполнения (повторение товара):

```
| Изменение товара |
Название товара : стул
Имя продавца : оби
[1] Изменить название
[2] Изменить продавца
[3] Изменить цена
Выбор действия >> 2
Новое имя продавца : петрович
> Ошибка : такой товар, у данного продавца - уже существует!
```

- **Отсортировать список - action\_sort\_list()**

Функция производит сортировку списка товаров, с помощью метода класса каталога `sort_itemList()`. На выбор пользователю предлагается 4 вида сортировки: 1. По названию товара (алфавитная) 2. По имени продавца (алфавитная) 3. По цене (от большей к меньшей) 4. По цене (От меньшей к большей).

Различные виды сортировки, реализованы с помощью, четырех функций-компараторов, передаваемых в метод класса, как у казатель на функцию.

```
bool pname_comp(catalog::PRICE& a, catalog::PRICE& b);
bool sname_comp(catalog::PRICE& a, catalog::PRICE& b);
bool price_lower_comp(catalog::PRICE& a, catalog::PRICE& b);
bool price_higher_comp(catalog::PRICE& a, catalog::PRICE& b);
```

Пример выполнения:

```
| Сортировка списка товаров |
Как отсортировать список ? :
[1] По названию товара
[2] По продавцам
[3] По цене (от меньшей)
[4] По цене (от большей)
Выбор действия >> 1
Список товаров - отсортирован
```

### 3.3 Методы класса каталога

Описание методов класса каталога, определённого в файле *catalog.cpp*.

- **Метод – Открытие базы данных:**

`Catalog::open_db(std::wstring path)`

Данная функция, открывает (загружает в память существующую базу данных) или создает новую, по данному пути. В случае неудачи открытия базы данных, выводит сообщение об ошибке, и выбрасывает исключение. В случае каких-либо ошибок при чтении существующей базы данных, выводит сообщения об ошибках, в каких конкретно строках она была вызвана и по какой причине.

В начале, функция создает два объекта потока (ввод / вывод), и настраивает кодировку страницы, для поддержки русского языка, для этих потоков.

Затем функция пытается открыть, существующий файл, по заданному пути. Если такой файл найден, производится чтение и загрузка базы данных в память программы, иначе создается новый файл базы данных.

В конце метода, заданный путь, сохраняется в приватном поле класса, который можно получить с помощью геттера `get_db_path()`.

Так же функция, возвращает тип выполненной операции, либо база данных создана, либо открыта существующая.

- **Метод – Сохранение базы данных:**

`Catalog::save_db(void)`

Данная функция, сохраняет базу данных, в формате таблицы *.csv*. Сначала функция, создает выходной поток, и настраивает кодировку страницы, для поддержки русского языка, затем пытается открыть файл, если файл не был открыт, до выбрасывается исключение.

Далее, если список товаров не пуст, происходит запись в файл, где каждая строка представляет собой три поля: название товара, имя продавца, цена, разделенных символом `','`.

- **Методы – получения итераторов на список товаров:**

```
List<PRICE>::Iterator get_itemList_begin(void);
List<PRICE>::Iterator get_itemList_end(void);
```

Данные методы класса каталога, позволяют получить итераторы на начало и конец списка товаров.

- **Метод – получение размера списка товаров:**

```
size_t catalog::Catalog::get_size()
```

Возвращает актуальный размер списка товаров.

- **Метод – проверка на наличие товара в списке:**

```
bool check_itemExists(std::wstring p_name, std::wstring s_name);
bool check_itemExists(PRICE &prodd);
```

Возвращает `True` (истина), если товар существует в списке, иначе `false`.

- **Метод – получение ссылки (доступа) на конкретный товар в списке:**

```
PRICE& get_item(std::wstring p_name, std::wstring s_name);
```

Данный метод класса каталога, возвращает ссылку на найденный товар (структура `catalog::PRICE`), по названию и имени продавца.

- **Метод – получение ссылки (доступа) на конкретный товар в списке:**

```
PRICE& get_item(std::wstring p_name, std::wstring s_name);
```

Данный метод класса каталога, возвращает ссылку на найденный товар (структура `PRICE`), по названию и имени продавца.

- **Методы – удаления:**

```
void del_item(std::wstring p_name, std::wstring s_name);
bool del_seller(std::wstring s_name);
bool del_product(std::wstring p_name);
```

Первый метод – удаляет конкретную позицию/товар, по данному названию и имени продавца.

Второй метод – удаляет все товары, конкретного продавца.

Третий метод – удаляет указанный товар, у всех продавцов.

## 4. Описание пользовательского интерфейса

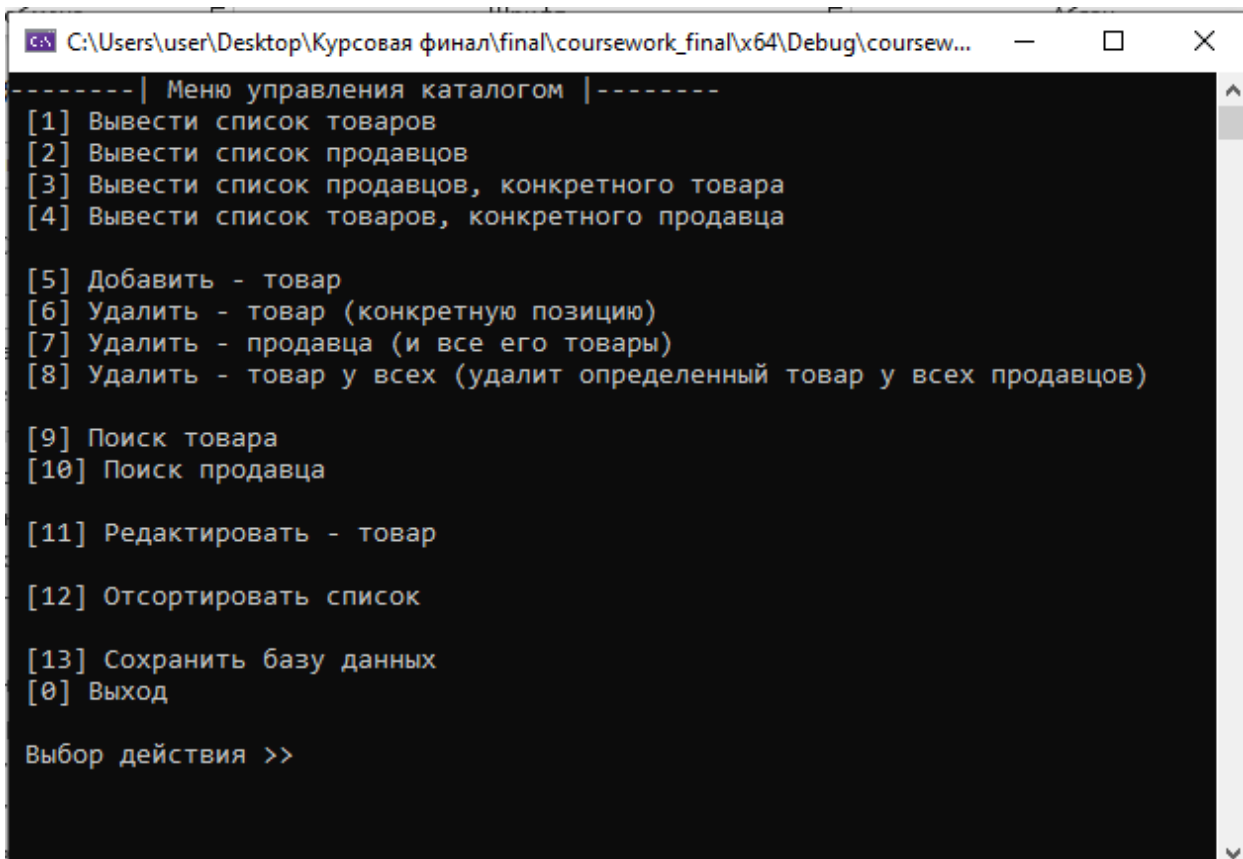
### 4.1 Открытие базы данных

При запуске программы, пользователю будет предложено ввести с клавиатуры путь до файла базы данных (если пользователь явно не указал в аргументах запуска приложения, путь до базы данных).

### 4.2 Главное меню

После открытия базы данных, на экран выводится консольное приложение с меню пользователя. При нажатии на клавиатуре на определенную цифру выполняется команда, соответствующая описанию.

Для удобства пользования экраном и восприятия информации пользователем, при каждом выборе определенной команды, происходит очистка экрана, и 'пауза ввода', чтобы пользователь мог оценить результат выбранной операции, так же возвращение в основное меню сопровождается очисткой экрана.



```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\Debug\coursew...
-----| Меню управления каталогом |-----
[1] Вывести список товаров
[2] Вывести список продавцов
[3] Вывести список продавцов, конкретного товара
[4] Вывести список товаров, конкретного продавца

[5] Добавить - товар
[6] Удалить - товар (конкретную позицию)
[7] Удалить - продавца (и все его товары)
[8] Удалить - товар у всех (удалит определенный товар у всех продавцов)

[9] Поиск товара
[10] Поиск продавца

[11] Редактировать - товар
[12] Отсортировать список

[13] Сохранить базу данных
[0] Выход

Выбор действия >>
```

### 4.3 Вывод списка товаров

```
C:\Users\user\Desktop\Курсовая финал\final\course
| Вывод списка всех товаров |
-----<Товар #0>-----
Название : стол
Продавец : оби
Цена : 10,000

-----<Товар #1>-----
Название : abs
Продавец : qwerty
Цена : 0,240

-----<Товар #2>-----
Название : горшок для цветов
Продавец : оби
Цена : 725,500

-----<Товар #3>-----
Название : стул
Продавец : петрович
Цена : 1000,000

-----<Товар #4>-----
Название : стул
Продавец : оби
Цена : 3333,000

-----<Товар #5>-----
Название : диван
Продавец : петрович
Цена : 10000,000

Для продолжения нажмите любую клавишу . . .
```

### 4.4 Под-меню редактирования товара

При редактировании товара, пользователь может выбрать какое именно поле структуры PRICE, он хочет изменить.

```
C:\Users\user\Desktop\Курсовая финал\final\course
| Изменение товара |
Название товара : стул
Имя продавца : оби

[1] Изменить название
[2] Изменить продавца
[3] Изменить цена

Выбор действия >>
```

## 4.5 Меню поиска товара

C:\Users\user\Desktop\Курсовая финал\final\coursework\_final\x64\Debu

```
| Поиск товара |  
Название товара : сту  
  
-----[Товары подходящие описанию]-----  
1. | стул          | петрович      | 1000,000000  
2. | стул          | оби           | 3333,000000  
  
Для продолжения нажмите любую клавишу . . .
```

## 4.6 Под-меню сортировки

Консоль отладки Microsoft Visual Studio

```
| Сортировка списка товаров |  
Как отсортировать список ? :  
[1] По названию товара  
[2] По продавцам  
[3] По цене (от меньшей)  
[4] По цене (от большей)  
  
Выбор действия >>
```

## 5. Результаты тестирования программы

### 5.1 Юнит-тестирование классов

Для тестирования классов на предмет корректности исполнения, и утечек памяти, был выбран стандартный инструмент юнит-тестирования C++ кода, поставляемый Microsoft Visual Studio – *CppUnitTest*.

Класс юнит-теста, определён в листинге *coursework\_tests.cpp*, исходный код юнит-тестов находится в [ПРИЛОЖЕНИЕ В].

Для проверок на утечки памяти, используется общие средства библиотеки отладки и времени выполнения Microsoft::CRTDBG, проверка осуществляется по следующему алгоритму:

1. Делается снимок кучи (heap сегмент), в начале вызова функции.
2. Далее в локальном блоке, происходит вызов тестируемых функций, классов и т.д.
3. Затем сразу за пределами локального блока, делается снимок текущей кучи, и сравнивается, с прошлой.
4. Если снимки не совпадают, то в результате работы кода, в локальном блоке, произошла утечка памяти, и будет выведено соответственное сообщение и расшифровка снимка памяти, и тест будет – провален.

Схема блок-теста:

```
MEMORY_LEAK_CHECK_BEGIN(); // Начальный Снимок кучи
{ // Начало локально блока

    /* Тут тестируемый код, объекты, ... */

} // Конец локального блока, деструкторы точно отработают после
него.
MEMORY_LEAK_CHECK_END(); // Конечный снимок кучи
```

Определение макросов, для снимков кучи:

```
#define MEMORY_LEAK_CHECK_BEGIN() \
    _CrtMemState sOld; \
    _CrtMemState sNew; \
    _CrtMemState sDiff; \
    _CrtMemCheckpoint(&sOld); \
```

```

#define MEMORY_LEAK_CHECK_END() \
    _CrtMemCheckpoint(&sNew); \
    int status = _CrtMemDifference(&sDiff, &sOld, &sNew); \
    if (status) \
    { \
        Assert::Fail(L"Memory leak - detected!"); \
        OutputDebugString(L"----- _CrtMemDumpStatistics - \
-----"); \
        _CrtMemDumpStatistics(&sDiff); \
        OutputDebugString(L"----- \
_CrtMemDumpAllObjectsSince -----"); \
        _CrtMemDumpAllObjectsSince(&sOld); \
        OutputDebugString(L"----- _CrtDumpMemoryLeaks --- \
-----"); \
        _CrtDumpMemoryLeaks(); \
    }

```

### Программа и содержание тестов:

Примечание: после выполнения каждого теста, происходит проверка на утечки памяти. Во всех тестах предусмотрен обход всех исключительных ситуаций, на сколько это возможно для тестируемой функции.

#### 1. Тестирования класса списка : (`List<T>`)

- a. [DefaultConstructor] : Тестирование конструктора по умолчанию для различных типов.
- b. [PushTests] : Тестирование операций добавления в конец, и в начало списка.
- c. [PopTests] : Тестирование операций удаление из конца и начала списка.
- d. [IteratorsTests] : Тестирование итераторов списка.
- e. [InsertDeleteTests] : Тестирование функций вставки и удаления по индексу.
- f. [SortTests] : Тестирование различных сортировок списка.

#### 2. Тестирования класса каталога : (`catalog::Catalog`)

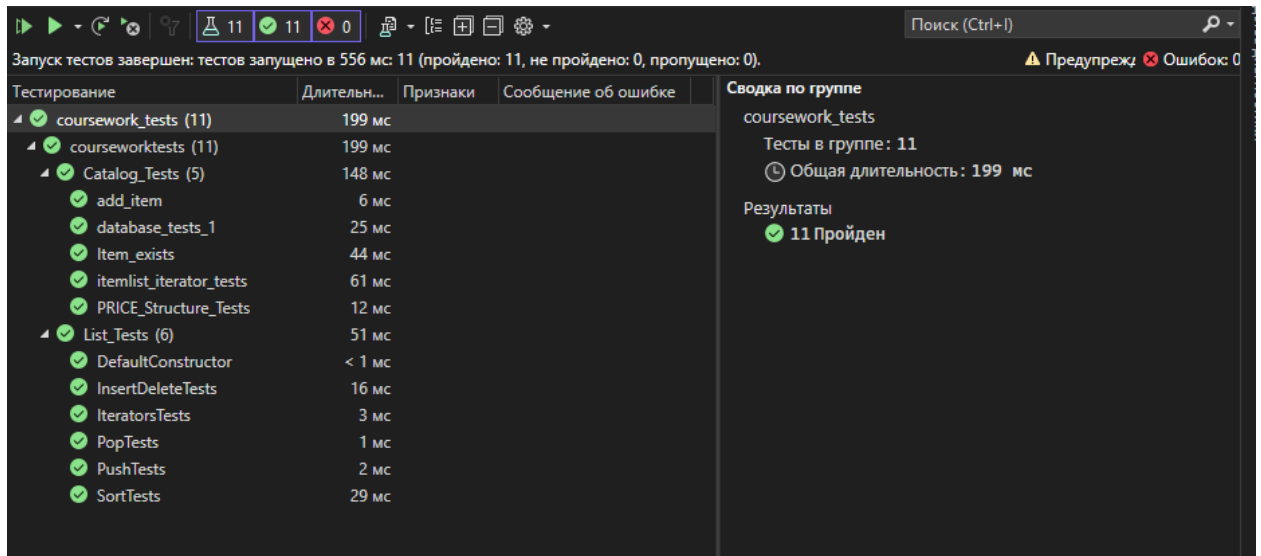
- a. [PRICE\_Structure\_tests] : Тестирование конструкторов структуры товара.
- b. [database\_tests\_1] : Тестирование функций для работы с базами данных, класса каталога
- c. [itemlist\_iterator\_tests] : Тестирование функций для работы с итераторами списка товаров.



- d. [Item\_exists] : Тестирование методов наличие товара в списке.
- e. [add\_item] : Тестирование методов добавления товара.

### Результаты тестирования:

Все тесты пройдены – успешно.



## 5.2 Итоговое тестирование приложения

Выполнения тестирования для всей программы:

**Входные** (тестируемые) **данные**, представлены в виде существующей базы данных (таблица *test\_case.csv*):

Название товара	Имя продавца	Цена
стол	оби	10,000
abs	qwerty	0,240
горшок для цветов	оби	725,500
стул	петрович	1000,000
стул	оби	3333,000
диван	петрович	10000,000

### 5.2.1 Тестирование на открытие/чтения базы данных

Суть теста - проверка корректно ли обрабатывается неправильно заданные строки в исходной базе данных.

Для тестирования корректности чтения, были добавлены следующие 'сломанные' строки (№ 4,5,6,7,8,9,13):

```
1 стол;оби;10,000000
2 abs;qwerty;0,240000
3 горшок для цветов;оби;725,500000
4 фиалка
5 диван;
6 ;;
7 товар1;продавец1;
8 товар2;продавец2;-100
9 ;продавец3;
10 стул;петрович;1000,000000
11 стул;оби;3333,000000
12 диван;петрович;10000,000000
13
```

Программа корректно обработала ошибки, вывела строки где возникла проблема чтения, а другие корректные товары были загружены в память:

```
[DataBase] Открытие базы данных [C:\Users\user\Desktop\Курсовая
финал\final\coursework_final\coursework_final\hello.csv]
[ОШИБКА БД] - чтения строки #4
[ОШИБКА БД] - чтения строки #5
[ОШИБКА БД] - чтения строки базы данных (невозможно получить
название товара) #6
[ОШИБКА БД] - чтения строки базы данных (невозможно получить
цену товара) #7
[ОШИБКА БД] - чтения строки базы данных (невозможно получить
цену товара) #8
[ОШИБКА БД] - чтения строки базы данных (невозможно получить
название товара) #9
[DataBase] Загружена существующая база данных : [hello.csv]
```

Итоговый, загруженный список товаров:

```
-----<Товар #0>-----
Название : стол
Продавец : оби
Цена : 10,000
```

```

-----<Товар #1>-----
Название : abs
Продавец : qwerty
Цена : 0,240

-----<Товар #2>-----
Название : горшок для цветов
Продавец : оби
Цена : 725,500

-----<Товар #3>-----
Название : стул
Продавец : петрович
Цена : 1000,000

-----<Товар #4>-----
Название : стул
Продавец : оби
Цена : 3333,000

-----<Товар #5>-----
Название : диван
Продавец : петрович
Цена : 10000,000

```

### 5.2.2 Тестирование на некорректный ввод в выводе товаров

Проверка команд [3] и [4]. Вывести список продавцов, конкретного товара и Вывести список товаров, конкретного продавца.

Ввод несуществующих значений – успешно обрабатывается и сигнализируется ошибкой:

**Вывести список продавцов, конкретного товара:**

```

C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\
| Вывод списка продавцов, определенного товара |
Введите название товара: максидом
Продавцы товара (максидом):
Товары по вашему запросу - не найдены

Для продолжения нажмите любую клавишу . . .

```

## Вывести список товаров, конкретного продавца:

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\De
| Вывод списка товаров, определенного продавца |
Введите имя продавца: максимдом
Товары продавца (максимдом):
Продавец - не найден

Для продолжения нажмите любую клавишу . . .
```

### 5.2.3 Тестирование добавления товара

Проверка команды [2] - Добавить товар.

## Обработка некорректного ввода:

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\Debug
| Добавление товара |
Название товара : @1231##$)(8%4
> Неправильный ввод, повторите снова!
Название товара : 1341351351351351351351
> Неправильный ввод, повторите снова!
Название товара : Шампанское
Продавец : Перекресток
Цена : -999
Цена : 1500
Товар (шампанское) - добавлен!
Для продолжения нажмите любую клавишу . . .
```

## Обработка случая, если добавляемый товар уже существует:

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\Debug\coursework_fi
| Добавление товара |
Название товара : стул
Продавец : оби
Цена : 1000
> Ошибка : такой товар, у данного продавца - уже существует!
Для продолжения нажмите любую клавишу . . .
```

#### 5.2.4 Тестирование удаление товара

Проверка команд [6] - Удалить - товар (конкретную позицию), [7] - Удалить - продавца (и все его товары), [8] - Удалить - товар у всех (удалит определенный товар у всех продавцов).

##### Удаление несуществующего товара:

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final>
| Удаление товара |
Название товара : лампа
Имя продавца : оби
Ошибка : Товар не найден!

Для продолжения нажмите любую клавишу . . .
```

##### Удаление несуществующего продавца:

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final>
| Удаление продавца |
Имя продавца : максидом
Ошибка : Продавец - не найден!

Для продолжения нажмите любую клавишу . . .
```

##### Удаление несуществующего товара у всех продавцов:

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\Debug>c
| Удаление определенного товара, у всех продавцов |
Название товара : книга
Ошибка : Товар - не найден!

Для продолжения нажмите любую клавишу . . .
```

### 5.2.5 Тестирование поиска товара/продавца

Проверка команд [9] - Поиск товара, [10] - Поиск продавца.

**Поиск товара по подстроке:**

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\Debug>
| Поиск товара |
Название товара : сту

-----[Товары подходящие описанию]-----
1. | стул      | петрович   | 1000,000000
2. | стул      | оби        | 3333,000000

Для продолжения нажмите любую клавишу . . .
```

**Поиск продавца по подстроке:**

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\Debug>
| Поиск продавца |
Имя продавца : петро

-----[Продавцы подходящие описанию]-----
1. | стул      | петрович   | 1000,000000
2. | диван     | петрович   | 10000,000000

Для продолжения нажмите любую клавишу . . .
```

### 5.2.6 Тестирование редактирования товара

Проверка команд [11] – Редактирование товара.

**Корректная обработка ошибки, при которой искомого товара не существует:**

```
C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\Debug>
| Изменение товара |
Название товара : вино
Имя продавца : дикси
> Товар не найден (наберите exit - для выхода)
> Выйти y/n :
```

**Корректная обработка, ситуации при которой, измененный товар будет повторяться (такая позиция уже существует):**

```

C:\Users\user\Desktop\Курсовая финал\final\coursework_final\x64\Debug\coursework_final.exe
| Изменение товара |
Название товара : стул
Имя продавца : оби

[1] Изменить название
[2] Изменить продавца
[3] Изменить цена

Выбор действия >> 1
Новое название товара : стол
> Ошибка : такой товар, у данного продавца - уже существует!
Для продолжения нажмите любую клавишу . . .
  
```

### 5.2.7 Тестирование сортировки списка товаров

Вид сортировки			
По названию товара	По продавцам	По цене (от меньшей)	По цене (от большей)
-----<Товар #0>----- Название : abs Продавец : qwerty Цена : 0,240 -----<Товар #1>--- Название : горшок для цветов Продавец : оби Цена : 725,500 -----<Товар #2>-- - Название : диван Продавец : петрович Цена : 10000,000 -----<Товар #3>-- - Название : стол Продавец : оби Цена : 10,000 -----<Товар #4>-- - Название : стул Продавец : оби Цена : 3333,000 -----<Товар #5>-- - Название : стул Продавец : петрович Цена : 1000,000	-----<Товар #0>----- Название : abs Продавец : qwerty Цена : 0,240 -----<Товар #1>--- Название : стул Продавец : оби Цена : 3333,000 -----<Товар #2>----- Название : стол Продавец : оби Цена : 10,000 -----<Товар #3>-- - Название : горшок для цветов Продавец :оби Цена : 725,500 -----<Товар #4>-- - Название : стул Продавец : петрович Цена : 1000,000 -----<Товар #5>--- - Название : диван Продавец : петрович Цена : 10000,000	-----<Товар #0>----- Название : abs Продавец : qwerty Цена : 0,240 -----<Товар #1>----- Название : стол Продавец : оби Цена : 10,000 -----<Товар #2>----- Название : горшок для цветов Продавец : оби Цена : 725,500 -----<Товар #3>--- - Название : стул Продавец : петрович Цена : 1000,000 -----<Товар #4>--- - Название : стул Продавец : оби Цена : 3333,000 -----<Товар #5>----- - Название : диван Продавец : петрович Цена : 10000,000	-----<Товар #0>----- Название : диван Продавец : петрович Цена : 10000,000 -----<Товар #1>----- Название : стул Продавец : оби Цена : 3333,000 -----<Товар #2>----- Название : стул Продавец : петрович Цена : 1000,000 -----<Товар #3>----- Название : горшок для цветов Продавец : оби Цена : 725,500 -----<Товар #4>----- Название : стол Продавец : оби Цена : 10,000 -----<Товар #5>----- Название : abs Продавец : qwerty Цена : 0,240

## ЗАКЛЮЧЕНИЕ

Результатом курсовой работы является реализованная программа – “Каталог товаров”, с использованием консольного пользовательского интерфейса, на языке программирования C++. Эта программа, позволяет пользователям работать с различными каталогами товаров.

Так же в результате проделанной работы, были закреплены знания структурного и объектно-ориентированного программирования, и реализована структура данных – линейный двусвязный список.

Все части и ветви программы, были протестированы на все возможные нестандартные ситуации, так был проведен контроль на утечки памяти.

Одним из ключевых недостатков программы, является невозможность загрузки – очень больших баз данных (превышает размеры ОЗУ), и следовательно не возможность работать с ними. В перспективе эту проблему можно решить так как, структура связанного списка прекрасно подходит для обработки дробных данных, следует организовать буффер чтения/записи, и обрабатывать базу данных порционно (кусками), а так же сортировка слиянием (используемая в программе, в классе списка) тоже прекрасно работает с неполными массивами данных.

Основные достоинства программы – быстро действие и простота, а так же легкость в модификации и портируемости кода. Приложение разбито на множество отдельных функций, классов, объектов – что позволяет другому программисту, без особых сложностей, модифицировать программу под себя.



## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Ключарев А. А., Матяш В. А., Щекин С. В., Структуры и алгоритмы обработки данных –Учеб. пособие/СПб. ГУАП. СПб., 2003. – 172 с.
2. CppReference.com (самый большой онлайн справочник по C++). URL: <https://en.cppreference.com/w/cpp>

## ПРИЛОЖЕНИЕ А

### ЛИСТИНГ 1 – MAIN.CPP

```
#define NOMINMAX

#include <Windows.h>
#include <iostream>
#include <vector>
#include "catalog.h"

extern void get_db_path(int argc, char* argv[]); // Получение
пути к базе данных
extern void open_catalog(void); // Открытие или создание базы
данных
extern void save_catalog(void); // Сохранение базы данных

extern void run_interface(int argc, char* argv[]);
extern int get_action(void);
extern int process_action(int action);

extern void action_print_all_goods(void);
extern void action_print_all_sellers(void);
extern void action_print_all_good_sellers(void);
extern void action_print_all_seller_goods(void);
extern void action_add_good(void);
extern void action_del_good(void);
extern void action_del_seller(void);
extern void action_del_same_good(void);
extern void action_find_good(void);
extern void action_find_seller(void);
extern void action_edit_good(void);
extern void action_sort_list(void);

/*~~~~~ Функции компараторы, передаваемые в сортировку ~~~~~*/

// Сравнение по названию товара (в алфавитном порядке)
bool pname_comp(catalog::PRICE& a, catalog::PRICE& b)
{
    return (wcscmp(a.product_name.c_str(),
b.product_name.c_str()) < 0) ? true : false;
}

// Сравнение по имени продавца (в алфавитном порядке)
bool sname_comp(catalog::PRICE& a, catalog::PRICE& b)
{
    return (wcscmp(a.seller_name.c_str(),
b.seller_name.c_str()) < 0) ? true : false;
}

// Сравнение по цене товара (от меньшего к большему)
```

```

bool price_lower_comp(catalog::PRICE& a, catalog::PRICE& b)
{
    return a.price < b.price;
}

// Сравнение по цене товара (от большего к меньшему)
bool price_higher_comp(catalog::PRICE& a, catalog::PRICE& b)
{
    return a.price > b.price;
}

// Функция для обработки входящей строки от пользователя
extern std::wstring get_user_string(std::wstring msg);

// Макросы, для очистки и паузы командной строки
#define CLEAR_SCREEN() system("cls")
#define PAUSE_INPUT() system("pause")

// Перечисление всех возможных действий в меню
enum MENU_ACTIONS
{
    EXIT_SIGNAL,

    PRINT_ALL_GOODS,
    PRINT_ALL_SELLERS,
    PRINT_ALL_GOOD_SELLERS,
    PRINT_ALL_SELLER_GOODS,

    ADD_GOOD,
    DEL_GOOD,
    DEL_SELLER,
    DEL_GOOD_FROM_ALL,

    FIND_GOOD,
    FIND_SELLER,

    EDIT_GOOD,

    SORT_LIST,

    SAVE_DB,
};

// Максимальный путь до файла (базы данных)
#ifndef MAX_PATH
#define MAX_PATH 260
#endif

catalog::Catalog CAT;
std::wstring db_f_path;
std::wstring db_f_abs_path;

```

```

int main(int argc, char* argv[])
{
    // Установка кодировки windows-1251 для поддержки русского
    языка
    setlocale(LC_ALL, "Russian");
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    // Запуск интерфейса
    run_interface(argc, argv);

    return 0;
}

void get_db_path(int argc, char* argv[])
{
    again:
    wchar_t path_tmp[MAX_PATH] = { 0 };
    wchar_t abs_path_tmp[MAX_PATH] = { 0 };

    if ((argc - 1) < 1)
    {
        std::wcout << L"[DataBase] Введите путь до базы данных
: ";
        //std::wcin >> path;
        wscanf_s(L"%ls", path_tmp, MAX_PATH);
    }
    else
        mbstowcs_s(nullptr, path_tmp, argv[1], MAX_PATH);

    if (wcslen(path_tmp) > 0)
    {
        if (wcscmp(path_tmp + wcslen(path_tmp) - 4, L".csv")
== 0)
        {
            GetFullPathNameW(path_tmp, MAX_PATH,
abs_path_tmp, nullptr);
            db_f_path = path_tmp;
            db_f_abs_path = abs_path_tmp;
        }
        else
        {
            wprintf(L"[ERROR] Файл базы данных должен иметь
расширение [.csv] !\n");
            goto again;
        }
    }
}

```

```

    }
    else
    {
        wprintf(L"[DataBase] Неправильный путь!\n");
        goto again;
    }
}

void open_catalog(void)
{
    wprintf(L"[DataBase] Открытие базы данных [%ls]\n",
db_f_abs_path.c_str());
    int status = 0;
    try
    {
        status = CAT.open_db(db_f_abs_path);
    }
    catch (const std::exception& e)
    {
        std::wcout << L"[ERROR] " << e.what() << std::endl;
        exit(-1);
    }
    if (status == catalog::DB_LOADED)
        wprintf(L"[DataBase] Загружена существующая база
данных : [%ls]\n", db_f_path.c_str());
    else if (status == catalog::DB_CREATED)
        wprintf(L"[DataBase] Создана новая базы данных :
[%ls]\n", db_f_path.c_str());
}

void save_catalog(void)
{
    try
    {
        CAT.save_db();
    }
    catch (const std::exception& e)
    {
        std::wcout << L"[ERROR] " << e.what() << std::endl;
        exit(-2);
    }

    wprintf(L"[DataBase] База данных (%ls) сохранена\n по
пути (%ls)\n", db_f_path.c_str(), db_f_abs_path.c_str());
}

void run_interface(int argc, char* argv[])
{

```

```

    get_db_path(argc, argv);
    open_catalog();

    int action = 0xDEADBEEF;

    while (action != EXIT_SIGNAL)
    {
        wprintf(L"-----| Меню управления каталогом |-----
        -\n");

        wprintf(L" [%d] Вывести список товаров \n",
PRINT_ALL_GOODS);
        wprintf(L" [%d] Вывести список продавцов \n",
PRINT_ALL_SELLERS);
        wprintf(L" [%d] Вывести список продавцов, конкретного
товара \n", PRINT_ALL_GOOD_SELLERS);
        wprintf(L" [%d] Вывести список товаров, конкретного
продавца \n", PRINT_ALL_SELLER_GOODS);
        wprintf(L"\n");
        wprintf(L" [%d] Добавить - товар \n", ADD_GOOD);
        wprintf(L" [%d] Удалить - товар (конкретную позицию)
\n", DEL_GOOD);
        wprintf(L" [%d] Удалить - продавца (и все его
товары) \n", DEL_SELLER);
        wprintf(L" [%d] Удалить - товар у всех (удалит
определенный товар у всех продавцов) \n", DEL_GOOD_FROM_ALL);
        wprintf(L"\n");
        wprintf(L" [%d] Поиск товара \n", FIND_GOOD);
        wprintf(L" [%d] Поиск продавца \n", FIND_SELLER);
        wprintf(L"\n");
        wprintf(L" [%d] Редактировать - товар \n", EDIT_GOOD);
        wprintf(L"\n");
        wprintf(L" [%d] Отсортировать список \n", SORT_LIST);
        wprintf(L"\n");
        wprintf(L" [%d] Сохранить базу данных \n", SAVE_DB);
        wprintf(L" [%d] Выход \n", EXIT_SIGNAL);

        action = get_action();
        if (action == EXIT_SIGNAL) break;
        process_action(action);
    }

    save_catalog();
}

int get_action(void)
{
    using namespace std;
    int action;
again:
    wprintf(L"\n Выбор действия >> ");
    wcin >> action;

```

```

        if (wcin.fail() || (action < 0) || (action > 14))
        {
            std::wcin.clear();
            std::wcin.ignore(numeric_limits<streamsize>::max(),
L'\n');
            wprintf(L"Ошибка ввода, повторите снова\n");
            action = 0;
            goto again;
        }

        return action;
    }

int process_action(int action)
{
    CLEAR_SCREEN();
    switch (action)
    {
        case PRINT_ALL_GOODS:
            action_print_all_goods();
            break;
        case PRINT_ALL_SELLERS:
            action_print_all_sellers();
            break;
        case PRINT_ALL_GOOD_SELLERS:
            action_print_all_good_sellers();
            break;

        case PRINT_ALL_SELLER_GOODS:
            action_print_all_seller_goods();
            break;

        case ADD_GOOD:
            action_add_good();
            break;

        case FIND_GOOD:
            action_find_good();
            break;

        case FIND_SELLER:
            action_find_seller();
            break;

        case DEL_GOOD:
            action_del_good();
            break;

        case DEL_SELLER:
            action_del_seller();
            break;
    }
}

```

```

        case DEL_GOOD_FROM_ALL:
            action_del_same_good();
            break;

        case EDIT_GOOD:
            action_edit_good();
            break;

        case SORT_LIST:
            action_sort_list();
            break;

        default:
            break;
    }
    PAUSE_INPUT();
    CLEAR_SCREEN();

    return 0;
}

void action_print_all_goods(void)
{
    size_t list_size = CAT.get_size();
    wprintf(L" | Вывод списка всех товаров |\n");

    if (list_size == 0)
    {
        wprintf(L"\n Список товаров пуст..\n\n");
        return;
    }

    List<catalog::PRICE>::Iterator it =
    CAT.get_itemList_begin();

    for (size_t i = 0; i < list_size; i++, it++)
    {
        wprintf(L" -----<Товар #%lu>-----\n", i);
        wprintf(L" Название : %ls \n",
        (*it).product_name.c_str());
        wprintf(L" Продавец : %ls \n",
        (*it).seller_name.c_str());
        wprintf(L" Цена : %4.3f Р\n", (*it).price);
        wprintf(L"\n\n");
    }
}

void action_print_all_sellers(void)
{
    size_t list_size = CAT.get_size();
    wprintf(L" | Вывод списка всех продавцов |\n");

```



```

        if (list_size == 0)
        {
            wprintf(L"\n Список товаров пуст..\n\n");
            return;
        }

        List<catalog::PRICE>::Iterator it =
CAT.get_itemList_begin();
        std::vector<std::wstring> sellers_set;
        std::wstring s_name;
        bool flag = true;

        for (size_t i = 0; i < list_size; i++, it++)
        {
            flag = true;
            s_name = (*it).seller_name;
            for (auto s : sellers_set)
                if (s == s_name)
                {
                    flag = false;
                    break;
                }

            if (flag)
                sellers_set.push_back(s_name);
        }
        size_t i = 1;
        for (auto s : sellers_set)
            wprintf(L" %lu. %ls\n", i++, s.c_str());
        wprintf(L"\n");
    }

void action_print_all_good_sellers(void)
{
    size_t list_size = CAT.get_size();
    if (list_size == 0)
    {
        wprintf(L"\n Список товаров пуст..\n\n");
        return;
    }

    std::wstring p_name;
    wprintf(L" | Вывод списка продавцов, определенного товара
|\n");
    getline(std::wcin, p_name);
    p_name = get_user_string(L" Введите название товара: ");

    size_t index = 1;
    wprintf(L" Продавцы товара (%ls): \n", p_name.c_str());
    List<catalog::PRICE>::Iterator it =
CAT.get_itemList_begin();

```

```

        for (size_t i = 0; i < CAT.get_size(); i++, it++)
            if ((*it).product_name == p_name)
                wprintf(L" %lu. | %-12ls | %0.3f\n", index++,
(*it).seller_name.c_str(), (*it).price);

        if (index == 1)
            wprintf(L" Товары по вашему запросу - не найдены \n");

        wprintf(L"\n");
    }

void action_print_all_seller_goods(void)
{
    size_t list_size = CAT.get_size();
    if (list_size == 0)
    {
        wprintf(L"\n Список товаров пуст..\n\n");
        return;
    }

    std::wstring s_name;
    wprintf(L" | Вывод списка товаров, определенного продавца
| \n");
    getline(std::wcin, s_name);
    s_name = get_user_string(L" Введите имя продавца: ");
    wprintf(L" Товары продавца (%ls): \n", s_name.c_str());
    size_t index = 1;

    List<catalog::PRICE>::Iterator it =
CAT.get_itemList_begin();
    for (size_t i = 0; i < list_size; i++, it++)
        if ((*it).seller_name == s_name)
            wprintf(L" %lu. | %-12ls | %0.3f\n", index++,
(*it).product_name.c_str(), (*it).price);
        if (index == 1)
        {
            wprintf(L" Продавец - не найден \n");
        }

        wprintf(L"\n");
    }

void action_add_good(void)
{
    using namespace std;

    wstring p_name;
    wstring s_name;
    double price = 0;

    getline(wcin, p_name);
    wprintf(L" | Добавление товара | \n");

```

```

        p_name = get_user_string(L" Название товара : ");
        s_name = get_user_string(L" Продавец : ");

price_again:
    wprintf(L" Цена : ");
    wcin >> price;
    if (price <= 0)
        goto price_again;

    if (CAT.check_itemExists(p_name, s_name) == true)
    {
        wprintf(L" > Ошибка : такой товар, у данного продавца
- уже существует! \n");
        return;
    }

    CAT.add_item(p_name, s_name, price);
    wprintf(L"Товар (%ls) - добавлен!\n", p_name.c_str());
}

void action_del_good(void)
{
    std::wstring p_name;
    std::wstring s_name;

    wprintf(L" | Удаление товара |\n");
    getline(std::wcin, p_name);

    p_name = get_user_string(L" Название товара : ");
    s_name = get_user_string(L" Имя продавца : ");

    if (CAT.check_itemExists(p_name, s_name))
    {
        CAT.del_item(p_name, s_name);
        wprintf(L" Товар (%ls) у продавца (%ls) удален!\n",
p_name.c_str(), s_name.c_str());
    }
    else
        wprintf(L" Ошибка : Товар не найден!\n");
    wprintf(L"\n");
}

void action_del_seller(void)
{
    std::wstring s_name;
    wprintf(L" | Удаление продавца |\n");
    getline(std::wcin, s_name);
    s_name = get_user_string(L" Имя продавца : ");

    bool state = CAT.del_seller(s_name);

```

```

        if (!state)
            wprintf(L" Ошибка : Продавец - не найден!\n");
        else
            wprintf(L" Продавец (%ls) - удален \n",
s_name.c_str());

        wprintf(L"\n");
    }

void action_del_same_good(void)
{
    std::wstring p_name;
    wprintf(L" | Удаление определенного товара, у всех
продавцов |\n");
    getline(std::wcin, p_name);
    p_name = get_user_string(L" Название товара : ");

    bool state = CAT.del_product(p_name);

    if (!state)
        wprintf(L" Ошибка : Товар - не найден!\n");
    else
        wprintf(L" Товар (%ls) - удален \n", p_name.c_str());

    wprintf(L"\n");
}

void action_find_good(void)
{
    if (CAT.get_size() == 0)
    {
        wprintf(L"\n Список товаров пуст..\n\n");
        return;
    }

    std::wstring p_name;
    wprintf(L" | Поиск товара |\n");
    getline(std::wcin, p_name);
    p_name = get_user_string(L" Название товара : ");

    List<catalog::PRICE>::Iterator it =
CAT.get_itemList_begin();
    wprintf(L"\n -----[Товары подходящие описанию]-----
----- \n");

    size_t index = 1;

    for (size_t i = 0; i < CAT.get_size(); i++, it++)
    {
        std::wstring guess = (*it).product_name;

```

```

        if (guess.find(p_name) != std::wstring::npos)
        {
            wprintf(L" %lu. | %-12ls | %-12ls | %f\n",
index++, (*it).product_name.c_str(), (*it).seller_name.c_str(),
(*it).price);
        }
    }
    if (index == 1)
        wprintf(L" Товары по вашему запросу - не найдены.
\n");
    wprintf(L"\n");
}

void action_find_seller(void)
{
    if (CAT.get_size() == 0)
    {
        wprintf(L"\n Список товаров пуст..\n\n");
        return;
    }

    std::wstring s_name;
    wprintf(L" | Поиск продавца |\n");
    getline(std::wcin, s_name);
    s_name = get_user_string(L" Имя продавца : ");

    List<catalog::PRICE>::Iterator it =
CAT.get_itemList_begin();
    wprintf(L"\n -----[Продавцы подходящие описанию]----
----- \n");

    size_t index = 1;

    for (size_t i = 0; i < CAT.get_size(); i++, it++)
    {
        std::wstring guess = (*it).seller_name;
        if (guess.find(s_name) != std::wstring::npos)
        {
            wprintf(L" %lu. | %-12ls | %-12ls | %f\n",
index++, (*it).product_name.c_str(), (*it).seller_name.c_str(),
(*it).price);
        }
    }
    if (index == 1)
        wprintf(L" Продавцы по вашему запросу - не найдены.
\n");
    wprintf(L"\n");
}

```

```

void action_edit_good(void)
{
    std::wstring s_name;
    std::wstring p_name;

    wprintf(L" | Изменение товара |\n");
    getline(std::wcin, s_name);
again:
    p_name = get_user_string(L" Название товара : ");
    s_name = get_user_string(L" Имя продавца : ");

    if (CAT.check_itemExists(p_name, s_name) == false)
    {
        wprintf(L" > Товар не найден (наберите exit - для
выхода)\n");
        wprintf(L" > Выйти y/n : ");
        getline(std::wcin, s_name);
        if (s_name == L"y")
            return;
        goto again;
    }

    catalog::PRICE& t = CAT.get_item(p_name, s_name);

    wprintf(L"\n [1] Изменить название \n");
    wprintf(L" [2] Изменить продавца \n");
    wprintf(L" [3] Изменить цена \n");
    int action = get_action();

    switch (action)
    {
    case 1:
        getline(std::wcin, p_name);
        p_name = get_user_string(L" Новое название товара :
");
        if (CAT.check_itemExists(p_name, s_name) == true)
        {
            wprintf(L" > Ошибка : такой товар, у данного
продавца - уже существует! \n");
            break;
        }
        t.product_name = p_name;
        wprintf(L"Название товара - изменено!\n");
        break;
    case 2:
        getline(std::wcin, s_name);
        s_name = get_user_string(L" Новое имя продавца : ");
        if (CAT.check_itemExists(p_name, s_name) == true)
        {
            wprintf(L" > Ошибка : такой товар, у данного
продавца - уже существует! \n");
            break;
        }
    }
}

```

```

        }
        t.seller_name = s_name;
        wprintf(L"Имя продавца товара - изменено!\n");
        break;
    case 3:
        double price;
        wprintf(L" Новая цена : ");
        std::wcin >> price;
        t.price = price;
        wprintf(L"Цена товара измена!\n");
    default:
        break;
    }
}

void action_sort_list(void)
{
    wprintf(L" | Сортировка списка товаров |\n");
    wprintf(L" Как отсортировать список ? : \n");
    wprintf(L" [1] По названию товара\n");
    wprintf(L" [2] По продавцам\n");
    wprintf(L" [3] По цене (от меньшей)\n");
    wprintf(L" [4] По цене (от большей)\n");

    int action = get_action();

    switch (action)
    {
    case 1:
        CAT.sort_itemList(pname_comp);
        break;
    case 2:
        CAT.sort_itemList(sname_comp);
        break;
    case 3:
        CAT.sort_itemList(price_lower_comp);
        break;
    case 4:
        CAT.sort_itemList(price_higher_comp);
        break;
    default:
        break;
    }

    wprintf(L"\nСписок товаров - отсортирован\n");
}

std::wstring get_user_string(std::wstring msg)
{
    using namespace std;

```

```

        wstring s;

again:
    wcout << msg;
    getline(wcin, s);

    if ((iswalnum(s[0]) == 0) || (iswspace(s[0]) != 0) ||
        (s.size() < 3))
    {
        wprintf(L" > Неправильный ввод, повторите снова!\n");
        goto again;
    }

    for (size_t i = 0; i < s.size(); i++)
        s[i] = towlower(s[i]);

    return s;
}

```

## ЛИСТИНГ 2 – CATALOG.H

```

#pragma once
#ifndef CATALOG_H
#define CATALOG_H

#include "list.h"
#include <string>

namespace catalog
{
    /* Список состояний-констант базы данных */
    enum DB_STATES
    {
        DB_CREATED,
        DB_LOADED,
        DB_SAVED,
        DB_FAIL
    };

    /* Структура описывающая товар */
    struct PRICE
    {
        std::wstring product_name;    // Название товара
        std::wstring seller_name;     // Название продавца
        double price;                 // Цена в рублях

        /*~~~ Конструкторы ~~~*/
        PRICE();
        PRICE(std::wstring prod_name, std::wstring sell_name,
            double _price);
    };
}

```



```

        PRICE(PRICE& product_class);
    };

    /* Класс каталога */
    class Catalog
    {
    private:
        std::wstring db_path;          // Относительный путь к
        файлу базы данных
        List<PRICE> items_list;        // Список товаров
    public:
        /*--- Функции для работы с базой данных ---*/

        std::wstring get_db_path(void); //
        Получить относительный путь к базе данных

        int open_db(std::wstring path); // Открыть базу
        данных

        int save_db(void);              // Сохранить базу
        данных

        /*--- Итераторы списка товаров ---*/
        List<PRICE>::Iterator get_itemList_begin(void);
        List<PRICE>::Iterator get_itemList_end(void);

        /*--- Размеры ---*/
        size_t get_size(); // Возвращает размер, кол-во
        товаров в списке

        /*--- Поиск/Получение товара ---*/
        bool check_itemExists(std::wstring p_name,
        std::wstring s_name); // Проверяет, существует ли данный
        товар в списке
        bool check_itemExists(PRICE &prodd);
        // Проверяет, существует ли данный
        товар в списке

        PRICE& get_item(std::wstring p_name, std::wstring
        s_name); // Получить ссылку на конкретный на товар

        /*--- Добавление товара ---*/
        void add_item(std::wstring p_name, std::wstring
        s_name, double price); // Добавление товара (название ; продавец
        ; цена)
        void add_item(PRICE& prod);

        /*--- Удаление товара ---*/
        void del_item(std::wstring p_name, std::wstring
        s_name); // Удаление конкретного товара (название ; продавец)
        bool del_seller(std::wstring s_name); // Удалить
        продавца (удалить все товары, которые продает данный продавец)

```

```

        bool del_product(std::wstring p_name); // Удалить
        товар у всех (удаляет данный товар, у всех продавцов)

        /*--- Сортировка ---*/
        void sort_itemList(bool (*compare_function) (PRICE& a,
        PRICE& b));

        };

};

#endif // CATALOG_H

```

### ЛИСТИНГ 3 – CATALOG.CPP

```

#include <iostream>
#include <fstream>
#include "catalog.h"

catalog::PRICE::PRICE() : product_name(), seller_name(),
price(0)
{ /* empty */ }

catalog::PRICE::PRICE(std::wstring prod_name, std::wstring
sell_name, double _price) :
    product_name(prod_name), seller_name(sell_name),
price(_price)
{ /* empty */ }

catalog::PRICE::PRICE(PRICE & product_class)
{
    product_name = product_class.product_name;
    seller_name = product_class.seller_name;
    price = product_class.price;
}

std::wstring catalog::Catalog::get_db_path(void)
{
    return this->db_path;
}

int catalog::Catalog::open_db(std::wstring path)
{
    int status;
    std::wifstream db_in;
    std::wofstream db_out;

    // Настройка кодировки (для поддержки русского языка)
    db_out.imbue(std::locale("rus_RUS.866"));
}

```

```

db_in.imbue(std::locale("rus_RUS.866"));

/*****
*****
        Алгоритм открытия базы данных:
База данных по данному пути существует ?
        Да: Открываем существующую базу данных по данному
пути
        Нет: Создаем новую по данному пути
*****
*****/

// Попытка открыть базу данных
db_in.open(path);

if (db_in.is_open())
{
    //-- База данных открыта
    // Чтение бд в память

    size_t nLine = 0;
    std::wstring line;

    while (getline(db_in, line))
    {
        nLine++;
        std::wstring p_name;
        std::wstring s_name;
        std::wstring price_str;
        double price_f = 0;

        if ((line.size() == std::string::npos) ||
(line.size() == 0))
        {
            std::cerr << "[ОШИБКА БД] - чтения строки #"
<< nLine << std::endl;
            continue;
        }

        int commaCnt = 0;
        for (int i = 0; i < line.size(); i++)
            if (line[i] == L';')
                commaCnt++;
        if (commaCnt != 2)
        {
            std::cerr << "[ОШИБКА БД] - чтения строки #"
<< nLine << std::endl;
            continue;
        }

        size_t f_comma = line.find(L';');
        size_t l_comma = line.rfind(L';');

```

```

        if ((f_comma == std::string::npos) || (l_comma ==
std::string::npos) || (f_comma == l_comma)) {
            std::cerr << "[ОШИБКА БД] - чтения строки #"
<< nLine << std::endl;
            continue;
        }

        p_name = line.substr(0, f_comma);
        if (p_name.size() <= 2)
        {
            std::cerr << "[ОШИБКА БД] - чтения строки
базы данных (невозможно получить название товара) #" << nLine <<
std::endl;
            continue;
        }

        s_name = line.substr(f_comma + 1, l_comma -
f_comma - 1);
        if (s_name.size() <= 2)
        {
            std::cerr << "[ОШИБКА БД] - чтения строки
базы данных (невозможно получить имя продавца) #" << nLine <<
std::endl;
            continue;
        }

        price_str = line.substr(l_comma + 1);
        price_f = _wtof(price_str.c_str());
        if ((price_str.size() == std::string::npos) ||
(price_f <= 0))
        {
            std::cerr << "[ОШИБКА БД] - чтения строки
базы данных (невозможно получить цену товара) #" << nLine <<
std::endl;
            continue;
        }

        PRICE prod(p_name, s_name, price_f);
        items_list.push_back(prod);
    }
    db_in.close();
    status = DB_LOADED;
}
else
{
    //-- Создание новой базы данных
    db_out.open(path, std::ios::out);
    if (db_out.fail() || !db_out.is_open())
    {

```

```

        throw std::exception("Can't open/create database
file!");
    }
    db_out.close();
    status = DB_CREATED;
}

db_path = path;
return status;
}

int catalog::Catalog::save_db(void)
{
    using namespace std;

    wofstream db_out;
    db_out.imbue(std::locale("rus_RUS.866"));
    db_out.open(db_path, ios::out);
    if (db_out.fail() || !db_out.is_open())
    {
        throw exception("Can't save database!");
    }

    if (items_list.size() > 0)
    {
        List<PRICE>::Iterator it;
        it = items_list.begin();
        for (size_t i = 0; i < items_list.size(); i++, it++)
        {
            db_out.write((*it).product_name.c_str(),
(*it).product_name.size());
            db_out << " ";

            db_out.write((*it).seller_name.c_str(),
(*it).seller_name.size());
            db_out << " ";

            db_out << to_wstring((*it).price) << endl;
        }
    }
    db_out.flush();
    db_out.close();
    return DB_SAVED;
}

List<catalog::PRICE>::Iterator
catalog::Catalog::get_itemList_begin(void)
{
    return this->items_list.begin();
}

```

```

List<catalog::PRICE>::Iterator
catalog::Catalog::get_itemList_end(void)
{
    return this->items_list.end();
}

size_t catalog::Catalog::get_size()
{
    return items_list.size();
}

bool catalog::Catalog::check_itemExists(std::wstring p_name,
std::wstring s_name)
{
    if (items_list.size() == 0)
        return false;

    auto it = items_list.begin();
    for (size_t i = 0; i < items_list.size(); i++, it++)
        if (((*it).product_name == p_name) &&
            ((*it).seller_name == s_name))
            return true;

    return false;
}

bool catalog::Catalog::check_itemExists(PRICE& prodd)
{
    return check_itemExists(prodd.product_name,
prodd.seller_name);
}

catalog::PRICE& catalog::Catalog::get_item(std::wstring p_name,
std::wstring s_name)
{
    List<PRICE>::Iterator it = items_list.begin();
    for (size_t i = 0; i < items_list.size(); i++, it++)
    {
        if (((*it).product_name == p_name) &&
            ((*it).seller_name == s_name))
            return *it;
    }
}

void catalog::Catalog::add_item(std::wstring p_name,
std::wstring s_name, double price)
{
    PRICE prod;
    prod.product_name = p_name;
    prod.seller_name = s_name;
    prod.price = price;
}

```

```

        add_item(prod);
    }

void catalog::Catalog::add_item(PRICE& prod)
{
    items_list.push_back(prod);
}

void catalog::Catalog::del_item(std::wstring p_name,
std::wstring s_name)
{
    List<PRICE>::Iterator it = items_list.begin();

    for (size_t i = 0; i < items_list.size(); i++, it++)
    {
        if (((*it).product_name == p_name) &&
            ((*it).seller_name == s_name))
        {
            items_list.erase(i);
            break;
        }
    }
}

bool catalog::Catalog::del_seller(std::wstring s_name)
{
    size_t index = 0;
    for (size_t i = 0; i < items_list.size(); i++)
    {
        if (items_list[i].seller_name == s_name)
        {
            items_list.erase(i);
            i--;
            index++;
        }
    }
    if (index == 0)
        return false;
    else
        return true;
}

bool catalog::Catalog::del_product(std::wstring p_name)
{
    size_t index = 0;
    for (size_t i = 0; i < items_list.size(); i++)
    {
        if (items_list[i].product_name == p_name)
        {
            items_list.erase(i);

```

```

        i--;
        index++;
    }
}
if (index == 0)
    return false;
else
    return true;
}

void
catalog::Catalog::sort_itemList(bool(*compare_function)(PRICE&
a, PRICE& b))
{
    items_list.sort(compare_function);
}

```



## ПРИЛОЖЕНИЕ Б

### ЛИСТИНГ 4 – LIST.H

```
/******
 *                               Реализация                               *
 *                               Линейного двусвязного списка             *
 *                               в виде шаблонного контейнера             *
 *****/

#pragma once
#ifndef LIST_H
#define LIST_H

#ifndef NDEBUG
    #include <iostream>
#endif
#include <stdexcept>

/*===== Класс =====*/

template <typename T>
class List
{
private:
    // Узел списка
    struct LNode
    {
        T Item;
        LNode* Next;
        LNode* Prev;
    };

    size_t listSize; // Размер списка
    LNode* listHead; // Указатель на голову списка
    LNode* listTail; // Указатель на хвост списка

    inline LNode* safeCreateNode(T item); // Функция создания
    узла

    /*--- Скрытые Функции сортировки слиянием ---*/
    LNode* mergeSort(LNode* head, bool (*compare_function) (T&
a, T& b));
    void swap(T& a, T& b);
    LNode* split(LNode* head);
    LNode* merge(LNode* first, LNode* second, bool
(*compare_function) (T& a, T& b));

public:
    /*--- Класс итератор ---*/
    class Iterator
```

```

{
private:
    LNode* node_ptr;
public:

    Iterator(LNode* ptr);
    Iterator();

    void operator++ (int);
    T& operator+ (int n);
    void operator-- (int);
    bool operator!= (const Iterator& it);
    bool operator== (const Iterator& it);
    T& operator* ();
};

/*--- Функции члены ---*/

List(); // Конструктор по умолчанию
~List(); // Деструктор по умолчанию

/*--- Размер ---*/

bool isEmpty(void); // Проверка, пуст ли список
size_t size(void); // Возвращает размер списка (кол-во
элементов)

/*--- Операции ---*/

void push_back(T item); // Добавить элемент в конец
void push_front(T item); // Добавить элемент в начало

T pop_back(void); // Удалить и получить элемент с конца
T pop_front(void); // Удалить и получить элемент с начала

void insert(T item, int index); // Вставка элемента по
индексу
void erase(int index); // Удалить элемент по
индексу

T& index(int index); // Получить опередленный
элемент по индексу
T& operator[](int _index); // Получить опередленный
элемент по индексу

void clear(void); // Полная очистка списка
void copy(List<T>& another); // Копирование
void sort(bool (*compare_function) (T& a, T& b)); //
Сортировка слиянием

```

```

/*--- Итераторы ---*/

Iterator begin() // Получить итератор на начало списка
{ return Iterator(listHead); }
Iterator end()   // Получить итератор на конец списка
{ return Iterator(listTail); }

/*--- Отладочные функции ---*/
#ifdef NDEBBUG
    void displayNodes();           // Выводит 'список' узлов, с
    // полной печатью значений
    void printList(void);         // Печатает список от головы-
    // >хвосту
    void printListReverse(void); // Печатает список от хвоста-
    // >голове
#endif

}; // END OF CLASS DECLARATION

/*===== ОПРЕДЕЛЕНИЕ методов класса =====*/

template<typename T>
List<T>::List()
{
    listSize = 0;
    listHead = nullptr;
    listTail = nullptr;
}

template<typename T>
List<T>::~~List()
{
    if (!isEmpty())
    {
        LNode* prev;
        size_t i = 0;
        for (LNode* ptr = listHead; ptr != nullptr; ++i)
        {
            prev = ptr;
            ptr = ptr->Next;
            delete prev;
        }
    }
}

template<typename T>
inline
struct List<T>::LNode*
List<T>::safeCreateNode(T item)
{

```

```

    LNode* node;
    try
    {
        node = new LNode;
    }
    catch (const std::bad_alloc& e)
    {
        throw e;
    }
    node->Item = item;
    node->Next = nullptr;
    node->Prev = nullptr;

    return node;
}

template<typename T>
bool
List<T>::isEmpty(void)
{
    return (listSize == 0) ? true : false;
}

template<typename T>
size_t
List<T>::size(void)
{
    return listSize;
}

template<typename T>
void
List<T>::push_back(T item)
{
    LNode* newNode;
    newNode = safeCreateNode(item);

    if (isEmpty())
    {
        listHead = newNode;
        listTail = newNode;
    }
    else
    {
        listTail->Next = newNode;
        newNode->Prev = listTail;
        listTail = newNode;
    }
    listSize++;
}

```

```

template<typename T>
void
List<T>::push_front(T item)
{
    LNode* newNode;
    newNode = safeCreateNode(item);

    if (isEmpty())
    {
        listHead = newNode;
        listTail = newNode;
    }
    else
    {
        listHead->Prev = newNode;
        newNode->Next = listHead;
        listHead = newNode;
    }
    listSize++;
}

template<typename T>
T
List<T>::pop_back(void)
{
    LNode* node;
    T returnItem;

    if (isEmpty())
    {
        throw std::out_of_range("List is empty");
    }
    else
    {
        node = listTail;
        returnItem = listTail->Item;
        if (size() == 1)
        {
            listHead = nullptr;
            listTail = nullptr;
            delete node;
        }
        else
        {
            listTail = listTail->Prev;
            listTail->Next = nullptr;
            delete node;
        }
    }
    listSize--;
    return returnItem;
}

```

```

template<typename T>
T
List<T>::pop_front(void)
{
    LNode* node;
    T returnItem;

    if (isEmpty())
    {
        throw std::out_of_range("List is empty");
    }
    else
    {
        node = listHead;
        returnItem = listHead->Item;
        if (size() == 1)
        {
            listHead = nullptr;
            listTail = nullptr;
            delete node;
        }
        else
        {
            listHead = listHead->Next;
            listHead->Prev = nullptr;
            delete node;
        }
    }
    listSize--;
    return returnItem;
}

template<typename T>
void
List<T>::insert(T item, int index)
{
    LNode* node;
    LNode* prevNode;
    LNode* ptr;

    // check index
    if ((index < 0) || (index > size()))
        throw std::out_of_range("index is out of range!");

    if (index == 0) push_front(item);
    else if (index == size()) push_back(item);
    else
    {
        node = safeCreateNode(item);
        ptr = listHead;
        while (index-- > 0) ptr = ptr->Next;
    }
}

```

```

        prevNode = ptr->Prev;
        prevNode->Next = node;

        node->Prev = prevNode;
        node->Next = ptr;

        ptr->Prev = node;
        listSize++;
    }
}

template<typename T>
void
List<T>::erase(int index)
{
    LNode* ptr;
    LNode* nextNode;
    LNode* prevNode;

    if (isEmpty())
        throw std::out_of_range("List is empty");
    if ((index < 0) || (index > (size() - 1)))
        throw std::out_of_range("index is out of range!");

    if (index == 0) pop_front();
    else if (index == (size() - 1)) pop_back();
    else
    {
        ptr = listHead;
        while (index-- > 0) ptr = ptr->Next;
        prevNode = ptr->Prev;
        nextNode = ptr->Next;

        prevNode->Next = nextNode;
        nextNode->Prev = prevNode;

        delete ptr;
        listSize--;
    }
}

template<typename T>
T&
List<T>::index(int index)
{
    LNode* ptr;
    if (isEmpty())
        throw std::out_of_range("List is empty");
    if ((index < 0) || (index > (size() - 1)))
        throw std::out_of_range("index is out of range!");

```

```

        if (index == 0) return listHead->Item;
        else if (index == (size() - 1)) return listTail->Item;
        else
        {
            ptr = listHead;
            while (index-- > 0) ptr = ptr->Next;
            return ptr->Item;
        }
    }

template<typename T>
T&
List<T>::operator[](int _index)
{
    return index(_index);
}

#ifdef NDEBUG

template<typename T>
void
List<T>::displayNodes()
{
    size_t i = 1;
    LNode* ptr = listHead;
    while (ptr != nullptr)
    {
        printf(" _____\n");
        printf("| NODE #%llu | \n", i);
        printf("| ADDRESS   : %p | \n", ptr);
        printf("|-----| \n");
        printf("| NEXT NODE: %p | \n", ptr->Next);
        printf("|-----| \n");
        printf("| PREV NODE: %p | \n", ptr->Prev);
        printf("|-----| \n");

        ptr = ptr->Next;
        i++;
    }
}

template<typename T>
void
List<T>::printList(void)
{
    size_t i = 0;
    LNode* ptr = listHead;
    std::cout << "head->tail [ ";
    while (ptr != nullptr)
    {
        std::cout << ptr->Item << " ";
    }
}

```



```

        ptr = ptr->Next;
        i++;
    }
    std::cout << "]" i = " << i << std::endl;
}

template<typename T>
void
List<T>::printListReverse(void)
{
    size_t i = 0;
    LNode* ptr = listTail;
    std::cout << "tail->head [ ";
    while (ptr != nullptr)
    {
        std::cout << ptr->Item << " ";
        ptr = ptr->Prev;
        i++;
    }
    std::cout << "]" i = " << i << std::endl;
}

#endif

template<typename T>
inline void List<T>::clear(void)
{
    this->~List();
    listSize = 0;
    listHead = nullptr;
    listTail = nullptr;
}

template<typename T>
inline void List<T>::copy(List<T>& another)
{
    for (size_t i = 0; i < another.listSize; i++)
    {
        this->push_back(another.index(i));
    }
}

// Итеративное определение сортировки слиянием
template<typename T>
inline void List<T>::sort(bool(*compare_function)(T& a, T& b))
{
    if (listSize <= 1)
        return;

    listHead = mergeSort(listHead, compare_function);
}

```

```

}

template<typename T>
inline List<T>::Iterator::Iterator(LNode* ptr)
{
    if (ptr != nullptr)
        node_ptr = ptr;
    else
        throw std::out_of_range("Trying dereference nullptr");
}

template<typename T>
inline List<T>::Iterator::Iterator()
{
    node_ptr = nullptr;
}

template<typename T>
inline void List<T>::Iterator::operator++(int)
{
    if (node_ptr != nullptr)
        node_ptr = node_ptr->Next;
    else
        throw std::out_of_range("Out of range");
}

template<typename T>
inline T& List<T>::Iterator::operator+(int n)
{
    while (n > 0)
    {
        if (node_ptr != nullptr)
            node_ptr = node_ptr->Next;
        else
            throw std::out_of_range("Out of range");
        n--;
    }
}

template<typename T>
inline void List<T>::Iterator::operator--(int)
{
    if (node_ptr != nullptr)
        node_ptr = node_ptr->Prev;
    else
        throw std::out_of_range("Out of range");
}

template<typename T>
inline bool List<T>::Iterator::operator!=(const Iterator& it)

```

```

{
    return (node_ptr->Item) != (it.node_ptr->Item);
}

template<typename T>
inline bool List<T>::Iterator::operator==(const Iterator& it)
{
    return (node_ptr->Item) == (it.node_ptr->Item);
}

template<typename T>
inline T& List<T>::Iterator::operator*()
{
    if (node_ptr != nullptr)
        return node_ptr->Item;
    else
        throw std::out_of_range("Element doesnt exist");
}

template<typename T>
inline typename List<T>::LNode*
List<T>::mergeSort(List<T>::LNode* head, bool
(*compare_function) (T& a, T& b))
{
    if (!head || !head->Next)
        return head;

    LNode* second = split(head);

    head = mergeSort(head, compare_function);
    second = mergeSort(second, compare_function);

    return merge(head, second, compare_function);
}

template<typename T>
inline typename List<T>::LNode* List<T>::split(List<T>::LNode*
head)
{
    LNode* fast = head, * slow = head;
    while (fast->Next && fast->Next->Next)
    {
        fast = fast->Next->Next;
        slow = slow->Next;
    }
    LNode* temp = slow->Next;
    slow->Next = NULL;
    return temp;
}

template<typename T>

```

```

inline typename List<T>::LNode *List<T>::merge(List<T>::LNode *
first, List<T>::LNode * second, bool (*compare_function) (T & a,
T & b))
{
    if (!first)
        return second;

    if (!second)
        return first;

    if (compare_function(first->Item, second->Item))
    {
        first->Next = merge(first->Next, second,
compare_function);
        first->Next->Prev = first;
        first->Prev = NULL;
        return first;
    }
    else
    {
        second->Next = merge(first, second->Next,
compare_function);
        second->Next->Prev = second;
        second->Prev = NULL;
        return second;
    }
}

template<typename T>
inline void List<T>::swap(T& a, T& b)
{
    T temp = a;
    a = b;
    b = temp;
}

#endif // LIST_H

```

## ПРИЛОЖЕНИЕ В

### ЛИСТИНГ 5 – coursework\_tests.cpp

```
#include "CppUnitTest.h"
#include "../coursework_final/list.h"
#include "../coursework_final/catalog.h"

#include <Windows.h>
#include <vector>
#include <crtdbg.h>
#include <cstdlib>
#include <stdexcept>
#include <string>

#define __CRTDBG_MAP_ALLOC
#define DEBUG_NEW new(_NORMAL_BLOCK, __FILE__, __LINE__)
#define new DEBUG_NEW

using namespace Microsoft::VisualStudio::CppUnitTestFixture;

struct dummy_struct
{
    int a;
    int b;
    std::string str;
};

/*-----
    Проверка на утечки памяти.
    1. Делается снимок кучи (heap), в начале вызова функции

    2. Далее в локальном блоке, происходит вызов тестируемых
       функций, классов и т.д.

    3. Затем делается снимок текущей кучи
       и сравнивается, с прошлой.

    Если снимки не совпадают, то в результате
    работы кода, в локальном блоке, произошла утечка памяти.

    Это повлияет на успешность каждого теста.
    -----*/

#define MEMORY_LEAK_CHECK_BEGIN() \
    _CrtMemState sOld; \
    _CrtMemState sNew; \
    _CrtMemState sDiff; \
    _CrtMemCheckpoint(&sOld); \
```

```

#define MEMORY_LEAK_CHECK_END() \
    _CrtMemCheckpoint(&sNew); \
    int status = _CrtMemDifference(&sDiff, &sOld, &sNew); \
    if (status) \
    { \
        Assert::Fail(L"Memory leak - detected!"); \
        OutputDebugString(L"-----_CrtMemDumpStatistics -\n"); \
        _CrtMemDumpStatistics(&sDiff); \
        OutputDebugString(L"-----\n"); \
        _CrtMemDumpAllObjectsSince -----"); \
        _CrtMemDumpAllObjectsSince(&sOld); \
        OutputDebugString(L"-----_CrtDumpMemoryLeaks ---\n"); \
        _CrtDumpMemoryLeaks(); \
    }

__forceinline void check_memory_leaks(void)
{

    int status = _CrtDumpMemoryLeaks();
    if (status != 0)
    {
        Logger::WriteMessage("Memory leak - detected !\n");
    }
    Assert::IsTrue(status == 0, L"MEMORY LEAK test");
}

bool sort_comp(int& a, int& b)
{
    return a < b;
}

namespace courseworktests
{
    /* [Тесты для (класса) List] */
    TEST_CLASS(List_Tests)
    {
    public:
        TEST_METHOD(DefaultConstructor)
        {
            /*-----
            Тестирование конструктора по умолчанию
            для различных типов
            -----*/

            MEMORY_LEAK_CHECK_BEGIN();
            {
                List<int> l_int;

```

```

        List<std::string> l_string;
        List<std::vector<int>> l_vector;
        List<dummy_struct> l_struct;

        Assert::IsTrue(l_int.size() == 0);
        Assert::IsTrue(l_int.isEmpty() == true);

        Assert::IsTrue(l_string.size() == 0);
        Assert::IsTrue(l_string.isEmpty() == true);

        Assert::IsTrue(l_vector.size() == 0);
        Assert::IsTrue(l_vector.isEmpty() == true);

        Assert::IsTrue(l_struct.size() == 0);
        Assert::IsTrue(l_struct.isEmpty() == true);
    }
    MEMORY_LEAK_CHECK_END();
}

TEST_METHOD(PushTests)
{
    /*-----
       Тестирование Операций
       1. push_back()
       2. push_front()
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        List<int> l_int;
        List<std::string> l_string;

        //-----[Int] object tests-----
        l_int.push_back(1);
        l_int.push_back(2);
        l_int.push_back(3);
        l_int.push_front(4);
        l_int.push_front(5);
        l_int.push_front(6);
        l_int.push_back(7);
        l_int.push_front(8);
        Assert::IsTrue(l_int.size() == 8);
        Assert::IsTrue(l_int.isEmpty() == false);
        int comp_arr[8] = { 8,6,5,4,1,2,3,7 };

        for (int i = 0; i < 8; ++i)
            Assert::AreEqual(l_int[i],
comp_arr[i]);

        //-----[string] object tests-----
        l_string.push_back("Hello");
        l_string.push_back("World");
    }
}

```

```

        l_string.push_front("Default");
        l_string.push_back("Message");
        Assert::IsTrue(l_string.size() == 4);
        Assert::IsTrue(l_string.isEmpty() == false);
        Assert::AreEqual(l_string[0],
std::string("Default"));
        Assert::AreEqual(l_string[1],
std::string("Hello"));
        Assert::AreEqual(l_string[2],
std::string("World"));
        Assert::AreEqual(l_string[3],
std::string("Message"));
    }
    MEMORY_LEAK_CHECK_END();

}
TEST_METHOD(PopTests)
{
    /*-----
    Тестирование Операций
    1. pop_back()
    2. pop_front()
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        List<int> l_int;
        List<std::string> l_string;

        // Int object
        int popped;

        l_int.push_back(1);
        l_int.push_back(2);
        l_int.push_back(3);
        l_int.push_back(4);

        popped = l_int.pop_back();
        Assert::AreEqual(popped, 4);
        Assert::IsTrue(l_int.size() == 3);

        popped = l_int.pop_front();
        Assert::AreEqual(popped, 1);
        Assert::IsTrue(l_int.size() == 2);

        l_int.pop_back();
        l_int.pop_back();
        Assert::IsTrue(l_int.isEmpty() == true);

        // String object
        std::string popped_s;
        l_string.push_back("1");

```



```

        l_string.push_back("2");
        l_string.push_back("3");
        l_string.push_back("4");

        popped_s = l_string.pop_back();
        Assert::AreEqual(popped_s, std::string("4"));
        Assert::IsTrue(l_string.size() == 3);

        popped_s = l_string.pop_front();
        Assert::AreEqual(popped_s, std::string("1"));
        Assert::IsTrue(l_string.size() == 2);

        l_string.pop_back();
        l_string.pop_back();

        Assert::IsTrue(l_string.isEmpty() == true);

        std::string test_str;
        test_str = "Hello world!";
        l_string.push_back(test_str);

        Assert::IsTrue(l_string.isEmpty() == false);
        Assert::IsTrue(l_string.size() == 1);

        test_str.clear();
        Assert::IsTrue(test_str.size() == 0);
        Assert::AreNotEqual(l_string.pop_back(),
test_str);

        // Out of range, list is empty at this
moment
        try
        {
            popped = l_int.pop_back();
            Assert::Fail(L"Can't catch
exception!");
        }
        catch (const std::exception& e)
        {
            Assert::AreEqual(e.what(), "List is
empty");
        }

        // same for pop-front()
        try
        {
            popped = l_int.pop_front();
            Assert::Fail(L"Can't catch
exception!");
        }
        catch (const std::exception& e)
        {

```

```

        Assert::AreEqual(e.what(), "List is
empty");
    }

}

MEMORY_LEAK_CHECK_END();

}

TEST_METHOD(IteratorsTests)
{
    /*-----
    Тестирование Итераторов списка
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        List<int> list;

        list.push_back(1);
        list.push_back(2);
        list.push_back(3);

        List<int>::Iterator it;
        List<int>::Iterator it2;
        it = list.begin();
        Assert::AreEqual(*it, 1);

        *it = 228;
        Assert::AreEqual(*it, 228);
        it++;
        Assert::AreEqual(*it, 2);
        it++;
        Assert::AreEqual(*it, 3);

        it--;
        Assert::AreEqual(*it, 2);
        it--;
        Assert::AreEqual(*it, 228);

        //
        it = list.begin();
        it2 = list.begin();

        Assert::IsTrue(it == it2);
        it++;
        Assert::IsTrue(it != it2);

    }
    {
        List<int> list;
        for (int i = 0; i < 10; i++)

```

```

        list.push_back(i);

        int i = 0;
        List<int>::Iterator it = list.begin();
        for (; i < list.size(); i++, it++)
            Assert::AreEqual(*it, list[i]);

        i = list.size()-1;
        it = list.end();
        for (; i >= 0; i--, it--)
            Assert::AreEqual(*it, list[i]);
    }
    {
        List<dummy_struct> list;

        list.push_back({ 1,1,std::string("string_1")
});
        list.push_back({ 2,2,std::string("string_2")
});
        list.push_back({ 3,3,std::string("string_3")
});

        List<dummy_struct>::Iterator it =
list.begin();
        for (int i = 0; i < list.size(); i++, it++)
        {
            Assert::AreEqual((*it).a, list[i].a);
            Assert::AreEqual((*it).b, list[i].b);
            Assert::AreEqual((*it).str,
list[i].str);
        }

        it = list.begin();
        (*it).str = "Hacked_1";
        it = list.end();
        (*it).str = "Hacked_3";

        Assert::AreEqual(list[0].str,
std::string("Hacked_1"));
        Assert::AreEqual(list[2].str,
std::string("Hacked_3"));
    }
    {
        List<int> list;
        List<int>::Iterator it;

        list.push_back(1);
        list.push_back(2);
        list.push_back(3);
        list.push_back(4);

```

```

    }

    MEMORY_LEAK_CHECK_END();

}

TEST_METHOD(InsertDeleteTests)
{
    /*-----
       Тестирование функций
       вставки и удаления
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        List<int> list;
        list.push_back(1);
        list.push_back(2);
        list.push_back(3);
        list.push_back(4);

        list.insert(999, 1);
        Assert::AreEqual(list[1], 999);

        list.insert(666, 3);
        Assert::AreEqual(list[3], 666);

        list.erase(3);
        list.erase(1);

        Assert::AreEqual(list[0], 1);
        Assert::AreEqual(list[1], 2);
        Assert::AreEqual(list[2], 3);
        Assert::AreEqual(list[3], 4);

        list.erase(0);
        list.erase(0);
        list.erase(0);
        list.erase(0);

        try
        {
            list.erase(0);
        }
        catch (const std::exception& e)
        {
            Assert::AreEqual(e.what(), "List is
empty");
        }

        list.push_back(228);
    }
}

```

```

        try
        {
            list.erase(222);
        }
        catch (const std::exception& e)
        {
            Assert::AreEqual(e.what(), "index is
out of range!");
        }
    }
    MEMORY_LEAK_CHECK_END();
}

TEST_METHOD(SortTests)
{
    /*-----
    Тестирование Сортировки списка
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        List<int> list;
        list.push_back(5);
        list.push_back(3);
        list.push_back(1);
        list.push_back(2);
        list.push_back(4);

        list.sort(sort_comp);

        List<int>::Iterator it = list.begin();
        for (int i = 0; i < list.size(); i++, it++)
        {
            int a = *it;
            std::string tmp;
            tmp = std::to_string(a);
            tmp += " ";
            Logger::WriteMessage(tmp.c_str());
        }

    }
    MEMORY_LEAK_CHECK_END();
}

};
/* [Тесты для (класса) List] */
TEST_CLASS(Catalog_Tests)
{
public:
    TEST_METHOD(PRICE_Structure_Tests)

```

```

{
    /*-----
    Тестирование конструкторов
    структуры товара
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        catalog::PRICE prod;
        Assert::AreEqual(prod.price, 0.0);
        Assert::IsTrue(prod.product_name.size() ==
0);
        Assert::IsTrue(prod.seller_name.size() ==
0);

        catalog::PRICE prod2(L"стол", L"оби",
15000.0);
        Assert::AreEqual(prod2.product_name.c_str(),
L"стол");
        Assert::AreEqual(prod2.seller_name.c_str(),
L"оби");
        Assert::AreEqual(prod2.price, 15000.0);

        catalog::PRICE prod3(prod2);
        Assert::AreEqual(prod3.product_name.c_str(),
L"стол");
        Assert::AreEqual(prod3.seller_name.c_str(),
L"оби");
        Assert::AreEqual(prod3.price, 15000.0);
    }
    MEMORY_LEAK_CHECK_END();
}

TEST_METHOD(database_tests_1)
{
    /*-----
    Тестирование функций для работы
    с база данных класата Catalog
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        catalog::Catalog cat;
        // Открытие базы данных
        // -----
        int status;
        const wchar_t* new_db_path = L"test_db.csv";
        const wchar_t* exist_db_path = L"new.csv";

        DeleteFileW(new_db_path);
        status = cat.open_db(new_db_path);
    }
}

```

```

        Assert::IsTrue(status ==
catalog::DB_CREATED);

        status = cat.open_db(exist_db_path);
        Assert::IsTrue(status ==
catalog::DB_LOADED);

        // Сохранение базы данных
        // -----
        status = cat.save_db();
        Assert::IsTrue(status == catalog::DB_SAVED);

    }
    MEMORY_LEAK_CHECK_END();
}

TEST_METHOD(itemlist_iterator_tests)
{
    /*-----
        Тестирование функций для работы
        с итераторами списка товаров
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        catalog::Catalog cat;
        cat.open_db(L"new.csv");

        List<catalog::PRICE>::Iterator iter;
        catalog::PRICE tmp;

        iter = cat.get_itemList_begin();
        tmp = *iter;
        Assert::AreEqual(tmp.product_name.c_str(),
L"стул");
        Assert::AreEqual(tmp.seller_name.c_str(),
L"оби");
        Assert::AreEqual(tmp.price, 1337.0);

        iter++;
        tmp = *iter;

        Assert::AreEqual(tmp.product_name.c_str(),
L"черный стул винтажный");
        Assert::AreEqual(tmp.seller_name.c_str(),
L"оби");
        Assert::AreEqual(tmp.price, 999.0);

    }
    MEMORY_LEAK_CHECK_END();
}

```

```

TEST_METHOD(Item_exists)
{
    /*-----
                               Item exists
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        catalog::Catalog cat;
        cat.open_db(L"new.csv");

        Assert::IsFalse(cat.check_itemExists(L"бочка", L"Дикси"));
        Assert::IsTrue(cat.check_itemExists(L"стул",
L"оби"));
    }
    MEMORY_LEAK_CHECK_END();
}

TEST_METHOD(add_item)
{
    /*-----
                               Add item
    -----*/

    MEMORY_LEAK_CHECK_BEGIN();
    {
        catalog::Catalog cat;
        cat.open_db(L"new.csv");

        cat.add_item(L"aaa", L"seller", 222);
        cat.add_item(L"bbb", L"seller", 333);

        Assert::IsTrue(cat.get_size() == 4);
        auto it = cat.get_itemList_end();
        Assert::AreEqual((*it).product_name.c_str(),
L"bbb");
    }
    MEMORY_LEAK_CHECK_END();
}

};

}

```