

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Конструирование программ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
на тему

ПРОГРАММА ДЛЯ ОБСЛУЖИВАНИЯ КЛИЕНТОВ МАГАЗИНА
БГУИР КП 1-40 02 01 009 ПЗ

Студент: группы 910903,
Логвин В.В.

Руководитель: доцент каф. ЭВМ,
Насуро Е.В.

Минск 2020
Учреждение образования

«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой

(подпись)

2020г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Логвину Владиславу Валерьевичу

1. Тема проекта Написать программу для обслуживания клиентов магазина.
2. Срок сдачи студентом законченного проекта 20 декабря 2020 г.
3. Исходные данные к проекту Программа должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню. Работа с информацией должна производиться в окнах. Информация должна храниться в различных файлах, при этом каждая группа товаров должна иметь отдельный файл. Каждый товар имеет характеристики (группа, тип, индивидуальные особенности, страна происхождения и т. д.) и штрих-код. При обслуживании клиента необходимо подготовить электронный чек, в котором должно быть указано название товара, его цена, количество, общая сумма покупки, дата и время покупки. Чеки должны сохраняться в файлы. Все покупки, совершенные клиентами, должны записываться в файл. Разработать и использовать в программе классы контейнеров и итераторов. Общее для всех вариантов задание: реализовать авторизацию

для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе).

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке) Титульный лист, Задание по курсовой работе (заполненное и подписанное студентом и преподавателем), Содержание, 1. Требования к программе, 2. Конструирование программы (1.1 Разработка модульной структуры программы, 1.2 Выбор способа организации данных, 1.3 Разработка перечня пользовательских функций программы), 3. Разработка алгоритмов работы программы (3.1 Алгоритм функции main, 3.2 – 3.5 Алгоритм функций – в соответствии с перечнем функций), 4. Описание работы программы (4.1 Авторизация, 4.2 Модуль администратора, 4.3 Модуль пользователя, 4.4 Исключительные ситуации. Приложение (обязательное): листинг кода с комментариями.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) 1. Модульная структура программы. 2. Диаграмма классов, 3. Алгоритмы основных функций программы

6. Консультант по проекту Насуро Е.В.

7. Дата выдачи задания 10 сентября 2020 г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

разделы 1,2 к 1 октября 2020 г. – 20 %;

раздел 3 к 1 ноября 2020 г. – 50 %;

раздел 4 к 1 декабря 2020 г. – 80 %;

оформление пояснительной записки к 10 декабря 2020 г. – 100 %

Защита курсового проекта с 14 декабря 2020 г. по 21 декабря 2020 г.

РУКОВОДИТЕЛЬ _____ Е.В. Насуро

Задание принял к исполнению _____ В.В. Логвин

СОДЕРЖАНИЕ

Введение	5
1 Требования к программе	7
2 Обзор методов и алгоритмов поставленной задачи.....	12
3 Обоснование выбранных методов и алгоритмов	14
4 Конструирование программы	16
5 Разработка алгоритмов работы программы	19
5.1 Алгоритм функции WinMain	19
5.2 Алгоритм функции sortProductsByName	19
5.3 Алгоритм функции findProduct	19
5.4 Алгоритм функции buyProduct	19
5.5 Алгоритм функции editProductPrice	19
5.6 Алгоритм функции sortUsersByName	20
5.7 Алгоритм функции findUser.....	20
5.8 Алгоритм функции editUserName	21
5.9 Алгоритм функции getSummaryPrice	21
5.10 Алгоритм функции addAccount	22
6 Описание работы программы	23
6.1 Авторизация	23
6.2 Модуль администратора.....	24
6.3 Модуль пользователя.....	26
6.4 Исключительные ситуации	29
Заключение	30
Литература	31
Приложение А – Листинг программы.....	
Приложение Б – Диаграмма классов.....	
Приложение В – Модульная структура программы	
Приложение Г – Алгоритмы функций	
Приложение Д – Описание работы программы	

ВВЕДЕНИЕ

Скорость развития технологий в наш век рекордно большая для всей истории человечества. Мир движется к удобствам, экономии ресурсов, автоматизации и оптимизации всех процессов. В контексте информационных технологий во многом это удастся благодаря разработке эффективных сервисов, которые направлены на облегчение работы людей, занимающихся рутинной и кропотливой деятельностью, которая, благодаря сервисам, теперь может выполняться практически без человеческого участия или контроля.

Подобные сервисы способствуют более эффективному распределению кадров: компаниям требуется меньше работников, чтобы обеспечивать деятельность предприятия, и, соответственно, появляется большая денежная выгода, благодаря которой можно также обеспечить более высокие зарплаты сотрудникам, повышая таким образом их мотивацию и продуктивность. Кроме того, если функции, выполняемые сервисами, занимали определенный процент обязанностей работника, то вместе с автоматизацией этих функций данный сотрудник получает возможность перенаправить свое время и усилия на более важные аспекты своей работы.

Внедрение программы для обслуживания клиентов магазина является технологией абсолютной необходимости. Обусловлено это многими факторами. Во-первых, подобный сервис обеспечивает разумное распределение ресурсов и кадров магазина, при этом повышая эффективность и оперативность работы. Во-вторых, информация часто нуждается в своевременной коррекции, реализовать которую без помощи программного сервиса в разы дольше и сложнее. В-третьих, экономия времени в современном мире является приоритетной, особенно когда речь идет о таких процессах, как покупка и просмотр информации о различной продукции. Данная программная система позволит централизованно и структурированно хранить и обрабатывать данные о продуктах, а также удобно осуществлять

поиск и сортировку даже в самых объемных базах данных. Внедрение данного приложения приведет к автоматизации регистрации клиентов в системе, даст возможность клиентам самостоятельно осуществлять поиск по базе со всей необходимой информацией и покупать нужную пользователю продукцию всего в пару кликов.

C++ позволяет писать программы как в парадигме ООП, так и в процедурной. Этот язык является одним из самых производительных: приложения, написанные на нем, будут работать быстро, что является очень важным фактором в контексте разрабатываемого сервиса. По C++ есть очень много качественной и проверенной литературы, что облегчает его освоение и уверенное использование. Большое количество библиотек обеспечивает дополнительные удобства и производительность при написании кода. Имение опыта написания программ на данном языке стало окончательным аргументом в пользу выбора C++ для разработки курсового проекта.

Microsoft Visual Studio является одной из самых хороших и популярных сред разработки для операционной системы Windows. Данная среда позволяет писать красивый и правильный код, указывая на ошибки, совершаемые программистом, проводит анализ потока данных для поиска различных проблем в структуре программы. Более того, существует бесплатная лицензионная версия данной среды разработки, что является очень ценным в студенческой среде. Visual Studio обладает привлекательным и комфортным интерфейсом, поддерживает темную тему, что очень практично при ночной работе с кодом и позволяет уменьшать нагрузку на глаза.

1 ТРЕБОВАНИЕ К ПРОГРАММЕ

Исходные данные:

Заданием курсового проектирования является разработка программы для обслуживания клиентов магазина.

Сведения о продуктах включают: название, цену, группу, тип, индивидуальные особенности, страну происхождения и штрих-код.

При обслуживании клиента необходимо подготовить электронный чек, в котором должно быть указано название товара, его цена, количество, общая сумма покупки, дата и время покупки.

Необходимо также реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя.

Требования:

1. Тема «Программа для обслуживания клиентов магазина».
2. Язык программирования C++.
3. Среда разработки Microsoft Visual Studio.
4. Вид приложения – оконное.
5. Парадигма программирования – объектно-ориентированная.
6. Способ организации данных – классы (class).
7. Способ хранения данных – файлы.
8. Каждая логически завершенная подзадача программы должна быть реализована в виде отдельной функции.
9. Построение программного кода должно соответствовать соглашению о коде «C++ Code Convention».
10. К защите курсовой работы представляются: оконное приложение и пояснительная записка.

Функциональные требования к курсовой работе:

При первом запуске программы создается аккаунт создателя, чтобы взаимодействовать с дальнейшим функционалом. В последующих случаях

первым этапом работы программы будет являться авторизация, которая предоставит пользователю определённые права доступа.

При запуске программы происходит считывание из файла с аккаунтами следующей информации:

- Идентификатор аккаунта id,
- Login,
- HashPassword,
- Salt,
- Name,
- Surname,
- Age,
- Role (0 – пользователь, 1 – администратор, 2 – создатель).

Пароль маскируется в целях безопасности с помощью специального символа «*».

Регистрация аккаунтов выполняется:

1. Создателем, с возможностью выбрать роль новой учётной записи (1 – администратор, 0 – пользователь).
2. Администратором.
3. Пользователем.

После входа становится доступно меню баз данных. В режиме администратора доступны взаимодействия с

1. Базой данных продуктов:
 - Добавление нового продукта в базу данных,
 - Просмотр базы данных,
 - Удаление продукта из базы данных,
 - Сортировка списка,
 - Поиск продукта,
 - Редактирование информации о продукте,

- Покупка продукта,
- Возврат продукта,
- Создание чека.

2. Базой данных аккаунтов

- Просмотр списка аккаунтов,
- Добавление аккаунта (администратор или пользователь),
- Редактирование аккаунта,
- Просмотр чеков аккаунтов,
- Сортировка аккаунтов,
- Поиск аккаунта,
- Удаление аккаунта.

В режиме пользователя доступны взаимодействия:

1. С базой данных продуктов:

- Просмотр базы данных,
- Сортировка списка,
- Поиск аккаунта,
- Покупка продукта,
- Возврат продукта,
- Создание чека.

2. Со своим аккаунтом:

- Редактирование аккаунта.

В курсовой работе учтено:

1. Исключительные ситуации:

- запись определённого текста, при наведении мышкой на каждое поле предусмотрено всплывающее окно с разрешёнными символами;
- проверка на существование пользователя в базе и на заполнение полей при авторизации;

- проверки на существования названий продуктов, логинов пользователей к которым мы обращаемся;
- проверки, касающиеся покупки, продажи, получения чеков;
- проверки на существование файлов;
- проверки на изменение или удаление информации людей, которые выше по роли;
- проверка на то, при просмотре администратором чеков, покупал ли пользователь что-нибудь;
- проверки на невозможность сменить название продукта или логин пользователя на уже существующий.

2. Существование возможности выйти из того или иного окна.

Требования к программной реализации курсовой работы:

- Все переменные и константы должны иметь осмысленные имена в рамках тематики варианта курсовой работы.
- Имена функций должны быть осмысленными, начинаться с буквы нижнего регистра, строиться по принципу глагол + существительное (например, `addAccount`, `findStudentBySurname`). Если функция выполняет проверку и возвращает результат типа `bool`, то ее название должно начинаться с глагола `is` (например, `isNumberNumeric`, `isLoginUnique`).
- Не допускается использование оператора прерывания `goto`.
- Код не должен содержать неименованных числовых констант («магических» чисел), неименованных строковых констант (например, имен файлов и др.). Подобного рода информацию следует представлять как глобальные константы. По правилам качественного стиля программирования тексты всех информационных сообщений, выводимых пользователю в ответ на его действия, также оформляются как константы.
- Код необходимо комментировать (как минимум в части объявления структур, массивов/векторов, прототипов функций, нетривиальной логики).
- Код не должен дублироваться – для этого существуют функции.

– Одна функция решает только одну задачу (например, не допускается в одной функции считывать данные из файла и выводить их на консоль – это две разные функции). При этом внутри функции возможен вызов других функций.

– Выполнение операций чтения/записи в файл должно быть сведено к минимуму (т. е. после однократной выгрузки данных из файла в массив/вектор дальнейшая работа ведется с этим массивом/вектором, а не происходит многократное считывание данных из файла в каждой функции).

– Следует избегать глубокой вложенности условных и циклических конструкций: вложенность блоков должна быть не более трех.

– Следует избегать длинных функций: текст функции должен уместиться на один экран (размер текста не должен превышать 25–50 строк).

– Следует выносить код логически независимых модулей в отдельные `cpp` файлы и подключать их с помощью заголовочных `.h` файлов.

2 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

Выбранная парадигма программирования: объектно-ориентированная.

У объектно-ориентированного программирования много плюсов, и именно поэтому этот подход использует большинство современных программистов.

- Визуально код становится проще, и его легче читать. Когда всё разбито на объекты и у них есть понятный набор правил, можно сразу понять, за что отвечает каждый объект и из чего он состоит.

- Меньше одинакового кода. Если в обычном программировании одна функция считает повторяющиеся символы в одномерном массиве, а другая — в двумерном, то у них большая часть кода будет одинаковой. В ООП это решается наследованием.

- Сложные программы пишутся проще. Каждую большую программу можно разложить на несколько блоков, сделать им минимальное наполнение, а потом раз за разом подробно наполнить каждый блок.

- Увеличивается скорость написания. На старте можно быстро создать нужные компоненты внутри программы, чтобы получить минимально работающий прототип.

В качестве выбора способа описания входных данных приводится описание следующих типов class (с указанием конкретных полей):

- для авторизации учётных записей пользователей (класс состоит из login – уникальный, для каждого пользователя, набор символов, состоящий из цифр и букв, необходимый для доступа к системе, hashPassword – хешированный пароль, от учётной записи, salt – “соль” для хеширования пароля, role – поле для разделения в правах создателя (2), администраторов (1) и пользователей (0);

– для данных об учётных записях (класс состоит из name – имя пользователя, surname – фамилия пользователя, age – возраст пользователя, identifier – уникальный идентификатор пользователя, products – список существующих продуктов, receipts – чеки пользователя, receipt – нынешний чек, users – список пользователей);

– для данных о продуктах (класс состоит из name – название продукта, price – цена продукта, group – группа в которой состоит продукт, country – страна в которой произведён продукт, type – тип продукта, individual abilities – индивидуальные особенности продукта, barCode – штрих-код);

– для создания покупки (получения чека) (класс состоит из products – список продуктов купленных пользователем, summaryPrice – итоговая цена, identifier – идентификатор пользователя которому принадлежит чек, numberOfProducts – количество купленных продуктов);

– для времени (класс состоит из year – года, month – месяца, day – дня, hour – часа, minute – минут, second – секунд).

В качестве способа объединения данных указывается:

- использование самостоятельно-разработанного шаблона List;
- использование самостоятельно-разработанного шаблона Vector;
- его выбранная область видимости – локальная.

В качестве работы с файлами данных выбран форматированный файловый ввод/вывод.

3 ОБОСНОВАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ

При решении поставленной задачи необходимо определиться с методом программирования, а также с тем, как хранить информацию и как её обрабатывать.

При создании программ на основе объектно-ориентированной парадигмы из предметной области выделяются объекты, поведение и взаимодействие которых моделируются с помощью программы. Кроме того, в программе есть служебные объекты, служащие для хранения объектов, их визуализации и других необходимых функций.

В качестве способа объединения данных выбрано два контейнера STL: двунаправленный список – List (для быстрого добавления и удаления элементов) и вектор – Vector (для получения быстрого доступа к элементам вектора).

Обоснование выбранных методов и алгоритмов решения программы:

Поиск: линейный поиск

- в векторе, т.к. мы не можем быть уверены, что вектор отсортирован и не можем знать, в каком диапазоне будет находиться искомый элемент;

- в двунаправленном списке, т.к. для списка линейный поиск является самым быстрым.

Сортировка: пузырьком для вектора и двунаправленного списка, т.к. она не требует дополнительных массивов, легка в реализации и хороша для нашего количества элементов массива.

Файл: txt

- Универсальность — текстовый файл может быть прочитан (так или иначе) на любой системе или ОС, особенно если речь идёт об однобайтных кодировках вроде ASCII, которые не подвержены проблеме, характерной для других форматов файлов — для них не важна разница в порядке байтов или длине машинного слова на разных платформах.

– Устойчивость — каждое слово и символ в таком файле самодостаточно и, если случится повреждение байтов в таком файле, то обычно можно восстановить данные или продолжить обработку остального содержимого, в то время как у сжатых или двоичных файлов повреждение нескольких байтов может сделать файл совершенно невозможным. Многие системы управления версиями рассчитаны на текстовые файлы и с двоичными файлами могут работать только как с единым целым.

– Формат текстового файла крайне прост и его можно изменять текстовым редактором — программой, входящей в комплект практически любой ОС.

4 КОНСТРУИРОВАНИЕ ПРОГРАММЫ

Диаграммы классов используются для более удобного представления иерархии классов в программе.

Диаграммы классов показывают набор классов, интерфейсов, а также их связи. Диаграммы этого вида чаще всего используются для моделирования объектно-ориентированных систем. Они предназначены для статического представления системы.

Работа программы начинается с авторизации, для чего предусмотрен класс `Authorization` с полями:

- `login`,
- `hashPassword`,
- `salt`,
- `role`.

Поля отвечают за корректную авторизацию пользователей и хеширование пароля.

Классы `User`, `Admin`, `Creator` хранят данные о пользователях и включают в себя поля:

- `products`,
- `receipts`,
- `receipt`,
- `users`,
- `identifier`,
- `age`,
- `name`,
- `surname`.

Класс позволяет работать с пользователями, получать доступ к данным и управлять ими.

Класс Product создан для управления информацией о продуктах и включает в себя поля:

- name,
- price,
- group,
- type,
- individualAbility,
- country,
- barCode.

Поля класса Product хранят информацию о продукте.

Класс Receipt предусмотрен для объединения продуктов и пользователей и включает в себя поля:

- products,
- summaryPrice,
- identificator,
- numberOfProducts.

Данные поля предоставляют доступ к информации о взаимодействии пользователя с продуктами.

Класс Date создан для фиксации даты во время реализации покупки пользователем и имеет поля:

- year,
- month,
- day,
- hour,
- minute,
- second.

Поля класса позволяют записать в них данные о дате в выбранное время.

Диаграмма классов, отображающая отношения между приведенными

классами, приведена в приложении Б.

Модульная структура программы приведена в приложении В.

Курсовая работа включает в себя восемнадцать отдельных `cpp`-файлов и двадцать `h` - файлов:

- `AddProductForm.h` + `AddProductForm.cpp`;
- `Admin.h` + `Admin.cpp`;
- `Authorization.h` + `Authorization.cpp`;
- `AuthorizationForm.h` + `AuthorizationForm.cpp`;
- `Creator.h` + `Creator.cpp`;
- `Date.h` + `Date.cpp`;
- `Files.h` + `Files.cpp`;
- `FriendFunctions.h` + `FriendFunctions.cpp`;
- `OwnInformationForm.h` + `OwnInformationForm.cpp`;
- `Product.h` + `Product.cpp`;
- `ProductManagingForm.h` + `ProductManagingForm.cpp`;
- `Receipt.h` + `Receipt.cpp`;
- `ReceiptForm.h` + `ReceiptForm.cpp`;
- `Registration.h` + `Registration.cpp`;
- `User.h` + `User.cpp`;
- `UserForm.h` + `UserForm.cpp`;
- `UserInfoForm.h` + `UserInfoForm.cpp`;
- `UserManagingForm.h` + `UserManagingForm.cpp`;
- `List.h`;
- `Vector.h`.

Файлы, в которых хранятся данные:

- `usersFile.txt`;
- `productsFile.txt`;
- `receiptsFile.txt`.

5 РАЗРАБОТКА АЛГОРИТМОВ РАБОТЫ ПРОГРАММЫ

5.1 Алгоритм функции WinMain

Функция WinMain – точка входа в программу. Блок-схема алгоритма на рисунке Г.1 в приложении Г.

5.2 Алгоритм функции sortProductByName

Алгоритм функции sortProductsByName предусмотрен для сортировки продуктов по имени. Блок-схема алгоритма на рисунке Г.2 в приложении Г.

5.3 Алгоритм функции findProduct

Алгоритм функции findProduct предусмотрен для поиска продукта по имени. Блок-схема алгоритма на рисунке Г.3 в приложении Г.

5.4 Алгоритм функции buyProduct

Алгоритм функции buyProduct предусмотрен для покупки продукта по имени. Блок-схема алгоритма на рисунке Г.4 в приложении Г.

5.5 Алгоритм функции editProductPrice

Алгоритм функции editProductPrice предусмотрен для изменения цены продукта по имени. Блок-схема алгоритма на рисунке Г.5 в приложении Г.

5.6 Алгоритм функции `sortUsersByName`

Алгоритм функции `sortUsersByName` предназначен для сортировки пользователей по имени. Пошаговое описание:

Шаг 1: Инициализация `i = 0`;

Шаг 2: Инициализация `it1 = users.begin()`;

Шаг 3: Проверка, `i < users.getSize()-1`: если да – переход к шагу 4, если нет – переход к шагу 10;

Шаг 4: Инициализация `it2 = it1+1`;

Шаг 5: Проверка, `it2!=users.end()`: если да – к шагу 6, нет – к шагу 9;

Шаг 6: Вызов функции, которая проверяет, какая из строк больше, если первая строка меньше – переход к шагу 7, если нет – переход к шагу 8;

Шаг 7: Вызов функции, которая меняет данные в `it1` и `it2` местами;

Шаг 8: Инкремент `it2`, переход к шагу 5;

Шаг 9: Инкремент `it1`, переход к шагу 3;

Шаг 10: Выход.

5.7 Алгоритм функции `findUser`

Алгоритм функции `findUser` предназначен для поиска пользователей. Пошаговое описание:

Шаг 1: Инициализация `user = users.begin()`;

Шаг 2: Проверка условия: `user!=users.end()`, если да, переход к шагу 3, если нет, переход к шагу 6;

Шаг 3: Сравнение строк, если `(*user).getLogin()` равно `login` - переход к шагу 4, если нет, переход к шагу 5;

Шаг 4: Вернуть `*user`. Выход;

Шаг 5: Инкремент `user`. Переходу к шагу 2;

Шаг 6: Выход.

5.8 Алгоритм функции editUserName

Алгоритм функции editUserName предназначен для изменения пользовательского имени администратором. Пошаговое описание:

Шаг 1: Инициализация `it = users.begin()`;

Шаг 2: Проверка условия: `it!=users.end()`, если да, переход к шагу 3, если нет, переход к шагу 6;

Шаг 3: Сравнение строк, если `(*user).getLogin()` равно `login` - переход к шагу 4, если нет, переход к шагу 6;

Шаг 4: Вызов функции `*user.setName`, которая устанавливает новое имя пользователю.

Шаг 5: Вернуть `true`. Выход;

Шаг 6: Инкремент `it`. Переходу к шагу 2;

Шаг 7: Вернуть `false`. Выход.

5.9 Алгоритм функции getSummaryPrice

Алгоритм функции getSummaryPrice предназначен для получения итоговой суммы за покупку. Пошаговое описание:

Шаг 1: Инициализация `summaryPrice = 0.0`;

Шаг 2: Начало цикла по списку `products`;

Шаг 3: Добавляем с присваиванием `product.getPrice()` к `summaryPrice`;

Шаг 4: Переход к следующему элементу `products`, если элемент не указывает на конец коллекции, переход к шагу 3, если нет – переход к шагу 5;

Шаг 5: Вернуть `summaryPrice`. Выход.

5.10 Алгоритм функции addAccount

Алгоритм функции addAccount предназначен для добавления нового пользователя в систему администратором. Пошаговое описание:

Шаг 1: Инициализация `user = users.begin();`

Шаг 2: Проверка условия: `user!=users.end()`, если да, переход к шагу 3, если нет, переход к шагу 6;

Шаг 3: Сравнение строк, если `(*user).getLogin()` равно `login` - переход к шагу 4, если нет, переход к шагу 5;

Шаг 4: Вернуть `false`. Выход;

Шаг 5: Инкремент `user`. Переходу к шагу 2;

Шаг 6: Добавляем пользователя в конец списка `users.push_back(user);`

Шаг 7: Выход.

6 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

6.1 Авторизация

На начальном этапе программы пользователь должен пройти авторизацию. При запуске программы пользователю выводит окно, показанное на рисунке Д.1.

Для успешной авторизации должны быть заполнены поля «Username» и «Password». По соображениям безопасности в форме авторизации пароль замаскирован символами «звёздочка» *. Когда поля будут заполнены, следует нажать на кнопку «Log in» и только в случае корректного ввода данных, которые существует в системе, авторизация будет пройдена.

Если же в программу заходит новый пользователь, он должен зарегистрироваться. Для регистрации следует нажать кнопку «Register». После нажатия кнопки откроется форма, представленная на рисунке Д.2.

Для успешной регистрации достаточно заполнить лишь поля «Username» и «Password», можно заполнить все поля для дополнительной информации. После нажатия кнопки «Register» регистрируется аккаунт в системе, или же кнопки «Back», произойдёт возврат на окно авторизации (рисунок Д.1).

В случае, если пользователь хочет выйти из системы, ему следует нажать на кнопку «Application», а затем на «Exit», изображенную на рисунке Д.3. Далее пользователь должен будет подтвердить выход из системы (рисунок Д.4).

6.2 Модуль пользователя

После успешного прохождения авторизации пользователи попадают на окно, представленное на рисунке Д.5. Рассмотрим модуль пользователя.

Если пользователь хочет купить продукт, он должен ввести его название в поле «Product name» и затем нажать на кнопку «Buy product». В случае если такой продукт существует он добавится в таблицу «Receipt» (рисунок Д.6), если же такого продукта нет, выведется соответствующее сообщение, представленное на рисунке Д.7.

Если пользователь передумал, что хочет покупать этот продукт, он должен ввести его название в поле «Product name» и затем нажать на кнопку «Remove product». В случае если такой продукт вы покупали он уберётся из таблицы «Receipt», если же такого продукта вы не покупали, выведется соответствующее сообщение, представленное на рисунке Д.8.

Если пользователь решил, что хочет посмотреть информацию об каком-то продукте, он должен ввести его название в поле «Product name» и затем нажать на кнопку «Find product». В случае если такой продукт существует он будет выделен жёлтым цветом в таблице «Product list» (рисунок Д.9), если же такого продукта не существует, выведется соответствующее сообщение, представленное на рисунке Д.7.

В случае, если пользователь хочет отсортировать продукты по какому-либо критерию, есть выпадающий список «Sorting» (рисунок Д.10), где вы можете выбрать желаемую сортировку.

В тот момент, когда пользователь выбрал то, что он хочет купить, он нажимает на кнопку «Make a receipt». Далее ему нужно подтвердить его покупку (рисунок Д.11). В случае подтверждения, при успешной покупке, будет выведено оповещение, что покупка произошла успешно (рисунок Д.12)

Пользователь может изменить собственную информацию, нажав на кнопку «Changing information about yourself», после нажатия которой

откроется окно, представленное на рисунке Д.13. В целях безопасности пароль в окне не выводится и будет изменён лишь в том случае, если туда что-то ввести.

Пользователь может внести изменения и затем нажать на кнопку «Save input information about yourself», в следствие чего будет выведено оповещение с просьбой подтвердить сохранение изменений (рисунок Д.14). В случае, если пользователь подтвердит сохранение изменений, данные сохранятся, выведется оповещение с успешным сохранением (рисунок Д.15) и окно закроется. Также пользователь может нажать на кнопку «Cancel» и также окно закроется.

Кнопки «Managing products» и «Managing accounts» у обычного пользователя недоступны.

В случае, если пользователь хочет вернуться к авторизации, ему следует нажать на кнопку «File», а затем на «Exit», изображённую на рисунке Д.16. Далее пользователь должен будет подтвердить выход из системы (рисунок Д.17).

6.3 Модуль администратора

Все действия, которые есть в модуле пользователя, может произвести и модуль администратора.

В модуле администратора появился доступ к двум кнопкам: «Managing accounts» и «Managing products».

При нажатии кнопки «Managing products», откроется следующее окно, представленное на рисунке Д.18.

Если администратор хочет добавить новый продукт, он должен нажать на кнопку «Add new product». Откроется следующее окно, представленное на рисунке Д.19. Для того, чтобы добавить новый продукт, нужно заполнить все поля и нажать на кнопку «Add», после чего администратору нужно будет подтвердить добавление нового продукта (рисунок Д.20). В случае, если продукта с таким именем не существует, продукт успешно добавиться и вам выведется соответствующее оповещение (рисунок Д.21). Вернутся обратно можно с помощью кнопки «Back».

Если администратор хочет изменить информацию об уже существующем продукте, он должен написать в поле «Product name» название продукта и, затем, нажать на кнопку «Edit product information». Откроется следующее окно, представленное на рисунке Д.22. Для того, чтобы сохранить изменения о продукте, нужно заполнить все поля и нажать на кнопку «Save», после чего администратору нужно будет подтвердить сохранение изменений о продукте (рисунок Д.23). В случае успешного изменения информации выведется соответствующее оповещение (рисунок Д.24). Вернутся обратно можно с помощью кнопки «Back».

Если администратор хочет удалить продукт, он должен написать в поле «Product name» название продукта и, затем, нажать на кнопку «Delete product». Для того, чтобы удалить продукт, нужно подтвердить удаление продукта

(рисунок Д.25). В случае успешного удаления выведется соответствующее оповещение (рисунок Д.26).

Вернуться к предыдущей форме можно с помощью кнопки «Back to previous form» или кнопки «Exit», представленной на рисунке Д.27.

При нажатии кнопки «Managing accounts», откроется следующее окно, представленное на рисунке Д.28.

Если администратор хочет добавить нового пользователя, он должен нажать на кнопку «Add user». Откроется следующее окно, представленное на рисунке Д.29. Для того, чтобы добавить нового пользователя, нужно заполнить хотя бы логин и пароль (если администратор является создателем, то он может выбирать роль пользователя) и нажать на кнопку «Add», после чего администратору нужно будет подтвердить добавление нового пользователя (рисунок Д.30). В случае, если пользователя с таким логином не существует, пользователь успешно добавится и вам выведется соответствующее оповещение (рисунок Д.31). Вернуться обратно можно с помощью кнопки «Back».

Если администратор хочет изменить информацию об уже существующем пользователе, он должен написать в поле «User's username» логин пользователя и, затем, нажать на кнопку «Edit user information». Откроется следующее окно, представленное на рисунке Д.32 (если администратор является создателем, то он также может изменить поле «Role»). По соображениям безопасности пароль не отображается и будет изменён только в случае внесения каких-нибудь данных в поле «Password». Для того, чтобы сохранить изменения о пользователе, нужно заполнить все поля и нажать на кнопку «Save», после чего администратору нужно будет подтвердить сохранение изменений о пользователе (рисунок Д.33). В случае успешного изменения информации выведется соответствующее оповещение (рисунок Д.34). Вернуться обратно можно с помощью кнопки «Back».

Если администратор хочет удалить пользователя, он должен написать в поле «User's username» логин пользователя и, затем, нажать на кнопку «Delete user». Для того, чтобы удалить пользователя, нужно подтвердить удаление пользователя (рисунок Д.35). В случае успешного удаления выведется соответствующее оповещение (рисунок Д.36).

Если администратор хочет посмотреть информацию пользователя, он должен ввести в поле «User's username» его логин и нажать на кнопку «Find user». В случае если такой пользователь существует он будет выделен жёлтым цветом в таблице «User list» (рисунок Д.37), если же такого продукта не существует, выведется соответствующее сообщение, представленное на рисунке Д.38.

В случае, если администратор хочет отсортировать пользователей по какому-либо критерию, есть выпадающий список «Sorting» (рисунок Д.39), где вы можете выбрать желаемую сортировку.

Если администратор хочет посмотреть информацию о том, что покупали пользователи, он должен ввести в поле «User's username» его логин и нажать на кнопку «Show receipts». Откроется окно, изображённое на рисунке Д.40. На окне есть две кнопки «Previous receipt» и «Next receipt», которые позволяют перемещаться между чеками и кнопка «Back», для того, чтобы вернуться в прошлое окно.

6.4 Исключительные ситуации

В системе предусмотрена обработка различных исключительных ситуаций, таких как:

- запись определённого текста (рисунок Д.41), при наведении мышкой на каждое поле предусмотрено всплывающее окно с разрешёнными символами;
- проверка на существование пользователя в базе и на заполнение полей при авторизации (рисунок Д.42);
- проверки на существования названий продуктов, логинов пользователей к которым мы обращаемся (рисунки Д.43, Д.44);
- проверки, касающиеся покупки, продажи, получения чеков (рисунки Д.45, Д.46);
- проверки на существование файлов;
- проверки на изменение или удаление информации людей, которые выше по роли (рисунок Д.47, Д.48);
- проверка на то, при просмотре администратором чеков, покупал ли пользователь что-нибудь (рисунок Д.49);
- проверки на невозможность сменить название продукта или логин пользователя на уже существующий (рисунки Д.50, Д.51).

ЗАКЛЮЧЕНИЕ

В результате курсового проектирования была разработана программа для обслуживания клиентов магазина на языке C++.

В процессе проектирования данного программного модуля были задействованы новые возможности языка C++, которые не присутствовали в C, и которые существенно облегчают задачу во многих случаях и обеспечивают некоторую расширяемость, несмотря на процедурный стиль программирования.

Проект разработан в полном объеме и полностью соответствует поставленной цели. В качестве возможного направления развития разработанного программного модуля можно рассматривать перевод в парадигму ООП с использованием принципов SOLID и, если понадобится, паттернов проектирования для обеспечения намного лучшей масштабируемости и облегчения поддержки другими программистами данного программного модуля длительное время.

ЛИТЕРАТУРА

- [1] Роберт Лафоре. Объектно-ориентированное программирование в C++. 2018. — 928 с.
- [2] Шилдт, Г. C++. Базовый курс / Шилдт Г. — М.: Виллиамс, 2015. — 624 с.
- [3] Скотт Мэйерс — Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ, 2014. — 300 с.
- [4] Герб Саттер, Андрей Александреску — Стандарты программирования на C++, 2016. — 224 с.
- [5] Герб Саттер — Решение сложных задач на C++, 2008. — 400 с.

ПРИЛОЖЕНИЕ А – ЛИСТИНГ ПРОГРАММЫ

Файлы заголовков:

1) Authorization.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <ctime>
#include <string>
#include "FriendFunctions.h"
#include "Files.h"

using namespace std;

class Authorization
{
public:
    Authorization(const char* login, const char* password, int role = USER);           //constructor

    Authorization(const Authorization& authorization);                               //    copy constructor
    Authorization& operator= (const Authorization& authorization);                   //    operator = overload

    ~Authorization();                                                                //    destructor

    bool enter();                                                                    //    authorization methods that provides
    bool registration(const char* name = undefined,

        const char* surname = undefined,

        int age = defaultNumber);                                                  //    users access to system

    char* getLogin();                                                                //    login getter
    char* getHashPassword();                                                         //    hash password getter
    char* getSalt();                                                                //    salt getter
    int getRole();                                                                //    role getter

    void setLogin(const char* newLogin);                                             //    login setter
    void setPassword(const char* newPassword);                                       //    password setter

private:
```



```

bool isExist(); // checking for data existence

char* login; // login
char* hashPassword; // hash password

bool write(ostream& os,

           const char* name = undefined,
           const char* surname = undefined,

           int age = defaultNumber); // methods for data input output in file
bool read(istream& is); //

void makeHashPassword(const char* password); // method that makes hash password
char* returnUnhashPassword(); // method that unhashes password
void makeSalt(const char* password); // method that makes salt

protected:
char* salt; // salt for making hash password
int role; // role
Authorization(); // default constructor
void setHashPassword(const char* newHashPassword); // hash password setter
};

```

2) User.h

```

#pragma once

#include "D:\Влад БГУИР\КПрог Лабораторные работы\List\List.h"
#include "D:\Влад БГУИР\КПрог Лабораторные работы\Vector\Vector.h"
#include "Product.h"
#include "Receipt.h"
#include "Authorization.h"

class Receipt;

class User : public Authorization
{
public:
    User(); // default constructor
    User(const char* login); // constructor

    User(User& user); // copy constructor
    User& operator=(User& user); // operator = overload

    ~User(); // destructor

```

```

ostream& showProducts(ostream& os);          //    show products to os

                                           //    sort products by:
void sortProductsByName();                  //    name,
void sortProductsByCountry();              //    country,
void sortProductsByPrice();                //    price,
void sortProductsByGroup();                //    group,
void sortProductsByType();                 //    type,
void sortProductsByBarCode();              //    barcode.

Product& findProduct(const char* name);     //    find product in products vector

char* getName();                           //    user information getters
char* getSurname();                        //
int getAge();                              //
User& getFullUserInformation();            //
int getIdentifier();                       //

void setName(const char* newName);          //    user information setters
void setSurname(const char* newSurname);    //
void setAge(const int newAge);              //

void showReceipts(ostream& os);             //    show receipts to os
bool makeReceipt();                         //    make receipt from (Receipt* receipt)

bool buyProduct(const char* name);          //    add product to receipt
bool deleteBuyingProduct(const char* name); //    remove product from receipt

bool saveReceiptInformation();              //    save receipt information to file
bool saveUsersInformation();                //    save user information to file

Vector<Product> getProductsVector();         //    products vector getter
List<Receipt> getReceiptsList();             //    receipts list getter
Receipt* getReceiptPointer();                //    receipt pointer getter
List<User> getUsersList();                    //    users list getter

friend class Creator; //    make class Creator friendle for giving access for role

protected:
    Vector<Product> products;                 //    products vector
    List<Receipt> receipts;                   //    receipts list

```

```

    Receipt* receipt;           // receipt pointer
    List<User> users;           // users list
    bool write(ostream& os);    // write this(User) to file
    bool read(istream& is);     // read from file to this(User)

private:
    int identifier;             // user id
    int age;                    // user age
    char* name;                 // user name
    char* surname;              // user surname

    void showProduct(Product& product, ostream& os); // method that outputs
information about a single product
    bool areUsersWithId();      // are all identifiers in users file
};

```

3) Admin.h

```

#pragma once
#include "User.h"

class Admin : public User
{
public:
    Admin(const char* login) : User(login) {} // constructor
    Admin(Admin& admin) : User(admin) {}      // copy constructor

    bool deleteProduct(const char* name);      // delete Product from
products
    bool addProduct(Product& product);        // add Product to products

                                                    // edit
product:
    bool editProductCountry(const char* name, const char* newCountry); //
country,
    bool editProductPrice(const char* name, double newPrice);           // price,
    bool editProductType(const char* name, const char* newType);        // type,
    bool editProductIndividualAbility(const char* name, const char*
newIndividualAbility); // individual ability,
    bool editProductGroup(const char* name, const char* newGroup);      // group,

```

```

    bool editProductBarCode(const char* name, int newBarCode);          //
    barcode,
    bool editProductName(const char* name, const char* newName);        //    name.

    bool addAccount(User& user);          //    delete User from users
    bool deleteAccount(const char* login); //    add User to users

                                                    //    edit
user:
    bool editUserLogin(const char* login, const char* newLogin);        //    login,
    bool editUserPassword(const char* login, const char* newPassword); // password,
    bool editUserName(const char* login, const char* newName);          //    name,
    bool editUserSurname(const char* login, const char* newSurname);    //
    surname,
    bool editUserAge(const char* login, const int newAge);              //    age.

                                                    //    sort users by:
    void sortUsersByLogin();          //    login,
    void sortUsersByName();           //    name,
    void sortUsersBySurname();        //    surname,
    void sortUsersByAge();            //    age.

    User& findUser(const char* login); //    find user in users

    bool saveProductsToFile();        //    save products to file
};

```

4) Creator.h

```

#pragma once
#include "Admin.h"

class Creator : public Admin
{
public:
    Creator(const char* login) : Admin(login) {}          //    constructor
    bool changeRole(const char* login, const int newRole); //    change users role
};

```

5) Product.h

```
#pragma once

#include "FriendFunctions.h"
#include <iostream>
#include <ctime>

using namespace std;

class Product
{
public:
    Product(const char* name,
            const double price = static_cast<double>(defaultNumber),

            const char* group = undefined,

            const char* type = undefined,

            const char* individualAbility = undefined,

            const char* country = undefined,
            const int barCode = defaultNumber);    // constructor

    Product();                                     // default constructor

    Product(const Product& product);               // copy constructor
    Product& operator=(const Product& product);    // operator = overload

    ~Product();                                    // destructor

    char* getName();                               // product information getters
    double getPrice();                             //
    char* getGroup();                             //
    char* getIndividualAbility();                  //
    char* getType();                              //
    char* getCountry();                           //
    int getBarCode();                             //

    void setName(const char* newName);             // product information setters
    void setPrice(const double newPrice);          //
    void setGroup(const char* newGroup);           //
```

```

void setType(const char* newType);          //
void setIndividualAbility(const char* newIndividualAbility);

void setCountry(const char* newCountry); //
void setBarCode(const int newBarCode);    //

bool write(ostream& os);                   //    write this(Product) to file
bool read(istream& is);                   //    read from file to this(Product)

private:
    char* name;                           //    name
    double price;                         //    price
    char* group;                          //    group
    char* type;                           //    type
    char* individualAbility;              //    individual ability
    char* country;                        //    country
    int barCode;                          //    bar code
};

```

6) Receipt.h

```

#pragma once
#include "D:\Влад БГУИР\КПрог Лабораторные работы\List\List.h"
#include "Product.h"
#include "Date.h"
#include "User.h"

class Receipt : public Date
{
public:
    Receipt(int identificator = defaultNumber,
            int year = defaultNumber,

            int month = defaultNumber,

            int day = defaultNumber,

            int hour = defaultNumber,

            int minute = defaultNumber,

            int second = defaultNumber);    //    constructor

```

```

Receipt(const Receipt& receipt);           //    copy constructor
Receipt& operator=(const Receipt& receipt); //    operator = overload

~Receipt();                               //    destructor

ostream& showReceipt(ostream& os);        //    methods for using receipt


//    sort products by:
void sortByName();           //    name,
void sortByCountry();        //    country,
void sortByPrice();          //    price,
void sortByGroup();          //    group,
void sortByType();           //    type,
void sortByBarCode();        //    barcode.


Product& findProduct(const char* findName); //    find product in list products

void pushProduct(const Product& product); //    add product to products list
void popProduct(const char* productName); //    remove products from products list


double getSummaryPrice();      //    working with price
int getNumberOfProducts();     //    product number getter
List<Product> getProductsList(); //    products list getter


friend class User; // making User class frindly for making Receipts method
acceptable
private:
    List<Product> products;      //    product list for a filling the receipt
    double summaryPrice;        //    summary price
    int identificator;           //    identificator
    int numberOfProducts;       //    numberOfProducts


    bool read(istream& is);      //    read from file to this(Receipt)
    bool write(ostream& os);     //    write to this(Receipt) from file
    bool makeReceipt(int identificator); //    make Receipt
};

```

7) Date.h

```

#pragma once
#include <ctime>

class Date

```

```

{
public:
    Date();                //    constructor

    void setDate();        //    set time that is now

    int getYear();         //    getters for date
    int getMonth();        //
    int getDay();          //
    int getHour();         //
    int getMinute();       //
    int getSecond();       //

protected:
    int year;              //    year
    int month;             //    month
    int day;               //    day
    int hour;              //    hour
    int minute;            //    minute
    int second;            //    second
};

```

8) Files.h

```

#pragma once

extern const char* receiptsFile;    //    file for receipts
extern const char* productsFile;    //    file for products
extern const char* usersFile;       //    file for users

```

9) FrindlyFunctions.h

```

#pragma once

#include <iostream>

using namespace System;
using namespace Runtime::InteropServices;

#define LESS -1    //    values for comparing
#define EQUAL 0    //
#define MORE 1    //

```



```

#define USER 0      //      users roles
#define ADMIN 1     //
#define CREATOR 2   //

extern const char* undefined;
                //      underfined for default variables in classes for char*
extern const int defaultNumber;
                //      default number for default variables in classes for int and double
extern const char* lettersTip;
extern const char* numbersTip;
extern const char* lettersAndNumbersTip;
extern const char* lettersAndNumbersAndSpaceTip;
extern const char* doubleNumbersTip;
extern const char* lettersAndSpaceTip;

void copyString(const char* str1, char*& str2); //      copy str1 to str2
bool equal(const char* str1, const char* str2); //      is str1 == str2
int compareString(const char* str1, const char* str2); //      returns LESS EQUAL or MORE
depending on comparing str1 and str2
char* convertSpaceToUnderline(char* str); //      converts space into underline for file
char* convertUnderlineToSpace(char* str); //      converts underline into space for file
void MarshalString(String^ s, std::string& outputstring);
bool isCorrectNumberInput(System::Windows::Forms::KeyPressEventArgs^ e);
bool isCorrectLettersInput(System::Windows::Forms::KeyPressEventArgs^ e);
bool isCorrectLettersAndNumbersInput(System::Windows::Forms::KeyPressEventArgs^ e);
bool isCorrectLettersAndNumbersAndSpaceInput(System::Windows::Forms::KeyPressEventArgs^
e);
bool isCorrectDoubleNumberInput(System::Windows::Forms::KeyPressEventArgs^ e);
bool isCorrectLettersAndSpaceInput(System::Windows::Forms::KeyPressEventArgs^ e);

template <typename T>
void swapElements(T& t1, T& t2) { //      swap elements
    T swap = t1;
    t1 = t2;
    t2 = swap;
}

```

10) List.h

```

#pragma once
#include <iostream>

```

```

using namespace std;

template <typename T>
class List
{
private:
    template <typename T>                                //Node of List
    class Node
    {
    public:
        Node* next;                                     //pointer to
next Node
        Node* prev;                                     //pointer to
previous Node
        T value;                                         //value
        Node(T value = T(), Node* next = nullptr, Node* prev = nullptr)
        {
            this->value = value;
            this->next = next;
            this->prev = prev;
        }
    };

    int size;                                           //size of
List
    Node<T>* tail;                                       //pointer to
tail Node
    Node<T>* head;                                       //pointer to
head Node
public:
    List();
    ~List();

    List(const List& list);
    List<T>& operator=(const List& list);

    class Iterator
    {
    private:
        Node<T>* pointer;
    public:

```

```

Iterator(Node<T>* pointer)
{
    this->pointer = pointer;
}
Iterator operator++(int)
{
    Iterator it = *this;
    pointer = pointer->next;
    return it;
}
Iterator operator++()
{
    pointer = pointer->next;
    return pointer;
}
Iterator operator--(int)
{
    Iterator it = *this;
    pointer = pointer->prev;
    return it;
}
Iterator operator--()
{
    pointer = pointer->prev;
    return pointer;
}
Iterator operator+(int index)
{
    Iterator it = *this;
    for (int i = 0; i < index; i++)
    {
        it.pointer = it.pointer->next;
    }
    return it;
}
T& operator*()
{
    return pointer->value;
}
bool operator==(const Iterator& itr) {
    return pointer == itr.pointer;
}

```

```

        bool operator!=(const Iterator& itr)
        {
            return pointer != itr.pointer;
        }
};

Iterator begin() const //begin pointer
{
    return Iterator(this->head);
}

Iterator end() const //end pointer
{
    if (this->tail != nullptr)
    {
        return Iterator(this->tail->next);
    }
    else
    {
        return Iterator(nullptr);
    }
}

Iterator rbegin() const //reverse begin
pointer
{
    if (this->head != nullptr)
    {
        return Iterator(this->head->prev);
    }
    else
    {
        return Iterator(nullptr);
    }
}

Iterator rend() const //reverse end
pointer
{
    return Iterator(this->tail);
}

void push_back(T value); //add to tail
void push_front(T value); //add to head

```

```

        int getSize(); //return size
        T& operator[](int index); //get element by index
        T& front(); //address of
head
        T& back(); //address of
tail
        void pop_back(); //delete from tail
        void pop_front(); //delete from head
        void clear(); //clear all list
        bool isEmpty(); //returns
whether there are elements in list
        void insert(T value, int index); //add element by index
        void removeAt(int index); //delete element by index
        void reverse(); //reverse
list
};

```

```

template<typename T>
inline List<T>::List()
{
    this->size = 0;
    this->tail = nullptr;
    this->head = nullptr;
}

```

```

template<typename T>
inline List<T>::~~List()
{
    clear();
}

```

```

template<typename T>
inline List<T>::List(const List& list)
{
    this->tail = nullptr;
    this->head = nullptr;

    this->size = list.size;

    if (list.head != nullptr && list.tail != nullptr)
    {

```

```

        Node<T>* tempHead = list.head;

        Node<T>* temp = new Node<T>(tempHead->value, tempHead->next);
        this->head = temp;

        while (tempHead->next)
        {
            tempHead = tempHead->next;
            temp->next = new Node<T>(tempHead->value, nullptr, temp);
            temp = temp->next;
        }
        this->tail = temp;
    }
}

template<typename T>
inline List<T>& List<T>::operator=(const List& list)
{
    if (this->size != 0)
    {
        this->clear();
    }
    this->tail = nullptr;
    this->head = nullptr;

    this->size = list.size;

    if (list.head != nullptr)
    {
        Node<T>* tempHead = list.head;

        Node<T>* temp = new Node<T>(tempHead->value, tempHead->next);
        this->head = temp;

        while (tempHead->next)
        {
            tempHead = tempHead->next;
            temp->next = new Node<T>(tempHead->value, nullptr, temp);
            temp = temp->next;
        }
        this->tail = temp;
    }
}

```

```

        return *this;
    }

template<typename T>
inline void List<T>::push_back(T value)
{
    if (this->tail == nullptr)
    {
        this->head = this->tail = new Node<T>(value);
    }
    else
    {
        this->tail = new Node<T>(value, nullptr, this->tail);
        this->tail->prev->next = this->tail;
    }
    this->size++;
}

template<typename T>
inline void List<T>::push_front(T value)
{
    if (this->head == nullptr)
    {
        this->head = this->tail = new Node<T>(value);
    }
    else
    {
        this->head = new Node<T>(value, this->head);
        this->head->next->prev = this->head;
    }
    this->size++;
}

template<typename T>
inline int List<T>::getSize()
{
    return this->size;
}

template<typename T>
inline T& List<T>::operator[](int index)
{

```

```

        Node<T>* temp = head;
        for (int i = 0; i < index; i++)
        {
            temp = temp->next;
        }
        return temp->value;
    }

template<typename T>
inline T& List<T>::front()
{
    if (this->head != nullptr)
    {
        return this->head->value;
    }
}

template<typename T>
inline T& List<T>::back()
{
    if (this->tail != nullptr)
    {
        return this->tail->value;
    }
}

template<typename T>
inline void List<T>::pop_back()
{
    if (size != 0)
    {
        Node<T>* temp = tail;
        tail = tail->prev;
        if (tail != nullptr)
        {
            tail->next = nullptr;
        }
        delete temp;
        size--;
    }
}

```



```

template<typename T>
inline void List<T>::pop_front()
{
    if (size != 0)
    {
        Node<T>* temp = head;
        head = head->next;
        if (head != nullptr)
        {
            head->prev = nullptr;
        }
        delete temp;
        size--;
    }
}

template<typename T>
inline void List<T>::clear()
{
    Node<T>* temp = head;
    for (int i = 0; i < size; i++)
    {
        head = head->next;
        delete temp;
        temp = head;
    }
    tail = nullptr;
    head = nullptr;
    size = 0;
}

template<typename T>
inline bool List<T>::isEmpty()
{
    return (size == 0);
}

template<typename T>
inline void List<T>::insert(T value, int index)
{
    if (index == 0)
    {

```

```

        this->push_front(value);
    }
    else if (index == size - 1)
    {
        this->push_back(value);
    }
    else if (index <= size / 2)
    {
        Node<T>* temp = head;
        for (int i = 0; i < index - 1; i++)
        {
            temp = temp->next;
        }
        temp->next = new Node<T>(value, temp->next, temp);
        temp->next->next->prev = temp->next;
        size++;
    }
    else if (index > size / 2)
    {
        Node<T>* temp = tail;
        for (int i = size - 1; i > index; i--)
        {
            temp = temp->prev;
        }
        temp->prev = new Node<T>(value, temp, temp->prev);
        temp->prev->prev->next = temp->prev;
        size++;
    }
}

template<typename T>
inline void List<T>::removeAt(int index)
{
    if (index == 0)
    {
        this->pop_front();
    }
    else if (index == size - 1)
    {
        this->pop_back();
    }
    else if (index <= size / 2)

```

```

{
    Node<T>* temp = head;
    for (int i = 0; i < index - 1; i++)
    {
        temp = temp->next;
    }
    Node<T>* deleteNode = temp->next;
    temp->next = deleteNode->next;
    deleteNode->next->prev = temp;
    delete deleteNode;
    size--;
}
else if (index > size / 2)
{
    Node<T>* temp = tail;
    for (int i = size - 1; i > index + 1; i--)
    {
        temp = temp->prev;
    }
    Node<T>* deleteNode = temp->prev;
    temp->prev = deleteNode->prev;
    deleteNode->prev->next = temp;
    delete deleteNode;
    size--;
}
}

template<typename T>
inline void List<T>::reverse()
{
    Node<T>* temp = tail;
    for (int i = 0; i < size; i++)
    {
        Node<T>* temporary = temp->next;
        temp->next = temp->prev;
        temp->prev = temporary;
        temp = temp->next;
    }
    temp = head;
    head = tail;
    tail = temp;
}

```

11) Vector.h

```
#pragma once
```

```
template <typename T>
class Vector
{
public:
    Vector();
    //constructor
    Vector(const Vector& vector);
    ~Vector();
    //destructor
    T& at(int index); //get
element by index
    T& back();
    //returns reference to the last element
    void clear();
    //delete all
    T* data();
    //return pointer to the first element
    bool isEmpty();
    //is vector has elements
    T& front();
    //returns reference to the first element
    Vector<T>& operator=(const Vector<T>& vector); //operator = overload for
new vector takes another space in memory
    T& operator[](int index); //get element
by index
    void pop_back();
    //delete last element
    void push_back(T value); //add element
to the end
    int getSize();
    //returns size
    int getCapacity();
    //returns capacity
    void emplace(int index); //delete
element by index
    void insert(int index, T value); //add element by
index
```

```

private:
    T* arr;
        //array
    int size;
        //size of array
    int capacity;
        //capacity that vector takes space in memory
    int allocator;
        //number for increasing vector capacity
    void recreateArray();
        //recreating vector
};

template<typename T>
inline Vector<T>::Vector()
{
    arr = nullptr;
    size = 0;
    capacity = 0;
    allocator = 5;
}

template<typename T>
inline Vector<T>::Vector(const Vector& vector)
{
    this->capacity = vector.capacity;
    this->size = vector.size;
    this->allocator = vector.allocator;
    this->arr = new T[this->capacity];
    for (int i = 0; i < this->size; i++)
    {
        this->arr[i] = vector.arr[i];
    }
}

template<typename T>
inline Vector<T>::~~Vector()
{
    clear();
}

template<typename T>

```

```

inline T& Vector<T>::at(int index)
{
    if (arr != nullptr && index < size)
    {
        return arr[index];
    }
}

template<typename T>
inline T& Vector<T>::back()
{
    if (arr != nullptr)
    {
        return arr[size - 1];
    }
}

template<typename T>
inline void Vector<T>::clear()
{
    if (arr != nullptr)
    {
        delete[]arr;
        size = 0;
        capacity = 0;
        allocator = 5;
    }
    arr = nullptr;
}

template<typename T>
inline T* Vector<T>::data()
{
    return arr;
}

template<typename T>
inline bool Vector<T>::isEmpty()
{
    return (size == 0);
}

```

```

template<typename T>
inline T& Vector<T>::front()
{
    if (arr != nullptr)
    {
        return arr[0];
    }
}

template<typename T>
inline Vector<T>& Vector<T>::operator=(const Vector<T>& vector)
{
    if (this->arr != nullptr)
    {
        delete[]this->arr;
    }
    this->arr = new T[vector.capacity];
    for (int i = 0; i < vector.size; i++)
    {
        this->arr[i] = vector.arr[i];
    }
    this->size = vector.size;
    this->capacity = vector.capacity;
    this->allocator = vector.allocator;
    return *this;
}

template<typename T>
inline T& Vector<T>::operator[](int index)
{
    if (arr != nullptr && index < size)
    {
        return arr[index];
    }
}

template<typename T>
inline void Vector<T>::pop_back()
{
    if (arr != nullptr && size > 0)
    {
        size--;
    }
}

```

```

    }
}

template<typename T>
inline void Vector<T>::push_back(T value)
{
    if (arr == nullptr || size == capacity)
    {
        recreateArray();
    }
    arr[size] = value;
    size++;
}

template<typename T>
inline int Vector<T>::getSize()
{
    return this->size;
}

template<typename T>
inline int Vector<T>::getCapacity()
{
    return this->capacity;
}

template<typename T>
inline void Vector<T>::emplace(int index)
{
    if (arr != nullptr && size > 0 && index < size && index >= 0)
    {
        for (int i = index; i < size - 1; i++)
        {
            arr[i] = arr[i + 1];
        }
        size--;
    }
}

template<typename T>
inline void Vector<T>::insert(int index, T value)
{

```



```

if (index >= 0 && index < size)
{
    if (arr == nullptr || size == capacity)
    {
        recreateArray();
    }
    if (index == size)
    {
        push_back(value);
    }
    else
    {
        arr[size] = value;
        for (int i = size; i > index; i--)
        {
            T temp = arr[i];
            arr[i] = arr[i - 1];
            arr[i - 1] = temp;
        }
        size++;
    }
}
}

```

```

template<typename T>
inline void Vector<T>::recreateArray()
{
    capacity += allocator;
    allocator += 2;
    T* newArr = new T[capacity];
    for (int i = 0; i < size; i++)
    {
        newArr[i] = arr[i];
    }
    delete[] arr;
    arr = newArr;
}

```

Исходные файлы:

1) Authorization.cpp

```
#include "Authorization.h"
```

```

Authorization::Authorization()           //default constructor
{
    this->login = nullptr;
    this->hashPassword = nullptr;
    this->salt = nullptr;
    role = -1;
}

Authorization::Authorization(const char* login, const char* password, int role)
    //constructor
{
    copyString(login, this->login);
    makeHashPassword(password);
    this->role = role;
}

Authorization::Authorization(const Authorization& authorization)           //copy
constructor
{
    copyString(authorization.login, this->login);
    copyString(authorization.hashPassword, this->hashPassword);
    copyString(authorization.salt, this->salt);
    this->role = authorization.role;
}

Authorization& Authorization::operator=(const Authorization& authorization)
    //operator = overload
{
    copyString(authorization.login, this->login);
    copyString(authorization.hashPassword, this->hashPassword);
    copyString(authorization.salt, this->salt);
    this->role = authorization.role;
    return *this;
}

Authorization::~Authorization()           //destructor
{
    if (this->login != nullptr)
    {
        delete[]this->login;
    }
    if (this->hashPassword != nullptr)

```

```

        {
            delete[]this->hashPassword;
        }
        if (this->salt != nullptr)
        {
            delete[]this->salt;
        }
    }

    bool Authorization::enter()           //enter to the system
    {
        try
        {
            ifstream in;
            in.exceptions(ifstream::badbit | ifstream::failbit);
            in.open(usersFile);
            if (in.is_open())
            {
                Authorization temp;
                while (!in.eof())
                {
                    bool isRead = temp.read(in);
                    if (isRead == true && equal(this->login, temp.login))
                    {
                        if (equal(this->returnUnhashPassword(),
temp.returnUnhashPassword()))
                        {
                            in.close();
                            return true;
                        }
                    }
                }
                in.close();
                return false;
            }
            else
            {
                throw exception("File could not be open");
            }
        }
        catch (exception& ex)
        {

```

```

        cout << ex.what();
    }
}

bool Authorization::registration(const char* name, const char* surname, int age)
    //user registration
{
    try
    {
        if (this->isExist())
        {
            return false;
        }
        ofstream out;
        out.exceptions(ofstream::failbit | ofstream::badbit);
        out.open(usersFile, ios::app);
        if (out.is_open())
        {
            bool isWrite = this->write(out, name, surname, age);
            out.close();
            return isWrite;
        }
        else
        {
            throw exception("File could not be open");
        }
    }
    catch (exception& ex)
    {
        cout << ex.what();
    }
}

char* Authorization::getLogin()           //login getter
{
    return login;
}

char* Authorization::getHashPassword()    //hash password getter
{
    return hashPassword;
}

```

```

char* Authorization::getSalt()           //salt getter
{
    return this->salt;
}

int Authorization::getRole()             //role getter
{
    return this->role;
}

void Authorization::setLogin(const char* newLogin)           //login setter
{
    copyString(newLogin, this->login);
}

void Authorization::setHashPassword(const char* newHashPassword) //hash password
setter
{
    copyString(newHashPassword, this->hashPassword);
}

void Authorization::setPassword(const char* newPassword)    //password setter
{
    makeHashPassword(newPassword);
}

bool Authorization::isExist()           //is account exist in database
{
    ifstream in;
    in.open(usersFile);
    if (in.is_open())
    {
        string temp;
        while (!in.eof())
        {
            in >> temp;
            if (equal(this->login, temp.c_str()))
            {
                in.close();
                return true;
            }
        }
    }
}

```

```

        }
        getline(in, temp);
    }
    in.close();
    return false;
}
else
{
    return false;
}
}

bool Authorization::write(ostream& os, const char* name, const char* surname, int age)
    //write object to file
{
    os.seekp(0, ios::end);
    if (os.tellp() != 0)
    {
        os << endl;
    }
    else
    {
        this->role = CREATOR;
    }
    char* tempName, * tempSurname;
    copyString(name, tempName);
    copyString(surname, tempSurname);
    os << convertSpaceToUnderline(this->login)
        << " " << defaultNumber
        << " " << this->role
        << " " << convertSpaceToUnderline(this->hashPassword)
        << " " << convertSpaceToUnderline(this->salt)
        << " " << convertSpaceToUnderline(tempName)
        << " " << convertSpaceToUnderline(tempSurname)
        << " " << age;
    return !os.fail();
}

bool Authorization::read(istream& is)           //read object from file
{
    string fileLogin, fileSalt, fileHashPassword, temp;
    is >> fileLogin >> temp >> this->role >> fileHashPassword >> fileSalt;

```

```

        getline(is, temp);
        copyString(fileLogin.c_str(), this->login);
        copyString(fileHashPassword.c_str(), this->hashPassword);
        copyString(fileSalt.c_str(), this->salt);
        this->login = convertUnderlineToSpace(this->login);
        this->hashPassword = convertUnderlineToSpace(this->hashPassword);
        this->salt = convertUnderlineToSpace(this->salt);
        return !is.fail();
    }

    void Authorization::makeHashPassword(const char* password)           //method which make
    hash password
    {
        makeSalt(password);
        this->hashPassword = new char[strlen(password) + 1];
        for (int i = 0; i < strlen(password); i++)
        {
            this->hashPassword[i] = (password[i] + this->salt[i]) / 2;
        }
        this->hashPassword[strlen(password)] = '\0';
    }

    char* Authorization::returnUnhashPassword()                       //method which unhashes password
    {
        char* password = new char[strlen(this->hashPassword) + 1];
        for (int i = 0; i < strlen(this->hashPassword); i++)
        {
            password[i] = 2 * this->hashPassword[i] - this->salt[i];
        }
        password[strlen(this->hashPassword)] = '\0';
        return password;
    }

    void Authorization::makeSalt(const char* password)                 //method that makes salt
    {
        this->salt = new char[strlen(password) + 1];
        srand(time(NULL));
        for (int i = 0; i < strlen(password); i++)
        {
            this->salt[i] = rand() % ('z' - 'a') + 'a';
            if (static_cast<int>(password[i] + salt[i]) % 2 == 1)
            {

```

```

        this->salt[i] += 1;
    }
}
this->salt[strlen(password)] = '\0';
}

```

2) User.cpp

```
#include "User.h"
```

```

User::User()           //default constructor
{
    receipt = new Receipt(identifier);
    this->name = nullptr;
    this->surname = nullptr;
    this->setLogin(nullptr);
    this->setHashPassword(nullptr);
    this->salt = nullptr;
}

```

```

User::User(const char* login)           //constructor
{
    try
    {
        srand(time(NULL));
        receipt = new Receipt(identifier);
        ifstream in(usersFile);
        User temp;
        if (in.is_open())
        {
            while (!in.eof())
            {
                temp.read(in);
                if (temp.identifier == defaultNumber)
                {
                    temp.identifier = 0;
                    for (int i = 0; i < 8; i++)
                    {
                        temp.identifier += static_cast<int>(pow(10, i))
* (rand() % 10);
                    }
                }
                if (equal(temp.getLogin(), login))

```



```

        {
            this->setLogin(temp.getLogin());
            this->setHashPassword(temp.getHashPassword());
            this->setName(temp.getName());
            this->setSurname(temp.getSurname());
            this->age = temp.age;
            copyString(temp.salt, this->salt);
            this->role = temp.role;
            this->identifier = temp.identifier;
        }
        users.push_back(temp);
    }
    in.close();
}
else
{
    throw exception("file did not open");
}
in.open(productsFile);
Product product;
if (in.is_open())
{
    while (!in.eof())
    {
        product.read(in);
        products.push_back(product);
    }
    in.close();
}
else
{
    throw exception("file did not open");
}
in.open(receiptsFile);
Receipt tempReceipt;
if (in.is_open())
{
    bool find = false;
    while (!in.eof())
    {
        tempReceipt.read(in);
        find = false;
    }
}

```

```

        for (List<User>::Iterator it = users.begin(); it !=
users.end() && find == false; ++it)
        {
            if (tempReceipt.identificator == (*it).identifier)
            {
                (*it).receipts.push_back(tempReceipt);
                find = true;
            }
        }
        find = false;
        for (List<User>::Iterator it = users.begin(); it != users.end() &&
find == false; ++it)
        {
            if (equal((*it).getLogin(), this->getLogin()))
            {
                this->receipts = (*it).receipts;
                find = true;
            }
        }
        in.close();
    }
    else
    {
        throw exception("file did not open");
    }
}
catch (exception& ex)
{
    ex.~exception();
}
}

User::User(User& user)                //copy constructor
{
    this->setLogin(user.getLogin());
    this->setHashPassword(user.getHashPassword());
    copyString(user.salt, this->salt);
    this->role = user.role;
    copyString(user.name, this->name);
    copyString(user.surname, this->surname);
    this->identifier = user.identifier;
}

```

```

        this->age = user.age;
        this->receipts = user.receipts;
        this->users = user.users;
        this->products = user.products;
        this->receipt = new Receipt(identifier);
        *this->receipt = *user.receipt;
    }

User& User::operator=(User& user)           //operator = overload
{
    this->setLogin(user.getLogin());
    this->setHashPassword(user.getHashPassword());
    copyString(user.salt, this->salt);
    this->role = user.role;
    copyString(user.name, this->name);
    copyString(user.surname, this->surname);
    this->identifier = user.identifier;
    this->age = user.age;
    this->receipts = user.receipts;
    this->users = user.users;
    this->products = user.products;
    if (this->receipt != nullptr)
    {
        delete this->receipt;
    }
    this->receipt = new Receipt(identifier);
    *this->receipt = *user.receipt;
    return *this;
}

User::~User()           //destructor
{
    delete receipt;
    receipts.clear();
    users.clear();
    products.clear();
    delete[]this->surname;
    delete[]this->name;
}

ostream& User::showProducts(ostream& os)           //show products to os
{

```

```

        for (int i = 0; i < products.getSize(); i++)
        {
            showProduct(products[i], os);
        }
        return os;
    }

void User::sortProductsByName()           //sort product by name
{
    for (int i = 0; i < products.getSize() - 1; i++)
    {
        for (int j = i + 1; j < products.getSize(); j++)
        {
            if (compareString(products[i].getName(), products[j].getName()) < 0)
            {
                swapElements(products[i], products[j]);
            }
        }
    }
}

void User::sortProductsByCountry()       //sort product by country
{
    for (int i = 0; i < products.getSize() - 1; i++)
    {
        for (int j = i + 1; j < products.getSize(); j++)
        {
            if (compareString(products[i].getCountry(),
products[j].getCountry()) < 0)
            {
                swapElements(products[i], products[j]);
            }
        }
    }
}

void User::sortProductsByPrice()         //sort product by price
{
    for (int i = 0; i < products.getSize() - 1; i++)
    {
        for (int j = i + 1; j < products.getSize(); j++)
        {

```

```

        if (products[i].getPrice() > products[j].getPrice())
        {
            swapElements(products[i], products[j]);
        }
    }
}

void User::sortProductsByGroup()           //sort product by group
{
    for (int i = 0; i < products.getSize() - 1; i++)
    {
        for (int j = i + 1; j < products.getSize(); j++)
        {
            if (compareString(products[i].getGroup(), products[j].getGroup()) <
0)
            {
                swapElements(products[i], products[j]);
            }
        }
    }
}

void User::sortProductsByType()           //sort product by type
{
    for (int i = 0; i < products.getSize() - 1; i++)
    {
        for (int j = i + 1; j < products.getSize(); j++)
        {
            if (compareString(products[i].getType(), products[j].getType()) < 0)
            {
                swapElements(products[i], products[j]);
            }
        }
    }
}

void User::sortProductsByBarCode()       //sort product by barcode
{
    for (int i = 0; i < products.getSize() - 1; i++)
    {
        for (int j = i + 1; j < products.getSize(); j++)

```

```

        {
            if (products[i].getBarCode() > products[j].getBarCode())
            {
                swapElements(products[i], products[j]);
            }
        }
    }
}

Product& User::findProduct(const char* name)           //find product
{
    for (int i = 0; i < products.getSize(); i++)
    {
        if (equal(name, products[i].getName()))
        {
            return products[i];
        }
    }
}

char* User::getName()           //name getter
{
    return this->name;
}

char* User::getSurname()        //surname getter
{
    return this->surname;
}

int User::getAge()              //age getter
{
    return this->age;
}

User& User::getFullUserInformation()           //user(this) getter
{
    return *this;
}

int User::getIdentifier()          //identifier getter
{
    return this->identifier;
}

```

```

}

void User::setName(const char* newName)           //name setter
{
    copyString(newName, this->name);
}

void User::setSurname(const char* newSurname)      //surname setter
{
    copyString(newSurname, this->surname);
}

void User::setAge(const int newAge)               //age setter
{
    this->age = newAge;
}

void User::showReceipts(ostream& os)             //show receipts to os
{
    for (List<Receipt>::Iterator it = receipts.begin(); it != receipts.end(); ++it)
    {
        (*it).showReceipt(os);
    }
}

bool User::makeReceipt()                        //make receipt
{
    if (receipt->products.getSize() <= 0)
    {
        return false;
    }
    receipt->setDate();
    receipts.push_back(*receipt);
    bool isFind = false;
    for (List<User>::Iterator it = users.begin(); it != users.end() && isFind ==
false; ++it)
    {
        if (equal((*it).getLogin(), this->getLogin()))
        {
            (*it).receipts = this->receipts;
        }
    }
}

```

```

        delete receipt;
        receipt = new Receipt();
        return true;
    }

    bool User::buyProduct(const char* name)           //buy product and send to receipt
    {
        bool buy = false;
        for (int i = 0; i < products.getSize() && buy == false; i++)
        {
            if (equal(name, products[i].getName()))
            {
                receipt->pushProduct(products[i]);
                buy = true;
            }
        }
        return buy;
    }

    bool User::deleteBuyingProduct(const char* name)   //delete product from receipt
    {
        bool deleteBuying = false;
        for (int i = 0; i < products.getSize() && deleteBuying == false; i++)
        {
            if (equal(name, products[i].getName()))
            {
                receipt->popProduct(name);
                deleteBuying = true;
            }
        }
        return deleteBuying;
    }

    bool User::saveReceiptInformation()               //save receipt information
    {
        if (!this->areUsersWithId())
        {
            this->saveUsersInformation();
        }
        ofstream out;
        out.open(receiptsFile);
        if (out.is_open())
    
```



```

        {
            for (List<User>::Iterator usIt = users.begin(); usIt != users.end();
++usIt)
            {
                for (List<Receipt>::Iterator recIt = (*usIt).receipts.begin(); recIt
!= (*usIt).receipts.end(); ++recIt)
                {
                    (*recIt).makeReceipt((*usIt).getIdentifier());
                }
            }
            out.close();
            return !out.fail();
        }
    else
    {
        throw exception("file did not open");
    }
}

bool User::saveUsersInformation()           //save user information
{
    ofstream out;
    out.open(usersFile);
    if (out.is_open())
    {
        for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
        {
            if (equal((*it).getLogin(), this->getLogin()))
            {
                this->write(out);
            }
            else
            {
                (*it).write(out);
            }
        }
        out.close();
        return !out.fail();
    }
    else
    {
        throw exception("file did not open");
    }
}

```

```

    }
}

Vector<Product> User::getProductsVector()           //products getter
{
    return products;
}

List<Receipt> User::getReceiptsList()               //receipts getter
{
    return receipts;
}

Receipt* User::getReceiptPointer()                  //receipt getter
{
    return receipt;
}

List<User> User::getUsersList()                     //users getter
{
    return users;
}

bool User::write(ostream& os)                       //write user to file
{
    os.seekp(0, ios::end);
    if (os.tellp() != 0)
    {
        os << endl;
    }
    os << convertSpaceToUnderline(this->getLogin())
        << " " << this->identifier
        << " " << this->role
        << " " << convertSpaceToUnderline(this->getHashPassword())
        << " " << convertSpaceToUnderline(this->getSalt())
        << " " << convertSpaceToUnderline(this->name)
        << " " << convertSpaceToUnderline(this->surname)
        << " " << this->age;
    return !os.fail();
}

bool User::read(istream& is)                        //read user from file

```

```

{
    string fileLogin, fileSalt, fileHashPassword, fileName, fileSurname;
    is >> fileLogin >> this->identifier >> this->role >> fileHashPassword
        >> fileSalt >> fileName >> fileSurname >> this->age;
    setLogin(fileLogin.c_str());
    setHashPassword(fileHashPassword.c_str());
    copyString(fileSalt.c_str(), this->salt);
    setLogin(convertUnderlineToSpace(this->getLogin()));
    setHashPassword(convertUnderlineToSpace(this->getHashPassword()));
    this->salt = convertUnderlineToSpace(this->salt);
    copyString(fileName.c_str(), this->name);
    copyString(fileSurname.c_str(), this->surname);
    this->name = convertUnderlineToSpace(this->name);
    this->surname = convertUnderlineToSpace(this->surname);
    return !is.fail();
}

void User::showProduct(Product& product, ostream& os)           //show one product to os
{
    os << "Name: " << product.getName() << endl
        << "Price: " << product.getPrice() << endl
        << "Country: " << product.getCountry() << endl
        << "Group: " << product.getGroup() << endl
        << "Individual ability: " << product.getIndividualAbility() << endl
        << "BarCode: " << product.getBarCode() << endl << endl;
}

bool User::areUsersWithId()           //checks if users get id in file
{
    ifstream in;
    in.open(usersFile);
    string temp;
    if (in.is_open())
    {
        while (!in.eof())
        {
            in >> temp;
            in >> temp;
            if (temp == "-1")
            {
                return false;
            }
        }
    }
}

```

```

        getline(in, temp);
    }
}
return true;
}

```

3) Admin.cpp

```
#include "Admin.h"
```

```

bool Admin::deleteProduct(const char* name) //delete Product
from products
{
    for (int i = 0; i < products.getSize(); i++)
    {
        if (equal(products[i].getName(), name))
        {
            products.erase(i);
            return true;
        }
    }
    return false;
}

bool Admin::addProduct(Product& product) //add Product to products
{
    for (int i = 0; i < products.getSize(); i++)
    {
        if (equal(product.getName(), products[i].getName()))
        {
            return false;
        }
    }
    products.push_back(product);
    return true;
}

bool Admin::editProductCountry(const char* name, const char* newCountry) //edit
product country
{
    for (int i = 0; i < products.getSize(); i++)

```

```

    {
        if (equal(name, products[i].getName()))
        {
            products[i].setCountry(newCountry);
            return true;
        }
    }
    return false;
}

bool Admin::editProductPrice(const char* name, double newPrice) //edit
product price
{
    for (int i = 0; i < products.getSize(); i++)
    {
        if (equal(name, products[i].getName()))
        {
            products[i].setPrice(newPrice);
            return true;
        }
    }
    return false;
}

bool Admin::editProductType(const char* name, const char* newType) //edit
product type
{
    for (int i = 0; i < products.getSize(); i++)
    {
        if (equal(name, products[i].getName()))
        {
            products[i].setType(newType);
            return true;
        }
    }
    return false;
}

bool Admin::editProductIndividualAbility(const char* name, const char*
newIndividualAbility) //edit product individual ability
{
    for (int i = 0; i < products.getSize(); i++)

```

```

    {
        if (equal(name, products[i].getName()))
        {
            products[i].setIndividualAbility(newIndividualAbility);
            return true;
        }
    }
    return false;
}

bool Admin::editProductGroup(const char* name, const char* newGroup) //edit
product group
{
    for (int i = 0; i < products.getSize(); i++)
    {
        if (equal(name, products[i].getName()))
        {
            products[i].setGroup(newGroup);
            return true;
        }
    }
    return false;
}

bool Admin::editProductBarCode(const char* name, int newBarCode) //edit
product barcode
{
    for (int i = 0; i < products.getSize(); i++)
    {
        if (equal(name, products[i].getName()))
        {
            products[i].setBarCode(newBarCode);
            return true;
        }
    }
    return false;
}

bool Admin::editProductName(const char* name, const char* newName) //edit
product name
{
    for (int i = 0; i < products.getSize(); i++)

```

```

    {
        if (equal(newName, products[i].getName()))
        {
            return false;
        }
    }
    for (int i = 0; i < products.getSize(); i++)
    {
        if (equal(name, products[i].getName()))
        {
            products[i].setName(newName);
            return true;
        }
    }
    return false;
}

bool Admin::addAccount(User& user)           //add User to users
{
    for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
    {
        if (equal((*it).getLogin(), user.getLogin()))
        {
            return false;
        }
    }
    users.push_back(user);
    return true;
}

bool Admin::deleteAccount(const char* login) //delete User from users
{
    int counter = 0;
    for (List<User>::Iterator it = users.begin(); it != users.end(); counter++, ++it)
    {
        if (equal(login, (*it).getLogin()))
        {
            if ((*it).getRole() > this->role)
            {
                return false;
            }
            users.removeAt(counter);
        }
    }
}

```

```

        return true;
    }
}
return false;
}

bool Admin::editUserLogin(const char* login, const char* newLogin)           //edit User
login
{
    for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
    {
        if (equal(newLogin, (*it).getLogin()))
        {
            return false;
        }
    }
    for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
    {
        if (equal(login, (*it).getLogin()))
        {
            (*it).setLogin(newLogin);
            return true;
        }
    }
    return false;
}

bool Admin::editUserPassword(const char* login, const char* newPassword)     //edit
user password
{
    for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
    {
        if (equal(login, (*it).getLogin()))
        {
            (*it).setPassword(newPassword);
            return true;
        }
    }
    return false;
}

```



```

bool Admin::editUserName(const char* login, const char* newName)           //edit user
name
{
    for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
    {
        if (equal(login, (*it).getLogin()))
        {
            (*it).setName(newName);
            return true;
        }
    }
    return false;
}

bool Admin::editUserSurname(const char* login, const char* newSurname)     //edit
user surname
{
    for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
    {
        if (equal(login, (*it).getLogin()))
        {
            (*it).setSurname(newSurname);
            return true;
        }
    }
    return false;
}

bool Admin::editUserAge(const char* login, const int newAge)              //edit user age
{
    for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
    {
        if (equal(login, (*it).getLogin()))
        {
            (*it).setAge(newAge);
            return true;
        }
    }
    return false;
}

void Admin::sortUsersByLogin()                                           //sort users by login

```

```

{
    int i = 0;
    for (List<User>::Iterator it1 = users.begin(); i < users.getSize() - 1; i++,
        ++it1)
    {
        for (List<User>::Iterator it2 = it1 + 1; it2 != users.end(); ++it2)
        {
            if (compareString((*it1).getLogin(), (*it2).getLogin()) < 0)
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

void Admin::sortUsersByName()           //sort users by name
{
    int i = 0;
    for (List<User>::Iterator it1 = users.begin(); i < users.getSize() - 1; i++,
        ++it1)
    {
        for (List<User>::Iterator it2 = it1 + 1; it2 != users.end(); ++it2)
        {
            if (compareString((*it1).getName(), (*it2).getName()) < 0)
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

void Admin::sortUsersBySurname()        //sort users by surname
{
    int i = 0;
    for (List<User>::Iterator it1 = users.begin(); i < users.getSize() - 1; i++,
        ++it1)
    {
        for (List<User>::Iterator it2 = it1 + 1; it2 != users.end(); ++it2)
        {
            if (compareString((*it1).getSurname(), (*it2).getSurname()) < 0)
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

```

```

        }
    }
}

void Admin::sortUsersByAge()           //sort users by age
{
    int i = 0;
    for (List<User>::Iterator it1 = users.begin(); i < users.getSize() - 1; i++,
        ++it1)
    {
        for (List<User>::Iterator it2 = it1 + 1; it2 != users.end(); ++it2)
        {
            if ((*it1).getAge() > (*it2).getAge())
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

User& Admin::findUser(const char* login)    //find user in users
{
    for (List<User>::Iterator user = users.begin(); user != users.end(); ++user)
    {
        if (equal(login, (*user).getLogin()))
        {
            return (*user);
        }
    }
}

bool Admin::saveProductsToFile()          //save products to file
{
    ofstream out;
    out.open(productsFile);
    if (out.is_open())
    {
        for (int i = 0; i < products.getSize(); i++)
        {
            products[i].write(out);
        }
    }
}

```

```

        out.close();
        return !out.fail();
    }
    else
    {
        throw exception("file did not open");
    }
}

```

4) Creator.cpp

```

#include "Creator.h"

bool Creator::changeRole(const char* login, const int newRole)    //role setter
{
    for (List<User>::Iterator it = users.begin(); it != users.end(); ++it)
    {
        if (equal(login, (*it).getLogin()))
        {
            if ((*it).role == CREATOR)
            {
                return false;
            }
            (*it).role = newRole;
            return true;
        }
    }
    return false;
}

```

5) Product.cpp

```

#include "Product.h"

Product::Product(const char* name, double price, const char* group,
    const char* type, const char* individualAbility, const char* country, int barCode)
    //constructor
{
    copyString(name, this->name);
    this->price = price;
    copyString(group, this->group);
    copyString(type, this->type);
    copyString(individualAbility, this->individualAbility);
}

```

```

        copyString(country, this->country);
        this->barCode = barCode;
    }

Product::Product()           //default constructor
{
    this->name = nullptr;
    this->group = nullptr;
    this->type = nullptr;
    this->individualAbility = nullptr;
    this->country = nullptr;
}

Product::Product(const Product& product)           //copy constructor
{
    copyString(product.name, this->name);
    this->price = product.price;
    copyString(product.group, this->group);
    copyString(product.type, this->type);
    copyString(product.individualAbility, this->individualAbility);
    copyString(product.country, this->country);
    this->barCode = product.barCode;
}

Product& Product::operator=(const Product& product)           //operator = overload
{
    copyString(product.name, this->name);
    this->price = product.price;
    copyString(product.group, this->group);
    copyString(product.type, this->type);
    copyString(product.individualAbility, this->individualAbility);
    copyString(product.country, this->country);
    this->barCode = product.barCode;
    return *this;
}

Product::~~Product()           //destructor
{
    if (this->name != nullptr)
    {
        delete[]this->name;
    }
}

```

```

    }
    if (this->group != nullptr)
    {
        delete[]this->group;
    }
    if (this->type != nullptr)
    {
        delete[]this->type;
    }
    if (this->individualAbility != nullptr)
    {
        delete[]this->individualAbility;
    }
    if (this->country != nullptr)
    {
        delete[]this->country;
    }
}

char* Product::getName()           //name getter
{
    return this->name;
}

double Product::getPrice()         //price getter
{
    return this->price;
}

char* Product::getGroup()          //group getter
{
    return this->group;
}

char* Product::getIndividualAbility() //individual ability getter
{
    return this->individualAbility;
}

char* Product::getType()           //type getter
{
    return this->type;
}

```

```

}

char* Product::getCountry()           //country getter
{
    return this->country;
}

int Product::getBarCode()             //barCode getter
{
    return this->barCode;
}

void Product::setName(const char* newName)           //name setter
{
    copyString(newName, this->name);
}

void Product::setPrice(const double newPrice)        //price setter
{
    this->price = newPrice;
}

void Product::setGroup(const char* newGroup)         //group setter
{
    copyString(newGroup, this->group);
}

void Product::setType(const char* newType)           //type setter
{
    copyString(newType, this->type);
}

void Product::setIndividualAbility(const char* newIndividualAbility) //individual
ability setter
{
    copyString(newIndividualAbility, this->individualAbility);
}

void Product::setCountry(const char* newCountry)     //country setter
{
    copyString(newCountry, this->country);
}

```

```

void Product::setBarCode(const int newBarCode)           //barCode setter
{
    this->barCode = newBarCode;
}

bool Product::write(ostream& os)                        //write product to file
{
    os.seekp(0, ios::end);
    if (os.tellp() != 0)
    {
        os << endl;
    }
    os << convertSpaceToUnderline(this->name)
        << " " << this->price
        << " " << convertSpaceToUnderline(this->group)
        << " " << convertSpaceToUnderline(this->type)
        << " " << convertSpaceToUnderline(this->individualAbility)
        << " " << convertSpaceToUnderline(this->country)
        << " " << this->barCode;
    return !os.fail();
}

bool Product::read(istream& is)                        //read product from file
{
    string fileName, fileGroup, fileType, fileIndividualAbility, fileCountry;
    is >> fileName >> this->price >> fileGroup >> fileType
        >> fileIndividualAbility >> fileCountry >> this->barCode;
    copyString(fileName.c_str(), this->name);
    copyString(fileGroup.c_str(), this->group);
    copyString(fileType.c_str(), this->type);
    copyString(fileIndividualAbility.c_str(), this->individualAbility);
    copyString(fileCountry.c_str(), this->country);
    this->name = convertUnderlineToSpace(this->name);
    this->group = convertUnderlineToSpace(this->group);
    this->type = convertUnderlineToSpace(this->type);
    this->individualAbility = convertUnderlineToSpace(this->individualAbility);
    this->country = convertUnderlineToSpace(this->country);
    return !is.fail();
}

```

6) Receipt.cpp


```
#include "Receipt.h"
```

```
Receipt::Receipt(int identificator, int year, int month, int day, int hour, int minute,  
int second)           //constructor
```

```
{  
    this->identificator = identificator;  
    this->numberOfProducts = 0;  
    this->summaryPrice = 0.0;  
    this->year = year;  
    this->month = month;  
    this->day = day;  
    this->hour = hour;  
    this->minute = minute;  
    this->second = second;  
}
```

```
Receipt::Receipt(const Receipt& receipt)           //copy constructor
```

```
{  
    this->identificator = receipt.identificator;  
    this->summaryPrice = receipt.summaryPrice;  
    this->numberOfProducts = receipt.numberOfProducts;  
    this->year = receipt.year;  
    this->month = receipt.month;  
    this->day = receipt.day;  
    this->hour = receipt.hour;  
    this->minute = receipt.minute;  
    this->second = receipt.second;  
    this->products = receipt.products;  
}
```

```
Receipt& Receipt::operator=(const Receipt& receipt)           //operator = overload
```

```
{  
    this->identificator = receipt.identificator;  
    this->summaryPrice = receipt.summaryPrice;  
    this->numberOfProducts = receipt.numberOfProducts;  
    this->year = receipt.year;  
    this->month = receipt.month;  
    this->day = receipt.day;  
    this->hour = receipt.hour;  
    this->minute = receipt.minute;  
    this->second = receipt.second;
```

```

        this->products = receipt.products;
        return *this;
    }

    Receipt::~Receipt()           //destructor
    {
        products.clear();
    }

    ostream& Receipt::showReceipt(ostream& os)           //show receipts to os
    {
        for (Product product : products)
        {
            os << "Name: " << product.getName() << endl
                << "Price: " << product.getPrice() << endl
                << "Country: " << product.getCountry() << endl
                << "Group: " << product.getGroup() << endl
                << "Individual ability: " << product.getIndividualAbility() << endl
                << "BarCode: " << product.getBarCode() << endl << endl;
        }
        this->setDate();
        os << this->hour << ":" << this->minute << ":" << this->second << " "
            << this->day << "." << this->month << "." << this->year << endl << endl;
        return os;
    }

    bool Receipt::makeReceipt(int identificator)           //make a receipt
    {
        if (products.getSize() > 0)
        {
            this->identificator = identificator;
            ofstream out(receiptsFile, ios::app);
            if (out.is_open())
            {
                this->write(out);
                return true;
            }
        }
        return false;
    }

    void Receipt::sortByName()           //sort products by name

```

```

{
    int i = 0;
    for (List<Product>::Iterator it1 = products.begin(); i < products.getSize() - 1;
i++, ++it1)
    {
        for (List<Product>::Iterator it2 = it1 + 1; it2 != products.end(); ++it2)
        {
            if (compareString((*it1).getName(), (*it2).getName()) < 0)
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

void Receipt::sortByCountry()           //sort products by country
{
    int i = 0;
    for (List<Product>::Iterator it1 = products.begin(); i < products.getSize() - 1;
i++, ++it1)
    {
        for (List<Product>::Iterator it2 = it1 + 1; it2 != products.end(); ++it2)
        {
            if (compareString((*it1).getCountry(), (*it2).getCountry()) < 0)
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

void Receipt::sortByPrice()           //sort products by price
{
    int i = 0;
    for (List<Product>::Iterator it1 = products.begin(); i < products.getSize() - 1;
i++, ++it1)
    {
        for (List<Product>::Iterator it2 = it1 + 1; it2 != products.end(); ++it2)
        {
            if ((*it1).getPrice() > (*it2).getPrice())
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

```

```

        }
    }
}

void Receipt::sortByGroup()           //sort products by group
{
    int i = 0;
    for (List<Product>::Iterator it1 = products.begin(); i < products.getSize() - 1;
i++, ++it1)
    {
        for (List<Product>::Iterator it2 = it1 + 1; it2 != products.end(); ++it2)
        {
            if (compareString((*it1).getGroup(), (*it2).getGroup()) < 0)
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

void Receipt::sortByType()           //sort products by type
{
    int i = 0;
    for (List<Product>::Iterator it1 = products.begin(); i < products.getSize() - 1;
i++, ++it1)
    {
        for (List<Product>::Iterator it2 = it1 + 1; it2 != products.end(); ++it2)
        {
            if (compareString((*it1).getType(), (*it2).getType()) < 0)
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

void Receipt::sortByBarCode()        //sort products by barcode
{
    int i = 0;
    for (List<Product>::Iterator it1 = products.begin(); i < products.getSize() - 1;
i++, ++it1)

```

```

    {
        for (List<Product>::Iterator it2 = it1 + 1; it2 != products.end(); ++it2)
        {
            if ((*it1).getBarCode() > (*it2).getBarCode())
            {
                swapElements(*it1, *it2);
            }
        }
    }
}

Product& Receipt::findProduct(const char* findName)           //find product
{
    for (List<Product>::Iterator it = products.begin(); it != products.end(); ++it)
    {
        if (compareString((*it).getName(), findName) == EQUAL)
        {
            return (*it);
        }
    }
}

void Receipt::pushProduct(const Product& product)             //push product to products
{
    products.push_back(product);
}

void Receipt::popProduct(const char* productName)             //pop product from products
{
    int counter = 0;
    for (List<Product>::Iterator it = products.begin(); it != products.end();
counter++, ++it)
    {
        if (equal((*it).getName(), productName))
        {
            products.removeAt(counter);
            break;
        }
    }
}

double Receipt::getSummaryPrice()                             //get summary price

```

```

{
    summaryPrice = 0.0;
    for (Product product : products)
    {
        this->summaryPrice += product.getPrice();
    }
    return this->summaryPrice;
}

int Receipt::getNumberOfProducts()           //number of products getter
{
    return numberOfProducts;
}

List<Product> Receipt::getProductsList()      //product list getter
{
    return products;
}

bool Receipt::read(istream& is)              //read receipt from file
{
    int numberOfProducts = 0;
    is >> this->identificator
        >> numberOfProducts
        >> this->summaryPrice
        >> this->year
        >> this->month
        >> this->day
        >> this->hour
        >> this->minute
        >> this->second;

    Product temp;
    products.clear();
    for (int i = 0; i < numberOfProducts; i++)
    {
        bool isRead = temp.read(is);
        if (isRead == true)
        {
            products.push_back(temp);
        }
    }
    return !is.fail();
}

```

```

}

bool Receipt::write(ostream& os)           //write receipt to file
{
    os.seekp(0, ios::end);
    if (os.tellp() != 0)
    {
        os << endl;
    }
    os << this->identificator
        << " " << this->products.getSize()
        << " " << this->getSummaryPrice()
        << " " << this->year
        << " " << this->month
        << " " << this->day
        << " " << this->hour
        << " " << this->minute
        << " " << this->second;
    for (List<Product>::Iterator it = products.begin(); it != products.end(); ++it)
    {
        (*it).write(os);
    }
    return !os.fail();
}

```

7) Date.cpp

```

#include "Date.h"

Date::Date()           //default constructor
{
    struct tm new_time;
    time_t now = time(NULL);
    localtime_s(&new_time, &now);

    this->year = new_time.tm_year % 100 + 2000;
    this->month = new_time.tm_mon + 1;
    this->day = new_time.tm_mday;
    this->hour = new_time.tm_hour;
    this->minute = new_time.tm_min;
    this->second = new_time.tm_sec;
}

```

```

void Date::setDate()           //set date which is now
{
    struct tm new_time;
    time_t now = time(NULL);
    localtime_s(&new_time, &now);

    this->year = new_time.tm_year % 100 + 2000;
    this->month = new_time.tm_mon + 1;
    this->day = new_time.tm_mday;
    this->hour = new_time.tm_hour;
    this->minute = new_time.tm_min;
    this->second = new_time.tm_sec;
}

int Date::getYear()           //year getter
{
    return this->year;
}

int Date::getMonth()          //month getter
{
    return this->month;
}

int Date::getDay()            //day getter
{
    return this->day;
}

int Date::getHour()           //hour getter
{
    return this->hour;
}

int Date::getMinute()         //minute getter
{
    return this->minute;
}

int Date::getSecond()         //second getter
{
    return this->second;
}

```



```
}
```

8) File.cpp

```
#include "Files.h"

const char* usersFile = "usersFile.txt";           //file that store users
information
const char* productsFile = "productsFile.txt";     //file that store products
information
const char* receiptsFile = "receiptsFile.txt";     //file that store receipts
information
```

9) FriendlyFunctions.cpp

```
#include "FriendFunctions.h"

const char* undefined = "undefined";               //default constant
const int defaultNumber = -1;                      //default constant

//tips
const char* lettersTip = "The field can contain:\nletters from a to z;\nnuppercase letters
from A to Z;\nhyphen '-'.\n";
const char* numbersTip = "The field can contains:\nnnumbers from 0 to 9.\n";
const char* lettersAndNumbersTip = "The field can contain:\nletters from a to
z;\nnuppercase letters from A to Z;\nhyphen '-';\nnnumbers from 0 to 9.\n";
const char* lettersAndNumbersAndSpaceTip = "The field can contain:\nletters from a to
z;\nnuppercase letters from A to Z;\nhyphen '-';\nwhitespace ' ';\nnnumbers from 0 to
9.\n";
const char* doubleNumbersTip = "The field can contains:\nnnumbers from 0 to 9;\nfloating
point ','.\n";
const char* lettersAndSpaceTip = "The field can contain:\nletters from a to z;\nnuppercase
letters from A to Z;\nhyphen '-';\nwhitespace ' '.\n";

void copyString(const char* str1, char*& str2)      //copy string 1 to string 2
{
    try
    {
        if (str1 == nullptr)
        {
            str2 = nullptr;
        }
        else
    }
```

```

        {
            str2 = new char[strlen(str1) + 1];
            for (int i = 0; i < strlen(str1); i++)
            {
                str2[i] = str1[i];
            }
            str2[strlen(str1)] = '\0';
        }
    }
    catch (std::exception& ex)
    {
        std::cout << ex.what();
    }
}

bool equal(const char* str1, const char* str2)           //equal strings
{
    try
    {
        if (str1 == nullptr || str2 == nullptr)
        {
            return false;
        }
        if (strlen(str1) != strlen(str2))
        {
            return false;
        }
        for (int i = 0; i < strlen(str1); i++)
        {
            if (str1[i] != str2[i])
            {
                return false;
            }
        }
        return true;
    }
    catch (std::exception& ex)
    {
        std::cout << ex.what();
        return false;
    }
}

```

```

int compareString(const char* str1, const char* str2)           //equal strings
{
    for (int i = 0; str1[i] != '\0' && str2[i] != '\0'; i++)
    {
        if (str1[i] < str2[i])
        {
            return MORE;
        }
        else if (str1[i] > str2[i])
        {
            return LESS;
        }
    }
    if (strlen(str1) < strlen(str2))
    {
        return MORE;
    }
    if (strlen(str1) > strlen(str2))
    {
        return LESS;
    }
    return EQUAL;
}

char* convertSpaceToUnderline(char* str)                       //method which make from ' ' -> '_'
{
    for (int i = 0; i < strlen(str); i++)
    {
        if (str[i] == ' ')
        {
            str[i] = '_';
        }
    }
    return str;
}

char* convertUnderlineToSpace(char* str)                       //method which make from '_' -> ' '
{
    for (int i = 0; i < strlen(str); i++)
    {
        if (str[i] == '_')

```

```

        {
            str[i] = ' ';
        }
    }
    return str;
}

void MarshalString(String^ s, std::string& outputstring)           //method that
convert String^ to string
{
    const char* kPtoC = (const char*)(Marshal::StringToHGlobalAnsi(s)).ToPointer();
    outputstring = kPtoC;
    Marshal::FreeHGlobal(IntPtr((void*)kPtoC));
}

bool isCorrectNumberInput(System::Windows::Forms::KeyPressEventArgs^ e) {
    //method for input
    if ((e->KeyChar < '0' || e->KeyChar > '9') && e->KeyChar != 8)
    {
        return true;
    }
    return false;
}

bool isCorrectLettersInput(System::Windows::Forms::KeyPressEventArgs^ e) {
    //method for input
    if ((e->KeyChar < 'a' || e->KeyChar > 'z') &&
        (e->KeyChar < 'A' || e->KeyChar > 'Z') &&
        e->KeyChar != 8 &&
        e->KeyChar != '-')
    {
        return true;
    }
    return false;
}

bool isCorrectLettersAndNumbersInput(System::Windows::Forms::KeyPressEventArgs^ e) {
    //method for input
    if ((e->KeyChar < '0' || e->KeyChar > '9') &&
        (e->KeyChar < 'a' || e->KeyChar > 'z') &&
        (e->KeyChar < 'A' || e->KeyChar > 'Z') &&
        e->KeyChar != 8 &&
        e->KeyChar != '-')

```

```

        {
            return true;
        }
        return false;
    }
}

bool isCorrectLettersAndNumbersAndSpaceInput(System::Windows::Forms::KeyPressEventArgs^
e) {
    //method for input
    if ((e->KeyChar < '0' || e->KeyChar > '9') &&
        (e->KeyChar < 'a' || e->KeyChar > 'z') &&
        (e->KeyChar < 'A' || e->KeyChar > 'Z') &&
        e->KeyChar != 8 &&
        e->KeyChar != '-' &&
        e->KeyChar != ' ' &&
        e->KeyChar != ',')
    {
        return true;
    }
    return false;
}

bool isCorrectDoubleNumberInput(System::Windows::Forms::KeyPressEventArgs^ e) {
    //method for input
    if ((e->KeyChar < '0' || e->KeyChar > '9') && e->KeyChar != 8 && e->KeyChar !=
',')
    {
        return true;
    }
    return false;
}

bool isCorrectLettersAndSpaceInput(System::Windows::Forms::KeyPressEventArgs^ e)
    //method for input
{
    if ((e->KeyChar < 'a' || e->KeyChar > 'z') &&
        (e->KeyChar < 'A' || e->KeyChar > 'Z') &&
        e->KeyChar != 8 &&
        e->KeyChar != '-' &&
        e->KeyChar != ' ')
    {
        return true;
    }
    return false;
}
}

```

ПРИЛОЖЕНИЕ Б – ДИАГРАММА КЛАССОВ

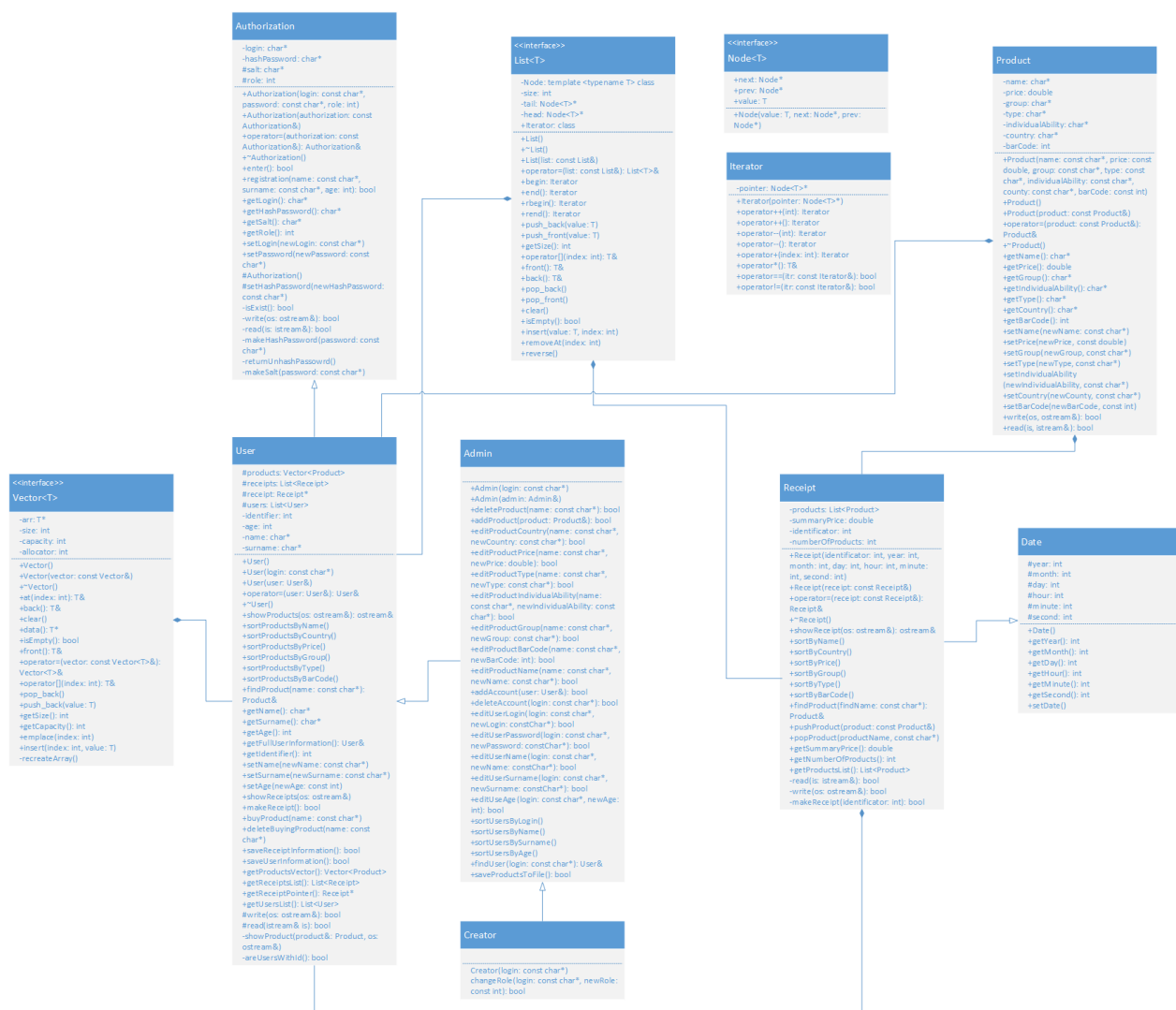


Рисунок Б.1 – Диаграмма классов

ПРИЛОЖЕНИЕ В – МОДУЛЬНАЯ СТРУКТУРА ПРОГРАММЫ

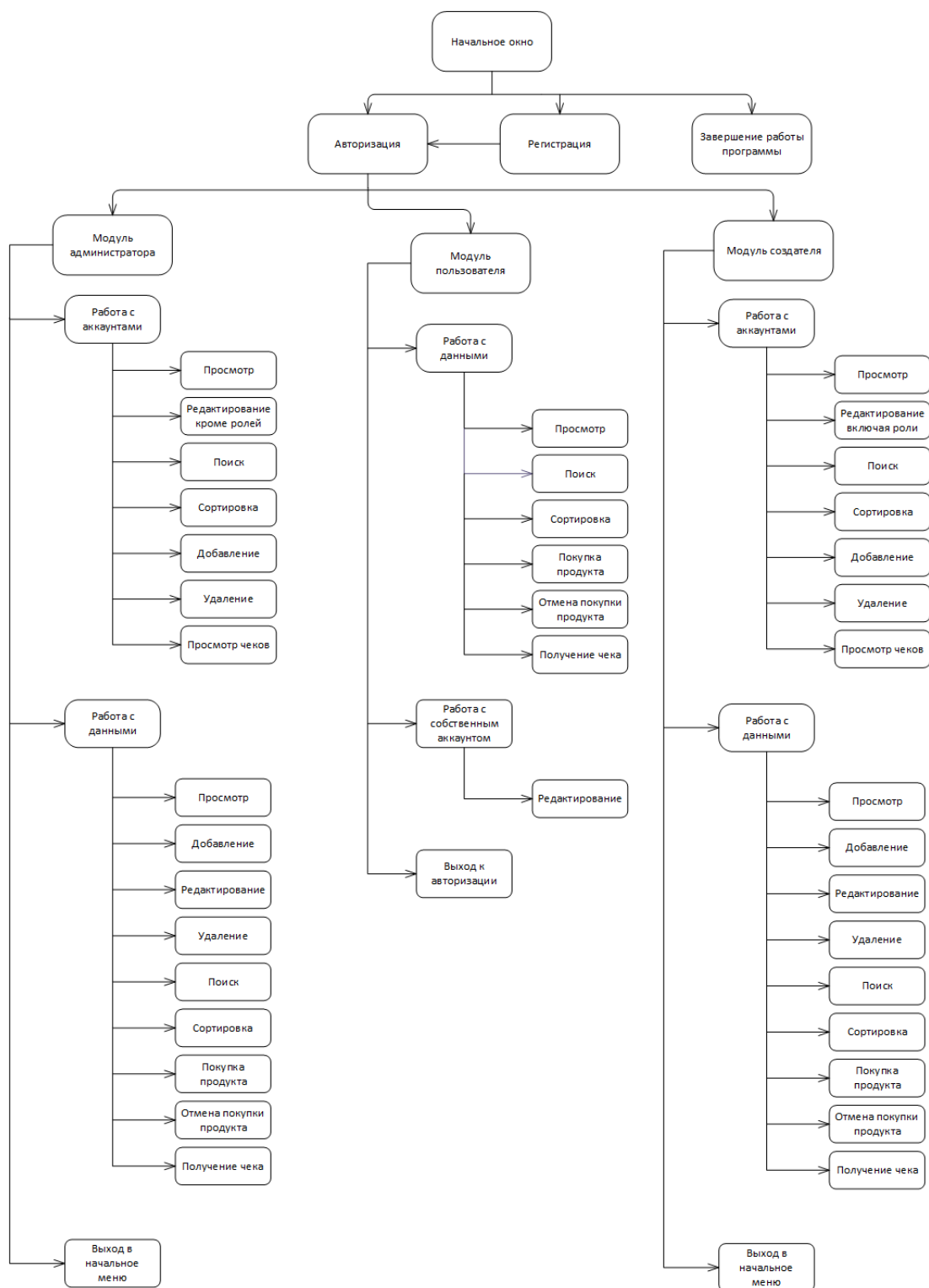


Рисунок В.1 – Модульная структура программы

ПРИЛОЖЕНИЕ Г – АЛГОРИТМЫ ФУНКЦИЙ

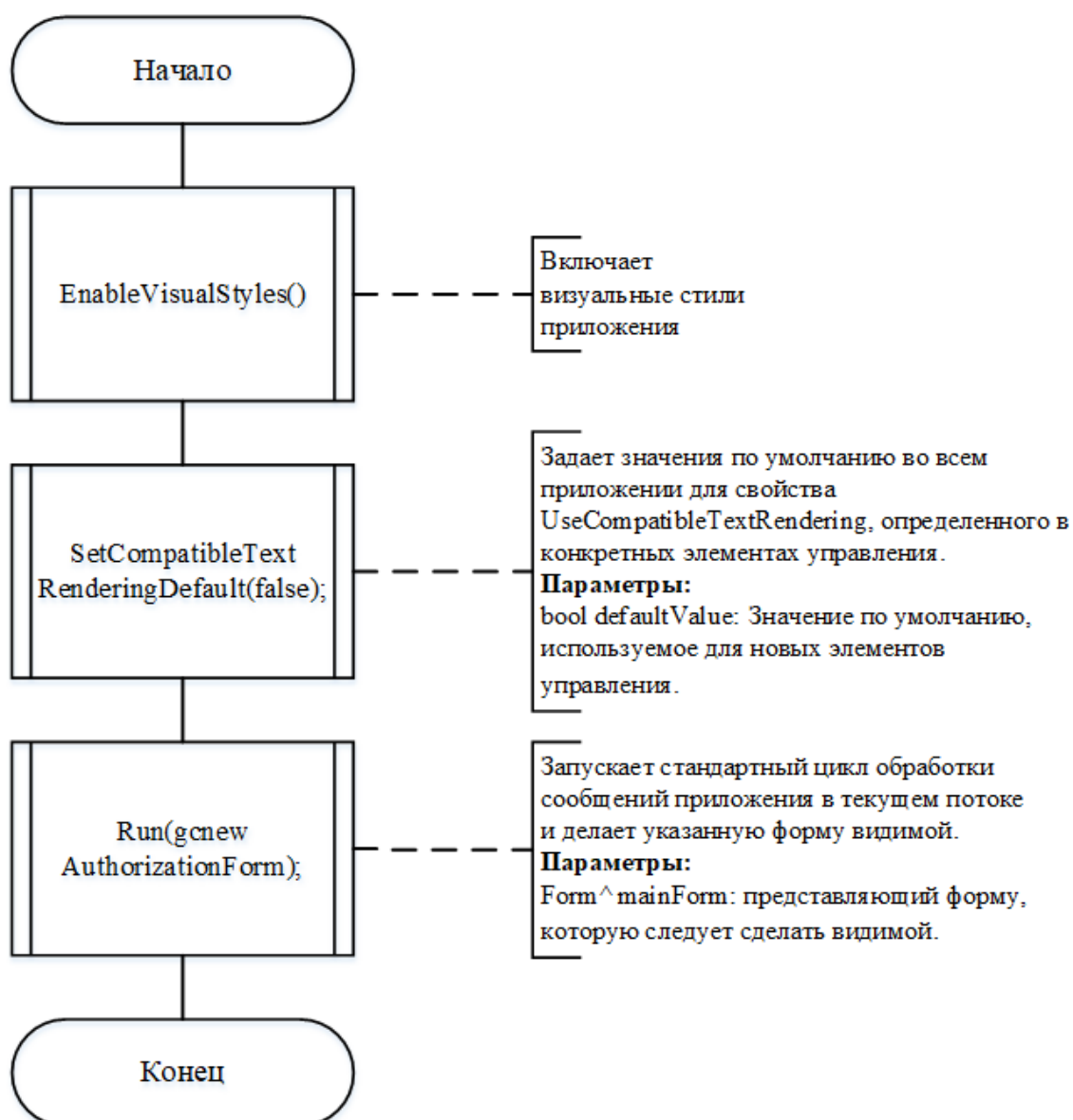


Рисунок Г.1 – Функция WinMain

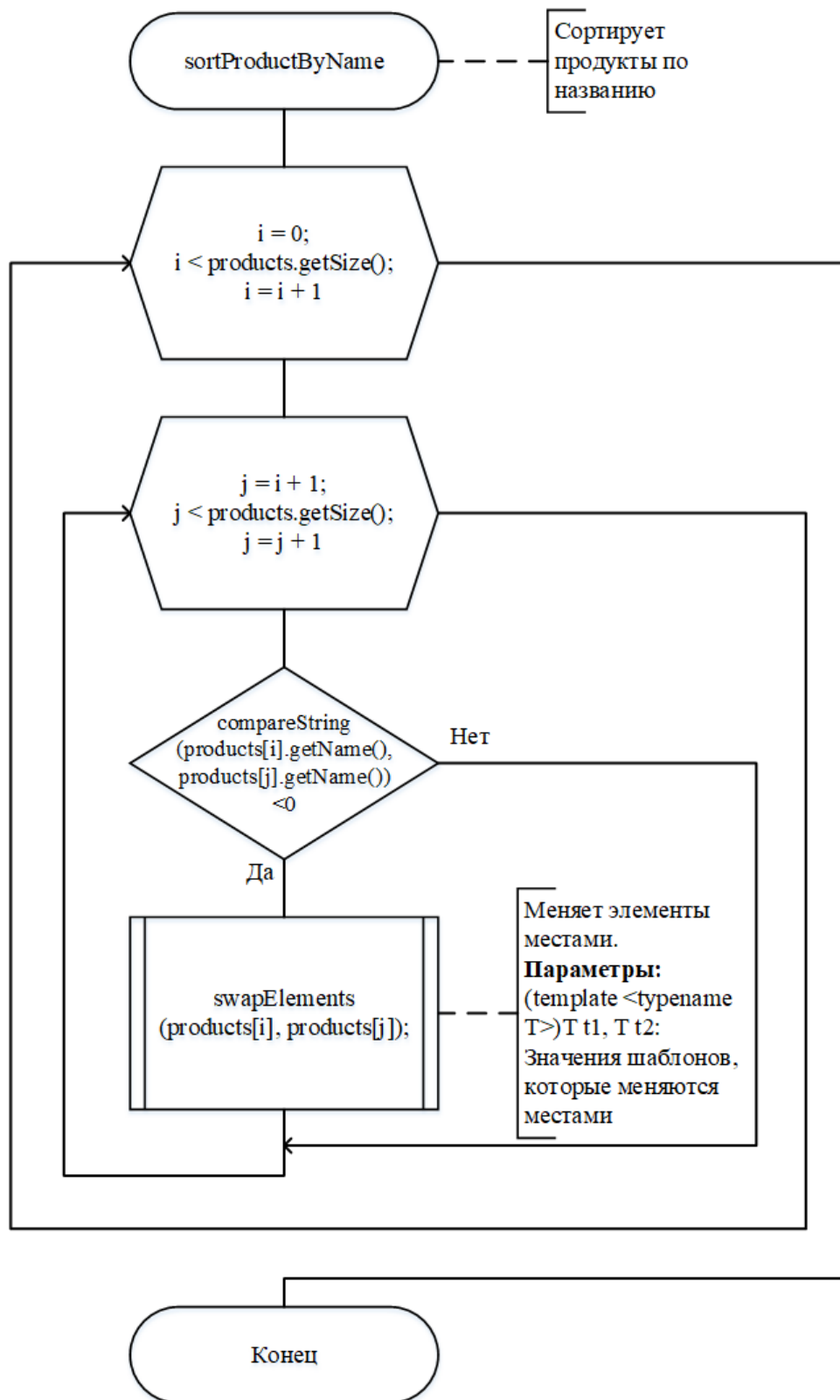


Рисунок Г.2 – Функция sortProductsByName

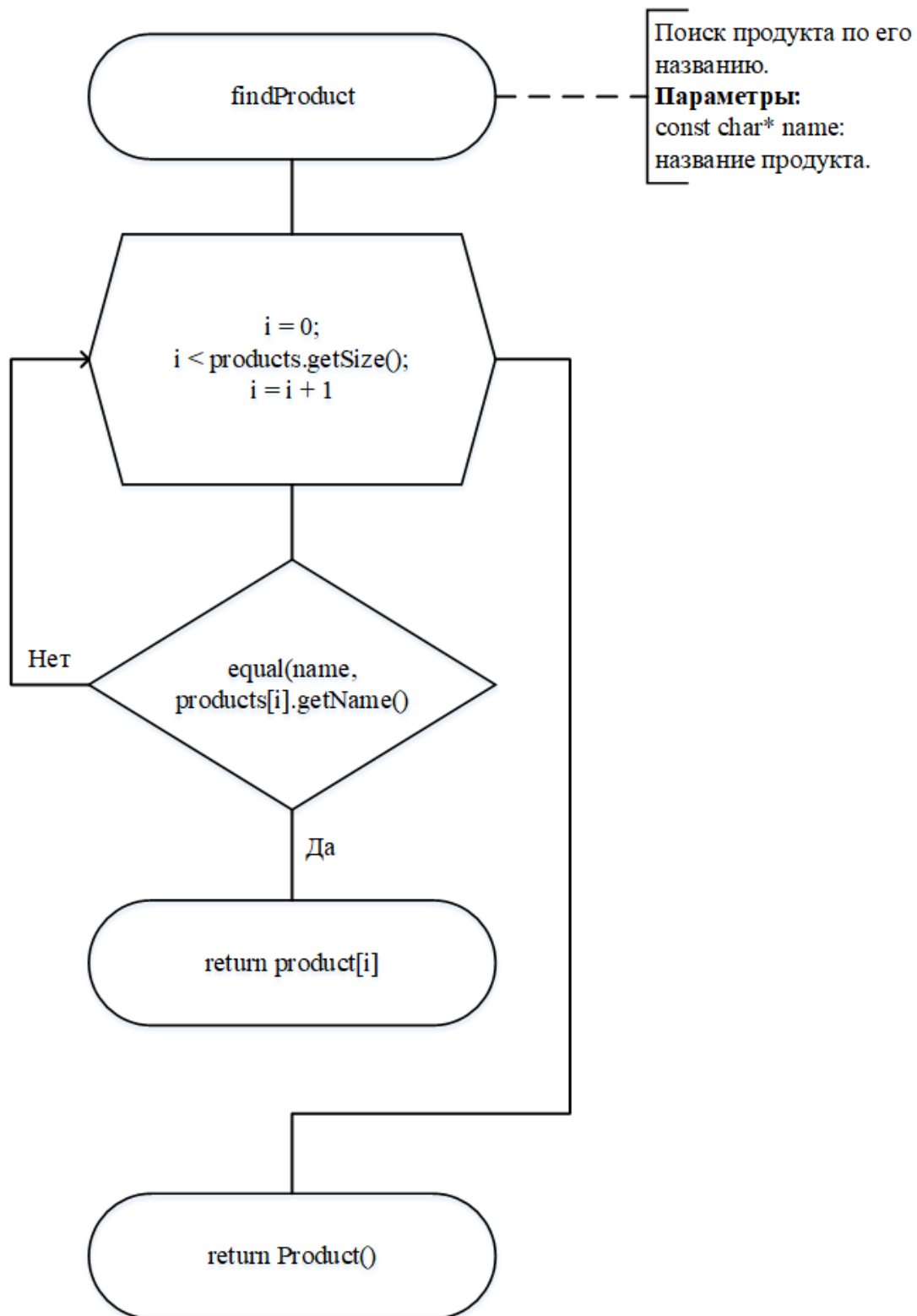


Рисунок Г.3 – Функция findProduct

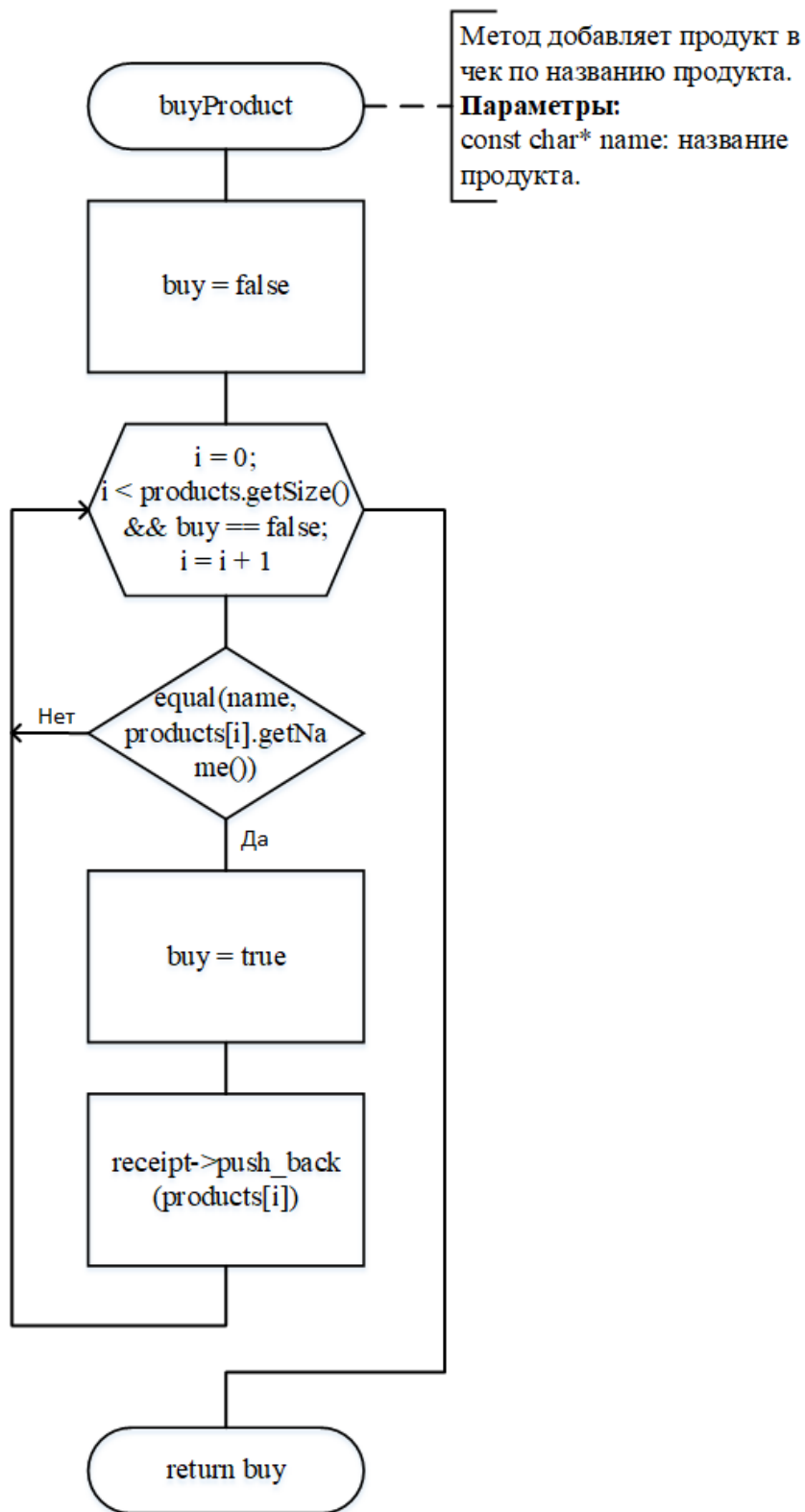


Рисунок Г.4 – Функция buyProduct

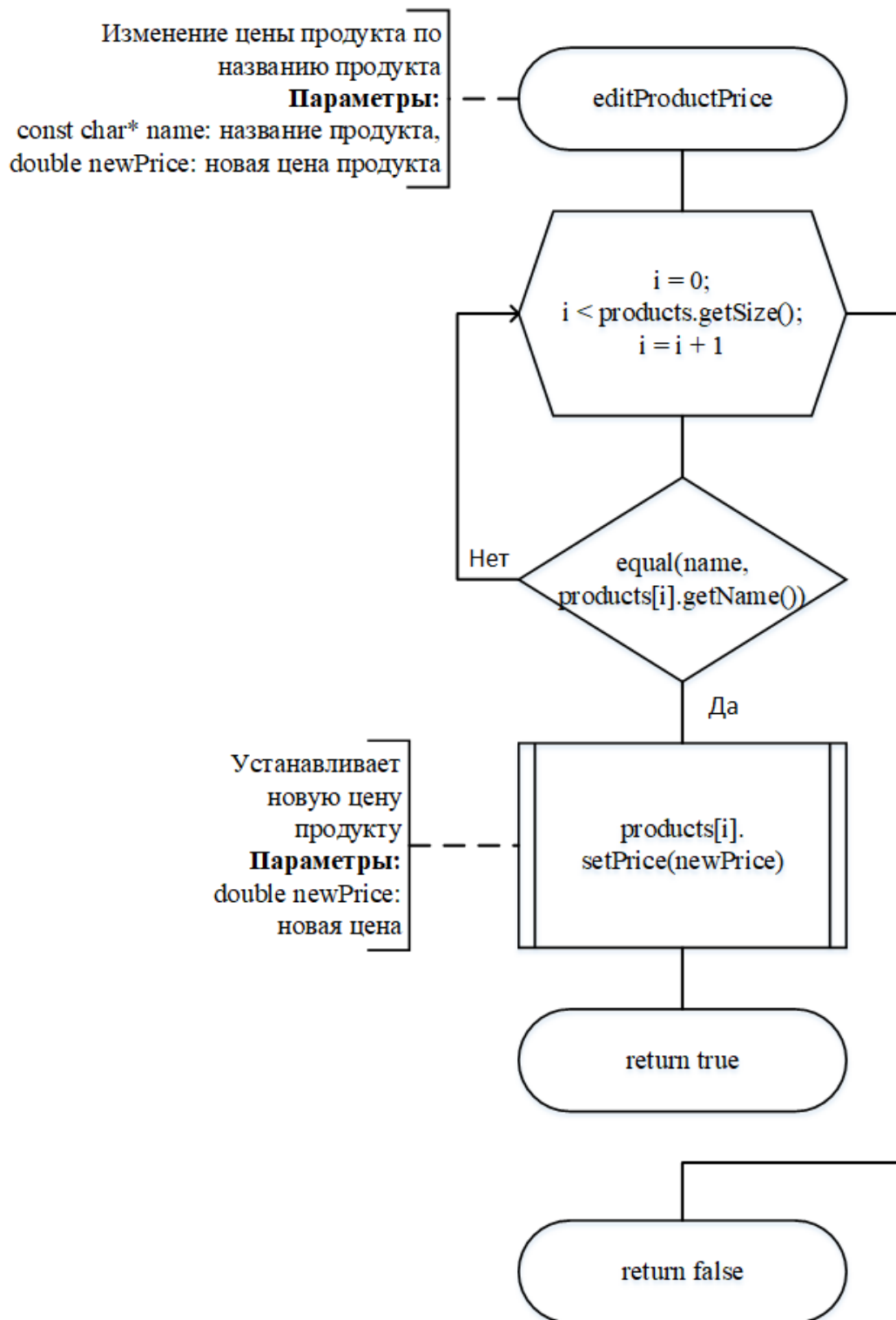


Рисунок Г.5 – Функция editProductPrice

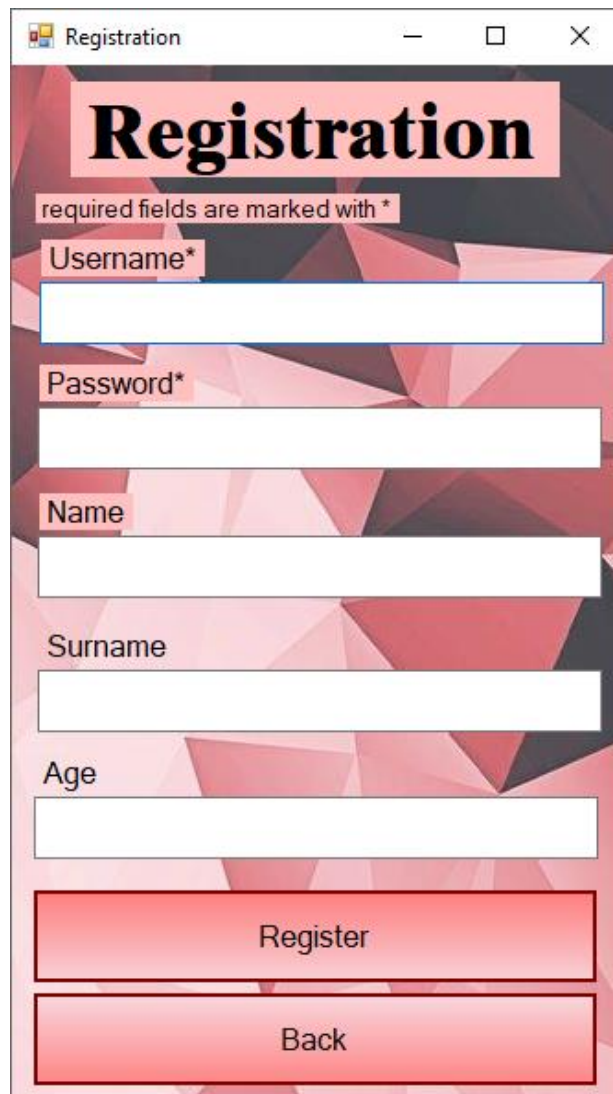
ПРИЛОЖЕНИЕ Д – ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ



The image shows a screenshot of a software application window titled "Application". The window contains a login form with a red and white geometric background. The form has the following elements:

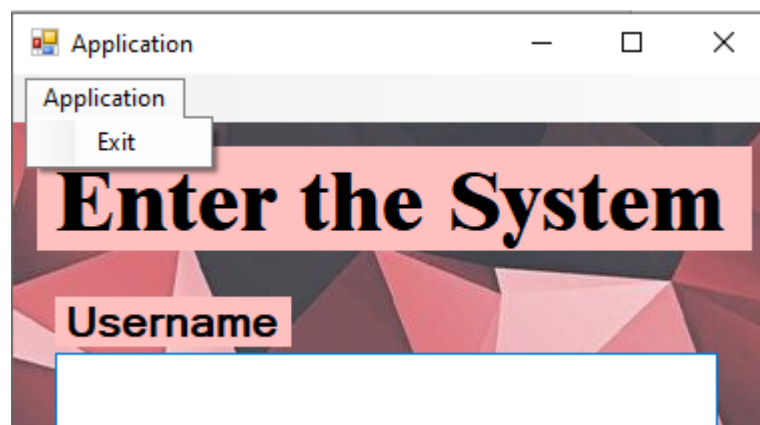
- A title "Enter the System" in a large, bold, black serif font, centered at the top.
- A label "Username" in a bold, black sans-serif font, positioned above a white text input field.
- A label "Password" in a bold, black sans-serif font, positioned above a white text input field.
- A "Log in" button with a red gradient and a black border, located below the password field.
- A "Register" button with a red gradient and a black border, located below the "Log in" button.

Рисунок Д.1 – Окно авторизации



A screenshot of a Windows application window titled "Registration". The window has a red and black geometric background. At the top, the title "Registration" is displayed in a large, bold, black serif font. Below the title, a small text label reads "required fields are marked with *". The form contains five input fields: "Username*" (with an asterisk), "Password*" (with an asterisk), "Name", "Surname", and "Age". Each field is a white rectangle with a thin blue border. At the bottom of the form are two red buttons with black text: "Register" and "Back". The window's title bar includes standard Windows window controls (minimize, maximize, close).

Рисунок Д.2 – Окно регистрации



A screenshot of a Windows application window titled "Application". The window has a red and black geometric background. At the top, the title "Enter the System" is displayed in a large, bold, black serif font. Below the title, there is a label "Username" in a black serif font, followed by a white input field with a blue border. In the top-left corner of the window, there is a menu bar with two items: "Application" and "Exit". The "Exit" item is currently selected, and a small white dropdown menu is visible next to it. The window's title bar includes standard Windows window controls (minimize, maximize, close).

Рисунок Д.3 – Выход из системы

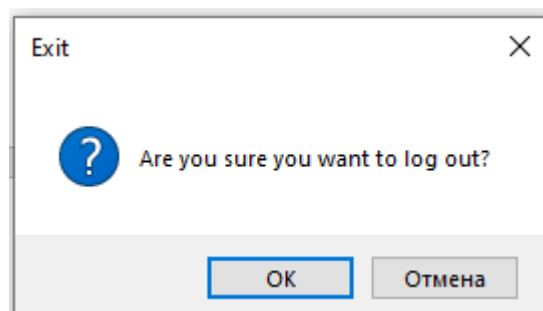


Рисунок Д.4 – Подтверждение выхода

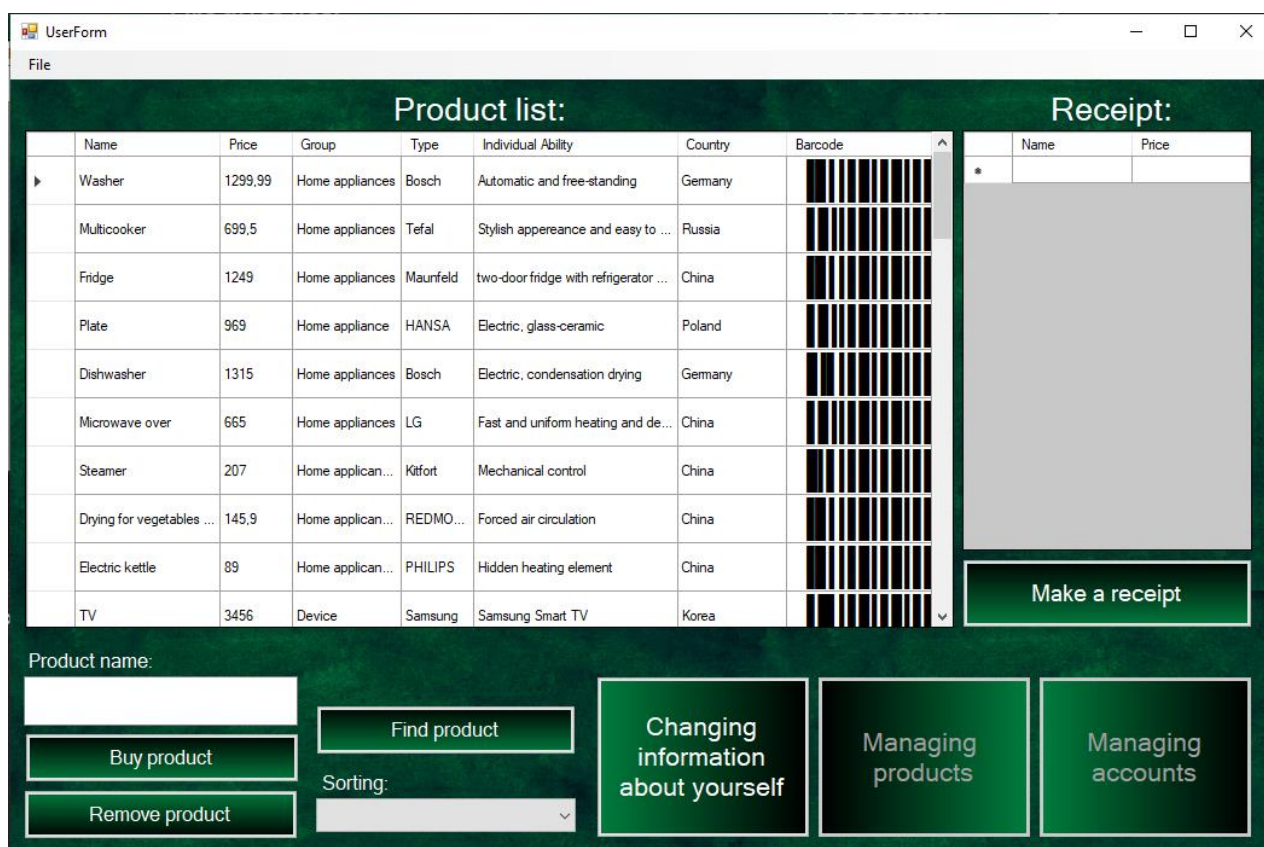


Рисунок Д.5 – Пользовательское окно

The screenshot shows a software window titled "UserForm" with a menu bar containing "File". The main area is divided into two sections: "Product list:" and "Receipt:".

Product list:

	Name	Price	Group	Type	Individual Ability	Country	Barcode
▶	Washer	1299,99	Home appliances	Bosch	Automatic and free-standing	Germany	[Barcode]
	Multicooker	699,5	Home appliances	Tefal	Stylish appereance and easy to ...	Russia	[Barcode]
	Fridge	1249	Home appliances	Maunfeld	two-door fridge with refrigerator ...	China	[Barcode]
	Plate	969	Home appliance	HANSA	Electric, glass-ceramic	Poland	[Barcode]
	Dishwasher	1315	Home appliances	Bosch	Electric, condensation drying	Germany	[Barcode]
	Microwave over	665	Home appliances	LG	Fast and uniform heating and de...	China	[Barcode]
	Steamer	207	Home applican...	Kitfort	Mechanical control	China	[Barcode]
	Drying for vegetables ...	145,9	Home applican...	REDMO...	Forced air circulation	China	[Barcode]
	Electric kettle	89	Home applican...	PHILIPS	Hidden heating element	China	[Barcode]
	TV	3456	Device	Samsung	Samsung Smart TV	Korea	[Barcode]

Receipt:

	Name	Price
▶	Washer	1299,99
*		

Below the receipt table is a large grey rectangular area and a green button labeled "Make a receipt".

At the bottom of the window, there is a "Product name:" input field containing "Washer". To its right is a "Find product" button. Below the input field are "Buy product" and "Remove product" buttons. To the right of these is a "Sorting:" dropdown menu. Further right are four large green buttons: "Changing information about yourself", "Managing products", and "Managing accounts".

Рисунок Д.6 – Покупка продукта

The dialog box has a title bar with "Not found" and a close button (X). The main text reads "Product Bath doesn't exist!". At the bottom is an "OK" button.

Рисунок Д.7 – Отсутствие продукта

The dialog box has a title bar with "Not found" and a close button (X). The main text reads "Product Drying for vegetables and fruits was not purchased by you!". At the bottom is an "OK" button.

Рисунок Д.8 – Продукт не был куплен

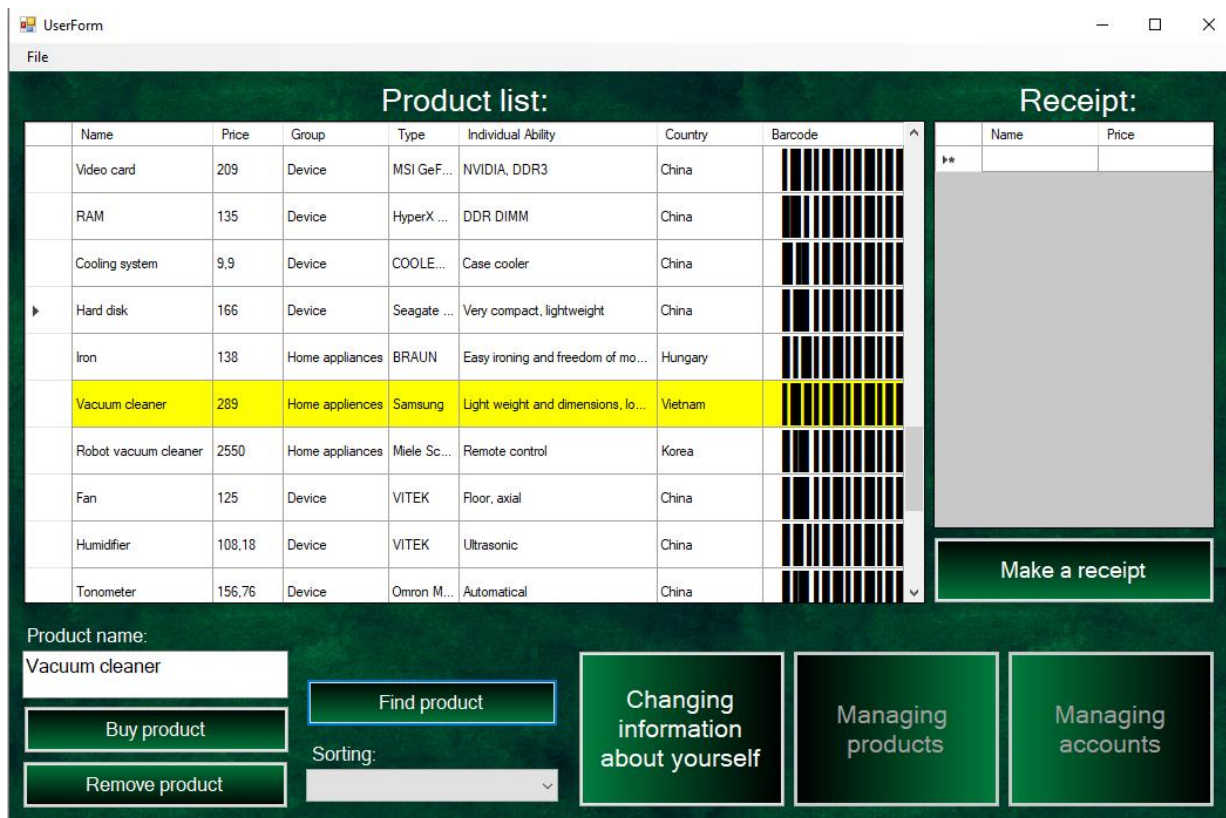


Рисунок Д.9 – Поиск продукта

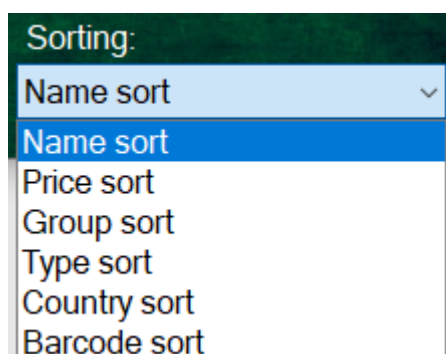


Рисунок Д.10 – Выпадающий список «Sorting»

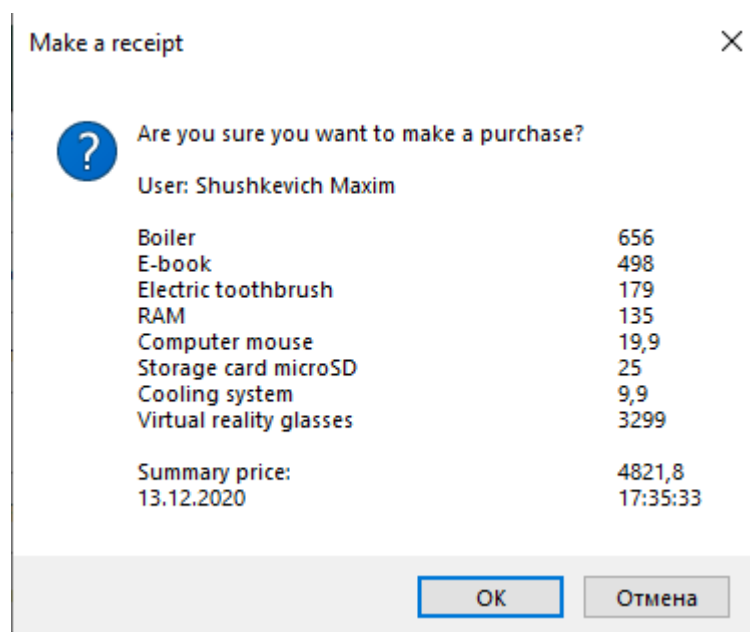


Рисунок Д.11 – Подтверждение покупки

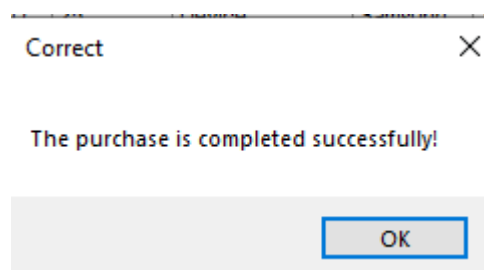


Рисунок Д.12 – Покупка успешна

OwnInformationForm

Own Information

Your login:

Your password:

Your name:

Your surname:

Your age:

Save input information about yourself

Cancel

Рисунок Д.13 – Окно с собственной информацией

The image shows a web application window titled "OwnInformationForm". The main form has a dark green header with the title "Own Information" in white. Below the header, there are three input fields: "Your login:" with the value "Maxim", "Your surname:" with the value "Shushkevich", and "Your age:" with the value "19". At the bottom of the form, there are two buttons: "Save input information about yourself" and "Cancel". Overlaid on top of the form is a smaller dialog box titled "Save changes". This dialog box contains a question mark icon and the text "Are you sure you want to change your personal information?". It has two buttons at the bottom: "OK" and "Отмена" (Cancel).

Рисунок Д.14 – Сохранение изменений о собственном аккаунте

The image shows a small dialog box titled "Complete" with a close button (X) in the top right corner. The text inside the dialog box says "The changes are saved!". At the bottom of the dialog box, there is an "OK" button.

Рисунок Д.15 – Информирование об успешности.

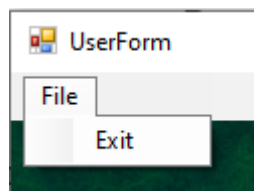


Рисунок Д.16 – Выход к авторизации

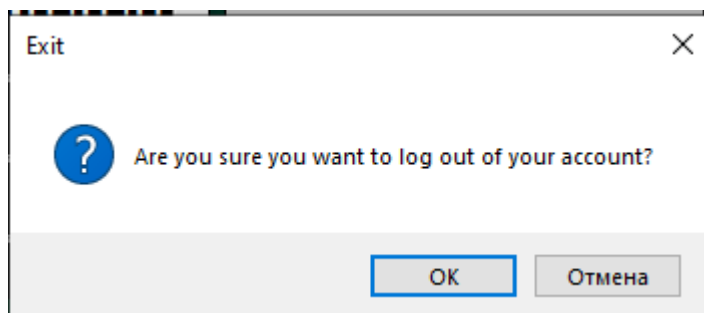


Рисунок Д.17 – Подтверждение выхода

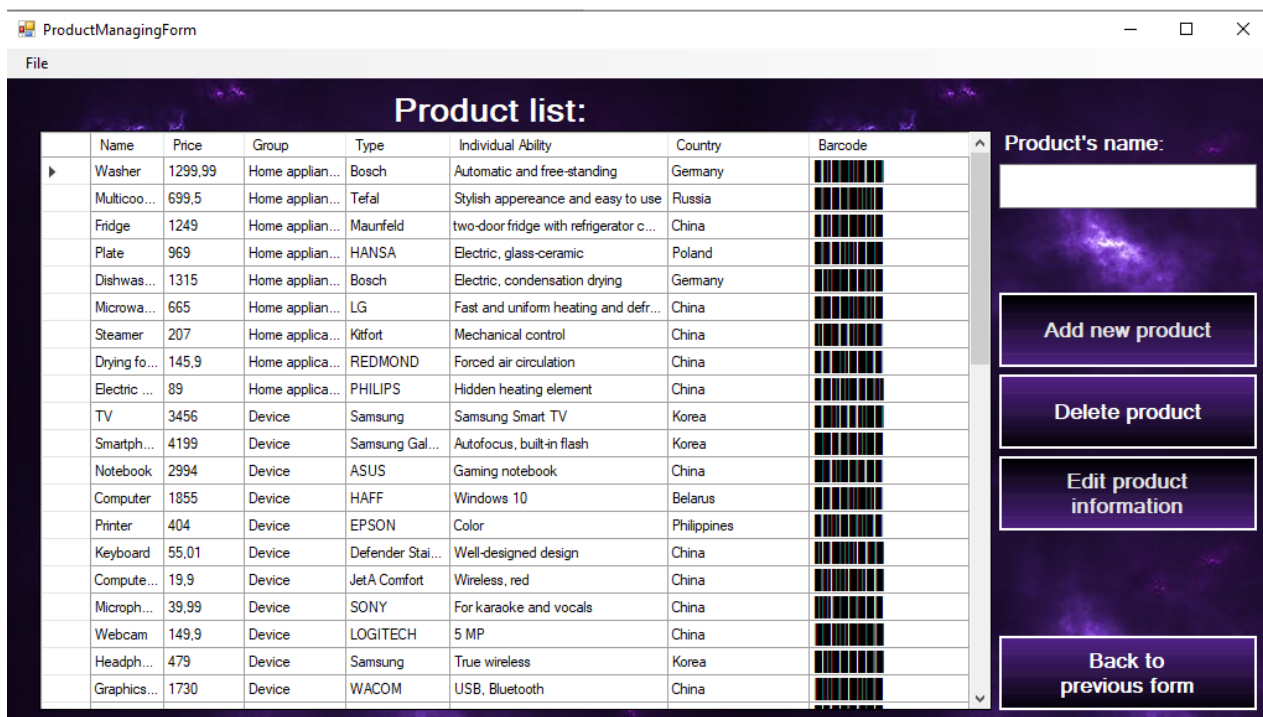


Рисунок Д.18 – Окно управления продуктами

The screenshot shows a window titled 'ProductForm' with a dark purple background and a white title bar. The main heading is 'Add product' in a large, bold, white serif font. Below the heading are seven input fields, each with a label in white text: 'Product name:', 'Product price:', 'Product group:', 'Product type:', 'Product individual ability:', 'Product country:', and 'Product barcode:'. At the bottom of the form are two buttons: 'Add' and 'Back', both with white text on a dark purple background.

Рисунок Д.19 – Добавление продукта

The screenshot shows a dialog box titled 'Add product' with a close button (X) in the top right corner. The dialog contains a blue circular icon with a white question mark and the text 'Are you sure you want to add a new product?'. At the bottom of the dialog are two buttons: 'OK' and 'Отмена' (Cancel), both with white text on a light gray background.

Рисунок Д.20 – Подтверждение добавления

The screenshot shows a small dialog box titled 'Correct' with a close button (X) in the top right corner. The dialog contains the text 'Product added!'. At the bottom of the dialog is a single button: 'OK', with white text on a light gray background.

Рисунок Д.21 – Успешное добавление

ProductForm

Edit product

Product name:
Microwave oven

Product price:
665

Product group:
Home appliances

Product type:
LG

Product individual ability:
Fast and uniform heating and defrosting

Product country:
China

Product barcode:
29374283

Save Back

Рисунок Д.22 – Окно изменения информации о продукте

Edit product

Are you sure you want to edit the information about this product?

OK Отмена

Рисунок Д.23 – Сохранение изменений о продукте

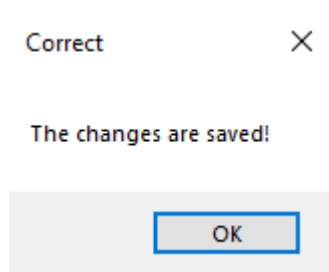


Рисунок Д.24 – Успешное сохранение

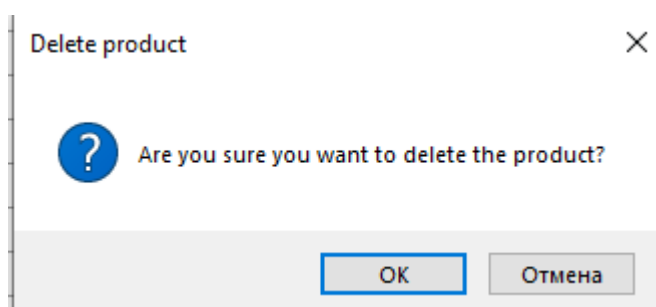


Рисунок Д.25 – Подтверждение удаления продукта

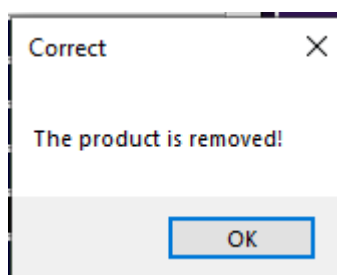


Рисунок Д.26 – Успешное удаление

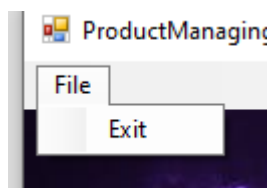


Рисунок Д.27 – Вернуться в предыдущую форму

UserManagingForm

User list:

User's username:

Sorting:

Find user

Add user

Edit user information

Show receipts

Delete user

Back to previous form

	Username	Name	Surname	Age
▶	Vlad	Vlad	Logvin	18
	Max	Max	Malinovskii	25
	Ilya	Ilya	Shepelev	20
	Dasha	Dasha	Bik	18
	Vadim	Vadim	Grashchenko	18
	Irina	Irina	Drozdovskaya	18
	Julia	Julia	Lodis	18
	Vika	Vika	Gavriluk	18
	Ksusha	Ksusha	Aleshko	18
	Irina2	Irina	Nehaichik	18
	Alexandr	Alexandr	Vasukovich	18
	Victor	Victor	Dichok	18
	Darina	Darina	Strahar	18
	Sasha	Sasha	Astashenok	18
	Anton	Anton	Vasilenko	18
	Nadezhda	Nadezhda	Minkevich	18
	Kostya	Kostya	Korchikov	19
	Igor	Igor	Kalenik	18
	Masha	Masha	Tishkevich	18
	Denis	Denis	Vasiliev	18
	Ksenia	Ksenia	Sharupich	18
	Maxim	Maxim	Shushkevich	19
	Alex	Alex	Belovskiy	18

Рисунок Д.28 – Окно управления аккаунтами

The screenshot shows a web application window titled "UserInformationForm". The main heading is "Add user". Below the heading, a note states "required fields are marked with *". The form contains the following fields: "Username*" (a text input field), "Password*" (a text input field), "Name" (a text input field), "Surname" (a text input field), "Age" (a text input field), and "Role" (a dropdown menu currently showing "0"). At the bottom of the form are two buttons: "Add" and "Back". The background of the form is a blue sky with white clouds.

Рисунок Д.29– Окно добавления нового пользователя

The screenshot shows a confirmation dialog box titled "Add user" with a close button (X) in the top right corner. The dialog contains a blue circular icon with a white question mark and the text "Are you sure you want to add a new user?". At the bottom of the dialog are two buttons: "ОК" (OK) and "Отмена" (Cancel).

Рисунок Д.30 – Подтверждение добавления нового пользователя

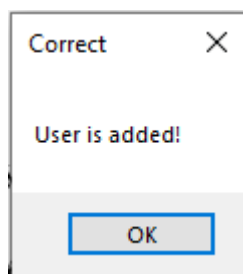


Рисунок Д.31 – Успешное добавление

A screenshot of a web application window titled 'UserInformationForm'. The main heading is 'Edit user'. Below it, a note says 'required fields are marked with *'. The form contains several input fields: 'Username*' with the value 'Kirill228', 'Password' (empty), 'Name' with 'Kirill', 'Surname' with 'Neforov', 'Age' with '23', and 'Role' with '0'. At the bottom are 'Save' and 'Back' buttons.

Рисунок Д.32 – Окно изменения пользователей

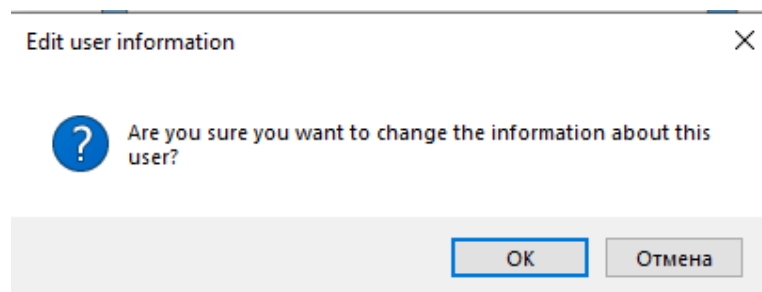


Рисунок Д.33 – Подтверждение сохранения изменений

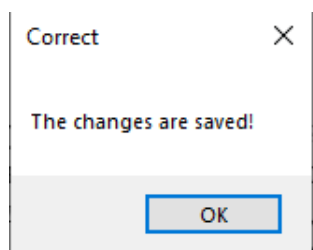


Рисунок Д.34 – Успешность сохранения

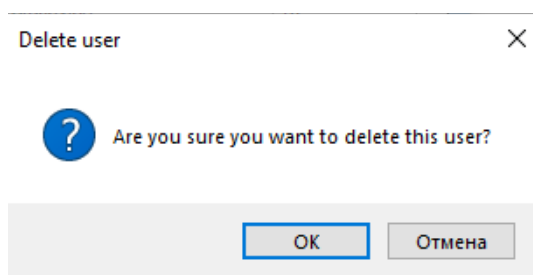


Рисунок Д.35 – Подтверждение удаления пользователя

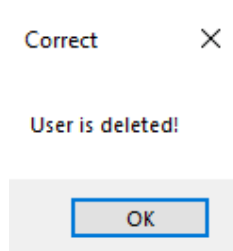


Рисунок Д.36 – Успешность удаления

The screenshot shows a web application window titled "UserManagingForm". On the left, there is a search form with the label "User's username:" and a text input field containing "Vika". Below the input field is a "Sorting:" dropdown menu. Further down are five buttons: "Find user", "Add user", "Edit user information", "Show receipts", and "Delete user". At the bottom is a "Back to previous form" button. On the right, under the heading "User list:", there is a table with the following columns: Username, Name, Surname, and Age. The table contains 20 rows of user data. The row for "Vika" is highlighted in yellow.

Username	Name	Surname	Age
Vlad	Vlad	Logvin	18
Max	Max	Malinovskii	25
Ilya	Ilya	Shepelev	20
Dasha	Dasha	Bik	18
Vadim	Vadim	Grashchenko	18
Irina	Irina	Drozdovskaya	18
Julia	Julia	Lodis	18
Vika	Vika	Gavrilyuk	18
Ksusha	Ksusha	Aleshko	18
Irina2	Irina	Nehaichik	18
Alexandr	Alexandr	Vasukovich	18
Victor	Victor	Dichok	18
Darina	Darina	Strahar	18
Sasha	Sasha	Astashenok	18
Anton	Anton	Vasilenko	18
Nadezhda	Nadezhda	Minkevich	18
Kostya	Kostya	Korchikov	19
Igor	Igor	Kalenik	18
Masha	Masha	Tishkevich	18
Denis	Denis	Vasiliev	18
Ksenia	Ksenia	Sharupich	18
Maxim	Maxim	Shushkevich	19
Alex	Alex	Butenko	18

Рисунок Д.37 – Поиск пользователя

The screenshot shows a small dialog box with the title "Not found" and a close button (X). The text inside the dialog box says "User Kirill doesn't exist!". At the bottom of the dialog box is an "OK" button.

Рисунок Д.38 – Отсутствие пользователя с указанным логино

The screenshot shows a dropdown menu for sorting. The label "Sorting:" is at the top. The dropdown is open, showing the following options: "Login sort" (selected), "Login sort", "Name sort", "Surname sort", and "Age sort".

Рисунок Д.39 – Сортировка для пользователей

ReceiptForm

Receipts of Malinovskii Max:

	Name	Price
▶	TV	3456
	Power supply	89
	Washer	1299,99
	Notebook	2994
	Portable device	53
*		

Summary price: 7891,99
 Date: 13.12.2020 Time: 15:49:6

Previous receipt Next receipt

Back

Рисунок Д.40 – Чеки

Enter the System

Username

Password

The field can contain:
 letters from a to z;
 uppercase letters from A to Z;
 hyphen '-';
 numbers from 0 to 9.

Рисунок Д.41 – Пример для ввода

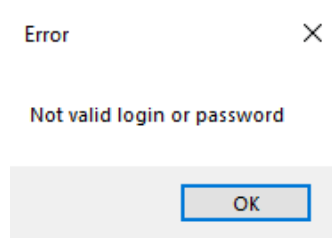


Рисунок Д.42 – Обработка ошибки ввода при авторизации

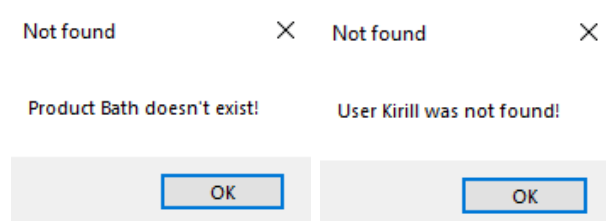


Рисунок Д.43 – Не существование продукта с введенным именем,
Рисунок Д.44 – Не существование пользователя с введенным логином

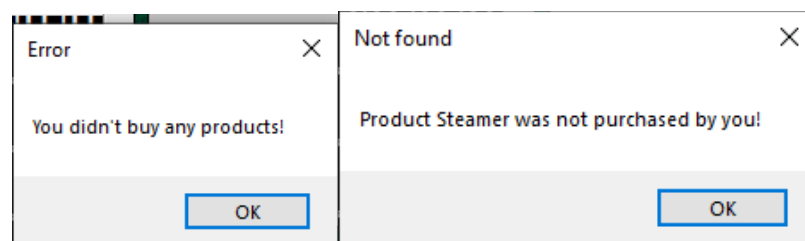


Рисунок Д.45 – В чеке нет продуктов,
Рисунок Д.46 – Невозможно вернуть продукт, который не покупали

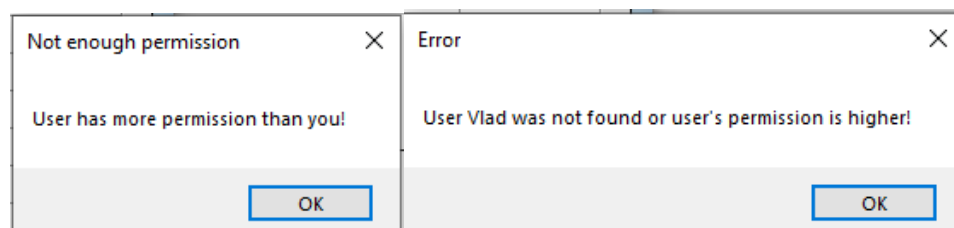


Рисунок Д.47 – Нельзя изменить информацию пользователя выше по роли,
Рисунок Д.48 – Невозможно удалить пользователя, который выше по роли

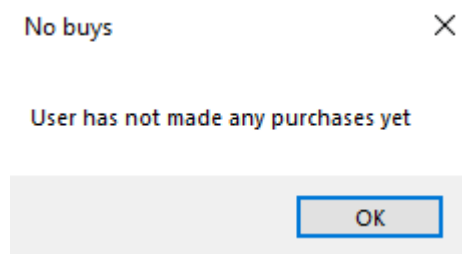


Рисунок Д.49 – Пользователь не делал покупок

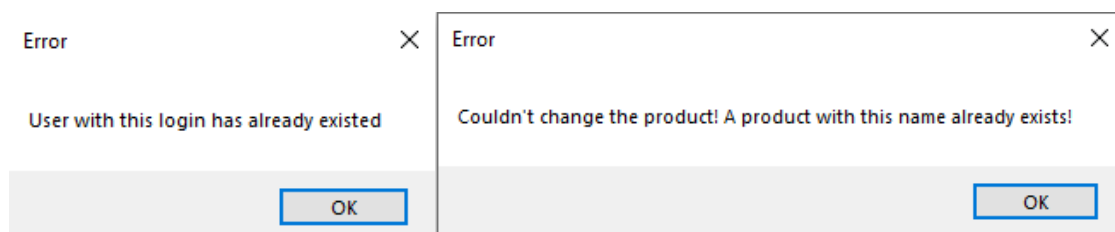


Рисунок Д.50 – Пользователь с таким логином уже существует,

Рисунок Д.51 – Продукт с таким названием уже существует