

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

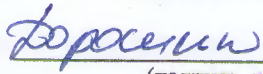
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**ПРОГРАММНЫЕ СРЕДСТВА
ФОРМАТИРОВАНИЯ, ФИЛЬТРАЦИИ И
СОРТИРОВКИ ДАННЫХ ДЛЯ
ТАБЛИЧНОГО ПРОЦЕССОРА**

Студент  31.05.2022 Сперанский Д.Д.
(подпись, дата)

Руководитель  31.05.2022 Дорошенко И.Н.
(подпись, дата)

Нормоконтролёр  31.05.2022 Бычков А.С.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 2.06.22 г. № 13)

Заведующий кафедрой  М.А. Митрохин
(подпись)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от
15.06.2022 № 7)

Секретарь ГЭК  Е.И. Гурин
(подпись)
Пенза 2022

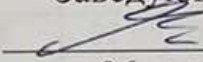
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«26» апреля 2022 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Сперанский Дмитрий Денисович гр. 18BB2 факультета вычислительной техники.

2. Тема работы «Программные средства форматирования, фильтрации и сортировки данных для табличного процессора»

Тема утверждена приказом ПГУ № 357/о от "26" апреля 2022 г.

3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Дорошенко И.Н.

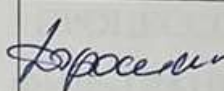
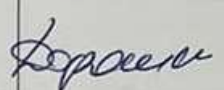
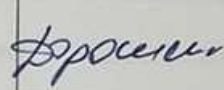
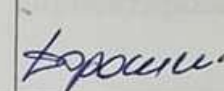
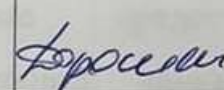
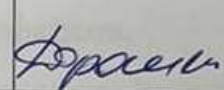
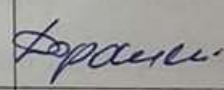
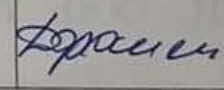
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программные средства, предназначенные для форматирования, фильтрации и сортировки данных для табличного процессора. Спроектировать для них графический интерфейс.

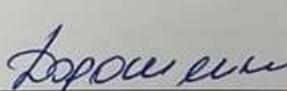
5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области, изучить актуальные на сегодняшний день решения, произвести их сравнение, определить архитектуру разрабатываемых модулей. Необходимо изучить методологии проектирования объектно-ориентированных приложений, функционал программных платформ *.net Framework* и *Windows Forms*.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	18	26.04.2022-28.04.2022	
2	Анализ требований	18	29.04.2022-01.05.2022	
3	Выбор технологий, применяемых при разработке	18	02.05.2022-04.05.2022	
4	Проектирование архитектуры	27	04.05.2022-07.05.2022	
5	Описание алгоритмов	27	08.05.2022-11.05.2022	
6	Разработка модуля форматирования данных	108	12.05.2022-19.05.2022	
7	Разработка модуля фильтрации и сортировки данных	60	20.05.2022-25.05.2022	
8	Тестирование и отладка	27	26.05.2022-28.05.2022	
9	Оформление выпускной работы	21	29.05.2022-31.05.2022	

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы  26.04.2022 Дорошенко И.Н.
(подпись, дата)

Задание к исполнению принял студент  26.04.2022 Сперанский Д.Д.
(подпись, дата)

Работу к защите допустить
Декан факультета  06.06.22 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 159 листов, 80 рисунков, 4 таблицы, 6 источников, 3 приложения.

ОФИСНЫЕ ПРОГРАММЫ, ТАБЛИЧНЫЙ ПРОЦЕССОР, ФОРМАТИРОВАНИЕ ДАННЫХ, СОРТИРОВКА И ФИЛЬТРАЦИЯ ДАННЫХ, СТАНДАРТ *OFFICE OPEN XML*.

Цель работы – разработать программные средства для табличного процессора «Ведомость», позволяющие пользователю выполнять форматирование, фильтрацию и сортировку данных, а также спроектировать для них графический интерфейс.

Объектами разработки являются программные средства, осуществляющие форматирование, фильтрацию и сортировку данных, состоящие из двух библиотек классов и двух графических интерфейсов.

В результате проведенной работы реализован проект, удовлетворяющей поставленной задаче, а именно, созданы программные средства, осуществляющие форматирование, фильтрацию и сортировку данных в табличном процессоре. Модуль фильтрации данных был успешно интегрирован в табличный процессор «Ведомость», разрабатываемый ООО НТП «Криптософт».

					ПГУ 2.090301.39.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства форматирования, фильтрации и сортировки данных для табличного процессора Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Сперанский Д.Д.					2	159	
Провер.		Дорошенко И.Н.							
Н. Контр		Бычков А.С.							
Утверд.		Митрохин М.А.				ФВТ, каф. ВТ гр. 18ВВ2			

Перечень принятых сокращений

IDE (Integrated development environment) – интегрированная среда разработки
TFS (Team Foundation Server) – многофункциональное решение, объединяющее в себе систему управления версиями, построение отчетов, сбор данных, отслеживание изменений по проекту

UI (User Interface) – пользовательский интерфейс

XML (eXtensible Markup Language) – расширяемый язык разметки

ОС – операционная система

ПО – программное обеспечение

ПК – персональный компьютер

ПФ – проверка функционала

НТП – научно-техническое предприятие

					ПГУ 2.090301.39.001 ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

Содержание

Реферат	2
Введение.....	6
1 Программные средства форматирования, фильтрации и сортировки данных для табличного процессора	7
1.1 Анализ предметной области	7
1.1.1 Постановка задачи	7
1.1.2 Анализ аналогов ПО	9
1.1.3 Анализ задачи.....	11
1.1.4 Структура кодов числовых форматов в табличных редакторах.....	12
1.1.5 Стандарт <i>Office Open XML</i>	13
1.1.6 Выбор инструментов разработки	16
1.1.7 Дополнительные плагины и технологии	18
1.2 Реализация программных средств.....	18
1.2.1 Описание архитектуры.....	18
1.2.2 Разработка библиотеки форматирования данных	23
1.2.3 Разработка библиотеки сортировки и фильтрации данных	26
1.2.4 Разработка графического интерфейса	27
1.3 Анализ результата	32
1.3.1 Тестирование и отладка	32
1.3.2 Тестирование в табличном процессоре «Ведомость».....	48
2 Безопасность жизнедеятельности.....	59
2.1 Анализ потенциальных опасностей и вредностей, возникающих при работе с ПЭВМ и способы их минимизации	59
2.2 Классификация и категорирование помещения, в котором производится разработка.....	59
2.2.1 Опасность поражения электрическим током	59
2.2.2 Пожарная безопасность.....	61
2.3 Санитарно-гигиенические требования к производству	62

2.4 Расчет пожарной нагрузки в помещениях категории В1-В4.....	63
Заключение	66
Список использованных источников	67
Приложение А. <i>UML</i> -диаграммы	68
Приложение Б. Листинг программы	76
Приложение В. Презентация.....	145

Введение

В последнее время актуальность импортозамещения заметно выросла. Для обеспечения устойчивости экономики страны необходимо прежде всего разрабатывать отечественные *IT*-продукты. Научно-техническое предприятие ООО НТП «Криптософт» около 20 лет занимается разработкой защищённой операционной системы *QP* ОС и предназначенных для нее прикладных программ. Важной целью разработки отечественной ОС является создание защищенной и безопасной системы.

Одним из разрабатываемых на предприятии продуктов является офисный пакет «*QP* Грамота». В пакет входят редактор текстовых документов, редактор электронных таблиц, менеджер презентаций и клиент электронной почты. При разработке особое внимание уделяется стабильности и быстродействию продукта.

Редактор текстовых документов предназначен для создания, редактирования, просмотра и форматирования текстовых статей, документов. Имеет широкий функционал: от набора и верстки текста до проверки орфографии, вставки в текст графического материала, печати текста.

Менеджер презентаций используется для подготовки и просмотра презентаций. Чаще всего материалы, подготовленные с его помощью, предназначены для отображения на большом экране – через проектор или телевизионный экран.

Редактор электронных таблиц позволяет создавать сложные электронные таблицы, а также производить тяжелые вычисления. С помощью него также можно проанализировать полученные данные, применяя удобные встроенные инструменты. Важными функциональными характеристиками табличного редактора является поддержка сортировки, фильтрации и форматирования данных.

Целью данной ВКР является разработка модулей форматирования, фильтрации и сортировки данных для табличного редактора.

					ПГУ 2.090301.39.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Программные средства форматирования, фильтрации и сортировки данных для табличного процессора

1.1 Анализ предметной области

1.1.1 Постановка задачи

Прежде всего, стоит отметить, что модули разрабатываются для табличного процессора «Ведомость», применяемого в отечественной операционной системе *QP ОС*, разработкой которой занимается научно-техническое предприятие «Криптософт».

QP ОС позиционируется, как защищенная система, при разработке системы особое внимание было уделено безопасности системы и контролю доступа к данным. Для нее с 2017 года разрабатывается собственный офисный пакет приложений, в который входят: редакторы текстовых документов, электронных таблиц и презентаций, медиаплеер.

Офисные программы поддерживают, как новые форматы *Open XML* (.docx, .xlsx, .pptx), так и старые форматы (.doc, .xls, .ppt), их поддержка осуществляется за счет встроенного конвертера, который выполняет свою основную функцию при открытии документа, имеющего старый формат.

Разрабатываемые офисные программы могут работать как под управлением *Windows*, так и в *QP ОС*.

Главным мотивом разработки табличного процессора является создание полностью отечественного продукта, не уступающего, в некоторых моментах даже превосходящего, привычного большинству пользователей табличного процессора *Microsoft Excel*. При этом необходимо сохранить привычный пользователю интерфейс, которыми обладают все имеющиеся на данный момент пакеты офисных программ, например *Microsoft Office*, *Libre Office*, *Open Office*, Мой Офис.

На данный момент «Ведомость» уже поддерживает основной функционал табличного процессора:

					ПГУ 2.090301.39.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

- открытие документа в формате *XLSX* или *XLS*;
- сохранение документа в формате *XLSX*;
- создание таблиц, путем управления строками и столбцами (изменения высоты/ширины, кол-ва и тп.);
- управление атрибутами ячеек (цвет фона, цвет текста, цвет/ширина границ);
- редактирование текста ячеек, задание шрифта, цвета, др. атрибутов редактируемого текста;
- выравнивание текста (левый край, центр, правый край, по ширине);
- возможность добавления/удаления строк и столбцов;
- поддержка объединения/разъединения ячеек;
- отображение картинок в ячейках;
- поддержка буфера обмена.

В рамках выполнения квалификационной работы была поставлена задача разработать программные средства для табличного процессора, которые должны реализовать следующие функции:

- форматирование данных в таблице, как в соответствии с форматом, созданным пользователем, так и с заранее заданными форматами;
- сортировка данных по критерию, задаваемому пользователем;
- фильтрация данных на основании выбранных условий.

Задачи, которые должны быть решены при реализации:

- разработка эффективного алгоритма разбиения и структурирования кода формата, введенного пользователем, на токены;
- реализация алгоритма применения введенного формата к выбранному массиву данных;
- разработка эффективного алгоритма сортировки данных;
- разработка эффективного алгоритма фильтрации данных;
- разработка графического интерфейса пользователя;
- тестирование разработанного ПО.

1.1.2 Анализ аналогов ПО

Наиболее популярными программными продуктами, включающими в себя аналог разрабатываемых программных средств, являются *Microsoft Excel*, *Open Office*, *Libre Office*. Главным недостатком каждого из них является то, что все они разработаны за рубежом. Из отечественных аналогов можно отметить лишь МойОфис.

Отличительной особенностью, разрабатываемого НТП «Криптософт» продукта, является полностью уникальная архитектура проекта и код, написанный с нуля, без использования *Open-Source* технологий. При этом реализована полная поддержка чтения и записи файлов согласно стандарту *ЕСМА-376*.

Каждый из представленных аналогов содержит модуль форматирования, фильтрации и сортировки данных. Для совместимости с ними в ходе разработки, будет реализована поддержка аналогичных кодов форматов.

Самый большой набор числовых форматов реализован в продукте *Microsoft Excel*, в остальных отсутствуют несколько форматов.

В *Excel* заданы следующие стандартные числовые форматы:

- общий;
- числовой;
- денежный;
- финансовый;
- дата;
- время;
- процентный;
- дробный;
- экспоненциальный;
- текстовый;
- дополнительный.

Некоторых из них содержат определенные настройки, а остальные заранее созданные типы. Дополнительно в *Microsoft Excel* реализован числовой формат «Пользовательский», в последнем пользователь сам вводит код формата. Стоит отметить, что «Мой Офис Таблица» не предоставляет пользователю эти форматы, что является его существенным недостатком.

Эталонным модулем форматирования данных обладает продукт *Microsoft Excel*. Но, он является зарубежной разработкой. Таким образом, есть смысл произвести сравнение с единственным отечественным аналогом «Мой Офис Таблица». По сравнению с *Excel*, модуль форматирования данных содержит меньшее количество настроек для различных форматов. Но главным его недостатком является невозможность ввода пользовательского кода формата. Также «Мой Офис Таблица» не обладает модулем фильтрации данных по различным условиям. В ходе разработки модуля эти критерии будут учтены и реализованы.

В *Microsoft Excel* содержится огромное количество фильтров для следующих типов данных: текстовый, числовой и дата. Для текстовых данных реализованы следующие фильтры: «Равно», «Не равно», «Начинается с», «Заканчивается на», «Содержит», «Не содержит». Для числовых: «Равно», «Не равно», «Больше», «Больше или равно», «Меньше», «Меньше или равно», «Между», «Первые 10», «Выше среднего», «Ниже среднего». Для даты реализовано огромное количество фильтров, позволяющих делать выборку данных за определенный период или интервал. Также существует возможность комбинировать любые два из фильтров с помощью операций «И», «ИЛИ».

Отечественный аналог «Мой Офис Таблица» имеет возможность фильтрации только по возрастанию или убыванию. В ходе разработки модуля фильтрации будет реализована поддержка всех фильтров из продукта компании *Microsoft*, а также их комбинирование.

Во всех рассмотренных выше редакторах электронных таблиц интерфейсы инструментов форматирования, фильтрации и сортировки данных схожи. Для

того, чтобы пользователю было привычно работать в табличном процессоре «Ведомость», будет реализован схожий интерфейс с аналогами интерфейса.

В конечном итоге, разрабатываемые модули не должны ничем уступать аналогичным модулям в *Microsoft Excel*, а также превосходить в функционале отечественные аналоги. Так же в заключении стоит отметить, что пакет офисных программ «Мой Офис» не является полностью отечественным продуктом, так как некоторый функционал был заимствован из западных разработок.

1.1.3 Анализ задачи

Разрабатываемые программные модули должны поддерживать форматирование, сортировку и фильтрацию данных. При этом для форматирования могут использоваться, как заранее заданные, так и введенные пользователем в виде кода форматы.

Код формата — это унифицированная последовательность, составленная из следующих основных частей (токенов), который могут комбинироваться в зависимости от желания пользователя:

- литерал;
- цифра;
- текстовый заполнитель (*placeholder*) (#, *, ?);
- экспонента (e+, e-);
- основной формат (*general*);
- часть даты (y, m, d, h, m, s, am/pm);
- продолжительность времени ([h], [m], [s]);
- цвет ([Красный]);
- условие (>=100).

Для начала должна производиться валидации формата, если она прошла успешно, то необходимо приступить к разбору последовательности. На следующем этапе требуется определить к какому типу принадлежит введенный

код (основной, дробный, экспоненциальный и т.д.) и выбрать подходящий алгоритм форматирования данных.

Для выполнения сортировки, необходимо определить тип данных (текст, числовой, дата, время и т.д.) и в зависимости от него выбрать алгоритмы сортировки. Аналогично и для фильтрации: в зависимости от типа данных, пользователю будет предложен определенный набор условий для осуществления этой операции.

1.1.4 Структура кодов числовых форматов в табличных редакторах

С помощью числовых форматов можно изменять внешний вид чисел, даты или времени, при этом фактически не меняя числа. Числовой формат не влияет на значение ячейки, которое табличный процессор использует для вычислений и записывает в *XML*. Реальное значение содержится в строке формул.

При создании пользовательского формата можно указывать до четырех разделов. Они записываются в следующем порядке:

- формат для положительных чисел (1);
- формат для отрицательных чисел (2);
- формат для нулевых значений (3);
- формат для текстовых данных (4).

При этом разделы кода разделяются знаком «;». На рисунке 1 изображен пример записи пользовательского кода формата [1].

Рисунок 1 - Пользовательский код формата

Если в коде указан не полный набор разделов, то применяются следующие правила определения принадлежности раздела к конкретным данным: если указан только один раздел, то он будет использоваться для всех чисел. Если указаны два раздела, первый раздел будет использоваться для положительных чисел и нулей, а второй для отрицательных чисел. Также

пользователь имеет возможность пропустить какой-либо раздел, для этого необходимо добавить точку с запятой.

1.1.5 Стандарт *Office Open XML*

ECMA-376 определяет семейство *XML* схем, вместе называемых *Office Open XML*, которые определяют правила для обработки текстовых документов, электронных таблиц и презентаций, а также упаковку этих документов, в соответствии с определенными схемами [2].

Главная цель создания данного стандарта – стандартизировать структуру всех документов, что позволит воспроизводить их одинаково различными офисными приложениями.

Стандарт *Open XML* включает в себя несколько более детальных стандартов. Одним из них является стандарт *OPC (Open Packaging Convention)*, который описывает структуру файла, взаимосвязь его частей, наличие различных типов данных в документе, а также, при необходимости, цифровую подпись документа.

Помимо *OPC* используются следующие стандарты:

- *WordprocessingML*. Описывает разметку текстовых документов;
- *SpreadSheetML*. Отвечает за структуру электронных таблиц;
- *PresentationML*. Описывает структуру презентаций;
- *DrawingML*. Указывает структуру графиков, диаграмм и некоторых графических объектов, также может описывать формулы, выражения и метаданные документа.

С точки зрения пользователя любой документ, соответствующий стандарту *OpenXML*, представляет собой *ZIP*-архив, содержащий в себе структурированные *XML* файлы, файлы ресурсов документа и файлы, содержащие в себе отношения между составными частями документа (*Relationships*).

Каждый из стандартных форматов данных имеет свой собственный *ID*, который никак не хранится в архиве документа. Коды форматов, которые вводит пользователь хранятся в файла *styles.xml*, который располагается по

					ПГУ 2.090301.39.001 ПЗ	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

пути *table.xlsx/xl/styles.xml*. За хранение таких кодов отвечает элемент *numFmts*, который имеет атрибут *count*, отвечающий за количество пользовательских форматов [3].

Элемент *numFmts* содержит в себе множество элементов с тегом *numFmt*. В свою очередь, *numFmt* содержит следующие атрибуты:

– *numFmtID*. Определяет уникальный *ID* формата, введенного пользователем;

– *formatCode*. Определяет код формата, введенный пользователем.

Ниже приведен пример хранения в *XML* созданного пользовательского числового формата.

```
<numFmts count="1">
  <numFmt numFmtId="164" formatCode="##,##0;\-##,##0"/>
</numFmts>
```

При задании форматов создается элемент *cellXfs* с атрибутом *count*, определяющим количество примененных форматов. Внутри себя этот элемент содержит множество элементов *xf*, имеющих атрибуты определяющие параметры этого стиля.

Данный элемент имеет следующий вид.

```
<cellXfs count="3">
  <xf numFmtId="0" fontId="0" fillId="0" borderId="0"
xfId="0"/>
  <xf numFmtId="9" fontId="0" fillId="0" borderId="0"
xfId="1" applyFont="1"/>
  <xf numFmtId="164" fontId="0" fillId="0" borderId="0"
xfId="0" applyNumberFormat="1"/>
</cellXfs>
```

Здесь как раз и определяется *ID* числового формата, установленного для стиля, за это отвечает атрибут *numFmtId*. Идентификатор формата 0 принадлежит стандартному формату, 9 – процентному, который является стандартным, 164 – это как раз введенный пользователем формат, *XML* элемент которого приведен выше.

Принадлежность определенного формата ячейке определяется в файле *sheet.xml*, располагающегося по пути *table.xlsx/xl/worksheets/sheet1.xml*. Внутри

него содержится элемент *sheetData*, внутри которого содержится множество элементов *row*, отвечающих за строки, внутри которых в свою очередь содержатся элементы *c* (*cell*). Атрибут *s* у элемента *c* как раз и отвечает за номер стиля, который определен в файле *styles.xml*, установленный ячейке. Стиль в свою очередь содержит атрибут с кодом формата (*numFmtId*).

Ниже приведен пример элемента *sheetData*.

```
<sheetData>
  <row r="1" spans="1:2" x14ac:dyDescent="0.3">
    <c r="A1" s="1">
      <v>767878</v>
    </c>
    <c r="B1" s="1">
      <v>965936954</v>
    </c>
  </row>
  <row r="2" spans="1:2" x14ac:dyDescent="0.3">
    <c r="A2" s="2">
      <v>423423</v>
    </c>
    <c r="B2" s="2">
      <v>654654</v>
    </c>
  </row>
  <row r="3" spans="1:2" x14ac:dyDescent="0.3">
    <c r="A3" s="1">
      <v>123</v>
    </c>
    <c r="B3" t="s">
      <v>0</v>
    </c>
  </row>
</sheetData>
```

Стиль, применяемый к ячейке, зависит от атрибута *s*:

– *s*=1 – стиль, где установлен числовой формат «процентный» с идентификатором 9;

– *s*=2 – стиль, где установлен пользовательский числовой формат с идентификатором 164.

Атрибут *t="s"* означает, что применяется стандартный стиль.

В ходе интеграции библиотеки форматирования данных в «Ведомость» все перечисленные выше моменты будут учтены и будет реализована полная

поддержка стандарта *Office Open XML*.

1.1.6 Выбор инструментов разработки

1.1.6.1 Язык программирования

В качестве языка программирования был выбран *C#*. Он является объектно-ориентированным и строго типизированным языком. Также *C#* – это один из мощных и востребованных языков программирования на рынке. Его синтаксис схож с *Java* и *C++*. Основными преимуществами *C#* являются: высокая скорость работы, надежность, простота в освоении, автоматическая сборка мусора и многое другое. На платформе *.Net* можно реализовывать любое приложение от консольных до оконных *WinForms* программ, веб-сайтов с помощью *ASP.NET*, мобильных и облачных приложений. Также стоит отметить, что разработка офисного пакета «*QP Грамота*» ведется преимущественно на языке *C#*, поэтому целесообразно выбрать именно его для разработки описанных ранее модулей.

1.1.6.2 Среда разработки

Выбор среды разработки пал на продукт *Microsoft – Visual Studio 2019*. Официальная, бесплатная и самая функциональная *IDE* для разработки на *C#*. Является самой распространенной среди *C#* разработчиков. *Visual Studio* полностью совместима с *TFS*, имеет огромное количество встроенных библиотек и инструментов, поддерживающих компиляцию, отладку, профилирование, юнит-тестирование. Также стоит отметить, что *Visual Studio* поддерживает подключение сторонних плагинов.

1.1.6.3 Платформа *.Net Framework*

.Net Framework – программная платформа, выпущенная компанией *Microsoft*. Первый релиз состоялся в 2002 году. Основой платформы является общезыковая среда исполнения *Common Language Runtime (CLR)*, которая подходит для различных языков программирования. [4]

Можно выделить следующие отличительные черты данной платформы:

– поддержка нескольких языков. Благодаря тому, что в основе платформы лежит *CLR*, *.Net Framework* поддерживает несколько языков

					ПГУ 2.090301.39.001 ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

программирования: *C#, VB.NET, C++, F#*, а также различные диалекты других языков, привязанные к *.NET*;

- кроссплатформенность. Используя различные технологии платформы, разработчик может создавать приложения для самых разных операционных систем: *Windows, Linux, MacOS, Android*;

- мощная библиотека классов. *.NET* представляет единую библиотеку классов для всех поддерживаемых языков программирования;

- разнообразие технологий. Стек технологий платформы *.NET* – огромен, отличительными из них являются: *ASP.NET, Windows Forms, WPF, WinUI, Entity Framework, ADO.NET* и другие;

- производительность. Согласно огромному количеству тестов, приложения, построенные на *.NET* отличаются высокой производительностью.

В ходе реализации проекта использовалась платформа *.Net Framework 4.5.2* для обеспечения совместимости с *QP ОС*.

1.1.6.4 Платформа пользовательского интерфейса

Для реализации пользовательского интерфейса с целью тестирования на ОС *Windows* было принято использовать платформу *Windows Forms*. Также стоит отметить, что для интеграции в табличный редактор «Ведомость» реализовывались аналогичные графические интерфейсы на основе собственной платформы, разработанной научно-техническим предприятием «Криптософт».

Windows Forms – это платформа пользовательского интерфейса для создания классических интерфейсов *Windows*, включенная в *.NET Framework*. Прежде всего она является объектно-ориентированной оболочкой над *Win32 API* [5].

Как и большинство аналогичных платформ, *WinForms* управляется событиями. Основным элементом является форма, которая содержит различные элементы управления (кнопки, текстовые поля, надписи и т.д.), с которыми пользователь напрямую взаимодействует. В ответ на взаимодействие эти элементы вызывают события, которые выполняются программой.

В *Windows Forms* можно разрабатывать сложные графические интерфейсы, которые легко обновлять и развертывать. Также с ними удобно работать, как в автономно режиме, так и в сети. Приложения *Windows Forms* могут получать доступ к файловой системе и оборудованию компьютера.

Существенным преимуществом является глубокая интеграция со средой разработки *Visual Studio*. Данная *IDE* содержит конструктор графических интерфейсов, что существенно упрощает реализацию приложения.

1.1.7 Дополнительные плагины и технологии

В ходе разработки применялся плагин *ReSharper*. Это одна из наиболее популярных разработок компании *JetBrains*, специально разработанный на основе партнерских отношений с *Microsoft*. Выбранный плагин позволяет повысить производительность, качество и чистоту кода.

Для разработки библиотеки фильтрации и сортировки использовалась технология *LINQ*, а конкретно провайдер *LINQ To Objects*. Данная технология позволяет выполнять запросы к коллекциям, реализующим интерфейс *IEnumerable*, что позволяет существенно повысить читаемость и производительность в написании кода. Также *LINQ to objects* обладает схожей производительностью по отношению к «ручному» коду [6].

1.2 Реализация программных средств

1.2.1 Описание архитектуры

Было принято решение разделить разрабатываемые программные средства на две библиотеки классов:

- библиотека сортировки и фильтрации данных;
- библиотека форматирования данных.

При построении архитектуры первой библиотеки было принято решение создать несколько вспомогательных перечислений и словарей с типами фильтров, это необходимо для сокращения количества кода в основных модулях и улучшения его читаемости. Так же для каждого типа фильтра был создан собственный класс со статическими функциями фильтрации. Для

передачи информации из графического интерфейса в библиотеку был создан класс «*FilterData*», содержащий в себе значения и условия для фильтрации, тип первого и второго фильтра, а также операцию (и, или), выполняемую над ними, это позволяет повысить читаемость кода. Главным модулем является класс «*Filter*», туда и передается объект, описанный выше. Основная его цель – выполнить фильтрацию данных.

В главном методе проектируемого модуля определяется тип переданных значений и фильтра, который затем применяется. Для определения типа используются описанные ранее словари, содержащие в себе название фильтра и элемент перечисления либо делегат, инкапсулирующий нужный метод фильтрации.

На рисунке 2 приведена схема взаимодействия модулей программы.

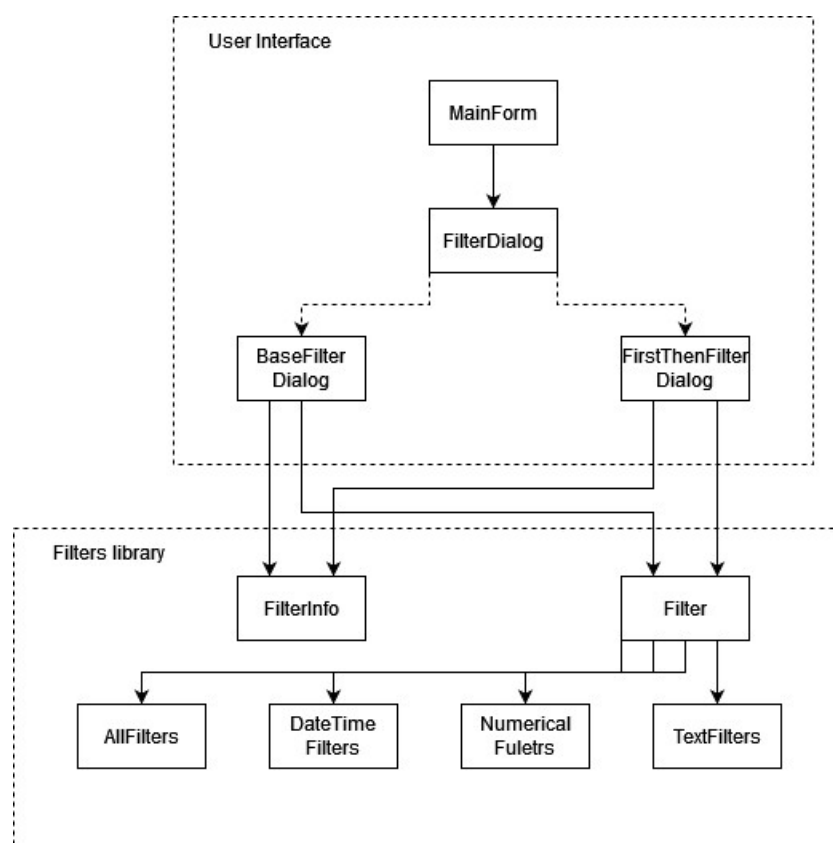


Рисунок 2 - Схема взаимодействия программы фильтрации данных

Графический интерфейс содержит 4 модуля. *MainForm* – это главное окно программы. Для задания фильтров было реализовано 2 диалога:

BaseFilterDialog, содержащий основные фильтры, *FirstThenFilterDialog*, содержащий фильтры «Первые 10...». Каждый из них унаследован от *FilterDialog*, содержащий в себе поле с массивом результирующим массивом индексов начального массива значений.

Передача данных из графического интерфейса на сторону библиотеки происходит с помощью вспомогательного класса *FilterArgs*, который содержит в себе исходные значения, название первого фильтра, значение первого фильтра, название второго фильтра, значение второго фильтра и операцию, выполняемую над результатами двух фильтров, при этом обязательными аргументами являются только первые два. Созданный объект передается в метод *ApplyFilter()*, находящийся в классе *Executor*. В нем определяется тип фильтра с помощью вспомогательного класса *AllFitlers*, содержащего перечисления, названия фильтров, а также делегаты, инкапсулирующие методы фильтрации. Они в свою очередь содержатся в классах *DateTimeFilters*, *NumericalFilters*, *TextFilters*.

Вторую библиотеку условно можно разделить на две части:

- парсер кода формата;
- модуль, форматирующий данные.

В ходе проектирования архитектуры парсера было принято создать несколько объектов-секций для каждого типа данных, на которые будет делиться код формата. Секции в свою очередь разбиваются на токены различных типов. Разбиение кода формата на секции будет производиться путем разделения строки по символу «;». Затем каждая секция будет считываться посимвольно для разбиения на токены, параллельно будет проверяться ее валидность, так как не все токены можно комбинировать в определенных разделах. Конечный результатом работы парсера - объект типа «*NumberFormat*», который содержит в себе список секций. На его основе будет производиться форматированные данных. Разбиение формата на мелкие части позволит структурировать написанный код и упростить процесс форматирования данных.

Модуль форматирования принимает на вход формируемое значение и созданный ранее объект типа «*Format*». На первом этапе определяется применяемая к данным секция. Затем, в зависимости от ее типа, будет вызван соответствующий метод для форматирования, где перебираются токены и постепенно собирается готовый результат.

На рисунке 3 приведена схема взаимодействия модулей программы.

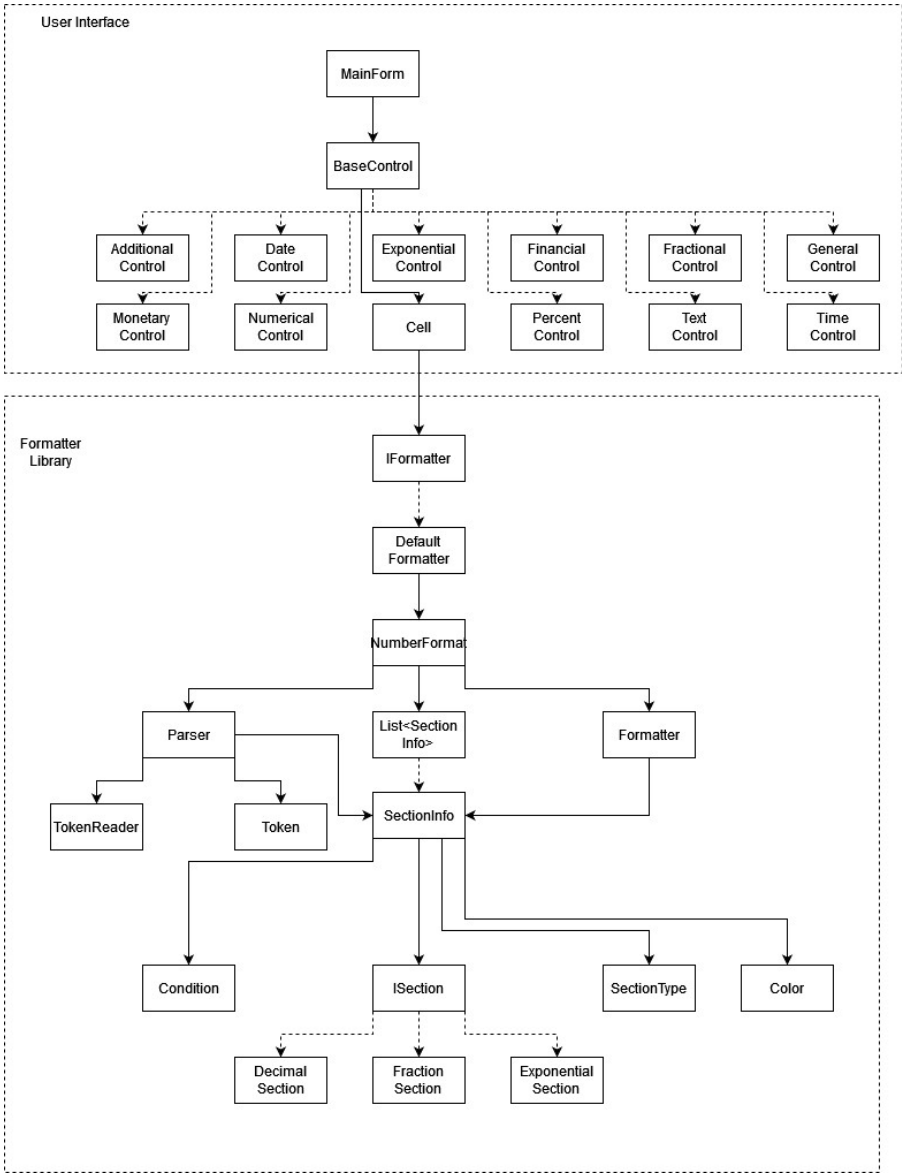


Рисунок 3 - Схема взаимодействия модулей программы форматирования данных

Графический интерфейс содержит 14 модулей. *MainForm* – главное окно программы. Для настройки форматов реализовано 11 пользовательских

элементов управления: *AdditionalControl*, *MonetaryControl*, *DateControl*, *NumericalControl*, *ExponentialControl*, *FinancialControl*, *PercentControl*, *FractionalControl*, *TextControl*, *TimeControl*. Все они содержат уникальный набор элементов графического интерфейса, для настройки соответствующего формата. Каждый из них унаследован от класса *BaseControl*, содержащий в себе метод *FormatValue()*, отвечающий за форматирование значения, свойства *Type*, *FormatCode* и *Cell*. Также в базовом классе содержится делегат *ValueHandler*, принимающий один параметра типа *Cell*, событие *ChangeFormattedValue*, которое представляет делегат *ValueHandler*, и метод *PerformChangeValue()*, вызывающий событие. Это необходимо для форматирования данных сразу после ввода их в соответствующее поле, без необходимости дополнительных действий от пользователя.

Форматирование значение происходит с помощью вызова метода *Format()* у класса *DefaultFormatter*, реализующего интерфейс *IFormatter*, хранящегося в виде свойства у объекта *Cell*. *DefaultFormatter* содержит свойство *FormatCode* типа *string* и объект типа *NumberFormat*, содержащий в себе «разобранный» по секциям формат, за что отвечает класс *Parser*, и метод *Format()*, который определяет нужную секцию и обращается к классу *Formatter* для получения результата. Разобранный формат хранится в виде списка объектов типа *SectionInfo*, который содержит тип и индекс секции, цвет, условие, саму секцию и токены типов «общий», «текст», «дата», «длительность». *Parser* в свою очередь в процессе работы создает объект типа *TokenReader*, который считывает все токены из строки, тип которых затем определяется с помощью класса «*Token*», что в свою очередь позволяет определить тип секции.

На рисунке 4 приведена *UML*-диаграмма вариантов использования разрабатываемых модулей.

					ПГУ 2.090301.39.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

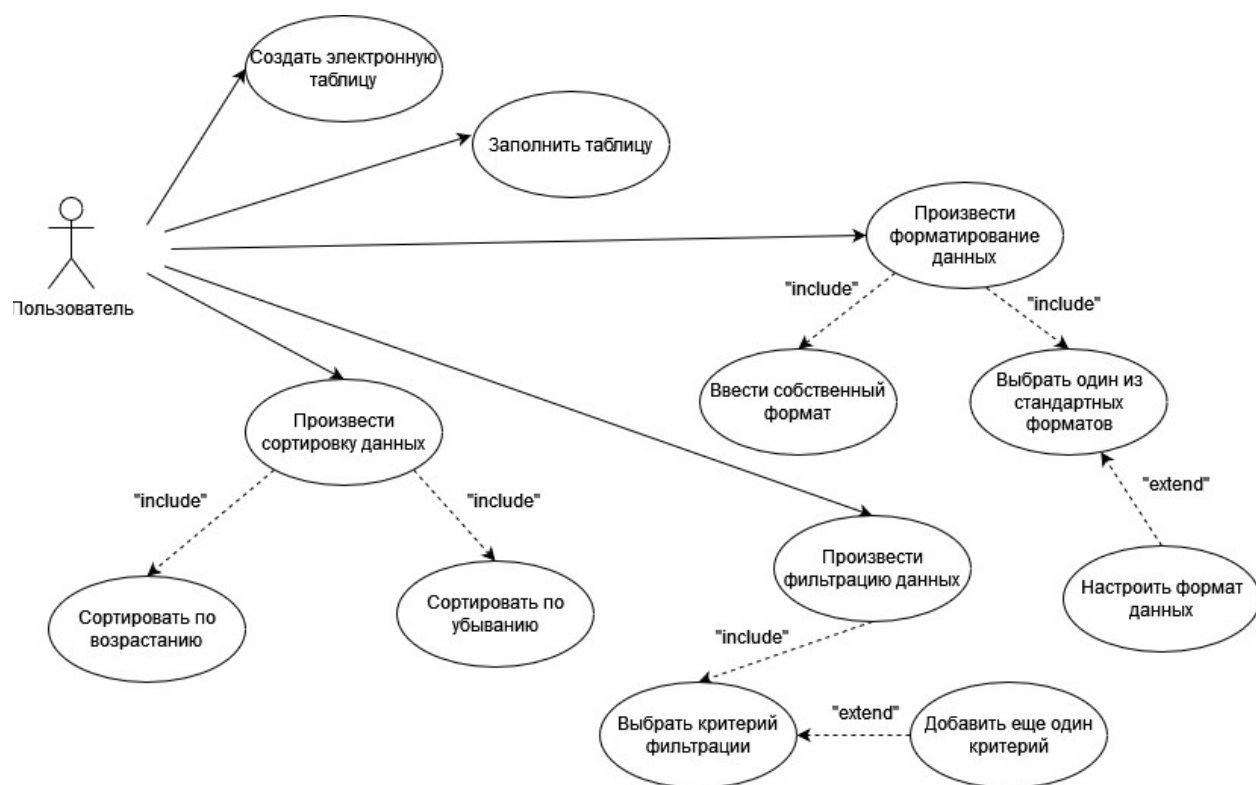


Рисунок 4 - UML-диаграмма вариантов использования

1.2.2 Разработка библиотеки форматирования данных

Условно, разрабатываемый модуль, можно разделить на две части: парсер и часть, отвечающая за форматирование данных.

Взаимодействие с графическим интерфейсом происходит с помощью класса *DefaultFormatter*, реализующий интерфейс *IFormatter*.

Он содержит в себе следующие свойства:

- *Code* – код формата;
- *FontColor* – цвет текста, который необходимо применить к отформатированному значению.

Также класс содержит поле типа *NumberFormat* – это разобранный на секции формат.

Код интерфейса *IFormatter*.

```

public interface IFormatter
{
    string Code { get; }
    Color FontColor { get; set; }
    string Format(string cellValue);
}
  
```

Код класса *DefaultFormatter*:

```
public class DefaultFormatter: IFormatter
{
    public string Code { get; }
    public Color FontColor { get; set; }

    private readonly NumberFormat _format;

    public DefaultFormatter(string formatCode, bool
isRuCode = false)
    {
        Code = formatCode;
        _format = new NumberFormat(formatCode, isRuCode);
    }

    public string Format(string cellValue)
    {
        var res = _format.Format(cellValue,
CultureInfo.CurrentCulture);
        FontColor = res.Color;
        return res.Text;
    }
}
```

Объект типа *DefaultFormatter* планируется хранить как свойство у ячейки, сделано это для реализации возможности кэширования объекта *NumberFormat*.

Для форматирования значение необходимо создать экземпляр класса *DefaultFormatter*, передав в него код применяемого формата. В свою очередь, в его конструкторе будет создан экземпляр класса *NumberFormat*. В конце необходимо вызывать метод *Format*, передав в него форматируемое значение. На выходе пользователь получит строку с отформатированным значением.

Как указывалось ранее, класс *NubmerFormat* содержит одну из секций формата, реализующей интерфейс *ISection*. Парсер секций вызывается в конструкторе данного класса, на выходе получается список объектов типа *Section*.

Класс *Section* содержит в себе несколько свойств:

- *SectionIndex* – индекс секции;
- *Type* – тип секции (элемент перечисления);
- *Color* – цвет, содержащийся в секции. По умолчанию – черный;

– *Condition* – условие, содержащиеся в секции. Реализован в виде отдельного класса, имеющего свойства *Op* (операция) и *Value* (значение). Имеет метод *Apply*, возвращающий результат применения условия. По умолчанию - true;

– одну из дополнительных секций: *Number*, *Exponential*, *Fraction*;

Для разбора секций был разработан класс *TokenReader*, содержащий методы посимвольного чтения строки формата для каждого типа токена.

После прочтения всех токенов, на основании полученных токенов, определяется тип формата, к которому принадлежит переданный в парсер код и и возвращается объект типа *Section*.

При вызове метода *Format* у объекта типа *NumberFormat* первым делом на основе переданного значения происходит определение применяемой секции. Затем у класса *Formatter* вызывается метод *Format*, куда передается значение и секция, в нем происходит форматирование значения по алгоритмам, разработанным для каждого типа формата.

На рисунке 5 приведена UML-диаграмма классов разработанной библиотеки.

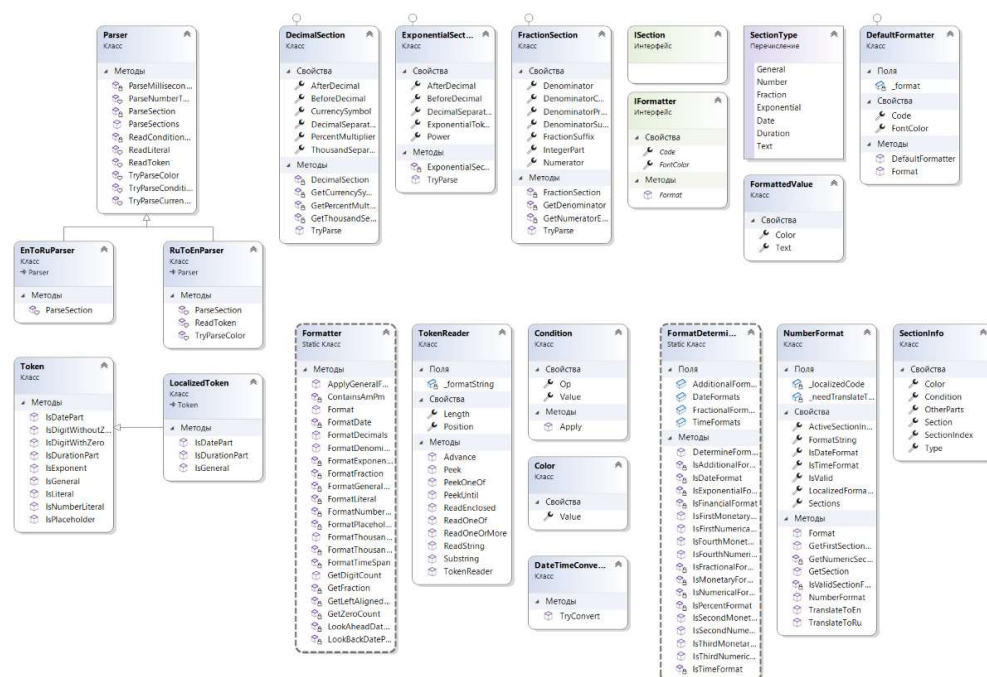


Рисунок 5 - UML диаграмма классов библиотеки форматирования данных

1.2.3 Разработка библиотеки сортировки и фильтрации данных

Разрабатываемый модуль позволяет применять различные фильтры к табличным данным и производить их сортировку.

Основным классом является – *Executer*. Он содержит в себе метод *ApplyFilter()*, применяющий фильтр и метод *Sort()*, выполняющий сортировку.

Вспомогательным классом для передачи информации между интерфейсом и библиотекой является класс *FilterArgs*, содержащий в себе следующие поля:

- *Values* – начальные значения;
- *NameOfFirstFilter* – название первого фильтра;
- *ValueOfFirstFilter* – значение первого фильтра;
- *NameOfSecondFilter* – название второго фильтра;
- *ValueOfSecondFilter* – значение второго фильтра;
- *FilterOp* – операция выполняемая над фильтрами.

Обязательными из них являются только первые два поля.

В метод *ApplyFilter()* передается объект типа *FilterArgs*, затем с помощью созданных ранее словарей и перечислений, определяется к какому типу принадлежит фильтр, после чего вызывается соответствующий метод фильтрации.

Классы *DateTimeFilters*, *NumericalFilters*, *TextFilters* содержат алгоритмы фильтрации для каждого из возможных типов данных. Возвращаемое значение – массив индексов начального массива данных.

На рисунке 6 приведена *UML*-диаграмма классов разработанной библиотеки.

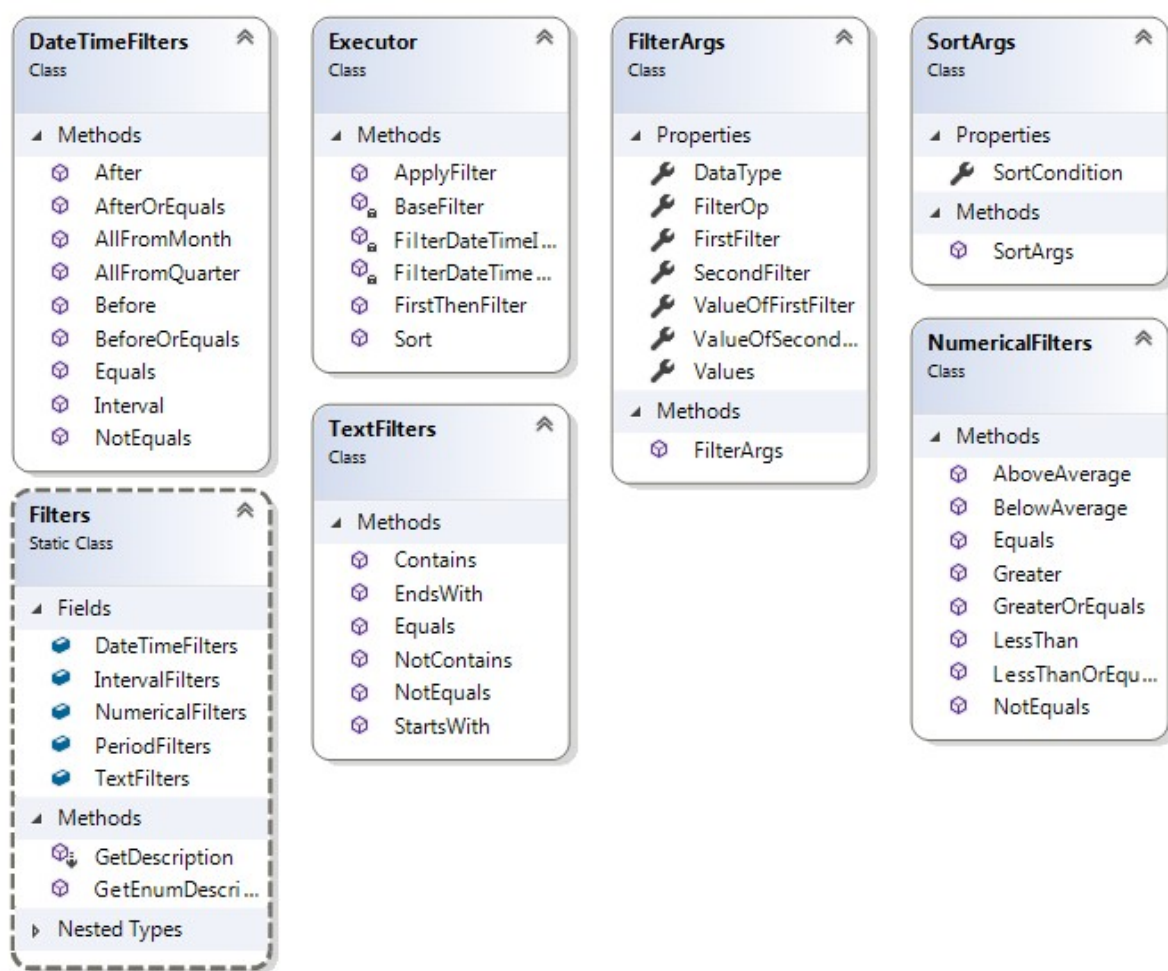


Рисунок 6 - UML-диаграмма классов библиотеки фильтрации и сортировки данных

1.2.4 Разработка графического интерфейса

Для реализации графического интерфейса использовалась платформа *Windows Forms*. В ходе разработки было реализовано два *UI*:

- графический интерфейс для тестирования библиотеки форматирования данных;
- графический интерфейс для тестирования библиотеки сортировки и фильтрации данных.

При их создании не преследовалась цель создать привлекательный и удобный для пользователя интерфейс. Главная цель – это демонстрации работы разработанных модулей. В дальнейшем графический интерфейс будет

разработан с использованием собственных технологий для применения в реальном табличном процессоре.

Основные элементы взаимодействия с программой – это три текстовых поля и список, в котором выбирается формат. В ходе разработки данного интерфейса были созданы 11 пользовательских элементов управления, для каждого типа формата, унаследованных от класса *BaseControl*. В каждом элементе содержатся соответствующие для него элементы управления, позволяющие настроить представление числа.

Форматирование введенного числа происходит после потери фокуса текстового поля, куда вводится начальное значение. Результат отображается в текстовом поле с подписью «Конвертированное число». Текущий код формата отображается в текстовом поле «*Format Code*», куда пользователь также может ввести собственный формат.

На рисунке 7 приведена *UML* диаграмма классов разработанного графического интерфейса.

Графический интерфейс библиотеки сортировки и фильтрации более прост в реализации, так как не требует множества элементов управления и привязки к большому числу событий. Интерфейс довольно прост: в текстовом поле слева вводится начальный массив, а в текстовом поле справа отображается результат сортировки или применения фильтра. Между ними есть кнопка, по нажатию на которую, программа определит тип введенных данных (текст, число, дата) и отобразит для пользователя возможные варианты сортировки и доступные фильтры.

На рисунке 8 представлен графический интерфейс, разработанный для библиотеки форматирования данных.

На рисунке 9 изображено главное окно интерфейса программы, разработанного для проверки работоспособности библиотеки.

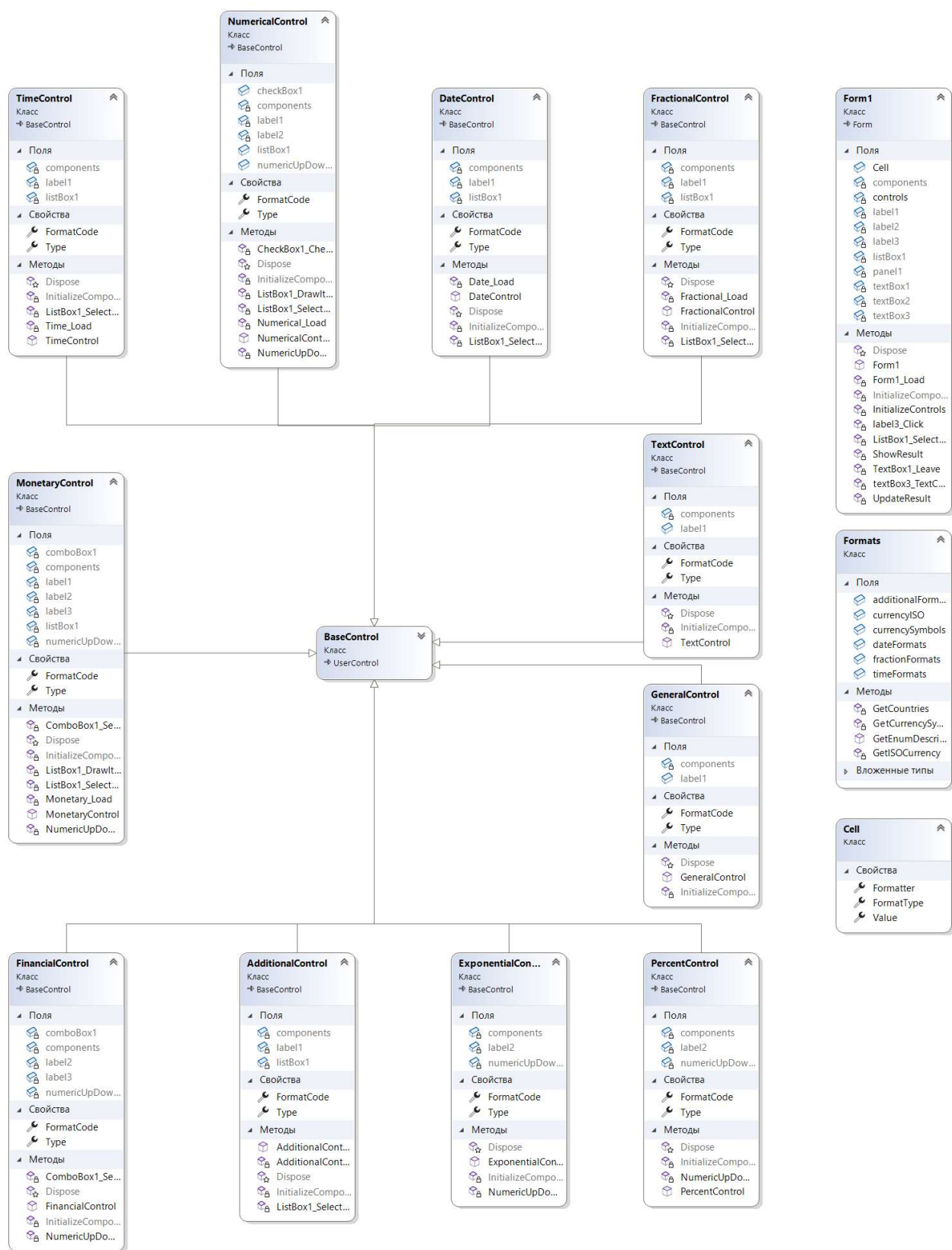


Рисунок 7 - UML диаграмма классов графического интерфейса библиотеки форматирования данных

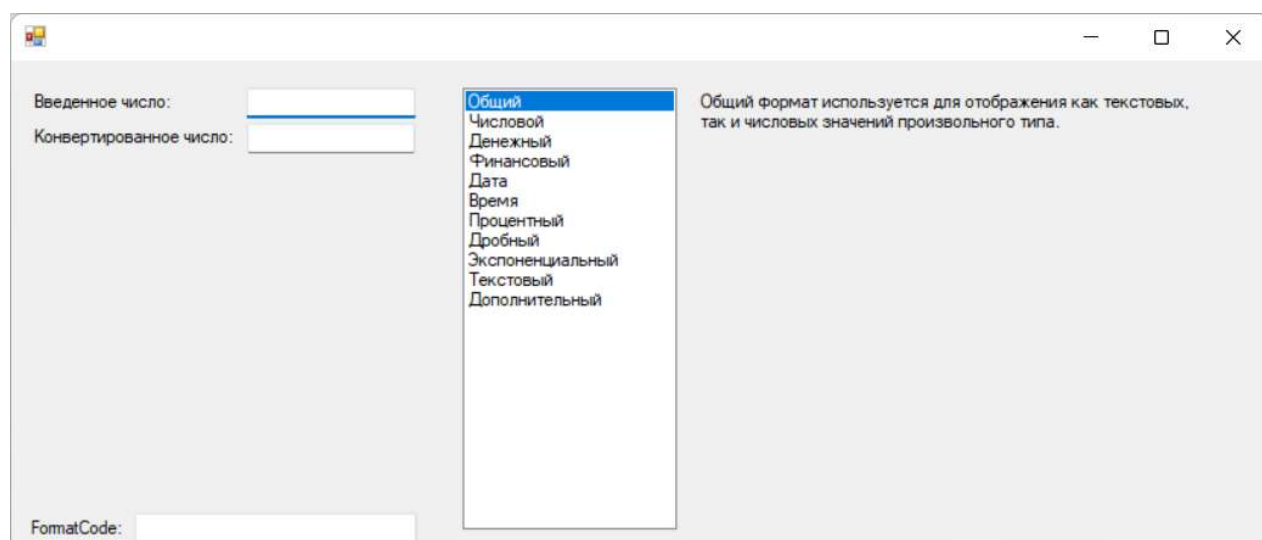


Рисунок 8 - Интерфейс, разработанный для библиотеки форматирования данных

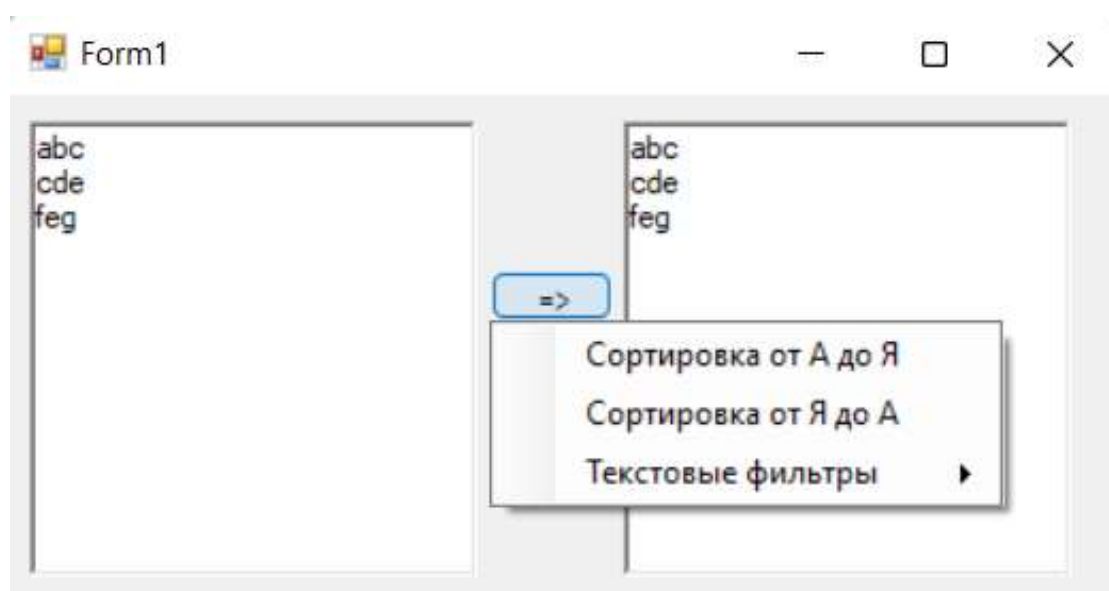


Рисунок 9 - Интерфейс, разработанный для библиотеки сортировки и фильтрации данных

Также для применения комбинированных фильтров была разработана отдельная форма, где выбирается первый и второй фильтр, критерии для них, а также логическая операция, выполняемая над результатами фильтрации.

На рисунке 10 изображено окно выбора фильтров.

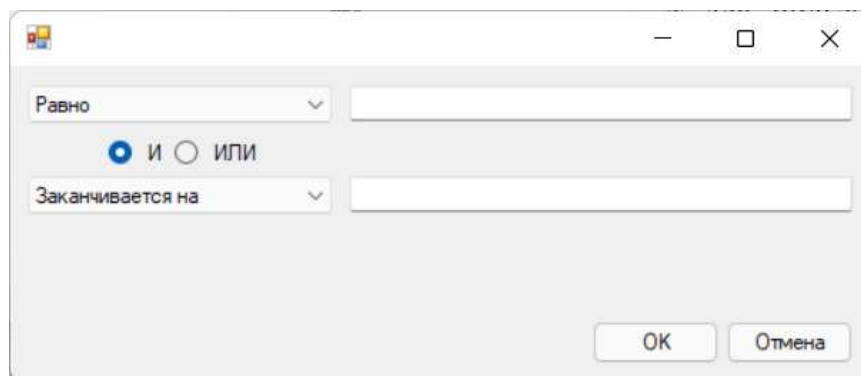


Рисунок 10 - Окно применения нескольких фильтров

На рисунке 11 приведена UML диаграмма классов разработанного графического интерфейса.

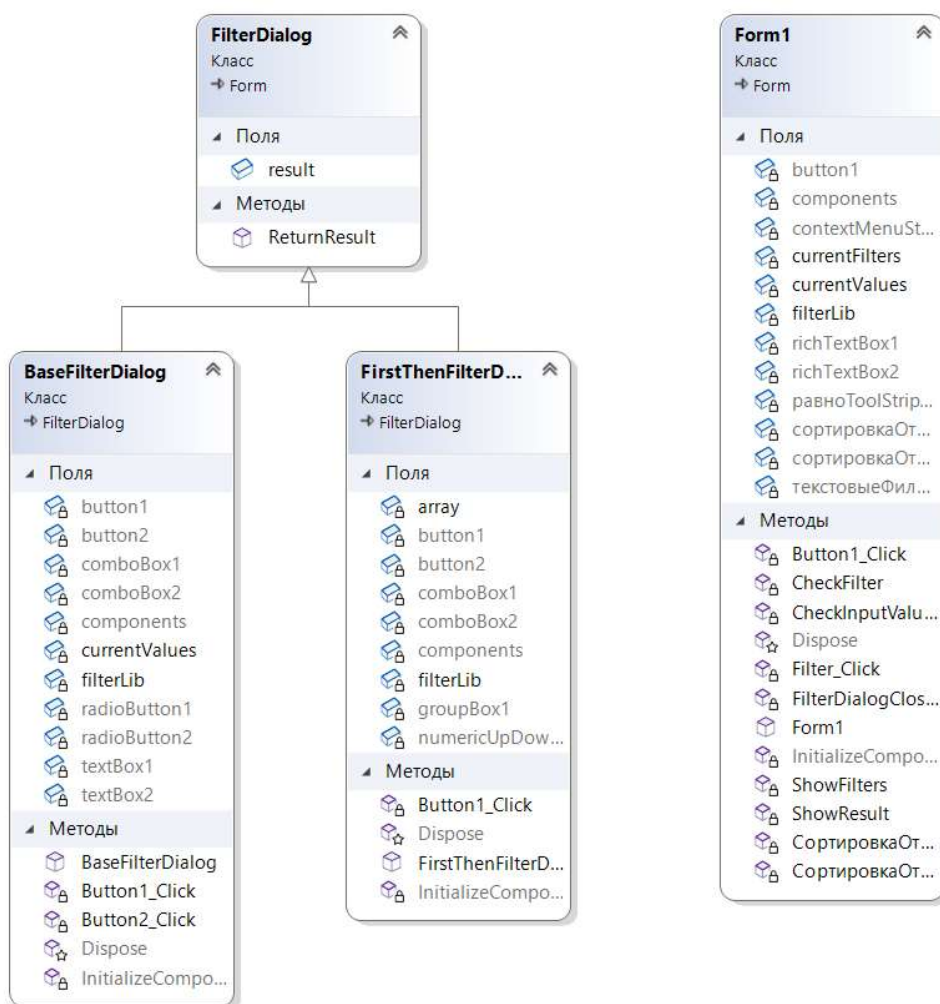


Рисунок 11 - UML диаграмма классов графического интерфейса библиотеки фильтрации и сортировки данных

1.3 Анализ результата

1.3.1 Тестирование и отладка

В ходе разработки программы было произведено функциональное тестирование, в процессе которого было проверено:

– ибблиотека форматирования данных. Применение следующих типов форматов:

- 1) «общий»;
- 2) «числовой»;
- 3) «денежный»;
- 4) «финансовый»;
- 5) «дата»;
- 6) «время»;
- 7) «процентный»;
- 8) «дробный»;
- 9) «экспоненциальный»;
- 10) «текстовый»;
- 11) «дополнительный»;
- 12) введенный вручную пользователем;

– библиотека сортировки и фильтрации данных:

- 1) сортировка текстовых данных;
- 2) сортировка числовых данных;
- 3) сортировка даты и времени;
- 4) фильтрация текстовых данных;
- 5) фильтрация числовых данных;
- 6) фильтрация даты и времени;
- 7) комбинированные фильтры;

Функциональное тестирование выявило проблемы, связанные с отличием в форматировании данных по сравнению с аналогами, ошибки, возникающие

при вводе некорректного пользовательского кода формата, некорректное применение комбинированных фильтров, все они были успешно исправлены.

1.3.1.1 Тестирование библиотеки форматирования данных

Далее приведена таблица 1, содержащая набор тестов, в которых описывается идея теста, входная информация, процедура выполнения, ожидаемый и полученный результаты. После таблицы для каждой проверки прилагается снимок экрана, на котором изображен результат выполнения теста.

Таблица 1 - Тестирование библиотеки форматирования данных

Проверяется: Библиотека форматирования данных					
Документ: ВКР. Пункт: 1					
Дата создания: 26.05.2022				Дата модификации 26.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.1.1	Проверка корректности и работы применения формата «Общий»	Форматируемое значение «123456,789»	Ввести в текстовое поле форматируемое значение, выбрать в списке формат «Общий»	В текстовом поле с результатом появится значение «123456,789»	Совпадает с ожидаемым
ПФ.1.2	Проверка корректности и работы применения формата «Числовой»	Форматируемое значение «-123456,789», выбрано 2 знака после запятой, установлен разделитель разрядов, формат отрицательных второй (Красный без знака)»	Ввести в текстовое поле форматируемое значение, выбрать в списке формат «Числовой», установить 2 знака после запятой и разделитель разрядов, выбрать второй формат	В текстовом поле с результатом появится значение «123 456,79». Цвет текста - красный	Совпадает с ожидаемым

Проверяется: Библиотека форматирования данных

Документ: ВКР. Пункт: 1

Дата создания: 26.05.2022

Дата модификации

26.05.2022

№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
			отрицательных чисел		
ПФ.1.3	Проверка корректности и работы применения формата «Денежный»	Форматируемое значение «123456,789», выбрана валюта «Российский рубль», формат отрицательных чисел без изменений, число знаков после запятой - 0	Ввести в текстовое поле форматируемое значение, выбрать в списке формат «Денежный», выбрать валюту «Российский рубль»	В текстовом поле с результатом появится значение «123 457 Р».	Совпадает с ожидаемым
ПФ.1.4	Проверка корректности и работы применения формата «Финансовый»	Форматируемое значение «123456,789», выбрана валюта «Российский рубль», число знаков после запятой - 2	Ввести в текстовое поле форматируемое значение, выбрать в списке формат «Финансовый», выбрать валюту «Российский рубль», установить число знаков после запятой - 2	В текстовом поле с результатом появится значение «123 456,79 Р».	Совпадает с ожидаемым
ПФ.1.5	Проверка корректности и работы применения формата «Дата»	Форматируемое значение «08.03.2022», выбран тип представления даты «2012, март»	Ввести в текстовое поле форматируемое значение, выбрать в списке тип представления	В текстовом поле с результатом появится значение «2022, Март»	Совпадает с ожидаемым

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

34

Проверяется: Библиотека форматирования данных

Документ: ВКР. Пункт: 1

Дата создания: 26.05.2022				Дата модификации 26.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
			даты «2012, март»		
ПФ.1.6	Проверка корректност и работы применения формата «Время»	Форматируемое значение «17:30», выбран тип представления времени «1:30 PM»	Ввести в текстовое поле форматируемое значение, выбрать в списке тип представления времени «1:30 PM»	В текстовом поле с результатом появится значение «5:30 PM»	Совпадает с ожидаемым
ПФ.1.7	Проверка корректност и работы применения формата «Процентны й»	Форматируемое значение «123456,789», количество знаков после запятой - 0	Ввести в текстовое поле форматируемое значение, установить количество знаков после запятой - 0	В текстовом поле с результатом появится значение «12345679% »	Совпадает с ожидаемым
ПФ.1.8	Проверка корректност и работы применения формата «Дробный»	Форматируемое значение «123456,789», вид представления дроби – «Шестнадцатыми долями»	Ввести в текстовое поле форматируемое значение, установить вид представления дроби – «Шестнадцатым и долями»	В текстовом поле с результатом появится значение «123456 13/16»	Совпадает с ожидаемым
ПФ.1.9	Проверка корректност и работы применения формата	Форматируемое значение «123456,789», количество знаков после запятой - 5	Ввести в текстовое поле форматируемое значение, установить	В текстовом поле с результатом появится значение	Совпадает с ожидаемым

Проверяется: Библиотека форматирования данных					
Документ: ВКР. Пункт: 1					
Дата создания: 26.05.2022				Дата модификации 26.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
	«Экспоненциальный»		количество знаков после запятой - 5	«1,23457E+05»	
ПФ.1.1 0	Проверка корректности работы применения формата «Текстовый»	Форматируемое значение «abcdefg»	Ввести в текстовое поле форматируемое значение	В текстовом поле с результатом появится значение «abcdefg»	Совпадает с ожидаемым
ПФ.1.1 1	Проверка корректности работы применения формата «Дополнительный»	Форматируемое значение «1234567890», тип формата «Номер телефона»	Ввести в текстовое поле форматируемое значение, установить тип формата «Номер телефона»	В текстовом поле с результатом появится значение «(123) 456-7890»	Совпадает с ожидаемым
ПФ.1.1 2	Проверка корректности работы применения формата, введенного пользователем	Форматируемое значение «12345,6», код формата «[Красный][>100] #.0»	Ввести в текстовое поле форматируемое значение, в текстовое поле с кодом формата ввести код формата	В текстовом поле с результатом появится значение «12345,6». Цвет текста - красный	Совпадает с ожидаемым

Исходя из полученных результатов, можно сделать вывод, что разработанная библиотека форматирования данных работает корректно и отвечает всем поставленным требованиям. Далее приведены снимки экранов, подтверждающие корректное выполнение каждого из тестов. Каждый из них имеет подпись с номером теста, к которому он относится.

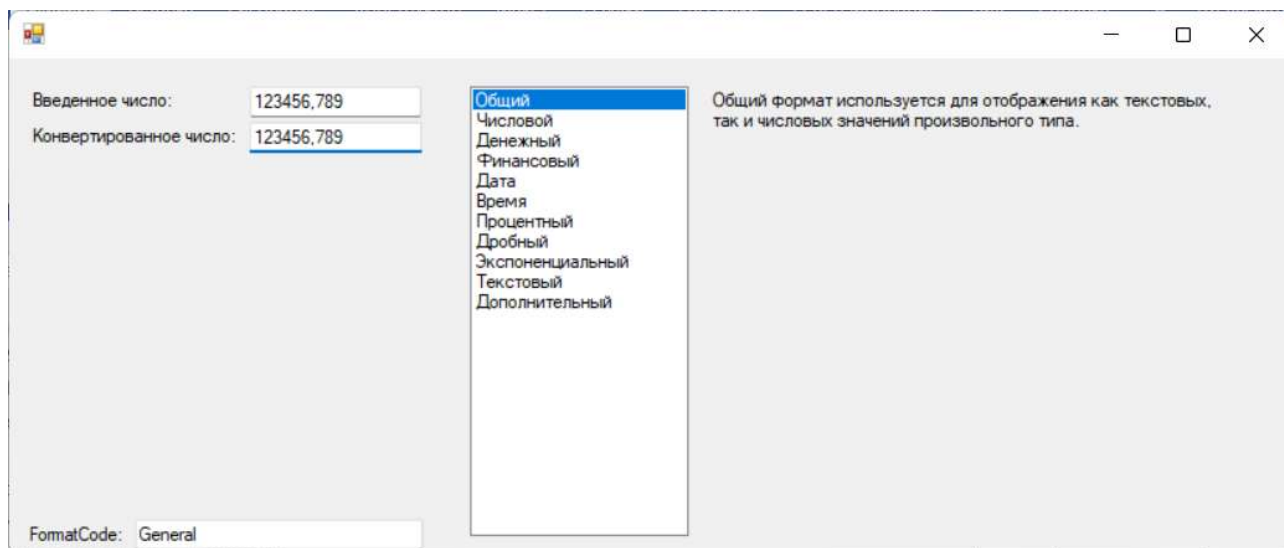


Рисунок 12 - ПФ.1.1

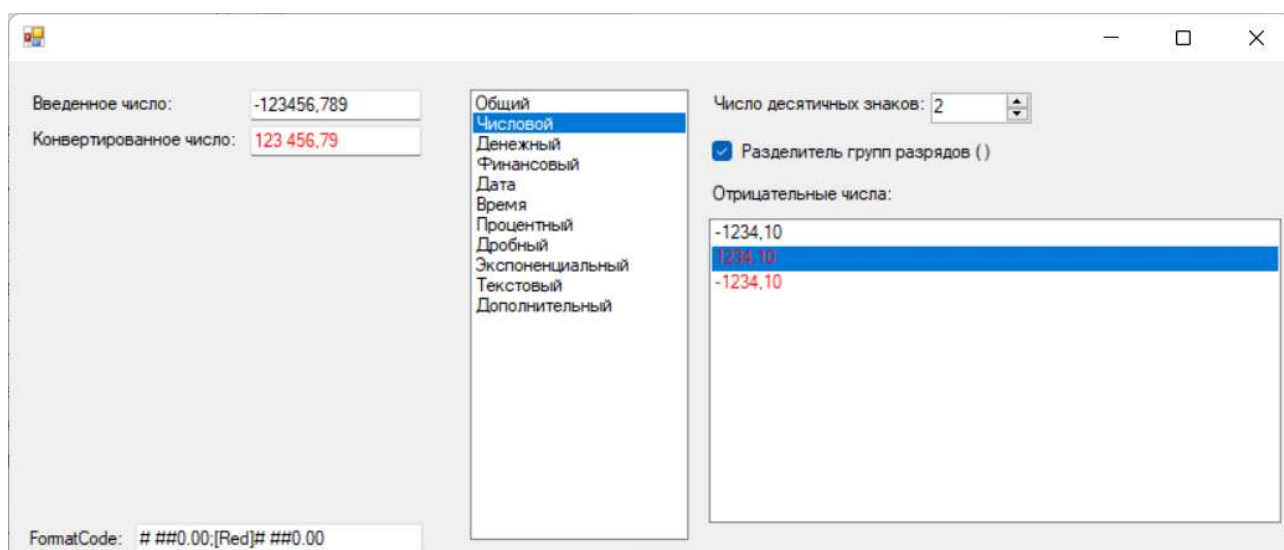


Рисунок 13 - ПФ.1.2

Введенное число:	123456,789	Общий Числовой Денежный Финансовый Дата Время Процентный Дробный Экспоненциальный Текстовый Дополнительный	Число десятичных знаков:	0
Конвертированное число:	123 457 P		Обозначение:	Russian Ruble
FormatCode: ##0 P		Отрицательные числа: -1 234,10 1 234,10 -1 234,10		

Рисунок 14 - ПФ.1.3

Введенное число:	123456,789	Общий Числовой Денежный Финансовый Дата Время Процентный Дробный Экспоненциальный Текстовый Дополнительный	Число десятичных знаков:	2
Конвертированное число:	123 456,79 P		Обозначение:	Russian Ruble
FormatCode: _* ##0.00 P_:_* ##0.00 P_:_* "				

Рисунок 15 - ПФ.1.4

Введенное число:	08.03.2022	Общий Числовой Денежный Финансовый Дата Время Процентный Дробный Экспоненциальный Текстовый Дополнительный	Тип:
Конвертированное число:	2022, Март		2012-03-1 1.3 1.3.12 1.03.12 1 мар 1 мар 12 01 мар 12 2012, март Март 2012 01 марта 2012 г. 01.03.12 1:30 PM 01.03.12 13:30 Март 2012, 14 марта 1.3.2012 1-мар-2012
FormatCode: yyyy, MMMM			

Рисунок 16 - ПФ.1.5

Введенное число:	17:30	Общий Числовой Денежный Финансовый Дата Время Процентный Дробный Экспоненциальный Текстовый Дополнительный	Тип:
Конвертированное число:	5:30 PM		13:30 1:30 PM 13:30:55 1:30:55 PM 30:55.2 37:30:55 14.03.12 1:30 PM 14.03.12 13:30
FormatCode: h:mm am/pm			

Рисунок 17 - ПФ.1.6

Введенное число: 123456,789

Конвертированное число: 12345679%

Число десятичных знаков: 0

FormatCode: 0%

- Общий
- Числовой
- Денежный
- Финансовый
- Дата
- Время
- Процентный**
- Дробный
- Экспоненциальный
- Текстовый
- Дополнительный

Рисунок 18 - ПФ.1.7

Введенное число: 123456,789

Конвертированное число: 123456 13/16

FormatCode: # " ? / 16

- Общий
- Числовой
- Денежный
- Финансовый
- Дата
- Время
- Процентный
- Дробный**
- Экспоненциальный
- Текстовый
- Дополнительный

Тип:

- Простыми дробями (1/4)
- Дробями до двух цифр (21/25)
- Дробями до трех цифр (312/948)
- Половинными долями (1/2)
- Четвертными долями (2/4)
- Восьмыми долями (4/8)
- Шестнадцатыми долями (8/16)**
- Десятыми долями (3/10)
- Сотыми долями (30/100)

Рисунок 19 - ПФ.1.8

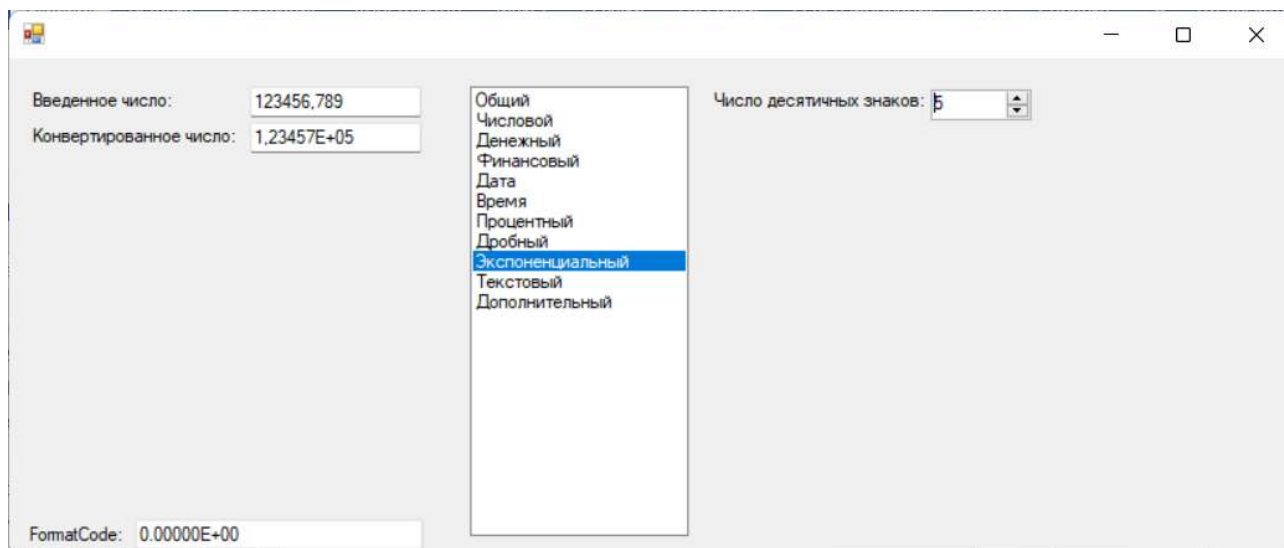


Рисунок 20 - ПФ.1.9

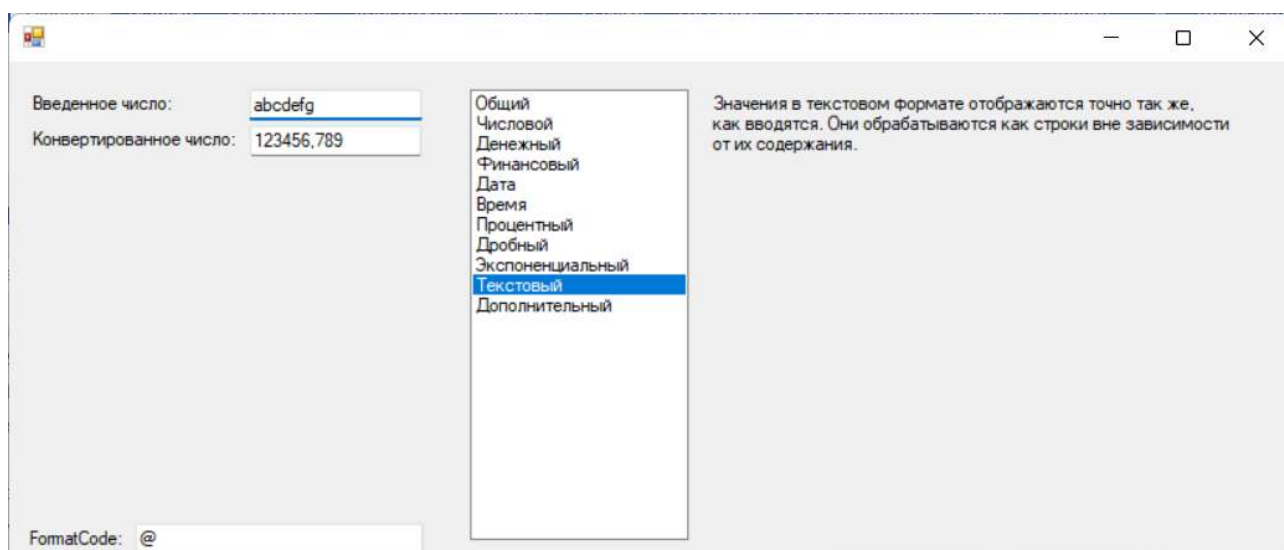


Рисунок 21 - ПФ1.10

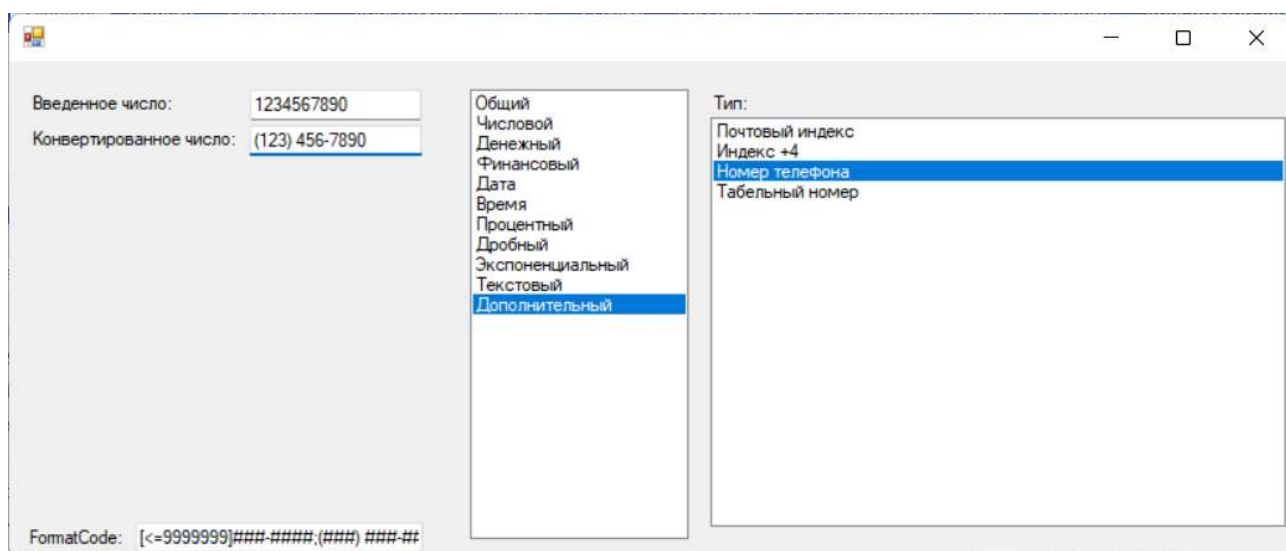


Рисунок 22 - ПФ1.11

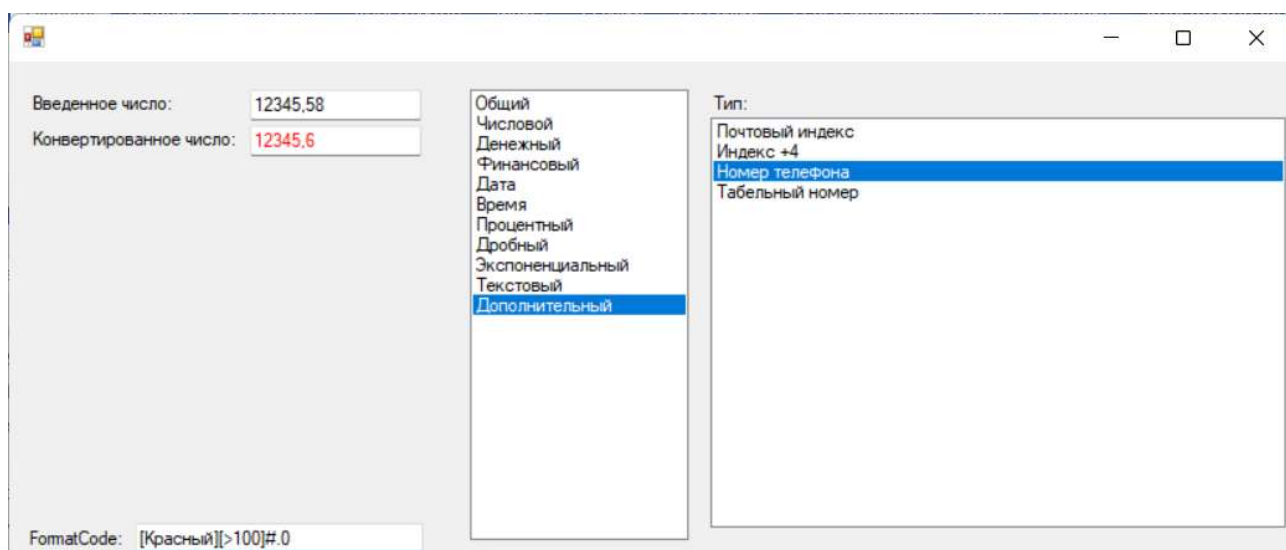


Рисунок 23 - ПФ.1.12

1.3.1.2 Тестирование библиотеки сортировки и фильтрации данных

Далее приведена таблица, содержащая набор тестов, в которых описывается идея теста, входная информация, процедура выполнения, ожидаемый и полученный результаты. После таблицы для каждой проверки прилагается снимок экрана, на котором изображен результат выполнения теста.

Таблица 2 - Тестирование библиотеки сортировки и фильтрации данных

Проверяется: Библиотека сортировки и фильтрации данных Документ: ВКР. Пункт: 2					
Дата создания: 27.05.2022				Дата модификации 27.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.2.1	Сортировка текстовых данных	Массив текстовых данных def abc zax efg klm	Ввести в текстовое поле массив, выбрать сортировку от «А до Я»	В текстовом поле с результатом появится текст «abc def efg klm zax»	Совпадает с ожидаемым
ПФ.2.2	Сортировка числовых данных	Массив числовых данных 543 123 432 999 875	Ввести в текстовое поле массив, выбрать сортировку по убыванию	В текстовом поле с результатом появится текст «999 875 543 432 123»	Совпадает с ожидаемым
ПФ.2.3	Сортировка даты и времени	Массив значений даты 09.12.2000 12.12.1998 08.03.2022 08.02.2022 01.01.2000	Ввести в текстовое поле массив, выбрать сортировку «От старых к новым»	В текстовом поле с результатом появится текст «12.12.1998 01.01.2000 09.12.2000 08.02.2022 08.03.2022»	Совпадает с ожидаемым
ПФ.2.4	Фильтрация текстовых данных	Массив текстовых	Ввести в текстовое поле массив,	В текстовом поле с	Совпадает с ожидаемым

Проверяется: Библиотека сортировки и фильтрации данных

Документ: ВКР. Пункт: 2

Дата создания: 27.05.2022				Дата модификации 27.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
		данных abc def aef jkl qwe anm	выбрать «Начинается С», в текстовом поле напротив названия фильтра ввести букву «а», нажать «ОК»	результатом появится текст «abc aef anm»	
ПФ.2.5	Фильтрация числовых данных	Массив числовых данных 123 8 54 987 342 11 17	Ввести в текстовое поле массив, выбрать фильтр «Больше», напротив названия фильтра ввести число «100», нажать «ОК»	В текстовом поле с результатом появится текст «123 987 342»	Совпадает с ожидаемым
ПФ.2.6	Фильтрация даты и времени	Массив значений даты 09.12.2000 12.12.1998 08.03.2022 08.02.2022 01.01.2000	Ввести в текстовое поле массив, выбрать фильтр «В этом году»	В текстовом поле с результатом появится текст «08.03.2022 08.02.2022»	Совпадает с ожидаемым
ПФ.2.7	Комбинированные фильтры	Массив значений даты 09.12.2000 12.12.1998 08.03.2022 08.02.2022 01.01.2000	Ввести в текстовое поле массив, выбрать фильтр «После», напротив названия фильтра ввести «02.01.2000», установить радио-	В текстовом поле с результатом появится текст «09.12.2000 08.02.2022»	Совпадает с ожидаемым

Проверяется: Библиотека сортировки и фильтрации данных					
Документ: ВКР. Пункт: 2					
Дата создания: 27.05.2022				Дата модификации 27.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
			кнопку «И», выбрать второй фильтр «До», напротив названия фильтра ввести «01.03.2022», нажать «ОК»		

Исходя из полученных результатов, можно сделать вывод, что разработанная библиотека сортировки и фильтрации данных работает корректно и отвечает всем поставленным требованиям. Далее приведены рисунки, подтверждающие корректное выполнение каждого из тестов. Каждый из них имеет подпись с номером теста, к которому он относится.

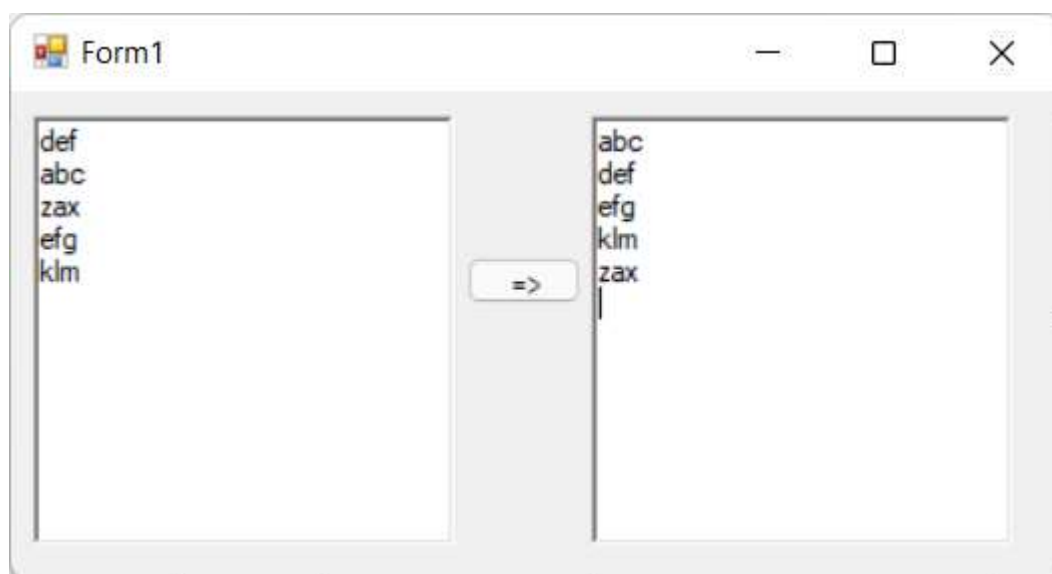


Рисунок 24 - ПФ.2.1

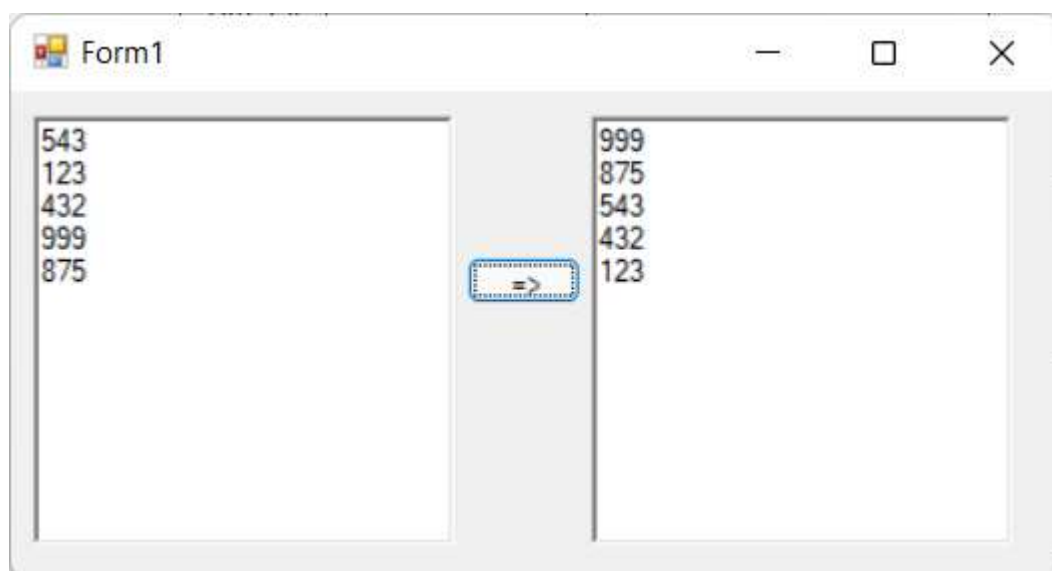


Рисунок 25 - ПФ.2.2

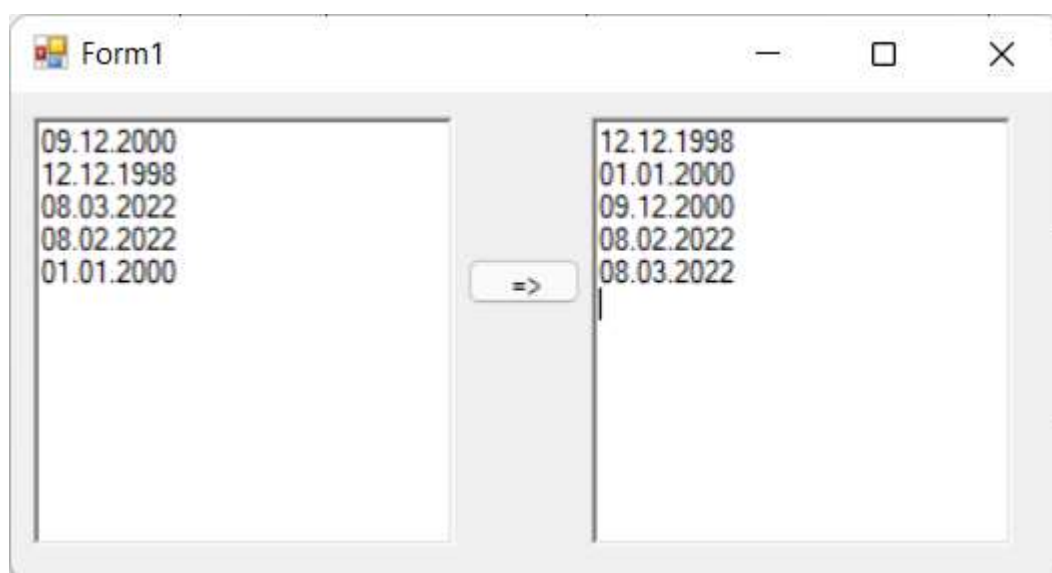


Рисунок 26 - ПФ.2.3

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

46

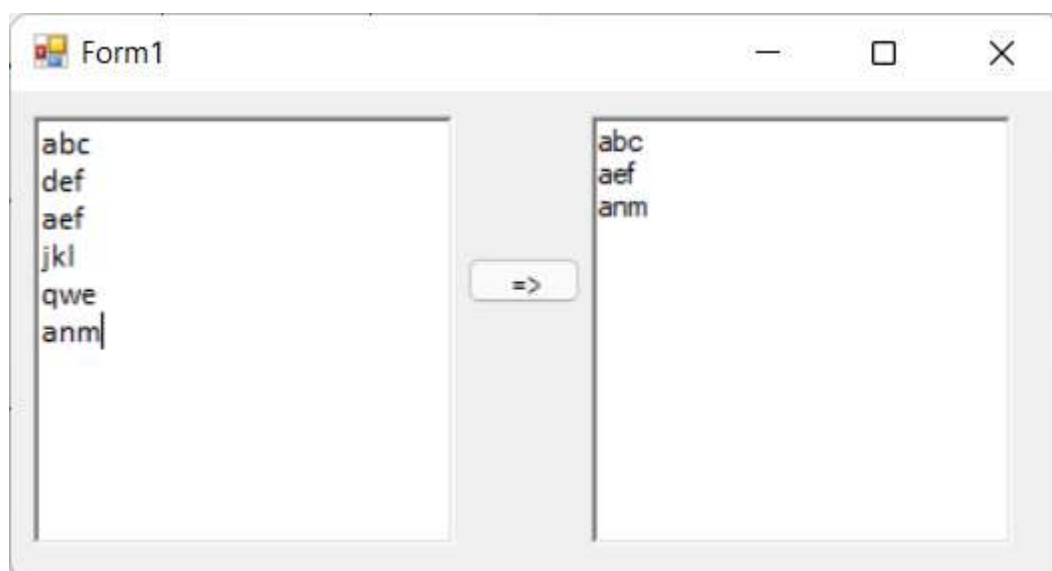


Рисунок 27 - ПФ.2.4

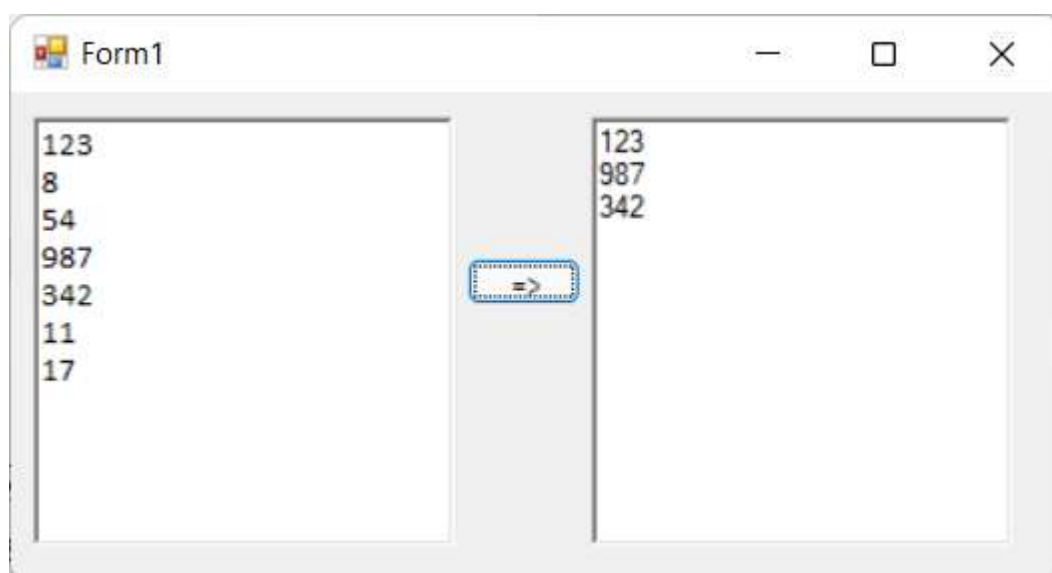


Рисунок 28 - ПФ.2.5

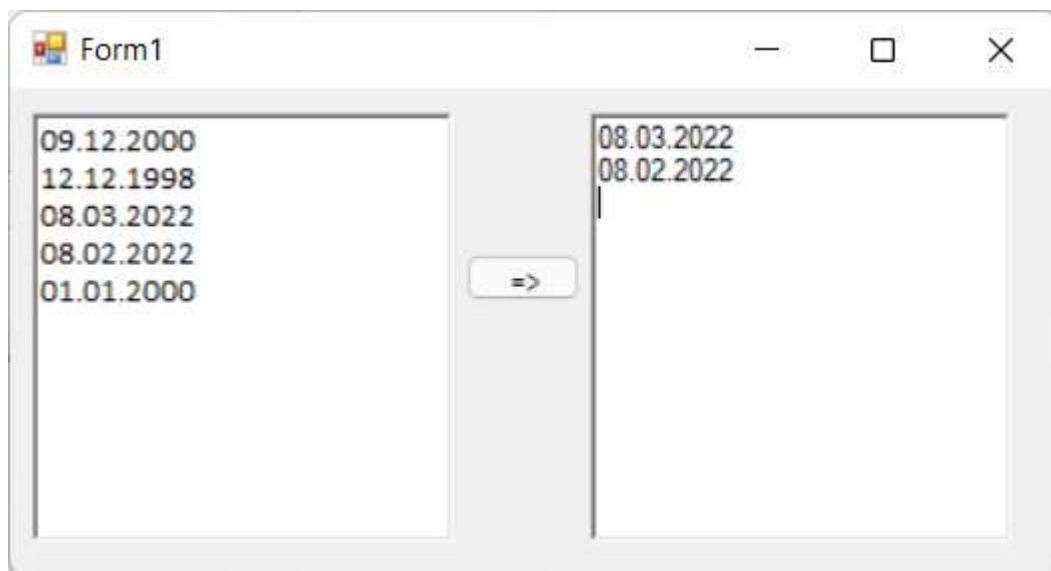


Рисунок 29 - ПФ.2.6

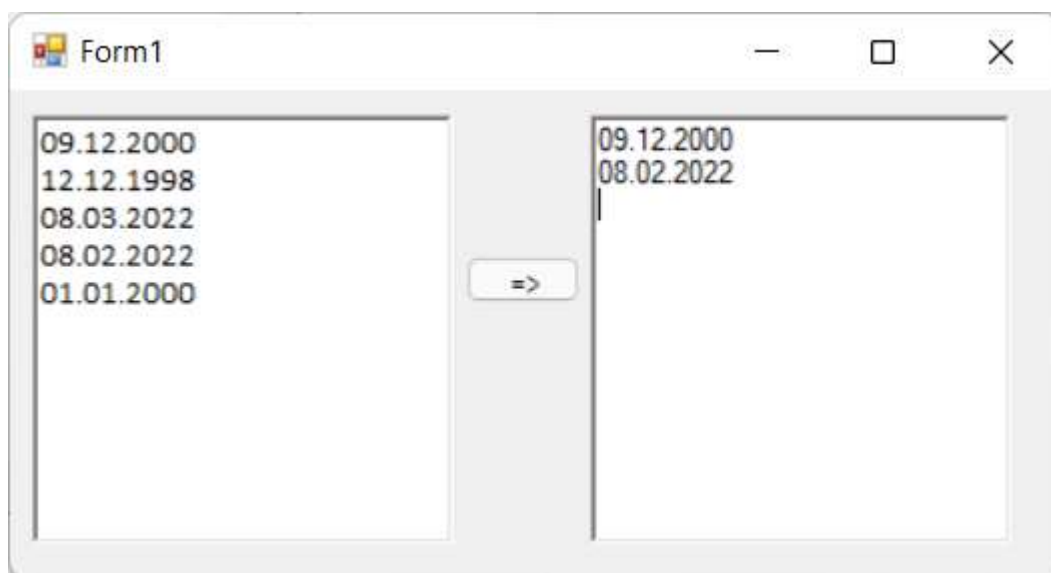


Рисунок 30 - ПФ.2.7

1.3.2 Тестирование в табличном процессоре «Ведомость»

Главным итогом выполнения ВКР, является интеграция модуля форматирования данных в табличный процессор «Ведомость», разрабатываемый научно-техническим предприятием «Криптософт».

Разработанный модуль соответствует всем поставленным требованиям и ничем не уступает по функционалу аналогам. Было принято решение оставить

графический интерфейс неизменным для удобства работы пользователей. Форматирование данных также происходит полностью корректно.

По итогам интеграции удалось добиться полной совместимости со стандартном *Office Open Xml*, что означает корректное считывание таблиц, созданных в других табличных редакторах, а также считывание ими таблиц созданных в Ведомости.

Далее представлены снимки экрана из табличного процессора, разрабатываемого НТП «Криптософт».

На рисунке 31 изображено главное окно табличного процессора, в красной рамке выделены элементы управления форматом данных.

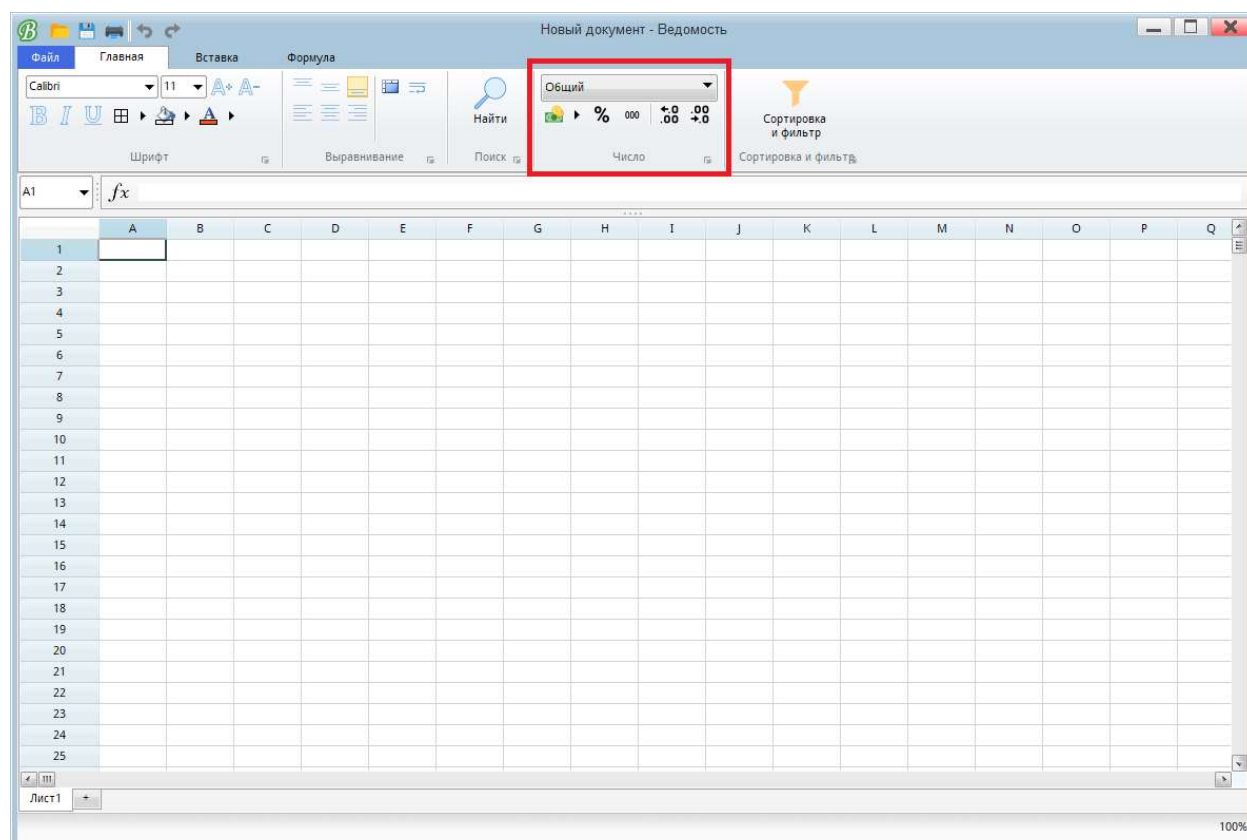


Рисунок 31 - Главное окно табличного процессора «Ведомость»

В выпадающем списке пользователь может выбрать один из форматов данных, не открывая окно форматирования. На рисунке 32 приведена структура данного списка:

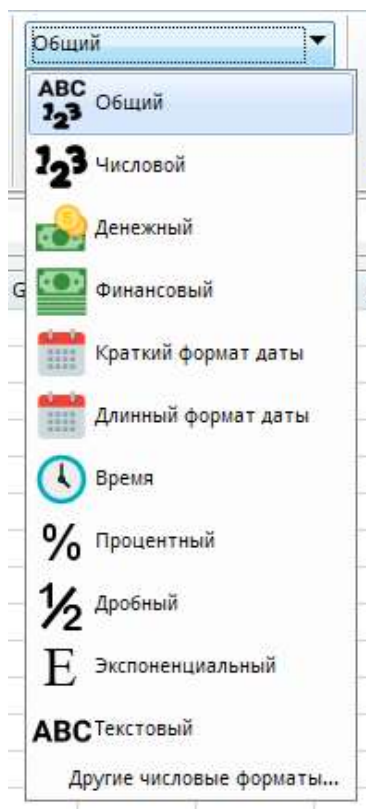


Рисунок 32 - Выпадающий список, содержащий форматы

Кнопки, находящиеся под выпадающим списком, отвечают за следующий функционал:

- выбор формата денежной единицы;
- отображение значение в процентах, отображение значения с разделителем групп разрядов;
- отображение более точного значения за счет увеличения числа знаков после запятой;
- отображение менее точного значения за счет уменьшения числа знаков после запятой.

На снимках экрана ниже приведены примеры форматирования данных для каждого из форматов.

На рисунке 33 изображен результат применения формата типа «Общий» к ячейке A1.

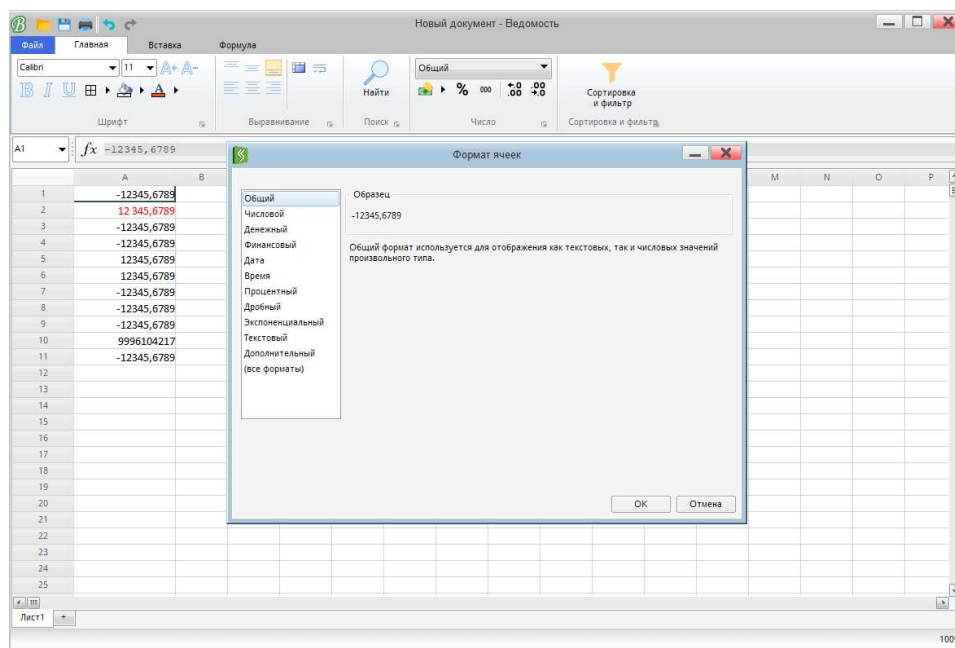


Рисунок 33 - Ведомость. Формат «Общий»

На рисунке 34 изображен результат применения к ячейке A2 формата типа «Числовой» с количеством десятичных знаков 4, разделителем и форматом для отрицательных чисел с красным цветом без знака «-».

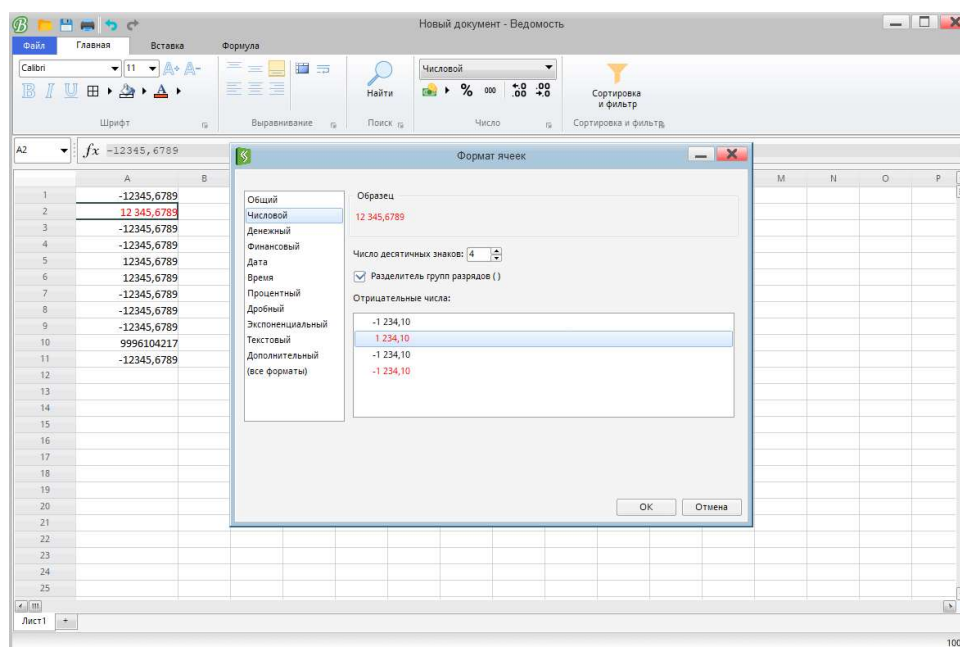


Рисунок 34 - Ведомость. Формат «Числовой»

На рисунке 35 изображен результат применения к ячейке A3 формата типа «Денежный» с количеством десятичных знаков 4, валютой евро и

форматом для отрицательных чисел с красным цветом без знака «-».

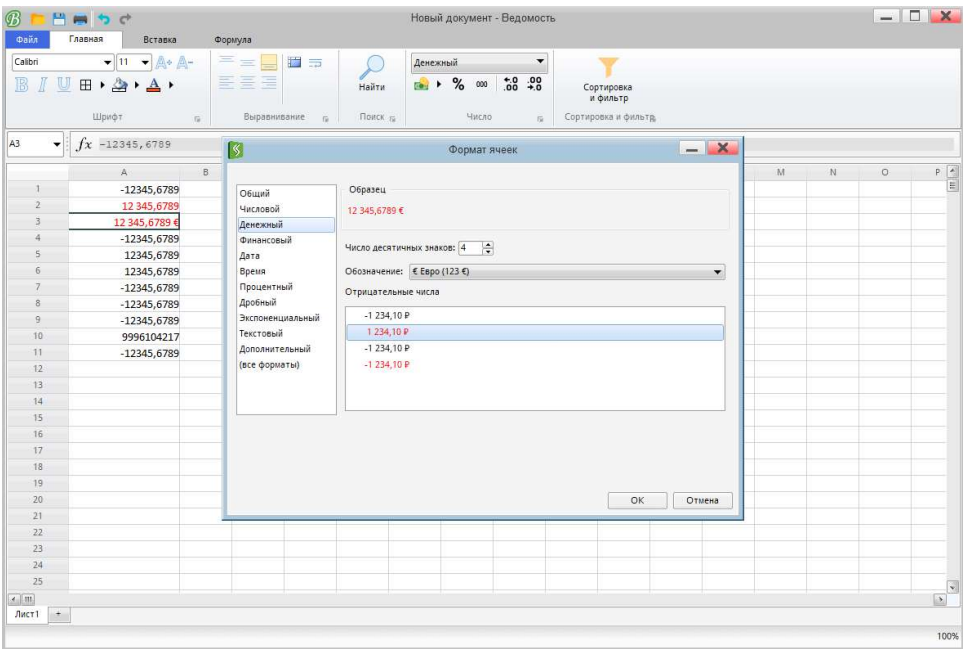


Рисунок 35 - Ведомость. Формат «Денежный»

На рисунке 36 изображен результат применения к ячейке A4 формата типа «Финансовый» с количеством десятичных знаков 5 и валютой доллар.

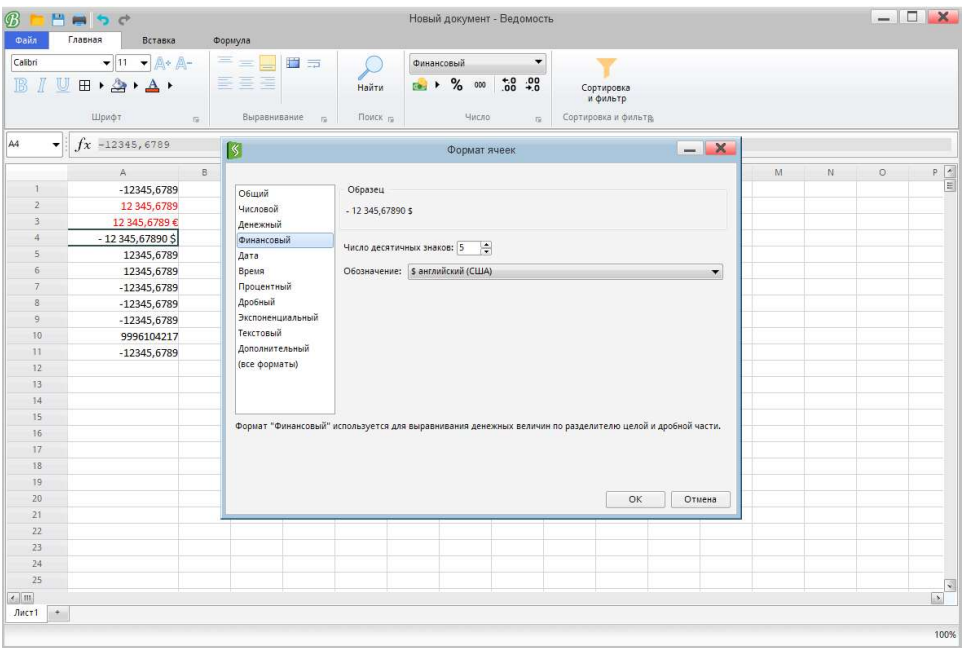


Рисунок 36 - Ведомость. Формат «Финансовый»

На рисунке 37 изображен результат применения к ячейке A5 формата типа «Дата» вида «ДД-МММ-ГГГГ».

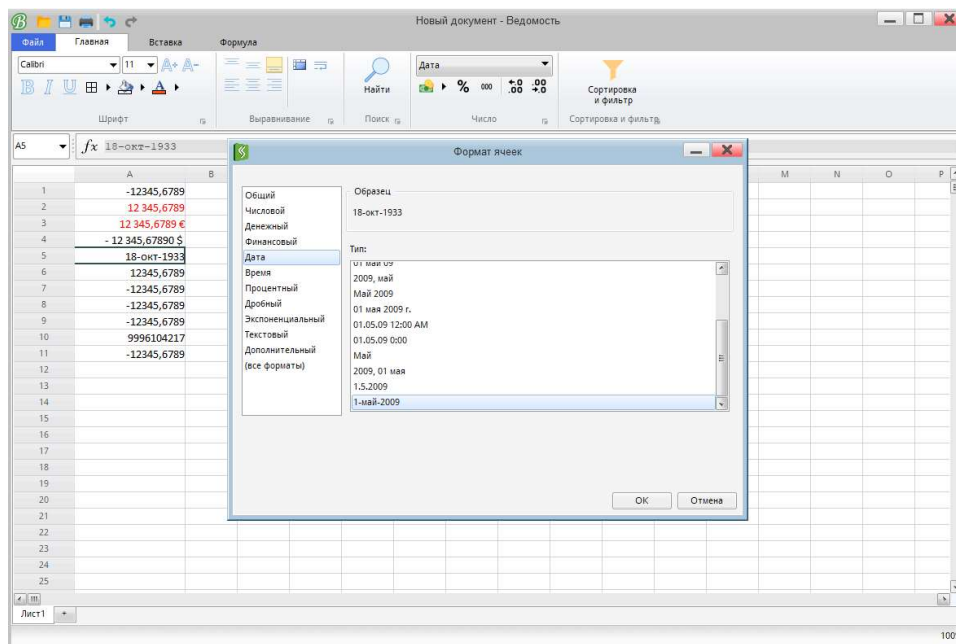


Рисунок 37 - Ведомость. Формат «Дата»

На рисунке 38 изображен результат применения к ячейке A6 формата типа «Время» вида «hh:mm AM/PM».

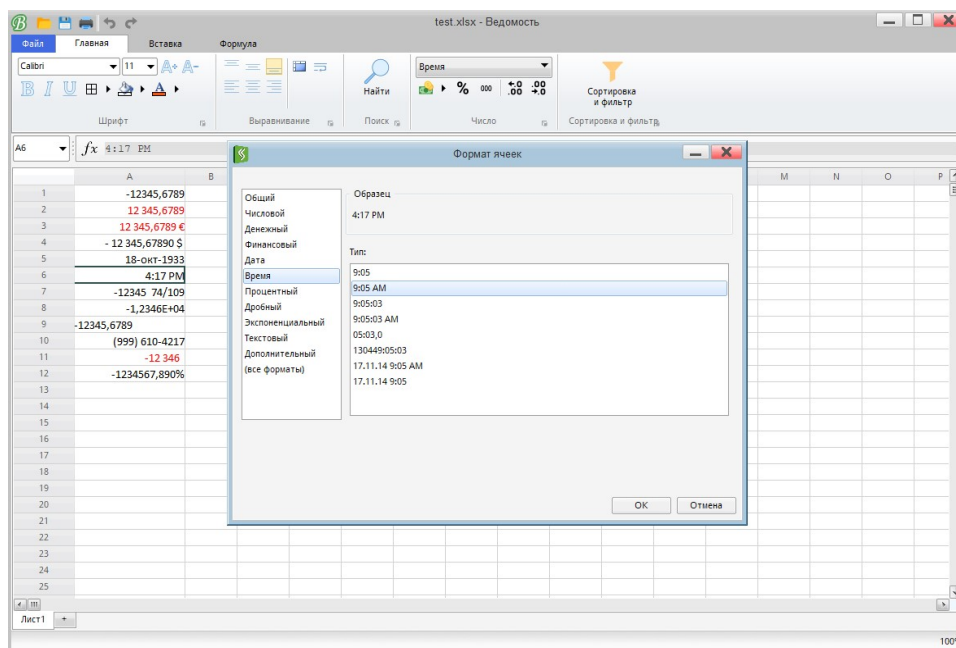


Рисунок 38 - Ведомость. Формат «Время»

На рисунке 39 изображен результат применения к ячейке A12 формата типа «Процентный» с количеством десятичных знаков 3.

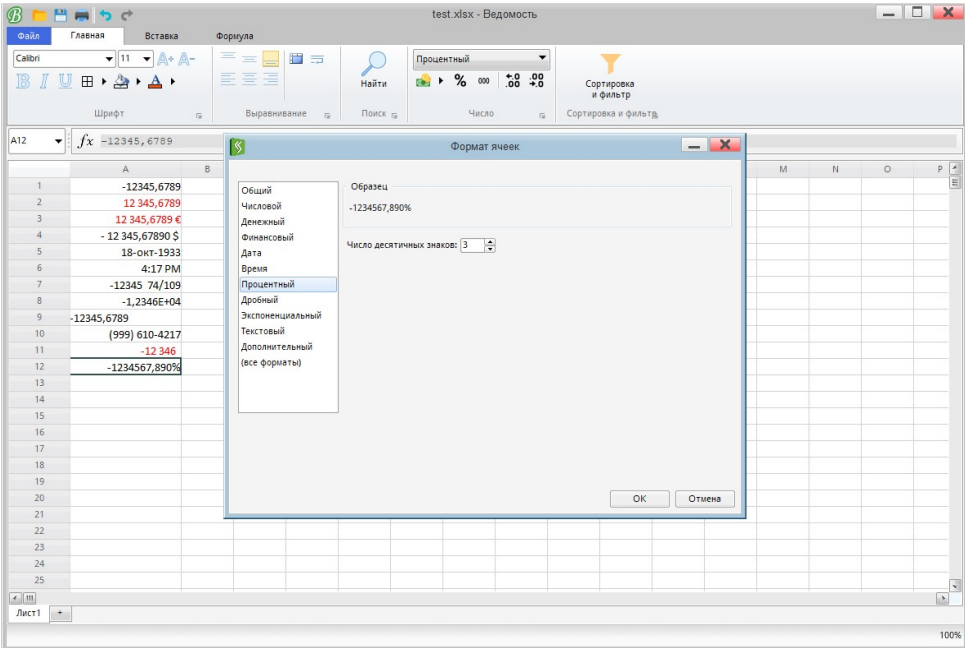


Рисунок 39 - Ведомость. Формат «Процентный»

На рисунке 40 изображен результат применения к ячейке A7 формата типа «Дробный» вида «Дробями до трех цифр».

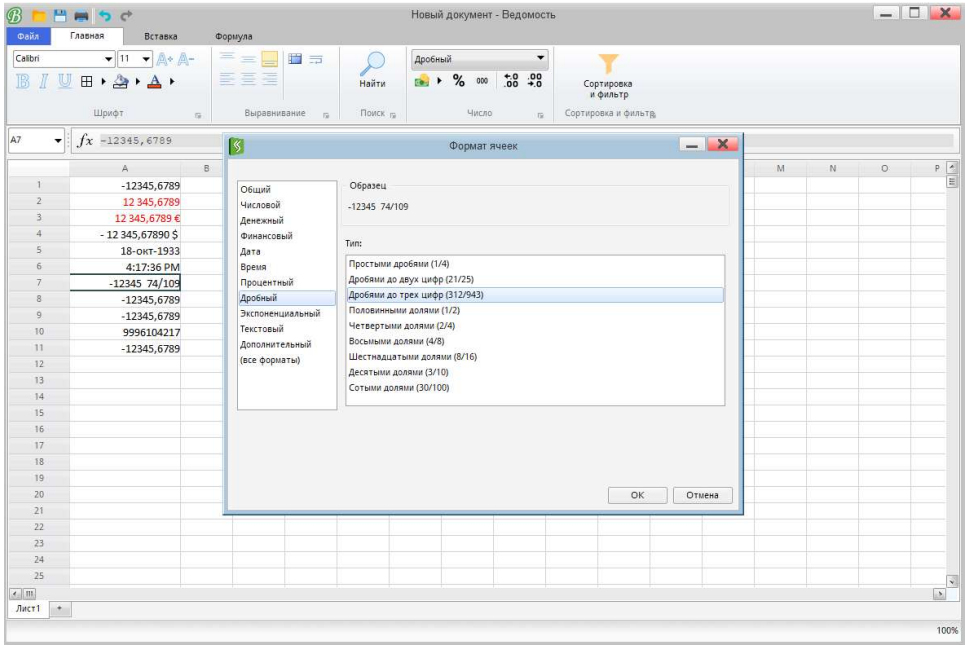


Рисунок 40 - Ведомость. Формат «Дробный»

На рисунке 41 изображен результат применения к ячейке A8 формата типа «Экспоненциальный» с количеством десятичных знаков 4.

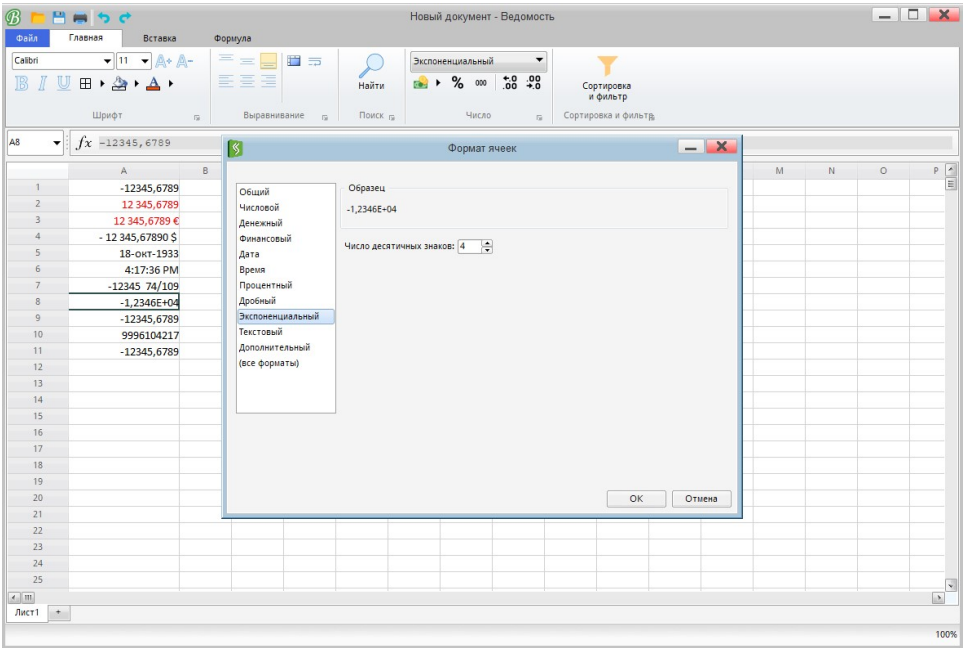


Рисунок 41 - Ведомость. Формат «Экспоненциальный»

На рисунке 42 изображен результат применения к ячейке A9 формата типа «Текстовый».

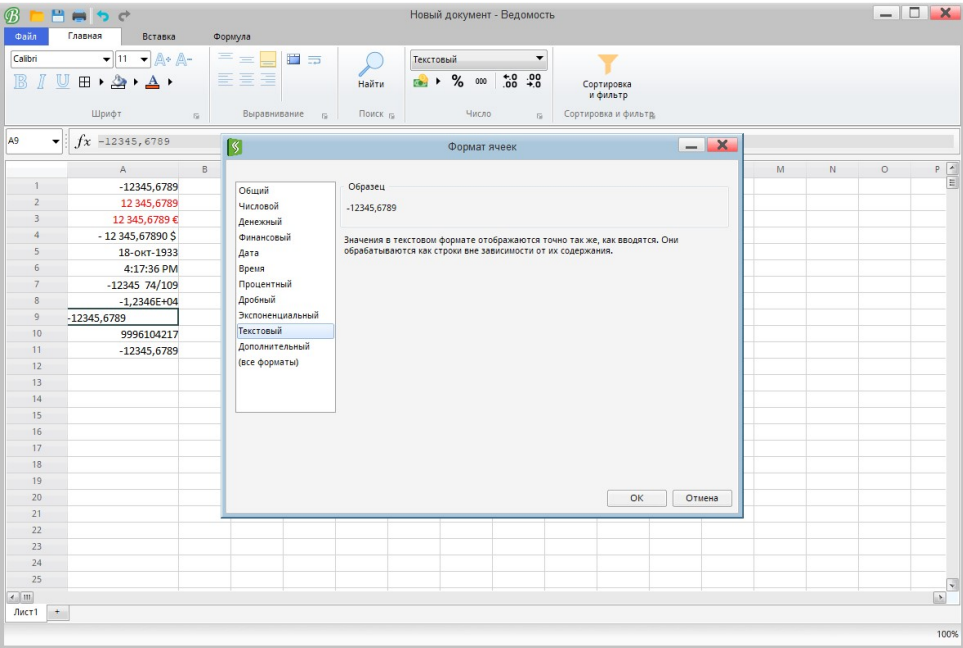


Рисунок 42 - Ведомость. Формат «Текстовый»

На рисунке 43 изображен результат применения к ячейке A10 формата типа «Дополнительный» вида «Номер телефона».

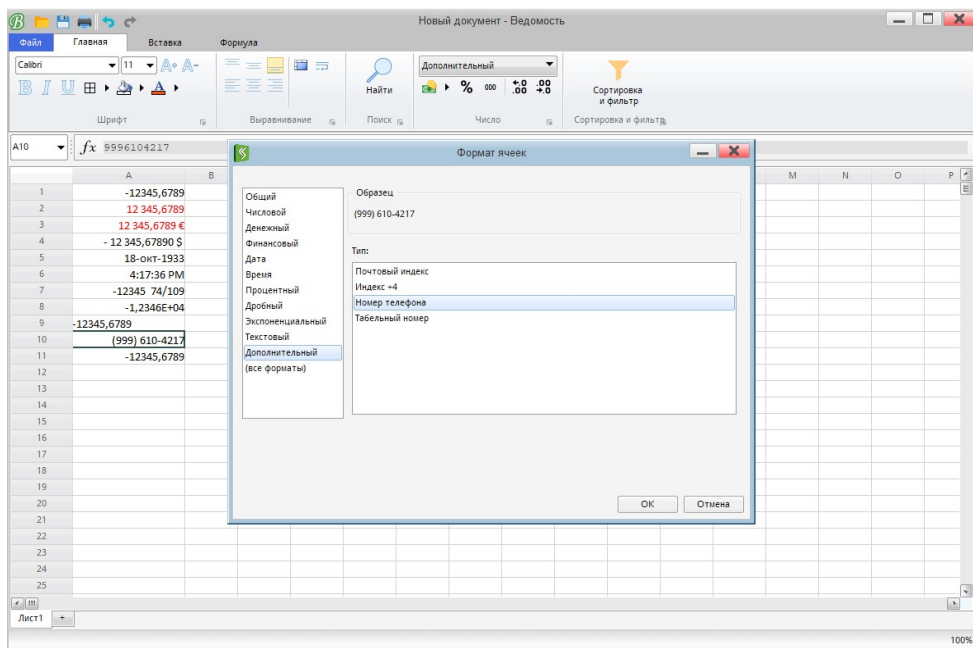


Рисунок 43 - Ведомость. Формат «Дополнительный»

На рисунке 44 изображен результат применения к ячейке A11 пользовательского кода формата.

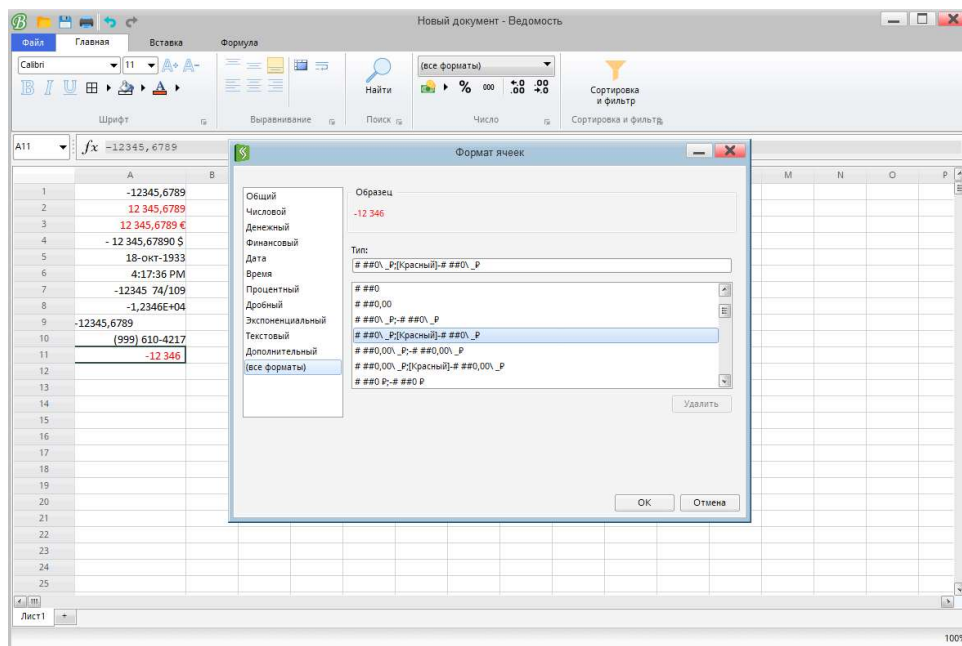


Рисунок 44 - Ведомость. Пользовательский формат

Конечный документ после применения всех форматов к диапазону ячеек A1-A13 имеет вид, изображенный на рисунке 45.

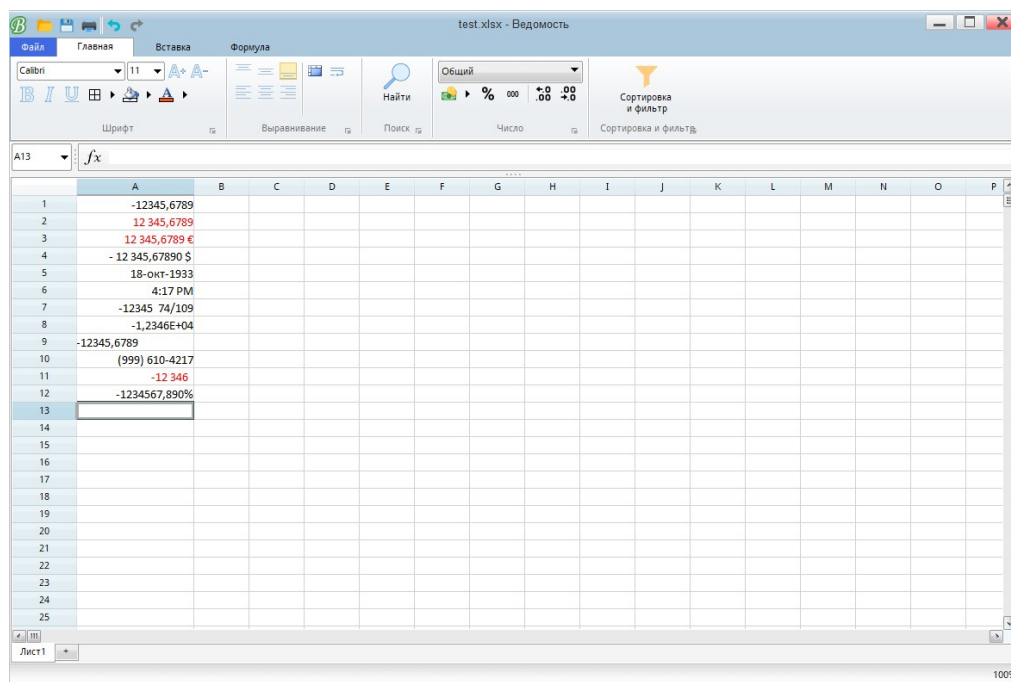


Рисунок 45 - Ведомость. Тестовый документ

Затем созданный документ был сохранен и открыт с помощью табличного редактора «Microsoft Excel». На рисунке 46 изображен открытый через него файл.

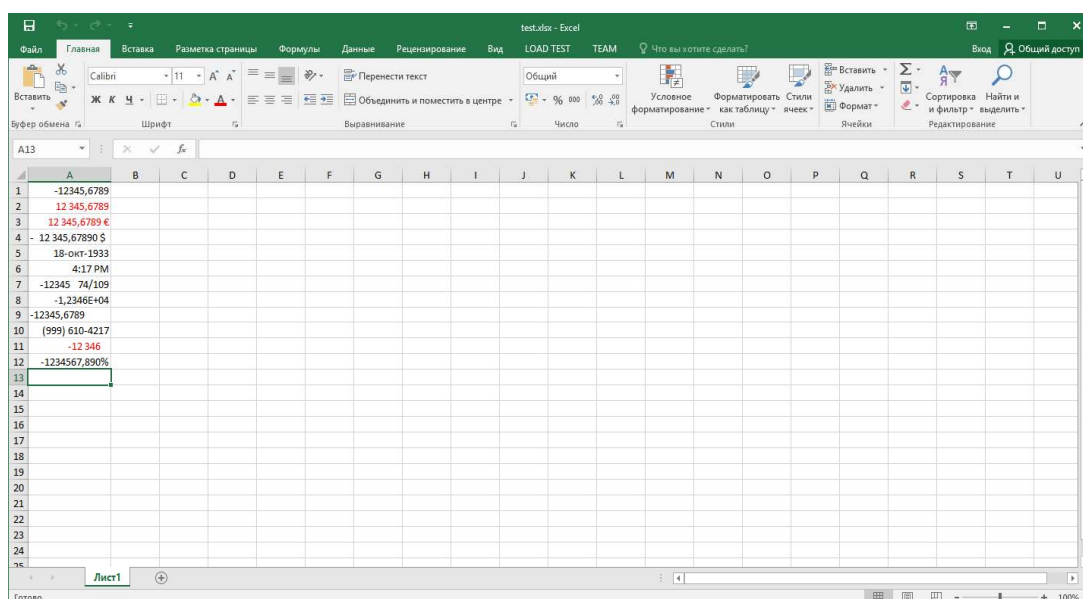


Рисунок 46 - Microsoft Excel. Тестовый файл

Сравнив результат из табличного редактора «Ведомость» и «Microsoft Excel», можно сделать вывод, что форматирование данных полностью идентично, что означает корректную работу разработанного модуля.

					ПГУ 2.090301.39.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

2 Безопасность жизнедеятельности

2.1 Анализ потенциальных опасностей и вредностей, возникающих при работе с ПЭВМ и способы их минимизации

В настоящее время персональные компьютеры стали неотъемлемой частью жизни почти всех людей во многих сферах жизнедеятельности. Но при этом нельзя отрицать их негативное взаимодействие на здоровье человека. Чтобы минимизировать вред, пользователю ПК необходимо соблюдать соответствующие меры безопасности. Основными из них являются: правильное позиционирование рабочего места, планировать режим работы за ПЭВМ, производить контроль освещенности и уровня шума помещения.

Далее в данном разделе ВКР будут рассматриваться требования по организации рабочего процесса для пользователей вычислительных машин, участвующих в разработке программного обеспечения, а также способы минимизации воздействия вредных факторов на здоровье человека.

2.2 Классификация и категорирование помещения, в котором производится разработка

Далее будет рассмотрена классификация помещений по уровню электробезопасности и пожарной безопасности.

2.2.1 Опасность поражения электрическим током

Использование электрического оборудования необходимо на любом производстве, где ведется разработка программного обеспечения. Однако оно может быть опасным для здоровья работников, если условия в помещении не соответствуют необходимым требованиям. Именно поэтому следует применять современные технологии в сфере безопасности жизнедеятельности, обеспечивающие максимальную защиту здоровью персонала.

С точки зрения электробезопасности выделяют три вида помещений:

– безопасные;

- с повышенной опасностью;
- особо опасные.

Чтобы определить условия, влияющие на опасность электрического оборудования для человека и возможность поражения током, следует разобраться в критериях, определяющих степень опасности помещения.

В безопасных помещения возможность поражения человека током сводится к минимуму.

Безопасные помещения должны соответствовать следующим критериям:

- температура воздуха составляет ниже 30°C, оптимальная температура должна быть от 18°C до 20°C;
- отсутствие токопроводящей пыли;
- процент заполнения площади предметами из металла должен составлять менее 20%;
- полы должны быть выполнены из материала, не проводящего ток;
- влажность воздуха не должна превышать 60%;
- помещение должно хорошо проветриваться.

В помещениях с повышенной опасностью вероятность поражения током выше, чем в безопасных поэтому в них персонал должен быть особенно осторожен при выполнении должностных обязанностей.

Для помещений с повышенной опасностью можно выделить следующие критерии:

- температура воздуха превышает 30°C;
- наличие токопроводящей пыли;
- наличие большого количества металлического оборудования;
- полы выполнены из токопроводящего материала;
- высокая влажность воздуха (70-80%);
- плохо проветриваемое помещение;

В особо опасных помещениях вероятность поражения током высокая, поэтому помимо техники безопасности, руководство должно обеспечить персонал специальным обмундированием, способным их защитить.

К особо опасным помещениям относятся следующие критерии:

- температура воздуха превышает 30-35°C;
- пожароопасные помещения, в которых производятся либо хранятся легковоспламеняющиеся вещества или материалы;
- взрывоопасные помещения;
- помещения с едкими парами или газами;
- влажность воздуха составляет 100%.

2.2.2 Пожарная безопасность

Применение электрического оборудования повышает не только риск поражения электрическим током, но и опасность возникновения возгорания или взрыва.

Помещения по степени пожарной и взрывопожарной опасности разделяют на 5 категорий:

- категория «А», объекты с повышенной пожарной активностью. К ней относятся помещения, в которых могут храниться опасные смеси и легко воспламеняющиеся составы с температурой возгорания не более 28°C;
- категория «Б», взрывопожароопасные объекты. К таким помещениям относятся комнаты, в которых хранится все то же самое, только с температурой возгорания более 28°C;
- категории «В1-В4». В нее включаются объекты, в которых разрешается хранить слабо горючие и горючие жидкости, а также твердые вещества, которые при определенных условиях поддерживают свободное горение;
- категория «Г». Допускается применять и содержать смеси и вещества, которые не поддерживают горение. При этом смеси разрешается хранить в разогретом или расплавленном состоянии. В процессе их обработки может

выделяться значительное количество искровых образований, которые в свою очередь провоцируют очаги возгорания;

– категория «Д». К ней принято относить помещения с пониженной пожарной опасностью. В них разрешено хранить негорючие материалы в не разогретом состоянии.

2.3 Санитарно-гигиенические требования к производству

С целью обеспечения благоприятных условий труда для персонала, руководство производства должно соблюдать установленные государственными органами санитарные нормы.

Прежде всего, производственные помещения должны быть теплыми, светлыми и сухими. Выделяют следующие основные требования:

- на каждого рабочего должно быть выделено не менее 15 м³;
- высота дверей и проходов должна составлять не менее 2 м;
- высота помещений не менее 3,2 м;
- расстояние от пола до выступающих конструктивных элементов – не менее 2,6 м;
- ширина дверей должна быть более 0,8 м;
- ширина коридоров должна быть более 1,4 м.

При проектировании помещения, необходимо расположить окна таким образом, чтобы освещалось каждое рабочее место. При этом каждое окно должно открываться. Световые фонари делают из армированного стекла, при применении простого стекла, необходимо под фонарями подвесить металлические сетки.

Температура в помещении чаще всего должна составлять 18-21°C, при относительной влажности 60-40% и скорости воздуха до 0,2 м/с.

Также руководство производства должно обеспечить каждое помещения питьевой водой температурой не менее 8°C и не более 20°C. Периодичность замены воды – 1 сутки.

					ПГУ 2.090301.39.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

Большую роль для сохранения здоровья человека на производстве играет освещенность рабочих мест. Для максимально эффективного использования естественного освещения необходимо очищать окна от пыли и грязи, а также правильно расставлять оборудование.

Помещения с повышенным уровнем шума должны быть обозначены соответствующими знаками. Также руководство обязано осуществлять контроль уровня шума на всех рабочих местах и определять правила безопасной работы в шумных условиях.

2.4 Расчет пожарной нагрузки в помещениях категории В1-В4

Расчет пожарной нагрузки в помещениях категории В1-В4 регулируется сводом правил 12.13130.2009 от 01.05.2009.

Пожарная нагрузка Q , МДж, определяется по формуле, изображенной ниже.

$$Q = \sum_{i=1}^n G_i + Q_{Hi}^p, \quad (1)$$

где G_i – количество i -го материала пожарной нагрузки, кг;

Q_{hi}^p – низшая теплота сгорания i -го материала пожарной нагрузки, МДж·кг⁻¹,

n – количество материалов.

Удельная пожарная нагрузка g , МДж·м⁻² определяется из соотношения $g = Q/S$, где S – площадь размещения пожарной нагрузки, м² (но не менее 10 м²).

Далее приведена таблица соответствия категорий В1-В4 к величине пожарной нагрузки.

Таблица 3 - Категории пожарной опасности помещения

№	Категория помещения	Удельная пожарная нагрузка g на участке, МДж·м ⁻²
1	В1	Более 2200
2	В2	1401-2200

№	Категория помещения	Удельная пожарная нагрузка g на участке, МДж·м ⁻²
3	B3	181-1400
4	B4	1-180

Если при определении категорий B2 или B3 количество пожарной нагрузки Q , определенное по формуле (1), отвечает неравенству $Q \geq 0,64 \cdot g_i \cdot H^2$, то помещение будет относиться к категориям B1 или B2 соответственно.

Ниже приведен список предметов рабочего помещения, их количество, вес, материал из которого они сделаны, значения низшей теплоты сгорания материалов.

Таблица 4 - Предметы рабочего помещения

№	Предмет	Количество	Вес одной единицы, кг.	Материал (с низшей теплотой сгорания)	Значение низшей теплоты сгорания, МДж/кг
1	Стол	5	25	Древесина	13,8
2	Стул	7	3	Древесина	13,8
3	Линолеум	20 м ²	3	Линолеум на тканной основе	20,292
4	Компьютер	5	15	Полистирол	39
5	Шкаф	2	40	Древесина	13,8
6	Рама оконная	3	35	Поливинилхлорид	14
7	Тумба	6	10	Древесина	13,8

Далее необходимо произвести расчет пожарной нагрузки:

$$Q = (5 \cdot 25 + 7 \cdot 3 + 2 \cdot 40 + 6 \cdot 10) \cdot 13,8 + 20 \cdot 3 \cdot 20,292 + 5 \cdot 15 \cdot 39 + 3 \cdot 35 \cdot 14 = 9559,32 \text{ МДж.}$$

Тогда удельная пожарная нагрузка равна:

					ПГУ 2.090301.39.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

$$g = Q/S = 9559,32 / 20 = 477,966 \text{ МДж} \cdot \text{м}^{-2}$$

Полученное значение соответствует категории В3.

Далее необходимо проверить неравенство

$$Q >= 0,64 * g_t * H^2.$$

$$Q = 9559,32.$$

$$0,64 * g_t * H^2 = 0,64 * 477,966 * 3^2 = 2753,08416.$$

$$9553,32 > 2753,08416.$$

Неравенство является верным, из этого можно сделать вывод, что помещение принадлежит к категории пожарной опасности В2. Так как в помещении находится компьютерное оборудование, то для пожаротушения следует применять огнетушитель углекислотный марки ОУ-5.

					ПГУ 2.090301.39.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

Поставленные для выполнения ВКР задачи были полностью решены. Были разработаны библиотеки форматирования, фильтрации и сортировки данных для табличного процессора. Также для тестирования и демонстрации работы был реализован графический интерфейс. Разработанный модуль форматирования данных был интегрирован в редактор электронных таблиц «Ведомость», разработкой которого занимается ООО НТП «Криптософт».

В ходе выполнения работы были улучшены навыки разработки на языке C# и разработки приложений *WinForms*. Также был получен новый опыт в объектно-ориентированном программировании и проектировании архитектур.

Были получены новые знания об основах разработки офисных приложений и стандарте *Open Office XML*, а также проектировании пользовательских элементов управления и применении их в графическом интерфейсе.

В дальнейшем разработанные программные средства можно улучшить, путем добавления поддержки различных языков в модуль форматирования данных, реализовать большой набор стандартных форматов и более широкий набор фильтров.

Список использованных источников

1. Коды числовых форматов: [Электронный ресурс]. URL: <https://support.microsoft.com/ru-ru/office/коды-числовых-форматов-5026bbd6-04bc-48cd-bf33-80f18b4eae68>.
2. ISO/IEC 29500-1:2016. Information technology — Document description and processing languages — Office Open XML FileFormats — Part 1: Fundamentals and Markup Language Reference, 2016.
3. Spreadsheet Styles: [Электронный ресурс]. URL: <http://officeopenxml.com/SSstyles.php>.
4. Документация по .NET Framework: [Электронный ресурс]. URL: <https://docs.microsoft.com/en-US/dotnet/framework>.
5. Руководство по рабочему столу .NET для Windows Forms: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/?view=netframeworkdesktop-4.8>.
6. Документация по C#: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>.

Приложение А

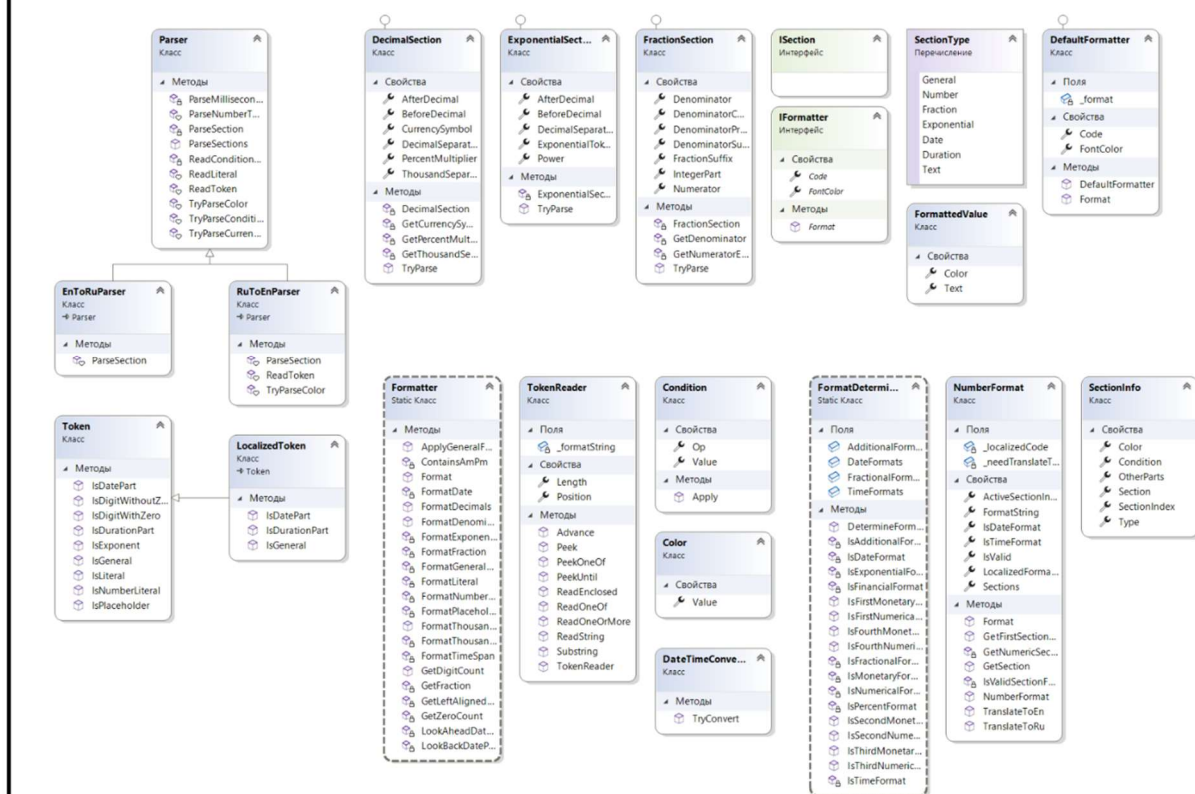
(Обязательное)

Программные средства форматирования, фильтрации и сортировки данных для
табличного процессора

UML-диаграммы

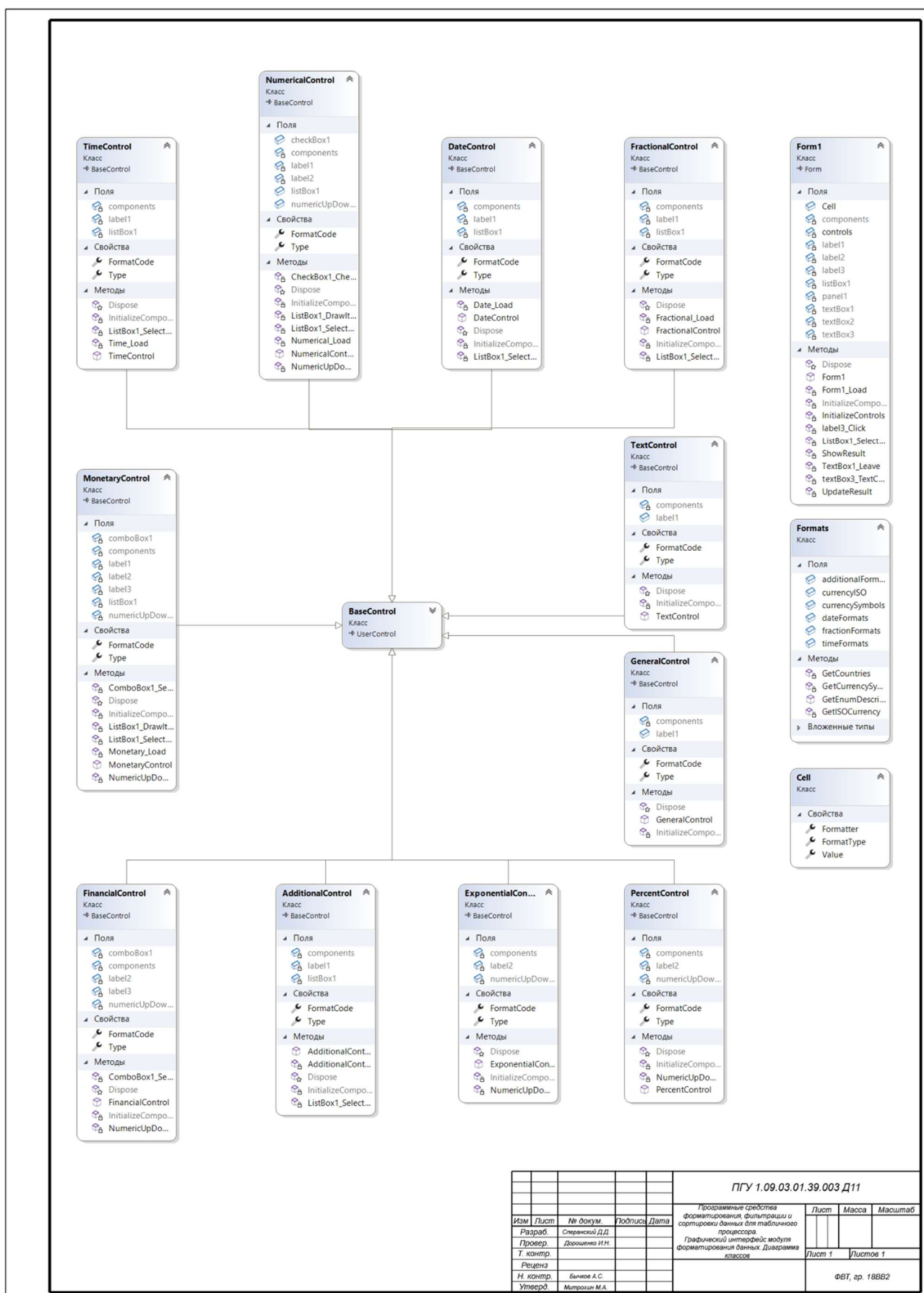
					ПГУ 2.090301.39.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 - Библиотека форматирования данных. Диаграмма классов



					ПГУ 1.09.03.01.39.002 Д11			
Изм.	Лист	№ докум.	Подпись	Дата	Програмные средства форматирования, фильтировки и сортировки данных для табличного процессора УМЛ-дизайнера классов библиотеки форматирования данных	Лист	Масса	Масштаб
Разраб.		Степанский Д.Д.						
Провер.		Дорошенко И.Н.						
Т. контрп.								
Реценза								
Утвердил:		Белов А. С. Медведев М.А.				Лист 1	Листов 1	
								ФВТ, гр. 18862

Приложение А.2 - Графический интерфейс модуля форматирования данных. Диаграмма классов



					ПГУ 1.09.03.01.39.003 Д11				
					Программные средства форматирования, фильтрации и сортировки данных для табличного процессора				
					Графический интерфейс модуля форматирования данных. Диаграмма классов				
Изм.	Лист	№ докум.	Подпись	Дата			Лист	Масштаб	Масштаб
Разраб.		Степанов Д.Д.					Лист 1	Листов 1	
Провер.		Дорошенко И.И.							
Т. контр.									
Реценз.									
И. контр.		Белые А.С.							
Утверд.		Митрохин М.А.							
					ФБТ, зр. 18882				

Изм.	Лист	№ докум.	Подпись	Дата

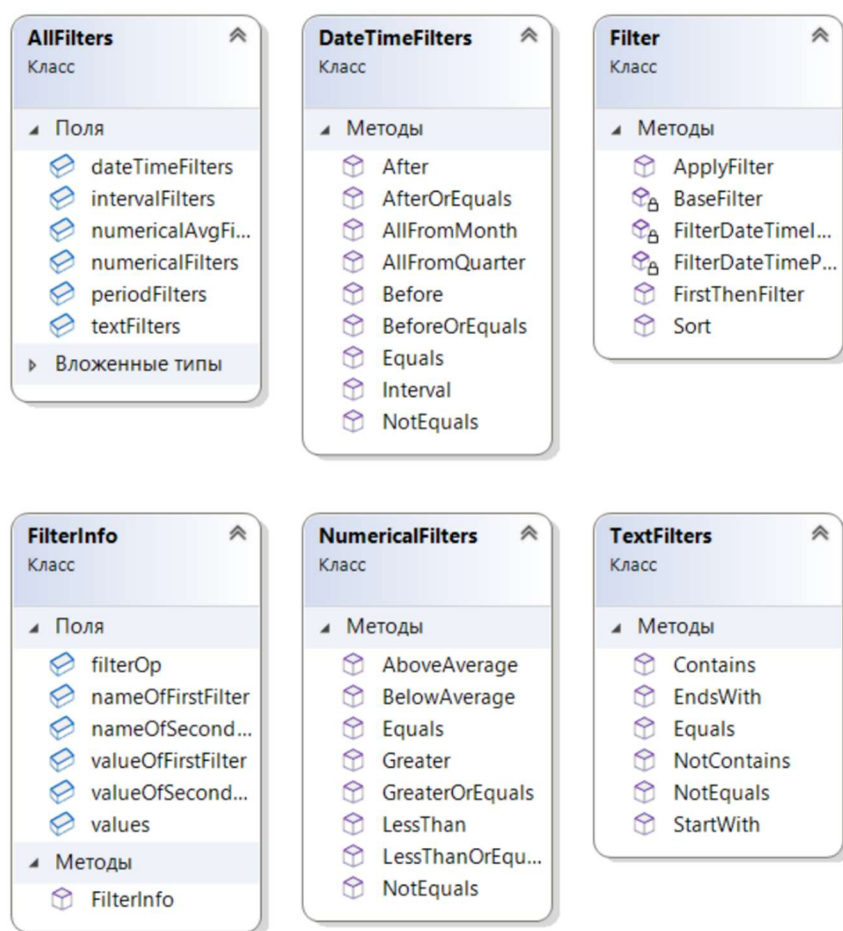
ПГУ 2.090301.39.001 ПЗ

Лист

70

Приложение А.3 - Библиотека фильтрации и сортировки данных.

Диаграмма классов



					ПГУ 1.09.03.01.39.004 Д11				
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства форматирования, фильтрации и сортировки данных для табличного процессора Библиотека фильтрации и сортировки данных. Диаграмма классов				
Разраб.		Степанов Д.Д.							
Провер.		Дорошенко И.Н.							
Т. контр.									
Реценз.									
Н. контр.		Белые А.С.			Лист 1 Листов 1				
Утверд.		Митрохин М.А.							
					ФБТ, гр. 18882				

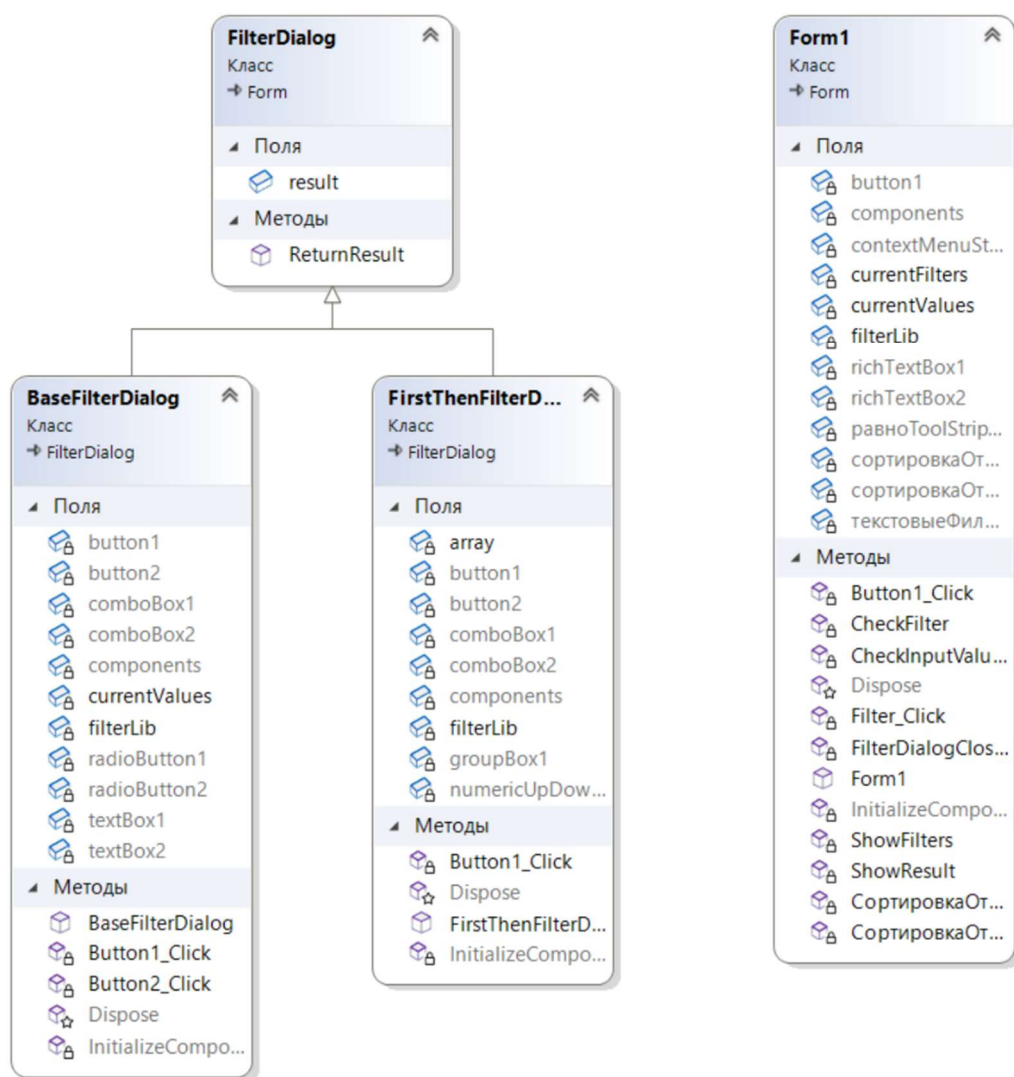
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

71

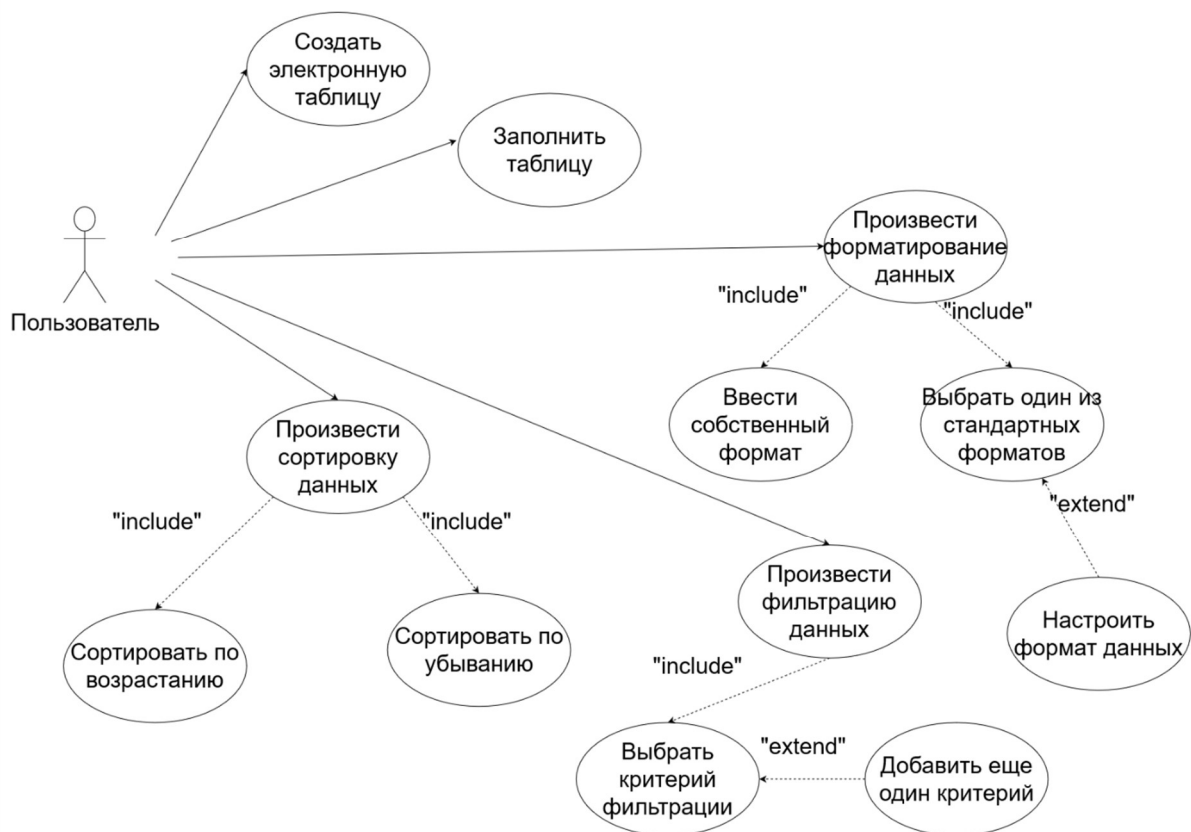
Приложение А.4 - Графический интерфейс модуля фильтрации и сортировки данных. Диаграмма классов



					ПГУ 1.09.03.01.39.005 Д11			
					Программные средства форматирования, фильтрации и сортировки данных для табличного процессора			
Изм.	Лист	№ докум.	Подпись	Дата	Лист			
Разраб.		Степанов Д.Д.			Масса			
Провер.		Дерюгина И.Н.			Масштаб			
Т. контр.					Лист 1			
Реценз.					Листов 1			
Н. контр.		Беляков А.С.			ФБТ, зр. 18882			
Утверд.		Митрохин М.А.						

Изм.	Лист	№ докум.	Подпись	Дата

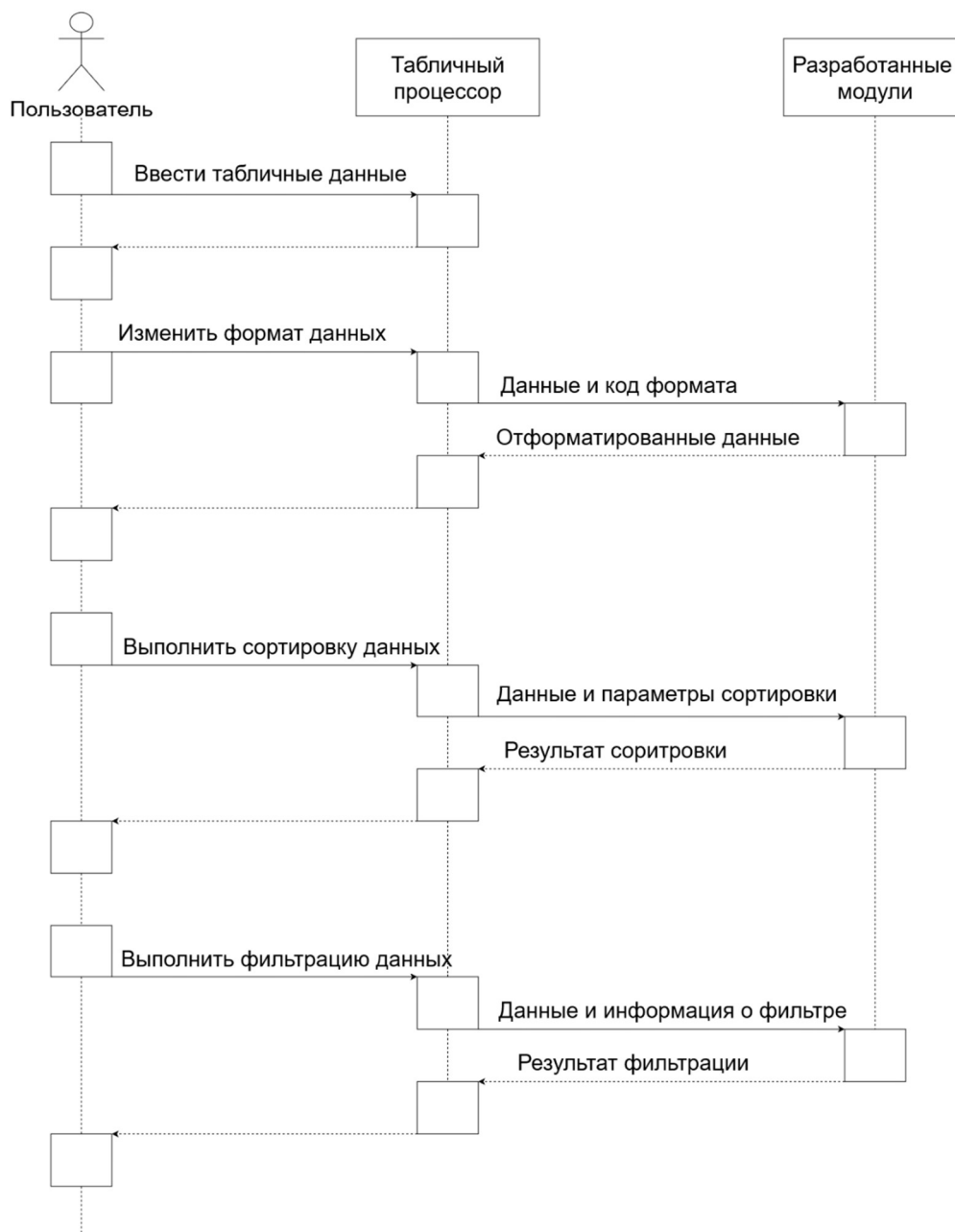
Приложение А.5 - Диаграмма прецедентов



					ПГУ 1.09.03.01.39.006 Д13			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства форматирования, фильтрации и сортировки данных для табличного процессора. Диаграмма прецедентов	Лист	Масса	Масштаб
Разраб.		Степанов Д.Д.						
Провер.		Дерюгина И.И.				Лист 1	Листов 1	
Т. контр.								
Реценз.								
Н. контр.		Беляков А.С.						
Утверд.		Митрохин М.А.						ФБТ, зр. 188В2

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.6 - Диаграмма последовательностей



					ПГУ 1.09.03.01.39.007 Д14			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства форматирования, фильтрации и сортировки данных для табличного процессора. Диаграмма последовательностей	Лист	Масса	Масштаб
Разраб.		Степанов Д.Д.						
Провер.		Дерюченко И.Н.						
Т. контр.						Лист 1	Листов 1	
Реценз.								
Н. контр.		Беляков А.С.						
Утверд.		Митрохин М.А.						ФБТ, зр. 188В2

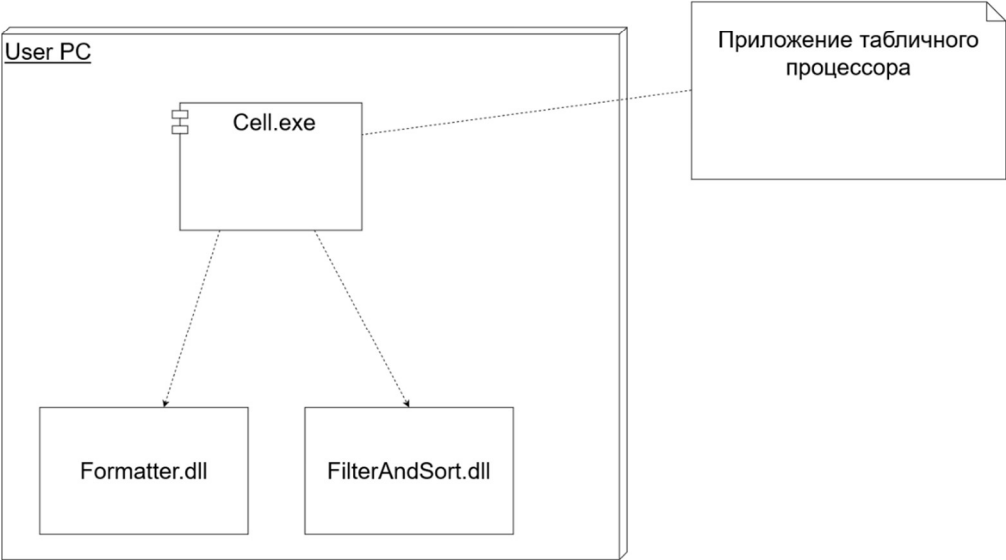
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

74

Приложение А.7 - Диаграмма развертывания



						ПГУ 1.09.03.01.39.008 Д19			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства форматирования, фильтрации и сортировки данных для табличного процессора. Диаграмма развертывания		Лист	Масса	Масштаб
Разраб.		Степанов Д.Д.							
Провер.		Дерюченко И.Н.							
Т. контр.							Лист 1	Листов 1	
Реценз.									
Н. контр.		Беляков А.С.					ФБТ, зр. 188В2		
Утверд.		Митрохин М.А.							

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(Обязательное)

Программные средства форматирования, фильтрации и сортировки данных для
табличного процессора

Листинг программы

					ПГУ 2.090301.39.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.1 - Модуль форматирования данных

Файл Color.cs

```
namespace FormatterLib
{
    public class Color
    {
        public string Value { get; set; }
    }
}
```

Файл Condition.cs

```
namespace FormatterLib
{
    public class Condition
    {
        public string Op { get; set; }
        public double Value { get; set; }
        public bool Apply(double lhs)
        {
            switch (Op)
            {
                case "<":
                    return lhs < Value;
                case "<=":
                    return lhs <= Value;
                case ">":
                    return lhs > Value;
                case ">=":
                    return lhs >= Value;
                case "<>":
                    return lhs != Value;
                case "=":
                    return lhs == Value;
            }
            return false;
        }
    }
}
```

Файл DateTimeConverter.cs

```
using System;
namespace FormatterLib
{
    public class DateTimeConverter
    {
    }
```

```

        public static bool TryConvert(object value, out
DateTime result)
        {
            switch (value)
            {
                case double doubleValue:
                    result =
DateTime.FromOADate(doubleValue);
                    return true;
                case int intValue:
                    result = DateTime.FromOADate(intValue);
                    return true;
                case short shortValue:
                    result = DateTime.FromOADate(shortValue);
                    return true;
                case DateTime dateTimeValue:
                    result = dateTimeValue;
                    return true;
                case string stringValue:
                    if (DateTime.TryParse(stringValue, out
result))
                    {
                        return true;
                    }
                    result = DateTime.MinValue;
                    return false;
                default:
                    result = DateTime.MinValue;
                    return false;
            }
        }
    }
}

```

Файл DecimalSection.cs

```

using System.Collections.Generic;
using System.Linq;

namespace FormatterLib
{
    public class DecimalSection : ISection
    {
        public List<string> BeforeDecimal { get; }
        public bool DecimalSeparator { get; }
        public List<string> AfterDecimal { get; }
        public double PercentMultiplier { get; }
        public string CurrencySymbol { get; }
        public bool ThousandSeparator { get; }
    }
}

```

```

        private DecimalSection(List<string> tokens,
List<string> beforeDecimal, bool decimalSeparator, List<string>
afterDecimal)
        {
            BeforeDecimal = beforeDecimal;
            DecimalSeparator = decimalSeparator;
            AfterDecimal = afterDecimal;
            PercentMultiplier = GetPercentMultiplier(tokens);
            ThousandSeparator = GetThousandSeparator(tokens);
            CurrencySymbol = GetCurrencySymbol(tokens);
        }

        private double
GetPercentMultiplier(IEnumerable<string> tokens)
        {
            return tokens.Any(token => token == "%") ? 100 :
1;
        }

        private string GetCurrencySymbol(List<string> tokens)
        {
            foreach (var token in tokens.Where(token => token
== "\"$\"" || token == "\"€\"" || token == "\"£\"" || token ==
 "\"₽\""))
            {
                return token.Replace("\"", "");
            }

            return string.Empty;
        }

        private bool
GetThousandSeparator(IReadOnlyList<string> tokens)
        {
            var hasLastPlaceholder = false;

            for (var j = 0; j < tokens.Count; j++)
            {
                var tokenIndex = tokens.Count - 1 - j;
                var token = tokens[tokenIndex];

                if (hasLastPlaceholder)
                {
                    if (token != ",") continue;

                    return true;
                }
            }

            return false;
        }

```

```

        public static bool TryParse(List<string> tokens, out
ISection format)
        {
            if (Parser.ParseNumberTokens(tokens, 0, out var
beforeDecimal, out var decimalSeparator, out var afterDecimal) ==
tokens.Count)
            {
                format = new DecimalSection(tokens,
beforeDecimal, decimalSeparator, afterDecimal);
                return true;
            }

            format = null;
            return false;
        }
    }
}

```

Файл DefaultFormatter.cs

```

using System.Globalization;

namespace FormatterLib
{
    public class DefaultFormatter : IFormatter
    {
        public string Code { get; }
        public Color FontColor { get; set; }

        private readonly NumberFormat _format;
        public DefaultFormatter(string formatCode, bool
isRuCode = false)
        {
            Code = formatCode;
            _format = new NumberFormat(formatCode, isRuCode);
        }

        public string Format(string cellValue)
        {
            var res = _format.Format(cellValue,
CultureInfo.CurrentCulture);
            FontColor = res.Color;
            return res.Text;
        }
    }
}

```

Файл EnToRuParser.cs

```

using System.Collections.Generic;

```

```

namespace FormatterLib
{
    internal class EnToRuParser : Parser
    {
        internal static List<string> ParseSection(TokenReader
reader, out bool syntaxError)
        {
            string token;
            var tokens = new List<string>();
            var hasHoursTokens = false;
            var lastDateToken = '\0';

            while ((token = ReadToken(reader, out
syntaxError)) != null)
            {

                if (token == ";")
                    break;

                Token.IsPlaceholder(token);

                if (Token.IsDatePart(token))
                {
                    var translatedToken = string.Empty;
                    Token.IsDurationPart(token);

                    foreach (var c in token.ToLower())
                    {
                        switch (c)
                        {
                            case 'y':
                                translatedToken += "г";
                                break;
                            case 'm':
                                if (hasHoursTokens)
                                {
                                    translatedToken += "М";
                                }
                                else
                                {
                                    translatedToken += "М";
                                }
                                break;
                            case 'd':
                                translatedToken += "д";
                                break;
                            case 'h':
                                translatedToken += "ч";

                                hasHoursTokens = true;
                                break;
                            case 's':
                                translatedToken += "с";

```

					ПГУ 2.090301.39.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

```

        if (lastDateToken == 'm')
        {
            for (int i = tokens.Count
- 1; i >= 0; i--)
            {
                if
(!tokens[i].Contains("M")) continue;

                tokens[i] =
tokens[i].Replace("M", "М");

                break;
            }
        }
        break;
    default:
        translatedToken += c;
        break;
    }

    lastDateToken = c;

}
tokens.Add(translatedToken);
}
else if (Token.IsGeneral(token))
{
    token =
token.ToLower().Replace("general", "основной");
    tokens.Add(token);
}
else if (token == "@")
{
    tokens.Add(token);
}
else if (token.StartsWith("["))
{
    var expression = token.Substring(1,
token.Length - 2);
    if (TryParseCondition(expression, out var
parseCondition))
tokens.Add($"[{parseCondition.Op}{parseCondition.Value}]");

    else if (TryParseColor(expression, out
var parseColor))
    {
        var localizedColor = string.Empty;
        switch (parseColor.Value.ToLower())
        {
            case "black":
                localizedColor = "Черный";

```

					ПГУ 2.090301.39.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		


```

        break;
    case "blue":
        localizedColor = "Синий";
        break;
    case "cyan":
        localizedColor = "Голубой";
        break;
    case "green":
        localizedColor = "Зеленый";
        break;
    case "magenta":
        localizedColor =
"Фиолетовый";
        break;
    case "red":
        localizedColor = "Красный";
        break;
    case "white":
        localizedColor = "Белый";
        break;
    case "yellow":
        localizedColor = "Желтый";
        break;
    }
    tokens.Add($"[{localizedColor}]");
}

else if
(TryParseCurrencySymbol(expression, out var parseCurrencySymbol))
    tokens.Add("\"" + parseCurrencySymbol
+ "\"");
    }
else
{
    tokens.Add(token);
}
}

if (syntaxError || tokens.Count == 0)
{
    return null;
}

return tokens;
}
}
}

```

Файл ExponentialSection.cs

```
using System.Collections.Generic;
```

					ПГУ 2.090301.39.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

namespace FormatterLib
{
    public class ExponentialSection : ISection
    {
        public List<string> BeforeDecimal { get; }
        public bool DecimalSeparator { get; }
        public List<string> AfterDecimal { get; }
        public string ExponentialToken { get; }
        public List<string> Power { get; }

        private ExponentialSection(List<string>
beforeDecimal, bool decimalSeparator, List<string> afterDecimal,
string exponentialToken, List<string> power)
        {
            BeforeDecimal = beforeDecimal;
            DecimalSeparator = decimalSeparator;
            AfterDecimal = afterDecimal;
            ExponentialToken = exponentialToken;
            Power = power;
        }

        public static bool TryParse(List<string> tokens, out
ExponentialSection format)
        {
            var position = Parser.ParseNumberTokens(tokens,
0, out var beforeDecimal, out var decimalSeparator, out var
afterDecimal);

            if (position != 0 && position < tokens.Count &&
Token.IsExponent(tokens[position]))
            {
                var exponentialToken = tokens[position];
                position++;
                format = new
ExponentialSection(beforeDecimal, decimalSeparator, afterDecimal,
exponentialToken, tokens.GetRange(position, tokens.Count -
position));

                return true;
            }

            format = null;
            return false;
        }
    }
}

```

Файл FormatDeterminant.cs

```

using System.Linq;
using System.Text.RegularExpressions;

```

					ПГУ 2.090301.39.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

```

namespace FormatterLib
{
    public static class FormatDeterminant
    {
        public static readonly string[] DateFormats =
        {
            "d/m;@", "d/m/yy;@", "dd/mm/yy;@", "d\\ mmm;@",
            "d\\ mmm\\ yy;@", "dd\\ mmm\\ yy;@", "yyyy\\,\\ mmmm;@", "mmmm\\
            yyyy;@",
            "dd\\ mmmm\\ yyyy\\ \\r\\.;@", "dd/mm/yy\\ h:mm\\
            AM/PM;@", "dd/mm/yy\\ h:mm;@", "mmmm;@", "yyyy\\,\\ dd\\ mmmm;@",
            "d/m/yyyy;@", "d\\-mmm\\-yyyy;@"
        };

        public static readonly string[] TimeFormats =
        {
            "h:mm;@", "h:mm\\ AM/PM;@", "h:mm:ss;@",
            "h:mm:ss\\ AM/PM;@", "mm:ss.0;@", "[h]:mm:ss;@",
            "dd/mm/yy\\ h:mm\\ AM/PM;@", "dd/mm/yy\\ h:mm;@",
        };

        public static readonly string[] FractionalFormats =
        {
            "#\" \"?/?", "\"\" \"?/?/?", "\"\" \"???/???",
            "\"\" \"?/2", "\"\" \"?/4", "\"\" \"?/8", "\"\" \"??/16", "\"\"
            \"?/10", "\"\" \"??/100";
        };

        public static readonly string[] AdditionalFormats =
        {
            "000000", "00000\\-0000", "[<=9999999]###\\-
            ####;\\(###\\)\\ ###\\-####", "0000";
        };

        public static int DetermineFormat(string formatCode)
        {
            if (formatCode == "General")
            {
                return 0;
            }

            if (IsNumericalFormat(formatCode))
            {
                return 1;
            }

            if (IsMonetaryFormat(formatCode))
            {
                return 2;
            }

            if (IsFinancialFormat(formatCode))
            {
                return 3;
            }
        }
    }
}

```

```

        if (IsDateFormat(formatCode))
        {
            return 4;
        }

        if (IsTimeFormat(formatCode))
        {
            return 5;
        }

        if (IsPercentFormat(formatCode))
        {
            return 6;
        }

        if (IsFractionalFormat(formatCode))
        {
            return 7;
        }

        if (IsExponentialFormat(formatCode))
        {
            return 8;
        }

        if (formatCode == "@")
        {
            return 9;
        }

        if (IsAdditionalFormat(formatCode))
        {
            return 10;
        }

        return 11;
    }

    private static bool IsNumericalFormat(string
formatCode)
    {
        return IsFirstNumericalFormat(formatCode) ||
IsSecondNumericalFormat(formatCode) ||
                IsThirdNumericalFormat(formatCode) ||
IsFourthNumericalFormat(formatCode);
    }

    private static bool IsMonetaryFormat(string
formatCode)
    {
        return IsFirstMonetaryFormat(formatCode) ||
IsSecondMonetaryFormat(formatCode) ||

```

```

        IsThirdMonetaryFormat(formatCode) ||
IsFourthMonetaryFormat(formatCode);
    }

    private static bool IsFinancialFormat(string
formatCode)
    {
        return new Regex("^_-\*\ #,##0\\.0*(\\|\\|\\|)?
(\\|)?_?.(\\|)?_-(\\|\\|\\|)?-\\*\ #,##0.0*(\\|\\|\\|)? (\\|)?_?.(\\|)?_-(\\|)?_-\*\
\\|"-\\|(\\|\\|\\|)? (\\|\\|\\|)? (\\|)?_?.(\\|)?_-(\\|)?_@_-$").IsMatch(formatCode);
    }

    private static bool IsDateFormat(string formatCode)
    {
        return DateFormats.Contains(new
Regex("\\[.*\\]").Replace(formatCode, string.Empty));
    }

    private static bool IsTimeFormat(string formatCode)
    {
        return TimeFormats.Contains(new
Regex("\\[\\$.*\\]").Replace(formatCode, string.Empty));
    }

    private static bool IsPercentFormat(string
formatCode)
    {
        return new
Regex("^0\\.0*%$").IsMatch(formatCode);
    }

    private static bool IsFractionalFormat(string
formatCode)
    {
        return FractionalFormats.Contains(formatCode);
    }

    private static bool IsExponentialFormat(string
formatCode)
    {
        return new Regex("^0\\.0*E\\|+00$",
RegexOptions.IgnoreCase).IsMatch(formatCode);
    }

    private static bool IsAdditionalFormat(string
formatCode)
    {
        return AdditionalFormats.Contains(formatCode);
    }

    public static bool IsFirstNumericalFormat(string
formatCode)
    {

```

```

        return new
Regex("(#,#)?0\\.0*$").IsMatch(formatCode);
    }

    public static bool IsSecondNumericalFormat(string
formatCode)
    {
        return new
Regex("(#,#)?0\\.0*;\[Red\\](#,#)?0\\.0*$",
RegexOptions.IgnoreCase).IsMatch(formatCode);
    }

    public static bool IsThirdNumericalFormat(string
formatCode)
    {
        return new Regex("(#,#)?0\\.0*_ ?;\[Red\\]?-
(#,#)?0\\.0*\[Red\\]? ?$").IsMatch(formatCode);
    }

    public static bool IsFourthNumericalFormat(string
formatCode)
    {
        return new Regex("(#,#)?0\\.0*_
?;\[Red\\]\[Red\\]?-(#,#)?0\\.0*\[Red\\]? ?$",
RegexOptions.IgnoreCase).IsMatch(formatCode);
    }

    public static bool IsFirstMonetaryFormat(string
formatCode)
    {
        return new Regex("^#,#0\\.0*\[Red\\]?
_?(\")?_?.(\")?$").IsMatch(formatCode);
    }

    public static bool IsSecondMonetaryFormat(string
formatCode)
    {
        return new Regex("^#,#0\\.0*\[Red\\]?
(\")?_?.(\")?;\[Red\\]#,#0\\.0*\[Red\\]? (\")?_?.(\")?",
RegexOptions.IgnoreCase).IsMatch(formatCode);
    }

    public static bool IsThirdMonetaryFormat(string
formatCode)
    {
        return new Regex("^#,#0\\.0*\[Red\\]?
(\")?_?.(\")?;\[Red\\]?-#,#0\\.0*\[Red\\]?
(\")?_?.(\")?$").IsMatch(formatCode);
    }

    public static bool IsFourthMonetaryFormat(string
formatCode)
    {

```

```

        return new Regex("^#,##0\\.0*\\\\\\?
(\\\")?_?.(\\\")?;\\\\[Red\\\\]\\\\\\\\?-#,##0\\.0*\\\\\\\\?
(\\\")?_?.(\\\")?").IsMatch(formatCode);
    }

}

```

Файл FormattedValue.cs

```

namespace FormatterLib
{
    public class FormattedValue
    {
        public string Text { get; set; }
        public Color Color { get; set; }
    }
}

```

Файл Formatter.cs

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
namespace FormatterLib
{
    public static class Formatter
    {
        public static string Format(object value, SectionInfo
sectionInfo, CultureInfo culture)
        {
            switch (sectionInfo.Type)
            {
                case SectionType.Number:
                    var number = Convert.ToDouble(value,
culture);
                    if ((sectionInfo.SectionIndex == 0 &&
sectionInfo.Condition != null) || sectionInfo.SectionIndex == 1)
                        number = Math.Abs(number);
                    if (!(sectionInfo.Section is
DecimalSection decimalSection))
                    {
                        throw new Exception("Invalid
Section");
                    }
                    return FormatNumber(number,
decimalSection, culture);
            }
        }
    }
}

```

```

        case SectionType.Date:
            if (DateTimeConverter.TryConvert(value,
out var conevrtdDateTime))
            {
                return FormatDate(conevrtdDateTime,
sectionInfo.OtherParts, culture);
            }
            else
            {
                throw new FormatException("Unexpected
date value");
            }
        case SectionType.Duration:
            if (value is TimeSpan ts)
            {
                return FormatTimeSpan(ts,
sectionInfo.OtherParts);
            }
            else
            {
                var d = Convert.ToDouble(value);
                return
FormatTimeSpan(TimeSpan.FromDays(d), sectionInfo.OtherParts);
            }
        case SectionType.General:
        case SectionType.Text:
            return
FormatGeneralText(ApplyGeneralFormat(value, culture),
sectionInfo.OtherParts);
        case SectionType.Exponential:
            return
FormatExponential(Convert.ToDouble(value, culture), sectionInfo,
culture);
        case SectionType.Fraction:
            return
FormatFraction(Convert.ToDouble(value, culture), sectionInfo,
culture);
        default:
            throw new
InvalidOperationException("Unknown number format sectionInfo");
    }
}
private static string FormatGeneralText(string text,
List<string> tokens)
{
    var result = new StringBuilder();
    foreach (var token in tokens)
    {
        if (Token.IsGeneral(token) || token == "@")
        {
            result.Append(text);
        }
        else
    }
}

```



```

        {
            FormatLiteral(token, result);
        }
    }
    return result.ToString();
}
private static string FormatTimeSpan(TimeSpan
timeSpan, IReadOnlyList<string> tokens)
{
    var result = new StringBuilder();
    var containsMilliseconds = false;
    for (var i = tokens.Count - 1; i >= 0; i--)
    {
        if (!tokens[i].StartsWith(".0")) continue;
        containsMilliseconds = true;
        break;
    }
    foreach (var token in tokens)
    {
        if (token.StartsWith("m",
StringComparison.OrdinalIgnoreCase))
        {
            var value = timeSpan.Minutes;
            var digits = token.Length;
            result.Append(value.ToString("D" +
digits));
        }
        else if (token.StartsWith("s",
StringComparison.OrdinalIgnoreCase))
        {
            var formatMs = containsMilliseconds ? 0 :
timeSpan.Milliseconds / 1000D;
            var value =
(int)Math.Round(timeSpan.Seconds + formatMs, 0,
MidpointRounding.AwayFromZero);
            var digits = token.Length;
            result.Append(value.ToString("D" +
digits));
        }
        else if (token.StartsWith("[h",
StringComparison.OrdinalIgnoreCase))
        {
            var value = (int)timeSpan.TotalHours;
            var digits = token.Length - 2;
            result.Append(value.ToString("D" +
digits));
            timeSpan = new TimeSpan(0, 0,
Math.Abs(timeSpan.Minutes), Math.Abs(timeSpan.Seconds),
Math.Abs(timeSpan.Milliseconds));
        }
        else if (token.StartsWith("[m",
StringComparison.OrdinalIgnoreCase))
        {

```

```

        var value = (int)timeSpan.TotalMinutes;
        var digits = token.Length - 2;
        result.Append(value.ToString("D" +
digits));
        TimeSpan = new TimeSpan(0, 0, 0,
Math.Abs(timeSpan.Seconds), Math.Abs(timeSpan.Milliseconds));
    }
    else if (token.StartsWith("[s",
StringComparison.OrdinalIgnoreCase))
    {
        var value = (int)timeSpan.TotalSeconds;
        var digits = token.Length - 2;
        result.Append(value.ToString("D" +
digits));
        TimeSpan = new TimeSpan(0, 0, 0, 0,
Math.Abs(timeSpan.Milliseconds));
    }
    else if (token.StartsWith(".0"))
    {
        var value = timeSpan.Milliseconds;
        var digits = token.Length - 1;
        result.Append("." + value.ToString("D" +
digits));
    }
    else
    {
        FormatLiteral(token, result);
    }
}
return result.ToString();
}
private static string FormatDate(DateTime date,
IEnumerable<string> tokens, CultureInfo culture)
{
    var containsAmPm = ContainsAmPm(tokens);
    var result = new StringBuilder();
    for (var i = 0; i < tokens.Count; i++)
    {
        var token = tokens[i];
        if (token.StartsWith("y",
StringComparison.OrdinalIgnoreCase))
        {
            // year
            var digits = token.Length;
            if (digits < 2)
                digits = 2;
            if (digits == 3)
                digits = 4;
            var year = date.Year;
            if (digits == 2)
                year = year % 100;
            result.Append(year.ToString("D" +
digits));

```

```

        }
        else if (token.StartsWith("m",
StringComparison.OrdinalIgnoreCase))
        {
            if (LookBackDatePart(tokens, i - 1, "h")
|| LookAheadDatePart(tokens, i + 1, "s"))
            {
                var digits = token.Length;

result.Append(date.Minute.ToString("D" + digits));
            }
            else
            {
                var digits = token.Length;
                switch (digits)
                {
                    case 3:

result.Append(culture.DateTimeFormat.AbbreviatedMonthNames[date.Mo
nth - 1]);

                    break;
                    case 4:

result.Append(culture.DateTimeFormat.MonthNames[date.Month - 1]);
                    break;
                    case 5:

result.Append(culture.DateTimeFormat.MonthNames[date.Month -
1][0]);

                    break;
                    default:

result.Append(date.Month.ToString("D" + digits));
                    break;

                }
            }
        }
        else if (token.StartsWith("d",
StringComparison.OrdinalIgnoreCase))
        {
            var digits = token.Length;
            switch (digits)
            {
                case 3:

result.Append(culture.DateTimeFormat.AbbreviatedDayNames[(int)date
.DayOfWeek]);

                    break;
                    case 4:

result.Append(culture.DateTimeFormat.DayNames[(int)date.DayOfWeek]
);

                    break;

```

```

        default:

result.Append(date.Day.ToString("D" + digits));
        break;
    }
    }
    else if (token.StartsWith("h",
StringComparison.OrdinalIgnoreCase))
    {
        var digits = token.Length;
        result.Append(containsAmPm
            ? ((date.Hour + 11) % 12 +
1).ToString("D" + digits)
            : date.Hour.ToString("D" + digits));
    }
    else if (token.StartsWith("s",
StringComparison.OrdinalIgnoreCase))
    {
        var digits = token.Length;
        result.Append(date.Second.ToString("D" +
digits));
    }
    else if (token.StartsWith("g",
StringComparison.OrdinalIgnoreCase))
    {
        var era =
culture.DateTimeFormat.Calendar.GetEra(date);
        var digits = token.Length;
        result.Append(digits < 3
            ?
culture.DateTimeFormat.GetAbbreviatedEraName(era)
            :
culture.DateTimeFormat.GetEraName(era));
    }
    else if (string.Compare(token, "am/pm",
StringComparison.OrdinalIgnoreCase) == 0)
    {
        result.Append(date.ToString("tt",
CultureInfo.InvariantCulture).ToUpperInvariant());
    }
    else if (string.Compare(token, "a/p",
StringComparison.OrdinalIgnoreCase) == 0)
    {
        var ampm = date.ToString("%t",
CultureInfo.InvariantCulture);
        result.Append(char.IsUpper(token[0]) ?
ampm.ToUpperInvariant() : ampm.ToLowerInvariant());
    }
    else if (token.StartsWith(".0"))
    {
        var value = date.Millisecond;
        var digits = token.Length - 1;

```

```

        result.Append(".") + value.ToString("D" +
digits));
    }
    else switch (token)
    {
        case "/":

result.Append(culture.DateTimeFormat.DateSeparator);

        break;
        case ",":
        {
            while (i < tokens.Count - 1
&& tokens[i + 1] == ",")
            {
                i++;
            }

            result.Append(",");
            break;
        }
        default:
            FormatLiteral(token, result);
            break;
    }
}
return result.ToString();
}
private static bool
LookAheadDatePart(IReadOnlyList<string> tokens, int fromIndex,
string startsWith)
{
    for (var i = fromIndex; i < tokens.Count; i++)
    {
        var token = tokens[i];
        if (token.StartsWith(startsWith,
StringComparison.OrdinalIgnoreCase))
            return true;
        if (Token.IsDatePart(token))
            return false;
    }
    return false;
}
private static bool
LookBackDatePart(IReadOnlyList<string> tokens, int fromIndex,
string startsWith)
{
    for (var i = fromIndex; i >= 0; i--)
    {
        var token = tokens[i];
        if (token.StartsWith(startsWith,
StringComparison.OrdinalIgnoreCase))
            return true;

```

```

        if (Token.IsDatePart(token))
            return false;
    }
    return false;
}
private static bool ContainsAmPm(IEnumerable<string>
tokens)
{
    foreach (var token in tokens)
    {
        if (string.Compare(token, "am/pm",
StringComparison.OrdinalIgnoreCase) == 0)
        {
            return true;
        }
        if (string.Compare(token, "a/p",
StringComparison.OrdinalIgnoreCase) == 0)
        {
            return true;
        }
    }
    return false;
}
private static string FormatNumber(double value,
DecimalSection format, CultureInfo culture)
{
    var thousandSeparator = format.ThousandSeparator;
    value *= format.PercentMultiplier;
    var result = new StringBuilder();
    FormatNumber(value, format.BeforeDecimal,
format.DecimalSeparator, format.AfterDecimal, thousandSeparator,
culture, result);
    return result.ToString();
}
private static void FormatNumber(double value,
List<string> beforeDecimal, bool decimalSeparator,
IReadOnlyList<string> afterDecimal, bool thousandSeparator,
CultureInfo culture, StringBuilder result)
{
    var significantDigits = 0;
    if (afterDecimal != null)
        significantDigits =
GetDigitCount(afterDecimal);
    var valueString = Math.Abs(value).ToString("F" +
significantDigits, CultureInfo.InvariantCulture);
    var valueStrings = valueString.Split('.');
    var thousandsString = valueStrings[0];
    var decimalString = valueStrings.Length > 1 ?
valueStrings[1].TrimEnd('0') : "";
    if (value < 0)
    {
        result.Append("-");
    }
}

```

```

        if (beforeDecimal != null)
        {
            FormatThousands(thousandsString,
thousandSeparator, false, beforeDecimal, culture, result);
        }
        if (afterDecimal == null) return;
        if (decimalSeparator)
        {
            result.Append(culture.NumberFormat.NumberDecimalSeparator);
        }
        FormatDecimals(decimalString, afterDecimal,
result);
    }
    public static void FormatThousands(string
valueString, bool thousandSeparator, bool significantZero,
IEnumerable<string> tokens, CultureInfo culture, StringBuilder
result)
    {
        var significant = false;
        var formatDigits = GetDigitCount(tokens);
        valueString = valueString.PadLeft(formatDigits,
'0');

        var tokenIndex = 0;
        for (; tokenIndex < tokens.Count; tokenIndex++)
        {
            var token = tokens[tokenIndex];
            if (Token.IsPlaceholder(token))
                break;
            else
                FormatLiteral(token, result);
        }
        var digitIndex = 0;
        for (; digitIndex < (valueString.Length -
formatDigits); digitIndex++)
        {
            significant = true;
            result.Append(valueString[digitIndex]);
            if (thousandSeparator)
                FormatThousandSeparator(valueString,
digitIndex, culture, result);
        }
        for (; tokenIndex < tokens.Count; ++tokenIndex)
        {
            var token = tokens[tokenIndex];
            if (Token.IsPlaceholder(token))
            {
                var c = valueString[digitIndex];
                if (c != '0' || (significantZero &&
digitIndex == valueString.Length - 1)) significant = true;
                FormatPlaceholder(token, c, significant,
result);
            }
        }
    }

```

```

        if (thousandSeparator && (significant ||
token.Equals("0")))
            FormatThousandSeparator(valueString,
digitIndex, culture, result);
            digitIndex++;
        }
        else
        {
            FormatLiteral(token, result);
        }
    }
}

private static void FormatThousandSeparator(string
valueString, int digit, CultureInfo culture, StringBuilder result)
{
    var positionInTens = valueString.Length - 1 -
digit;
    if (positionInTens > 0 && (positionInTens % 3) ==
0)
    {
        result.Append(culture.NumberFormat.NumberGroupSeparator);
    }
}

public static void FormatDecimals(string valueString,
IEnumerable<string> tokens, StringBuilder result)
{
    var unpaddedDigits = valueString.Length;
    var formatDigits = GetDigitCount(tokens);
    valueString = valueString.PadRight(formatDigits,
'0');

    var valueIndex = 0;
    foreach (var token in tokens)
    {
        if (Token.IsPlaceholder(token))
        {
            var c = valueString[valueIndex];
            var significant = valueIndex <
unpaddedDigits;
            FormatPlaceholder(token, c, significant,
result);
            valueIndex++;
        }
        else
        {
            FormatLiteral(token, result);
        }
    }
}

private static string FormatExponential(double value,
SectionInfo format, CultureInfo culture)
{

```



```

        if (!(format.Section is ExponentialSection
exponentialSection))
        {
            throw new Exception("Invalid section");
        }
        var baseDigits = 0;
        if (exponentialSection.BeforeDecimal != null)
        {
            baseDigits =
GetDigitCount(exponentialSection.BeforeDecimal);
        }
        var exponent =
(int)Math.Floor(Math.Log10(Math.Abs(value)));
        var mantissa = value / Math.Pow(10, exponent);
        var shift = Math.Abs(exponent) % baseDigits;
        if (shift > 0)
        {
            if (exponent < 0)
                shift = (baseDigits - shift);
            mantissa *= Math.Pow(10, shift);
            exponent -= shift;
        }
        var result = new StringBuilder();
        FormatNumber(mantissa,
exponentialSection.BeforeDecimal,
exponentialSection.DecimalSeparator,
exponentialSection.AfterDecimal, false, culture, result);

result.Append(exponentialSection.ExponentialToken[0]);
        if (exponentialSection.ExponentialToken[1] == '+'
&& exponent >= 0)
        {
            result.Append("+");
        }
        else if (exponent < 0)
        {
            result.Append("-");
        }

        FormatThousands(Math.Abs(exponent).ToString(CultureInfo.InvariantCulture), false, false, exponentialSection.Power, culture, result);
        return result.ToString();
    }

    private static string FormatFraction(double value,
SectionInfo format, CultureInfo culture)
    {
        if (!(format.Section is FractionSection
fractionSection))
        {
            throw new Exception("Invalid section");
        }
        var integral = 0;
        int numerator, denominator;

```

```

var sign = value < 0;
if (fractionSection.IntegerPart != null)
{
    integral = (int)Math.Truncate(value);
    value = Math.Abs(value - integral);
}
if (fractionSection.DenominatorConstant != 0)
{
    denominator =
fractionSection.DenominatorConstant;
    var rr = Math.Round(value * denominator);
    var b = Math.Floor(rr / denominator);
    numerator = (int)(rr - b * denominator);
}
else
{
    var denominatorDigits =
Math.Min(GetDigitCount(fractionSection.Denominator), 7);
    GetFraction(value, (int)Math.Pow(10,
denominatorDigits) - 1, out numerator, out denominator);
}
    var numeratorZeros =
GetZeroCount(fractionSection.Numerator);
    var hideFraction = (fractionSection.IntegerPart
!= null && numerator == 0 && numeratorZeros == 0);
    var result = new StringBuilder();
    if (sign)
        result.Append("-");
    if (fractionSection.IntegerPart != null)

FormatThousands(Math.Abs(integral).ToString("F0",
CultureInfo.InvariantCulture), false, hideFraction,
fractionSection.IntegerPart, culture, result);
        var numeratorString =
Math.Abs(numerator).ToString("F0", CultureInfo.InvariantCulture);
        var denominatorString =
denominator.ToString("F0", CultureInfo.InvariantCulture);
        var fraction = new StringBuilder();
        FormatThousands(numeratorString, false, true,
fractionSection.Numerator, culture, fraction);
        fraction.Append("/");
        if (fractionSection.DenominatorPrefix != null)
            FormatThousands("", false, false,
fractionSection.DenominatorPrefix, culture, fraction);
        if (fractionSection.DenominatorConstant != 0)

fraction.Append(fractionSection.DenominatorConstant.ToString());
        else
            FormatDenominator(denominatorString,
fractionSection.Denominator, fraction);
        if (fractionSection.DenominatorSuffix != null)
            FormatThousands("", false, false,
fractionSection.DenominatorSuffix, culture, fraction);

```

```

        result.Append(hideFraction ? new string(' ',
fraction.ToString().Length) : fraction.ToString());
        if (fractionSection.FractionSuffix != null)
            FormatThousands("", false, false,
fractionSection.FractionSuffix, culture, result);
        return result.ToString();
    }
    // Adapted from ssf.js 'frac()' helper
    private static void GetFraction(double x, int D, out
int nom, out int den)
    {
        var sgn = x < 0 ? -1 : 1;
        var B = x * sgn;
        var P_2 = 0.0;
        var P_1 = 1.0;
        var P = 0.0;
        var Q_2 = 1.0;
        var Q_1 = 0.0;
        var Q = 0.0;
        var A = Math.Floor(B);
        while (Q_1 < D)
        {
            A = Math.Floor(B);
            P = A * P_1 + P_2;
            Q = A * Q_1 + Q_2;
            if ((B - A) < 0.00000005) break;
            B = 1 / (B - A);
            P_2 = P_1; P_1 = P;
            Q_2 = Q_1; Q_1 = Q;
        }
        if (Q > D) { if (Q_1 > D) { Q = Q_2; P = P_2; }
else { Q = Q_1; P = P_1; } }
        nom = (int)(sgn * P);
        den = (int)Q;
    }
    public static void FormatDenominator(string
valueString, List<string> tokens, StringBuilder result)
    {
        var formatDigits = GetDigitCount(tokens);
        valueString = valueString.PadLeft(formatDigits,
'0');

        var significant = false;
        var valueIndex = 0;
        foreach (var token in tokens)
        {
            char c;
            if (valueIndex < valueString.Length)
            {
                c = GetLeftAlignedValueDigit(token,
valueString, valueIndex, significant, out valueIndex);
                if (c != '0')
                    significant = true;
            }
        }
    }

```

```

        else
        {
            c = '0';
            significant = false;
        }
        FormatPlaceholder(token, c, significant,
result);
    }
}

private static char GetLeftAlignedValueDigit(string
token, string valueString, int startIndex, bool significant, out
int valueIndex)
{
    char c;
    valueIndex = startIndex;
    if (valueIndex < valueString.Length)
    {
        c = valueString[valueIndex];
        valueIndex++;
        if (c != '0')
            significant = true;
        if (token != "?" || significant) return c;
        while (valueIndex < valueString.Length)
        {
            c = valueString[valueIndex];
            valueIndex++;
            if (c == '0') continue;
            break;
        }
    }
    else
    {
        c = '0';
    }
    return c;
}

private static void FormatPlaceholder(string token,
char c, bool significant, StringBuilder result)
{
    switch (token)
    {
        case "0" when significant:
            result.Append(c);
            break;
        case "0":
            result.Append("0");
            break;
        case "#":
            if (significant)
                result.Append(c);
            break;
        case "?" when significant:
            result.Append(c);

```

```

        break;
        case "?":
            result.Append(" ");
            break;
    }
}
public static int GetDigitCount(ReadOnlyList<string>
tokens)
{
    return tokens.Count(token =>
Token.IsPlaceholder(token));
}
private static int GetZeroCount(IEnumerable<string>
tokens)
{
    return tokens.Count(token => token == "0");
}
private static void FormatLiteral(string token,
StringBuilder result)
{
    var literal = string.Empty;
    if (token == ",")
    {
    }
    else switch (token.Length)
    {
        case 2 when (token[0] == '*' || token[0]
== '\\'):
            literal = token[1].ToString();
            break;
        case 2 when token[0] == '_':
            literal = " ";
            break;
        default:
            literal = token.StartsWith("\\") ?
token.Substring(1, token.Length - 2) : token;
            break;
    }
    result.Append(literal);
}
public static string ApplyGeneralFormat(object value,
IFormatProvider provider)
{
    switch (value)
    {
        case double d:
            return d.ToString("G15", provider);
        default:
            return Convert.ToString(value, provider);
    }
}
}
}

```

Файл FractionSection.cs

```
using System.Collections.Generic;

namespace FormatterLib
{
    public class FractionSection : ISection
    {
        public List<string> IntegerPart { get; }
        public List<string> Numerator { get; }
        public List<string> DenominatorPrefix { get; }
        public List<string> Denominator { get; }
        public int DenominatorConstant { get; }
        public List<string> DenominatorSuffix { get; }
        public List<string> FractionSuffix { get; }

        private FractionSection(List<string> numTokens,
            List<string> denomTokens, bool hasDenominatorConstant, int
            denominatorIndex, bool hasDenominatorPlaceholder)
        {
            var numEndIndex = GetNumeratorEndIndex(numTokens,
            out var hasIntegerPart);
            var denomEndIndex = GetDenominator(denomTokens,
            hasDenominatorConstant, hasDenominatorPlaceholder, out var
            constant);

            var fractionSuffixIndex = denomTokens.Count;
            while (fractionSuffixIndex > denomEndIndex)
            {
                if
            (Token.IsPlaceholder(denomTokens[fractionSuffixIndex - 1]))
                {
                    break;
                }
                fractionSuffixIndex--;
            }

            IntegerPart = hasIntegerPart ?
            numTokens.GetRange(0, numEndIndex) : null;
            Numerator = hasIntegerPart ?
            numTokens.GetRange(numEndIndex, numTokens.Count - numEndIndex) :
            numTokens;
            DenominatorPrefix = denominatorIndex > 0 ?
            denomTokens.GetRange(0, denominatorIndex) : null;
            Denominator =
            denomTokens.GetRange(denominatorIndex, denomEndIndex -
            denominatorIndex);
            DenominatorConstant = hasDenominatorConstant ?
            int.Parse(constant) : 0;
        }
    }
}
```

```

        DenominatorSuffix = denomEndIndex <
fractionSuffixIndex ? denomTokens.GetRange(denomEndIndex,
fractionSuffixIndex - denomEndIndex) : null;
        FractionSuffix = fractionSuffixIndex <
denomTokens.Count ? denomTokens.GetRange(fractionSuffixIndex,
denomTokens.Count - fractionSuffixIndex) : null;
    }

    private int GetNumeratorEndIndex(List<string> tokens,
out bool hasIntegerPart)
    {
        var hasPlaceholder = false;
        var hasSpace = false;
        var numeratorIndex = -1;
        var index = tokens.Count - 1;

        hasIntegerPart = false;
        while (index >= 0)
        {
            if (Token.IsPlaceholder(tokens[index]))
            {
                hasPlaceholder = true;

                if (hasSpace)
                {
                    hasIntegerPart = true;
                    break;
                }
            }
            else
            {
                if (hasPlaceholder && !hasSpace)
                {
                    hasSpace = true;
                    numeratorIndex = index + 1;
                }
            }
            index--;
        }

        return numeratorIndex;
    }

    private int GetDenominator(List<string> tokens, bool
hasConstant, bool hasPlaceholder, out string constant)
    {
        var index = 0;

        constant = string.Empty;
        while (index < tokens.Count)
        {

```

```

        if (hasConstant &&
Token.IsDigitWithZero(tokens[index]))
        {
            constant += tokens[index];
        }
        else if (!(hasPlaceholder &&
Token.IsPlaceholder(tokens[index])))
        {
            break;
        }
        index++;
    }

    return index;
}

public static bool TryParse(List<string> tokens, out
FractionSection format)
{
    format = null;
    var fractionSplitterIndex = tokens.IndexOf("/");
    if (fractionSplitterIndex == -1) return false;

    var numTokens = tokens.GetRange(0,
fractionSplitterIndex);
    fractionSplitterIndex++;
    var denomTokens =
tokens.GetRange(fractionSplitterIndex, tokens.Count -
fractionSplitterIndex);

    var hasDenominatorPlaceholder = false;
    var hasDenominatorConstant = false;
    var denominatorIndex = 0;

    while (denominatorIndex < denomTokens.Count)
    {
        var token = denomTokens[denominatorIndex];
        if (Token.IsPlaceholder(token))
        {
            hasDenominatorPlaceholder = true;
            break;
        }

        if (Token.IsDigitWithoutZero(token))
        {
            hasDenominatorConstant = true;
            break;
        }
        denominatorIndex++;
    }

    if (!hasDenominatorPlaceholder &&
!hasDenominatorConstant)

```



```

        {
            return false;
        }

        format = new FractionSection(numTokens,
denomTokens, hasDenominatorConstant, denominatorIndex,
hasDenominatorPlaceholder);

        return true;
    }
}

```

Файл IFormatter.cs

```

namespace FormatterLib
{
    public interface IFormatter
    {
        string Code { get; }
        Color FontColor { get; set; }
        string Format(string cellValue);
    }
}

```

Файл ISection.cs

```

namespace FormatterLib
{
    public interface ISection
    {
    }
}

```

Файл LocalizedToken.cs

```

using System;

namespace FormatterLib
{
    internal class LocalizedToken : Token
    {
        public new static bool IsGeneral(string token)
        {
            return string.Compare(token, "основной",
StringComparison.OrdinalIgnoreCase) == 0;
        }

        public new static bool IsDatePart(string token)
        {

```

```

        return
            token.StartsWith("r",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("m",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("д",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("с",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("ч",
StringComparison.OrdinalIgnoreCase) ||
            (token.StartsWith("g",
StringComparison.OrdinalIgnoreCase) && !IsGeneral(token)) ||
            string.Compare(token, "am/pm",
StringComparison.OrdinalIgnoreCase) == 0 ||
            string.Compare(token, "a/p",
StringComparison.OrdinalIgnoreCase) == 0 ||
            IsDurationPart(token);
    }

    public new static bool IsDurationPart(string token)
    {
        return
            token.StartsWith("[ч",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("[м",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("[с",
StringComparison.OrdinalIgnoreCase);
    }
}
}

```

Файл NumberFormat.cs

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;

namespace FormatterLib
{
    public class NumberFormat
    {
        private string _localizedCode;
        private readonly bool _needTranslateToEn;

        public bool IsValid { get; }
        public string FormatString { get; private set; }
        public List<SectionInfo> Sections { get; private set; }
    }
}

```

```

set; }

        public SectionInfo ActiveSectionInfo { get; private
        public bool IsDateFormat { get; }
        public bool IsTimeFormat { get; }

        public string LocalizedFormatString
        {
            get
            {
                return _localizedCode;
            }
            set
            {
                _localizedCode = value;
                if (_needTranslateToEn)
                {
                    FormatString = TranslateToEn(value);
                }
            }
        }

        public NumberFormat(string formatString, bool
isRuCode = false)
        {
            if (isRuCode)
            {
                _needTranslateToEn = true;
                LocalizedFormatString = formatString;
            }
            else
            {
                _needTranslateToEn = false;
                FormatString = formatString;
                LocalizedFormatString =
TranslateToRu(formatString);
            }

            var sections = Parser.ParseSections(FormatString,
out bool syntaxError);

            IsValid = !syntaxError;

            if (IsValid)
            {
                Sections = sections;
                IsDateFormat =
GetFirstSectionOfType(SectionType.Date) != null;
                IsTimeFormat =
GetFirstSectionOfType(SectionType.Duration) != null;
            }
            else
            {

```

```

        Sections = new List<SectionInfo>();
    }
}

    public FormattedValue Format(object value,
CultureInfo culture)
    {
        if (value == null)
        {
            return new FormattedValue { Color = null,
Text = null };
        }

        if (double.TryParse(value.ToString(), out var
dValue))
        {
            value = dValue;
        }
        else if (DateTime.TryParse(value.ToString(), out
var dtValue))
        {
            value = dtValue;
        }

        var section = GetSection(value);
        ActiveSectionInfo = section;

        if (section == null)
            return new FormattedValue()
            {
                Text =
Formatter.ApplyGeneralFormat(value, culture),
                Color = null
            };

        try
        {
            return new FormattedValue()
            {
                Text = Formatter.Format(value, section,
culture),
                Color = section.Color
            };
        }
        catch (Exception)
        {
            return new FormattedValue()
            {
                Text =
Formatter.ApplyGeneralFormat(value, culture),
                Color = null
            };
        }
    }
}

```

```

    }

    public static string TranslateToEn(string formatCode)
    {
        var translatedCode = string.Empty;
        var tokenizer = new TokenReader(formatCode);
        while (true)
        {
            var tokens =
RuToEnParser.ParseSection(tokenizer, out var sectionSyntaxError);

            if (sectionSyntaxError)
            {
                return null;
            }
            if (tokens == null)
            {
                translatedCode =
translatedCode.Remove(translatedCode.Length - 1);
                break;
            }

            translatedCode =
tokens.Aggregate(translatedCode, (current, token) => current +
token);

            translatedCode += ';';
        }

        return translatedCode;
    }

    public static string TranslateToRu(string formatCode)
    {
        var translatedCode = string.Empty;
        var tokenizer = new TokenReader(formatCode);
        while (true)
        {
            var tokens =
EnToRuParser.ParseSection(tokenizer, out var sectionSyntaxError);

            if (sectionSyntaxError)
            {
                return null;
            }

            if (tokens == null)
            {
                translatedCode =
translatedCode.Remove(translatedCode.Length - 1);
                break;
            }

```

```

        translatedCode =
tokens.Aggregate(translatedCode, (current, token) => current +
token);

        translatedCode += ';';
    }

    return translatedCode;
}

public SectionInfo GetSection(object value)
{
    switch (value)
    {
        case double d:
            return GetNumericSection(d);
        case int i:
            return GetNumericSection(i);
        case short s:
            return GetNumericSection(s);
        case string _:
            return Sections.Count >= 4 ? Sections[3]
: null;

        case DateTime _:
            return
GetFirstSectionOfType(SectionType.Date);
        case TimeSpan ts:
            return GetNumericSection(ts.TotalDays);
        default:
            return null;
    }
}

public SectionInfo GetFirstSectionOfType(SectionType
type)
{
    return Sections.FirstOrDefault(section =>
section.Type == type);
}

private bool IsValidSectionForValue(int sectionIndex,
double value)
{
    if (Sections[sectionIndex].Condition != null)
    {
        if
(Sections[sectionIndex].Condition.Apply(value))
        {
            return true;
        }
    }

    return false;
}

```

```

    }

    private SectionInfo GetNumericSection(double value)
    {
        if (Sections.Count < 1)
        {
            return null;
        }

        if (IsValidSectionForValue(0, value) ||
Sections.Count == 1 || (Sections.Count == 2 && value >= 0) ||
(Sections.Count >= 2 && value > 0))
        {
            return Sections[0];
        }

        if (Sections.Count < 2)
        {
            return null;
        }

        if (IsValidSectionForValue(1, value) || value < 0
|| (Sections.Count == 2 && Sections[0].Condition != null))
        {
            return Sections[1];
        }

        return Sections.Count < 3 ? null : Sections[2];
    }
}
}

```

Файл Parser.cs

```

using System.Collections.Generic;
using System.Globalization;

namespace FormatterLib
{
    internal class Parser
    {
        public static List<SectionInfo> ParseSections(string
formatString, out bool syntaxError)
        {
            var tokenReader = new TokenReader(formatString);
            var sections = new List<SectionInfo>();
            syntaxError = false;
            while (true)
            {
                var section = ParseSection(tokenReader,
sections.Count, out var sectionSyntaxError);

```

```

        if (sectionSyntaxError)
            syntaxError = true;

        if (section == null)
            break;

        sections.Add(section);
    }

    return sections;
}

private static SectionInfo ParseSection(TokenReader
reader, int index, out bool syntaxError)
{
    var hasDateParts = false;
    var hasDurationParts = false;
    var hasGeneralPart = false;
    var hasTextPart = false;
    var hasPlaceholders = false;
    Condition condition = null;
    Color color = null;
    string token;
    var tokens = new List<string>();

    while ((token = ReadToken(reader, out
syntaxError)) != null)
    {
        if (token == ";")
            break;

        hasPlaceholders |=
Token.IsPlaceholder(token);

        if (Token.IsDatePart(token))
        {
            hasDateParts = true;
            hasDurationParts |=
Token.IsDurationPart(token);
            tokens.Add(token);
        }
        else if (Token.IsGeneral(token))
        {
            hasGeneralPart = true;
            tokens.Add(token);
        }
        else if (token == "@")
        {
            hasTextPart = true;
            tokens.Add(token);
        }
        else if (token.StartsWith("["))
        {

```



```

token.Length - 2);
var expression = token.Substring(1,
if (TryParseCondition(expression, out var
parseCondition))
    condition = parseCondition;
else if (TryParseColor(expression, out
var parseColor))
    color = parseColor;
else if
(TryParseCurrencySymbol(expression, out var parseCurrencySymbol))
    tokens.Add("\"" + parseCurrencySymbol
+ "\"");
    }
    else
    {
        tokens.Add(token);
    }
}

if (syntaxError || tokens.Count == 0)
{
    return null;
}

if (
hasTextPart)) ||
    (hasDateParts && (hasGeneralPart ||
    (hasGeneralPart && (hasTextPart)))
{
    syntaxError = true;
    return null;
}

ISection section = null;
SectionType type;
List<string> generalTextDateDuration = null;

if (hasDateParts)
{
    type = hasDurationParts ?
SectionType.Duration : SectionType.Date;

    ParseMilliseconds(tokens, out
generalTextDateDuration);
}
else if (hasGeneralPart)
{
    type = SectionType.General;
    generalTextDateDuration = tokens;
}
else if (hasTextPart || !hasPlaceholders)
{
    type = SectionType.Text;

```

```

        generalTextDateDuration = tokens;
    }
    else if (FractionSection.TryParse(tokens, out var
fraction))
    {
        type = SectionType.Fraction;
        section = fraction;
    }
    else if (DecimalSection.TryParse(tokens, out var
number))
    {
        type = SectionType.Number;
        section = number;
    }
    else if (ExponentialSection.TryParse(tokens, out
var exponential))
    {
        type = SectionType.Exponential;
        section = exponential;
    }

    else
    {
        syntaxError = true;
        return null;
    }

    return new SectionInfo()
    {
        Type = type,
        SectionIndex = index,
        Color = color,
        Condition = condition,
        Section = section,
        OtherParts = generalTextDateDuration
    };
}

internal static int ParseNumberTokens(List<string>
tokens, int startPosition, out List<string> beforeDecimal, out
bool decimalSeparator, out List<string> afterDecimal)
{
    beforeDecimal = null;
    afterDecimal = null;
    decimalSeparator = false;

    var remainder = new List<string>();
    var index = 0;

    for (index = 0; index < tokens.Count; ++index)
    {
        var token = tokens[index];
        if (token == "." && beforeDecimal == null)

```

```

        {
            decimalSeparator = true;
            beforeDecimal = tokens.GetRange(0,
index);

            remainder = new List<string>();
        }
        else if (Token.IsNumberLiteral(token))
        {
            remainder.Add(token);
        }
        else if (token.StartsWith("["))
        {
        }
        else
        {
            break;
        }
    }

    if (remainder.Count <= 0) return index;

    if (beforeDecimal != null)
    {
        afterDecimal = remainder;
    }
    else
    {
        beforeDecimal = remainder;
    }

    return index;
}

private static void
ParseMilliseconds(IReadOnlyList<string> tokens, out List<string>
result)
{
    result = new List<string>();
    for (var i = 0; i < tokens.Count; i++)
    {
        var token = tokens[i];
        if (token == ".")
        {
            var zeros = 0;
            while (i + 1 < tokens.Count && tokens[i +
1] == "0")
            {
                i++;
                zeros++;
            }

```

```

        if (zeros > 0)
            result.Add("." + new string('0',
zeros));

        else
            result.Add(".");
    }
    else
    {
        result.Add(token);
    }
}

internal static string ReadToken(TokenReader reader,
out bool syntaxError)
{
    var offset = reader.Position;
    if (
        ReadLiteral(reader) ||
        reader.ReadEnclosed('[', ']') ||

        reader.ReadOneOf("#?;!&%+-
$€£₽0123456789{ } ( ) ; / . @ ") ||
        reader.ReadString("e+", true) ||
        reader.ReadString("e-", true) ||
        reader.ReadString("General", true) ||

        reader.ReadString("am/pm", true) ||
        reader.ReadString("a/p", true) ||
        reader.ReadOneOrMore('y') ||
        reader.ReadOneOrMore('Y') ||
        reader.ReadOneOrMore('m') ||
        reader.ReadOneOrMore('M') ||
        reader.ReadOneOrMore('d') ||
        reader.ReadOneOrMore('D') ||
        reader.ReadOneOrMore('h') ||
        reader.ReadOneOrMore('H') ||
        reader.ReadOneOrMore('s') ||
        reader.ReadOneOrMore('S') ||
        reader.ReadOneOrMore('g') ||
        reader.ReadOneOrMore('G'))
    {
        syntaxError = false;
        var length = reader.Position - offset;
        return reader.Substring(offset, length);
    }

    syntaxError = reader.Position < reader.Length;
    if (syntaxError)
    {
        syntaxError = true;
    }
    return null;
}

```

```

    }

    internal static bool ReadLiteral(TokenReader reader)
    {
        if (reader.Peek() == '\\\\' || reader.Peek() == '*'
|| reader.Peek() == '_')
        {
            reader.Advance(2);
            return true;
        }
        else if (reader.ReadEnclosed("'", "''))
        {
            return true;
        }

        return false;
    }

    internal static bool TryParseCondition(string token,
out Condition result)
    {
        var tokenizer = new TokenReader(token);

        if (tokenizer.ReadString("<=") ||
            tokenizer.ReadString("<>") ||
            tokenizer.ReadString("<") ||
            tokenizer.ReadString(">=") ||
            tokenizer.ReadString(">") ||
            tokenizer.ReadString("="))
        {
            var conditionPosition = tokenizer.Position;
            var op = tokenizer.Substring(0,
conditionPosition);

            if (ReadConditionValue(tokenizer))
            {
                var valueString =
tokenizer.Substring(conditionPosition, tokenizer.Position -
conditionPosition);

                result = new Condition()
                {
                    Op = op,
                    Value = double.Parse(valueString,
CultureInfo.InvariantCulture)
                };
                return true;
            }

            result = null;
            return false;
        }
    }
}

```

```

private static bool ReadConditionValue(TokenReader
tokenReader)
{
    tokenReader.ReadString("-");
    while (tokenReader.ReadOneOf("0123456789"))
    {
    }

    if (tokenReader.ReadString("."))
    {
        while (tokenReader.ReadOneOf("0123456789"))
        {
        }
    }

    if (!tokenReader.ReadString("e+", true) &&
!tokenReader.ReadString("e-", true)) return true;

    if (tokenReader.ReadOneOf("0123456789"))
    {
        while (tokenReader.ReadOneOf("0123456789"))
        {
        }
    }
    else
    {
        return false;
    }

    return true;
}

internal static bool TryParseColor(string token, out
Color color)
{
    var tokenizer = new TokenReader(token);
    if (
        tokenizer.ReadString("black", true) ||
        tokenizer.ReadString("blue", true) ||
        tokenizer.ReadString("cyan", true) ||
        tokenizer.ReadString("green", true) ||
        tokenizer.ReadString("magenta", true) ||
        tokenizer.ReadString("red", true) ||
        tokenizer.ReadString("white", true) ||
        tokenizer.ReadString("yellow", true))
    {
        color = new Color
        {
            Value = tokenizer.Substring(0,
tokenizer.Position)
        };
        return true;
    }
}

```

```

    }

    color = null;
    return false;
}

internal static bool TryParseCurrencySymbol(string
token, out string currencySymbol)
{
    if (string.IsNullOrEmpty(token)
        || !token.StartsWith("$"))
    {
        currencySymbol = null;
        return false;
    }

    currencySymbol = token.Contains("-") ?
token.Substring(1, token.IndexOf('-') - 1) : token.Substring(1);

    return true;
}
}
}

```

Файл RuToEnParser.cs

```

using System.Collections.Generic;

namespace FormatterLib
{
    internal class RuToEnParser : Parser
    {
        internal static List<string> ParseSection(TokenReader
reader, out bool syntaxError)
        {
            string token;
            var tokens = new List<string>();

            while ((token = ReadToken(reader, out
syntaxError)) != null)
            {
                if (token == ";")
                    break;

                Token.IsPlaceholder(token);

                if (LocalizedToken.IsDatePart(token))
                {
                    var translatedToken = string.Empty;
                    LocalizedToken.IsDurationPart(token);

```

```

        foreach (var c in token.ToLower())
        {
            switch (c)
            {
                case 'г':
                    translatedToken += "y";
                    break;
                case 'м':
                    translatedToken += "m";
                    break;
                case 'д':
                    translatedToken += "d";
                    break;
                case 'ч':
                    translatedToken += "h";
                    break;
                case 'с':
                    translatedToken += "s";
                    break;
                default:
                    translatedToken += c;
                    break;
            }

            tokens.Add(translatedToken);
        }
    }
    else if (LocalizedToken.IsGeneral(token))
    {
        token =
token.ToLower().Replace("основной", "general");
        tokens.Add(token);
    }
    else if (token == "@")
    {
        tokens.Add(token);
    }
    else if (token.StartsWith("[")
token.Length - 2);
        if (TryParseCondition(expression, out var
parseCondition))
        tokens.Add($"[{parseCondition.Op}{parseCondition.Value}]");

        else if (TryParseColor(expression, out
var parseColor))
        {
            var localizedColor = string.Empty;
            switch (parseColor.Value.ToLower())
            {

```



```

        case "черный":
            localizedColor = "black";
            break;
        case "синий":
            localizedColor = "blue";
            break;
        case "голубой":
            localizedColor = "cyan";
            break;
        case "зеленый":
            localizedColor = "green";
            break;
        case "фиолетовый":
            localizedColor = "magenta";
            break;
        case "красный":
            localizedColor = "red";
            break;
        case "белый":
            localizedColor = "white";
            break;
        case "желтый":
            localizedColor = "yellow";
            break;
    }
    tokens.Add($"[{localizedColor}]");
}
else if
(TryParseCurrencySymbol(expression, out var parseCurrencySymbol))
    tokens.Add("\"" + parseCurrencySymbol
+ "\"");
    }
    else
    {
        tokens.Add(token);
    }
}

if (syntaxError || tokens.Count == 0)
{
    return null;
}

return tokens;
}

internal new static string ReadToken(TokenReader
reader, out bool syntaxError)
{
    var offset = reader.Position;
    if (
        ReadLiteral(reader) ||
        reader.ReadEnclosed('[', ']') ||

```

```

        reader.ReadOneOf("#?;!&%+-
$€£0123456789{}() :;/.@# ") ||
        reader.ReadString("e+", true) ||
        reader.ReadString("e-", true) ||
        reader.ReadString("Основной", true) ||

        reader.ReadString("am/pm", true) ||
        reader.ReadString("a/p", true) ||
        reader.ReadOneOrMore('r') ||
        reader.ReadOneOrMore('Г') ||
        reader.ReadOneOrMore('м') ||
        reader.ReadOneOrMore('М') ||
        reader.ReadOneOrMore('д') ||
        reader.ReadOneOrMore('Д') ||
        reader.ReadOneOrMore('ч') ||
        reader.ReadOneOrMore('Ч') ||
        reader.ReadOneOrMore('с') ||
        reader.ReadOneOrMore('С') ||
        reader.ReadOneOrMore('g') ||
        reader.ReadOneOrMore('G'))
    {
        syntaxError = false;
        var length = reader.Position - offset;
        return reader.Substring(offset, length);
    }

    syntaxError = reader.Position < reader.Length;
    return null;
}

internal new static bool TryParseColor(string token,
out Color color)
{
    var tokenizer = new TokenReader(token);
    if (
        tokenizer.ReadString("черный", true) ||
        tokenizer.ReadString("синий", true) ||
        tokenizer.ReadString("голубой", true) ||
        tokenizer.ReadString("зеленый", true) ||
        tokenizer.ReadString("фиолетовый", true) ||
        tokenizer.ReadString("красный", true) ||
        tokenizer.ReadString("белый", true) ||
        tokenizer.ReadString("желтый", true))
    {
        color = new Color()
        {
            Value = tokenizer.Substring(0,
tokenizer.Position)
        };
        return true;
    }
}

```

```

        color = null;
        return false;
    }
}

```

Файл SectionInfo.cs

```

using System.Collections.Generic;
namespace FormatterLib
{
    public class SectionInfo
    {
        public int SectionIndex { get; set; }
        public SectionType Type { get; set; }
        public Color Color { get; set; }
        public Condition Condition { get; set; }
        public ISection Section { get; set; }
        // General, Text, Date, Duration Parts
        public List<string> OtherParts { get; set; }
    }
}

```

Файл SectionType.cs

```

namespace FormatterLib
{
    public enum SectionType
    {
        General,
        Number,
        Fraction,
        Exponential,
        Date,
        Duration,
        Text,
    }
}

```

Файл Token.cs

```

using System;
namespace FormatterLib
{
    internal class Token
    {
        public static bool IsExponent(string token)
        {
            return

```

```

        (string.Compare(token, "e+",
StringComparison.OrdinalIgnoreCase) == 0) ||
        (string.Compare(token, "e-",
StringComparison.OrdinalIgnoreCase) == 0);
    }
    public static bool IsLiteral(string token)
    {
        return
            token.StartsWith(" ") ||
            token.StartsWith("\\"") ||
            token.StartsWith("\"") ||
            token.StartsWith("*") ||
            token == "," ||
            token == "!" ||
            token == "&" ||
            token == "%" ||
            token == "+" ||
            token == "-" ||
            token == "$" ||
            token == "€" ||
            token == "£" ||
            token == "₽" ||
            token == "1" ||
            token == "2" ||
            token == "3" ||
            token == "4" ||
            token == "5" ||
            token == "6" ||
            token == "7" ||
            token == "8" ||
            token == "9" ||
            token == "{" ||
            token == "}" ||
            token == "(" ||
            token == ")" ||
            token == " ";
    }
    public static bool IsNumberLiteral(string token)
    {
        return
            IsPlaceholder(token) ||
            IsLiteral(token) ||
            token == ".";
    }
    public static bool IsPlaceholder(string token)
    {
        return token == "0" || token == "#" || token ==
"?";
    }
    public static bool IsGeneral(string token)
    {
        return string.Compare(token, "general",
StringComparison.OrdinalIgnoreCase) == 0;
    }

```

```

    }
    public static bool IsDatePart(string token)
    {
        return
            token.StartsWith("y",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("m",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("d",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("s",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("h",
StringComparison.OrdinalIgnoreCase) ||
            (token.StartsWith("g",
StringComparison.OrdinalIgnoreCase) && !IsGeneral(token)) ||
            string.Compare(token, "am/pm",
StringComparison.OrdinalIgnoreCase) == 0 ||
            string.Compare(token, "a/p",
StringComparison.OrdinalIgnoreCase) == 0 ||
            IsDurationPart(token);
    }
    public static bool IsDurationPart(string token)
    {
        return
            token.StartsWith("[h",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("[m",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("[s",
StringComparison.OrdinalIgnoreCase);
    }
    public static bool IsDigitWithZero(string token)
    {
        return token == "0" || IsDigitWithoutZero(token);
    }
    public static bool IsDigitWithoutZero(string token)
    {
        switch (token)
        {
            case "1":
            case "2":
            case "3":
            case "4":
            case "5":
            case "6":
            case "7":
            case "8":
            case "9":
                return true;
            default:
                return false;
        }
    }

```

```

    }
}
}

```

Файл TokenReader.cs

```

using System;
using System.Linq;
namespace FormatterLib
{
    internal class TokenReader
    {
        private readonly string _formatString;
        public TokenReader(string fmt)
        {
            _formatString = fmt;
        }
        public int Position { get; private set; }
        public int Length => _formatString?.Length ?? 0;
        public string Substring(int startIndex, int length)
        {
            return _formatString.Substring(startIndex,
length);
        }
        public int Peek(int offset = 0)
        {
            if (Position + offset >= Length)
                return -1;
            return _formatString[Position + offset];
        }
        public int PeekUntil(int startOffset, int until)
        {
            var offset = startOffset;
            while (true)
            {
                var c = Peek(offset++);
                if (c == -1)
                    break;
                if (c == until)
                    return offset - startOffset;
            }
            return 0;
        }
        public bool PeekOneOf(int offset, string s)
        {
            return s.Any(c => Peek(offset) == c);
        }
        public void Advance(int characters = 1)
        {
            Position = Math.Min(Position + characters,
_formatString.Length);
        }
    }
}

```

```

    }
    public bool ReadOneOrMore(int c)
    {
        if (Peek() != c)
            return false;
        while (Peek() == c)
            Advance();
        return true;
    }
    public bool ReadOneOf(string s)
    {
        if (!PeekOneOf(0, s)) return false;
        Advance();
        return true;
    }
    public bool ReadString(string s, bool ignoreCase =
false)
    {
        if (Position + s.Length > Length)
            return false;
        for (var i = 0; i < s.Length; i++)
        {
            var c1 = s[i];
            var c2 = (char)Peek(i);
            if (ignoreCase)
            {
                if (char.ToLower(c1) != char.ToLower(c2))
return false;
            }
            else
            {
                if (c1 != c2) return false;
            }
        }
        Advance(s.Length);
        return true;
    }
    public bool ReadEnclosed(char open, char close)
    {
        if (Peek() != open) return false;
        var length = PeekUntil(1, close);
        if (length <= 0) return false;
        Advance(1 + length);
        return true;
    }
}
}

```

Приложение Б.2 - Модуль сортировки и фильтрации данных

Файл AllFilters.cs

					ПГУ 2.090301.39.001 ПЗ	Лист
						129
Изм.	Лист	№ докум.	Подпись	Дата		

```

using System;
using System.Collections.Generic;

namespace FiltersLib
{
    public class AllFilters
    {
        public enum FilterOp
        {
            AND,
            OR
        }

        public enum MonthsNames
        {
            January = 1,
            February,
            March,
            April,
            May,
            June,
            July,
            August,
            September,
            October,
            November,
            December
        }

        public enum QuartersNames
        {
            FirstQuarter = 1,
            SecondQuarter,
            ThirdQuarter,
            FourthQuarter,
        }

        public enum Days
        {
            Yesterday = -1,
            Today,
            Tomorrow
        }

        public enum Weeks
        {
            LastWeek = -7,
            ThisWeek = 0,
            NextWeek = 7
        }

        public enum Months
        {

```



```

        LastMonth = -1,
        ThisMonth,
        NextMonth
    }

    public enum Quarters
    {
        LastQuarter = -1,
        ThisQuarter,
        NextQuarter
    }

    public enum Years
    {
        LastYear = -1,
        ThisYear,
        NextYear,
        FromBeginningYear
    }

    public static readonly Dictionary<string,
Func<string[], string, int[]>> textFilters =
        new Dictionary<string, Func<string[], string,
int[]>>()
    {
        { "Равно", TextFilters.Equals },
        { "Не равно", TextFilters.NotEquals },
        { "Начинается с", TextFilters.StartsWith },
        { "Заканчивается на", TextFilters.EndsWith },
        { "Содержит", TextFilters.Contains },
        { "Не содержит", TextFilters.NotContains }
    };

    public static readonly Dictionary<string,
Func<double[], double, int[]>> numericalFilters =
        new Dictionary<string, Func<double[], double,
int[]>>()
    {
        { "Равно", NumericalFilters.Equals },
        { "Не равно", NumericalFilters.NotEquals },
        { "Больше", NumericalFilters.Greater },
        { "Больше или равно",
NumericalFilters.GreaterOrEquals },
        { "Меньше", NumericalFilters.LessThan},
        { "Меньше или равно",
NumericalFilters.LessThanOrEquals },
        { "Между", null },
        { "Первые 10", null },
    };

    public static readonly Dictionary<string,
Func<double[], int[]>> numericalAvgFilters =
        new Dictionary<string, Func<double[], int[]>>()

```

```

        {
            { "Выше среднего",
NumericalFilters.AboveAverage },
            { "Ниже среднего",
NumericalFilters.BelowAverage }
        };

        public static readonly Dictionary<string,
Func<DateTime[], DateTime, int[]>> dateTimeFilters =
            new Dictionary<string, Func<DateTime[], DateTime,
int[]>>()
        {
            { "Равно", DateTimeFilters.Equals },
            { "Не равно", DateTimeFilters.NotEquals },
            { "После", DateTimeFilters.After },
            { "После или равно",
DateTimeFilters.AfterOrEquals },
            { "До", DateTimeFilters.Before },
            { "До или равно", DateTimeFilters.BeforeOrEquals
},
        };

        public static readonly Dictionary<string, object>
intervalFilters = new Dictionary<string, object>()
        {
            {"Завтра", Days.Tomorrow},
            {"Сегодня", Days.Today},
            {"Вчера", Days.Yesterday},
            {"На следующей неделе", Weeks.NextWeek},
            {"На этой неделе", Weeks.ThisWeek},
            {"На прошлой неделе", Weeks.LastWeek},
            {"В следующем месяце", Months.NextMonth},
            {"В этом месяце", Months.ThisMonth},
            {"В прошлом месяце", Months.LastMonth},
            {"В следующем квартале", Quarters.NextQuarter},
            {"В этом квартале", Quarters.ThisQuarter},
            {"В прошлом квартале", Quarters.LastQuarter},
            {"В следующем году", Years.NextYear},
            {"В этом году", Years.ThisYear},
            {"В прошлом году", Years.LastYear},
            {"С начала года", Years.FromBeginningYear},
            {"Все даты за период", null}
        };

        public static readonly Dictionary<string, object>
periodFilters = new Dictionary<string, object> {
            { "Квартал 1", QuartersNames.FirstQuarter },
            { "Квартал 2", QuartersNames.SecondQuarter },
            { "Квартал 3", QuartersNames.ThirdQuarter },
            { "Квартал 4", QuartersNames.FourthQuarter },
            { "Январь", MonthsNames.January },
            { "Февраль", MonthsNames.February },
            { "Март", MonthsNames.March },

```

```

        { "Апрель", MonthsNames.April },
        { "Май", MonthsNames.May },
        { "Июнь", MonthsNames.June },
        { "Июль", MonthsNames.July },
        { "Август", MonthsNames.August },
        { "Сентябрь", MonthsNames.September },
        { "Октябрь", MonthsNames.October },
        { "Ноябрь", MonthsNames.November },
        { "Декабрь", MonthsNames.December }
    };
}
}

```

Файл DateTimeFilters.cs

```

using System;
using System.Linq;

namespace FiltersLib
{
    class DateTimeFilters
    {
        public static int[] Equals(DateTime[] array, DateTime
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => (x.v > param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] NotEquals(DateTime[] array,
DateTime param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => !(x.v.Equals(param)))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] After(DateTime[] array, DateTime
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => (x.v > param))
                        .Select(x => x.i)
                        .ToArray();
        }
    }
}

```

```

        public static int[] AfterOrEquals(DateTime[] array,
DateTime param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v >= param))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] Before(DateTime[] array, DateTime
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v <= param))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] BeforeOrEquals(DateTime[] array,
DateTime param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v <= param))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] Interval(DateTime[] array,
DateTime param1, DateTime param2)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v >= param1) && (x.v <
param2))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] AllFromQuarter(DateTime[] array,
int quarterNum)
        {
            int firstMonthOfQuarter = 1 + (quarterNum - 1) *
3;
            int lastMonthOfQuarter = 1 + (quarterNum * 3);

            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v.Month >=
firstMonthOfQuarter) && (x.v.Month < lastMonthOfQuarter))
                .Select(x => x.i)

```

```

        .ToArray();
    }

    public static int[] AllFromMonth(DateTime[] array,
int month)
    {
        return array.Select((value, index) => new { v =
value, i = index })
            .Where(x => x.v.Month == month)
            .Select(x => x.i)
            .ToArray();
    }
}

```

Файл Filter.cs

```

using System;
using System.Linq;

namespace FiltersLib
{
    public class Executor
    {
        public int[] ApplyFilter(FilterArgs FilterArgs)
        {
            string filterName = FilterArgs.nameOfFirstFilter;
            if
(AllFilters.numericalAvgFilters.ContainsKey(filterName))
            {
                double[] doubleArray =
FilterArgs.values.Cast<double>().ToArray();

                return
AllFilters.numericalAvgFilters[FilterArgs.nameOfFirstFilter].Invok
e(doubleArray);
            }
            else if
(AllFilters.periodFilters.Keys.Contains(filterName))
            {
                return FilterDateTimePeriod(FilterArgs);
            }
            else if
(AllFilters.intervalFilters.Keys.Contains(FilterArgs.nameOfFirstFi
lter))
            {
                return FilterDateTimeInterval(FilterArgs);
            }
            else
            {
                return BaseFilter(FilterArgs);
            }
        }
    }
}

```

```

    }

    private int[] BaseFilter(FilterArgs FilterArgs)
    {
        bool useSecondFilter =
FilterArgs.nameOfSecondFilter != null;
        int[] result1 = null;
        int[] result2 = null;

        Type t = FilterArgs.values[0].GetType();

        if (FilterArgs.nameOfFirstFilter != null)
        {
            if (t == typeof(string))
            {
                string[] stringArray =
FilterArgs.values.Cast<string>().ToArray();
                result1 =
AllFilters.textFilters[FilterArgs.nameOfFirstFilter]
                    .Invoke(stringArray,
FilterArgs.valueOfFirstFilter);

                if (useSecondFilter)
                {
                    result2 =
AllFilters.textFilters[FilterArgs.nameOfSecondFilter]
                        .Invoke(stringArray,
FilterArgs.valueOfSecondFilter);
                }
            }
            else if (t == typeof(double))
            {
                double[] doubleArray =
FilterArgs.values.Cast<double>().ToArray();

                if
(double.TryParse(FilterArgs.valueOfFirstFilter, out double
doubleValue))
                {
                    result1 =
AllFilters.numericalFilters[FilterArgs.nameOfFirstFilter]
                        .Invoke(doubleArray,
doubleValue);

                    if (useSecondFilter)
                    {
                        if
(double.TryParse(FilterArgs.valueOfSecondFilter, out double
doubleValue2))
                        {
                            result2 =
AllFilters.numericalFilters[FilterArgs.nameOfSecondFilter]

```

```

doubleValue2);

        .Invoke(doubleArray,
        }
        else result1 = null;
    }
}
else if (t == typeof(DateTime))
{
    DateTime[] dateTimeArray =
FilterArgs.values.Cast<DateTime>().ToArray();
    if
(DateTime.TryParse(FilterArgs.valueOfFirstFilter, out DateTime
dateTimeValue))
    {
        result1 =
AllFilters.dateTimeFilters[FilterArgs.nameOfFirstFilter]
.Invoke(dateTimeArray,
dateTimeValue);

        if (useSecondFilter)
        {
            if
(DateTime.TryParse(FilterArgs.valueOfSecondFilter, out DateTime
dateTimeValue2))
            {
                result2 =
AllFilters.dateTimeFilters[FilterArgs.nameOfSecondFilter]
.Invoke(dateTimeArray,
dateTimeValue2);
            }
            else result1 = null;
        }
    }
}

if (result1 != null && result2 != null)
{
    if (FilterArgs.filterOp ==
AllFilters.FilterOp.AND)
    {
        result1 =
result1.Intersect(result2).ToArray();
    }
    else if (FilterArgs.filterOp ==
AllFilters.FilterOp.OR)
    {
        result1 =
result1.Union(result2).ToArray();
    }
}

```

```

        return result1;
    }

    private int[] FilterDateTimeInterval(FilterArgs
FilterArgs)
    {
        int[] result = null;

        DateTime[] array =
FilterArgs.values.Cast<DateTime>().ToArray();
        DateTime today = DateTime.Today;
        DateTime startOfThisWeek = today.AddDays(-(
(int)today.DayOfWeek + 1);
        DateTime startOfThisMonth = today.AddDays(-
today.Day + 1);
        DateTime startOfThisYear =
startOfThisMonth.AddMonths(-today.Month + 1);

        int currentQuarter =
(int)Math.Ceiling(today.Month / 3m);
        var selectedFilter =
AllFilters.intervalFilters[FilterArgs.nameOfFirstFilter];
        int num = (int)selectedFilter;

        if (selectedFilter.GetType() ==
typeof(AllFilters.Days))
        {
            result = DateTimeFilters.Interval(array,
today.AddDays(num), today.AddDays(num + 1));
        }
        else if (selectedFilter.GetType() ==
typeof(AllFilters.Weeks))
        {
            result = DateTimeFilters.Interval(array,
startOfThisWeek.AddDays(num), startOfThisWeek.AddDays(num + 7));
        }
        else if (selectedFilter.GetType() ==
typeof(AllFilters.Months))
        {
            result = DateTimeFilters.Interval(array,
startOfThisMonth.AddMonths(num), startOfThisMonth.AddMonths(num +
1));
        }
        else if (selectedFilter.GetType() ==
typeof(AllFilters.Quarters))
        {
            result = DateTimeFilters.Interval(array,
startOfThisYear.AddMonths((currentQuarter + num - 1) * 3),
startOfThisYear.AddMonths((currentQuarter + num) * 3));
        }
        else if (selectedFilter.GetType() ==
typeof(AllFilters.Years))
        {

```



```

        if (num == 2)
        {
            result = DateTimeFilters.Interval(array,
startOfThisYear, today);
        }
        else
        {
            result = DateTimeFilters.Interval(array,
startOfThisYear.AddYears(num), startOfThisMonth.AddYears(num +
1));
        }
    }

    return result;
}

private int[] FilterDateTimePeriod(FilterArgs
FilterArgs)
{
    int[] result = null;
    DateTime[] array =
FilterArgs.values.Cast<DateTime>().ToArray();
    var selectedFilter =
AllFilters.periodFilters[FilterArgs.nameOfFirstFilter];
    int num = (int)selectedFilter;

    if (selectedFilter.GetType() ==
typeof(AllFilters.MonthsNames))
    {
        result = DateTimeFilters.AllFromMonth(array,
num);
    }
    else if (selectedFilter.GetType() ==
typeof(AllFilters.QuartersNames))
    {
        result =
DateTimeFilters.AllFromQuarter(array, num);
    }

    return result;
}

/// <summary>
/// Фильтрует первые N чисел по заданному критерию.
/// </summary>
/// <param name="values">Значения</param>
/// <param name="showLargest">Первые N наибольших или
наименьших</param>
/// <param name="count">Количество чисел</param>
/// <param name="countIsPercent">Вид количества чисел
(процентный или числовой)</param>
/// <returns></returns>

```

```

        public int[] FirstThenFilter(double[] values, bool
showLargest, int count, bool countIsPercent)
        {
            int countOfElements = 0;

            if (!countIsPercent)
            {
                countOfElements = count;
            }

            else countOfElements =
(int)Math.Truncate((double)(count * values.Length / 100));

            if (countOfElements >= values.Length)
            {
                return null;
            }

            if (showLargest)
            {
                return values.Select((value, index) => new {
v = value, i = index })
                    .OrderByDescending(x => x.v)
                    .Select(x => x.i)
                    .Take(countOfElements)
                    .ToArray();
            }
            else
            {
                return values.Select((value, index) => new {
v = value, i = index })
                    .OrderBy(x => x.v)
                    .Select(x => x.i)
                    .Take(countOfElements)
                    .ToArray();
            }
        }

        public int[] Sort(object[] values, bool byDescending)
        {
            if (byDescending)
            {
                return values.Select((value, index) => new {
v = value, i = index })
                    .OrderByDescending(x => x.v)
                    .Select(x => x.i)
                    .ToArray();
            }
            else
            {
                return values.Select((value, index) => new {
v = value, i = index })

```

```

        .OrderBy(x => x.v)
        .Select(x => x.i)
        .ToArray();
    }
}
}

```

Файл FilterArgs.cs

```

namespace FiltersLib
{
    public class FilterArgs
    {
        public object[] values;
        public string nameOfFirstFilter;
        public string valueOfFirstFilter;
        public string nameOfSecondFilter;
        public string valueOfSecondFilter;
        public AllFilters.FilterOp filterOp;

        /// <param name="_values">Значения для
        фильтрации</param>
        /// <param name="_nameOfFirstFilter">Имя первого
        фильтра</param>
        /// <param name="_valueOfFirstFilter">Значение
        первого фильтра</param>
        /// <param name="_nameOfSecondFilter">Имя второго
        фильтра. По умолчанию null</param>
        /// <param name="_valueOfSecondFilter">Значение
        второго фильтра</param>
        /// <param name="_filterOp">Операция над
        фильтрами</param>
        public FilterArgs(object[] _values, string
        _nameOfFirstFilter, string _valueOfFirstFilter = null,
        string _nameOfSecondFilter = null, string
        _valueOfSecondFilter = null, AllFilters.FilterOp _filterOp = 0)
        {
            values = _values;
            nameOfFirstFilter = _nameOfFirstFilter;
            valueOfFirstFilter = _valueOfFirstFilter;
            nameOfSecondFilter = _nameOfSecondFilter;
            valueOfSecondFilter = _valueOfSecondFilter;
            filterOp = _filterOp;
        }
    }
}

```

Файл NumericalFilters.cs

```

using System.Linq;

```

```

namespace FiltersLib
{
    class NumericalFilters
    {
        public static int[] Equals(double[] array, double
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                                .Where(x => x.v.Equals(param))
                                .Select(x => x.i)
                                .ToArray();
        }

        public static int[] NotEquals(double[] array, double
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                                .Where(x => !(x.v.Equals(param)))
                                .Select(x => x.i)
                                .ToArray();
        }

        public static int[] Greater(double[] array, double
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                                .Where(x => (x.v > param))
                                .Select(x => x.i)
                                .ToArray();
        }

        public static int[] GreaterOrEquals(double[] array,
double param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                                .Where(x => (x.v >= param))
                                .Select(x => x.i)
                                .ToArray();
        }

        public static int[] LessThan(double[] array, double
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                                .Where(x => (x.v < param))
                                .Select(x => x.i)
                                .ToArray();
        }
    }
}

```

```

        public static int[] LessThanOrEquals(double[] array,
double param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v <= param))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] AboveAverage(double[] array)
        {
            double avg = array.Average();

            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v > avg))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] BelowAverage(double[] array)
        {
            double avg = array.Average();

            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v < avg))
                .Select(x => x.i)
                .ToArray();
        }
    }
}

```

Файл TextFilters.cs

```

using System.Linq;

namespace FiltersLib
{
    class TextFilters
    {
        public static int[] Equals(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => x.v.Equals(param))
                .Select(x => x.i)
                .ToArray();
        }
    }
}

```

```

        public static int[] NotEquals(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => !x.v.Equals(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] StartWith(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => x.v.StartsWith(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] EndsWith(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => x.v.EndsWith(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] Contains(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => x.v.Contains(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] NotContains(string[] array,
string param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => !x.v.Contains(param))
                        .Select(x => x.i)
                        .ToArray();
        }
    }
}

```

Приложение В

(Обязательное)

Программные средства форматирования, фильтрации и сортировки данных для
табличного процессора

Презентация

					ПГУ 2.090301.39.001 ПЗ	Лист
						145
Изм.	Лист	№ докум.	Подпись	Дата		

Программные средства форматирования, фильтрации и сортировки данных для табличного процессора

Автор работы

студент гр. 18BB2 Сперанский Дмитрий Денисович

Научный руководитель:

к.т.н., доцент, Дорошенко Иван Николаевич

Пенза, 2022

Цель и задачи работы

Цель работы – разработка программных средств для табличного процессора «Ведомость», позволяющих пользователю выполнять форматирование, фильтрацию и сортировку данных, а также проектирование для них графического интерфейса.

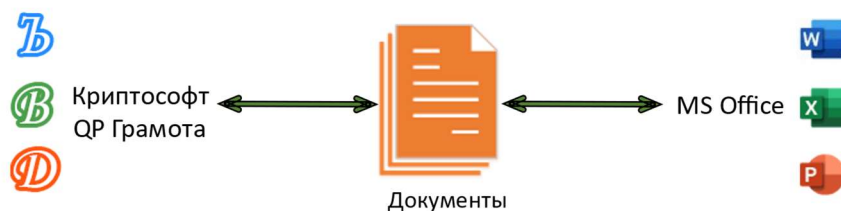
Объекты работы:

1. Модуль форматирования данных . Задачи:
 - Форматирование данных в табличном процессоре
 - Редактирование XML файлов в соответствии с полученным результатом
2. Модуль сортировки и фильтрации данных . Задачи:
 - Сортировка данных в табличном процессоре.
 - Фильтрация данных в табличном процессоре .
 - Редактирование XML файлов в соответствии с полученным результатом

2

Офисный пакет QR Грамота

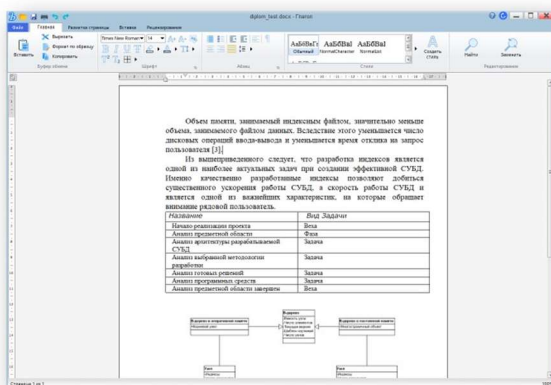
- Текстовый процессор Глагол ;
- Табличный процессор Ведомость ;
- Менеджер презентацией Доклад ;



3



Глагол



- Форматы: DOC, DOCX;
- Текст;
- Списки;
- Таблицы;
- Рисунки;
- Оглавление;
- Колонтитулы.

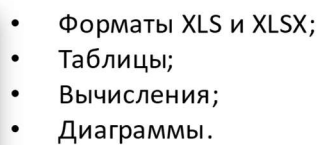
4

Изм.	Лист	№ докум.	Подпись	Дата

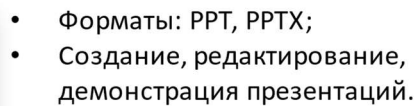
ПГУ 2.090301.39.001 ПЗ

Лист

147



5



6

Задачи модуля форматирования данных

Разрабатываемый программный модуль должен поддерживать форматирование данных согласно стандартным форматам и введенному пользователем кода формата.

Стандартные форматы:

- «Дата», «Время», «Дробный» и др.;
- Всего их 11 штук;
- Имеют базовый код, который может изменяться в зависимости от настроек пользователя.

Пример структуры кода формата приведен на рисунке ниже. Содержит 4 части:

- 1 – для положительных чисел;
- 2 – для отрицательных чисел;
- 3 – для нуля;
- 4 – для текста.


#,###.00_); [Red] (#,###.00); 0.00; "gross receipts for"@

7

Задачи модуля сортировки и фильтрации данных

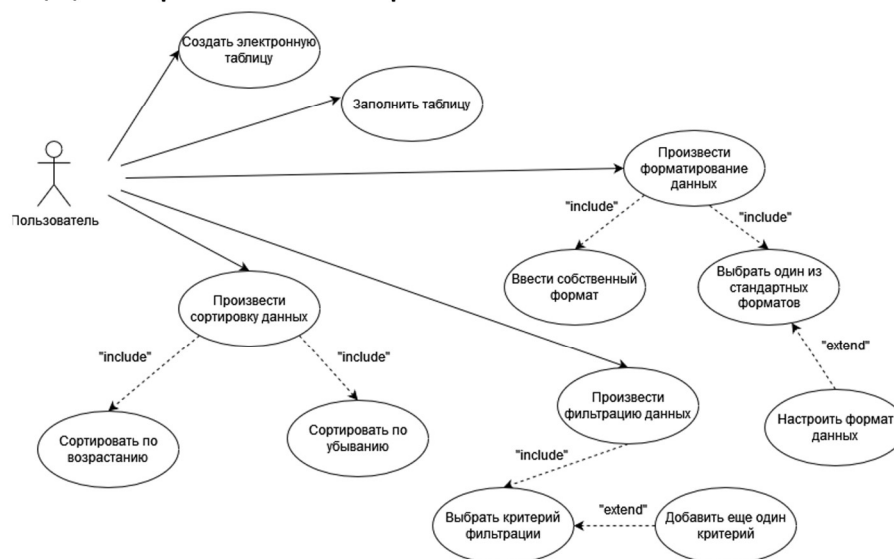
- Поддержка сортировки и фильтрации данных ;
- Поддержка всех фильтров , реализованных в других популярных табличных процессорах;
- Методы сортировки данных, также как и фильтры , должны зависеть от типа введенных пользователем данных.

Всего в табличных процессорах принято разделять три типа данных:

- Числовые;
- Текстовые;
- Дата.

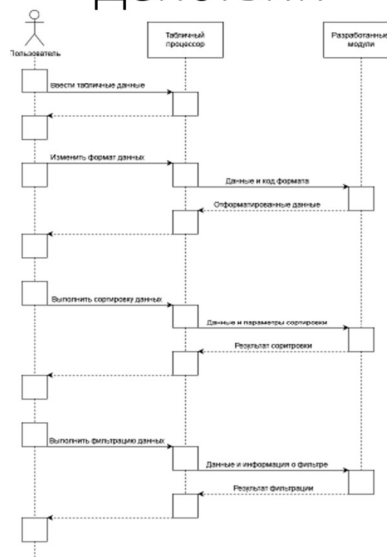
8

UML-диаграмма вариантов использования



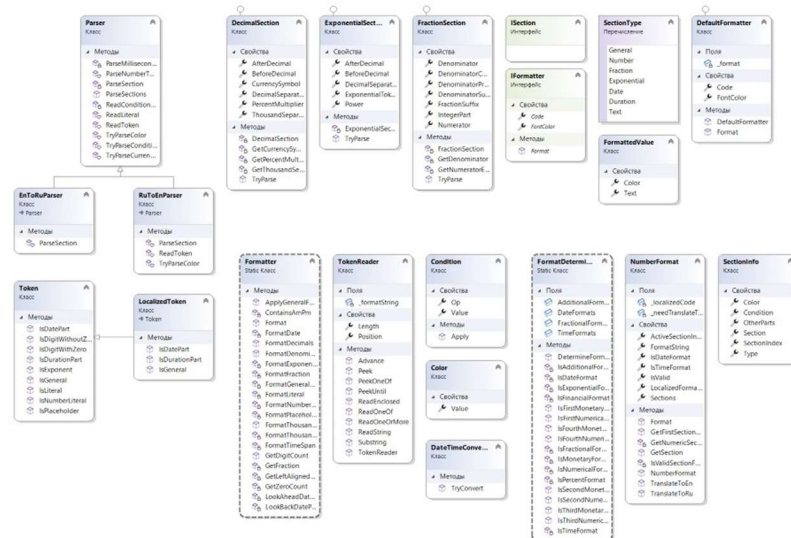
9

UML-диаграмма последовательности действий



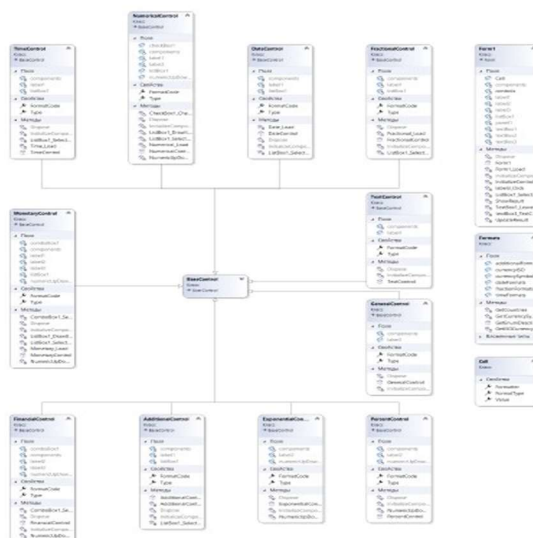
10

UML-диаграмма классов библиотеки форматирования данных



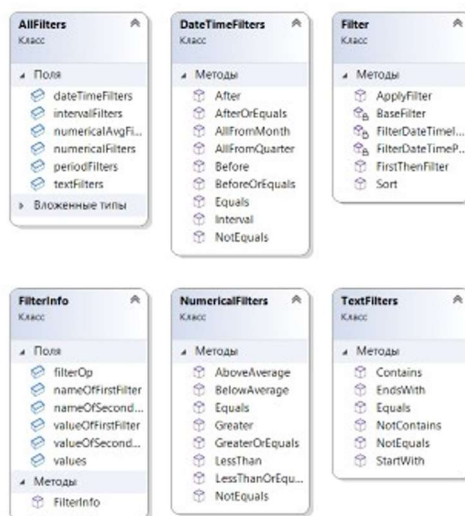
11

UML-диаграмма графического интерфейса модуля форматирования данных



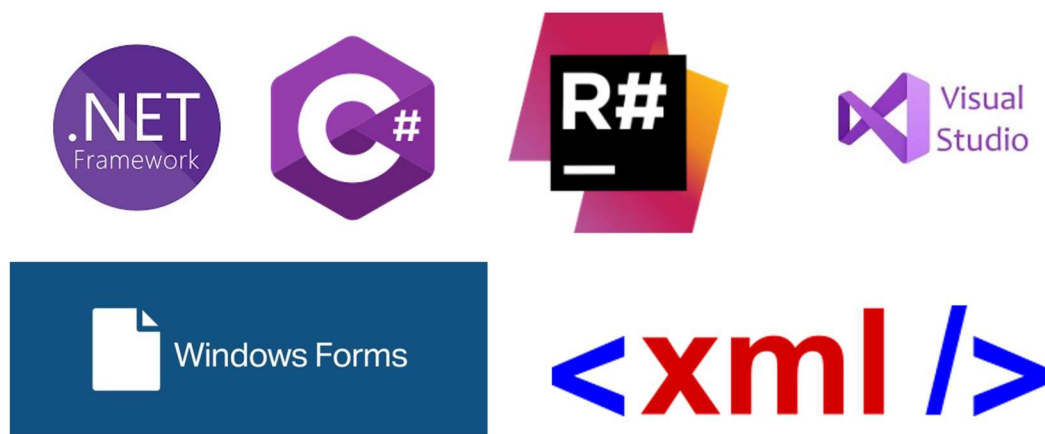
12

UML-диаграмма классов библиотеки фильтрации и сортировки данных



13

Используемые инструменты и технологии



14

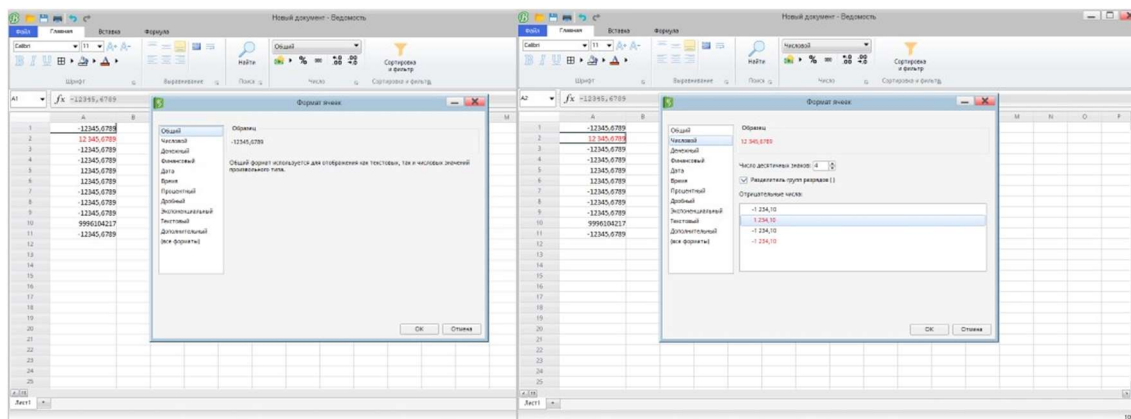
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

152

Формат «Общий» и «Числовой»

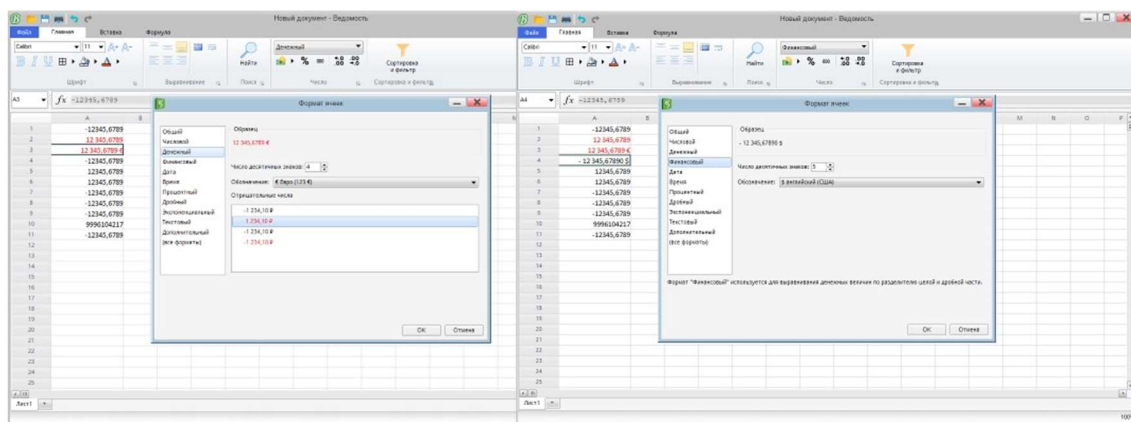


На рисунке изображен результат применения формата «Общий»

На рисунке изображен результат применения формата «Числовой»

15

Формат «Денежный» и «Финансовый»

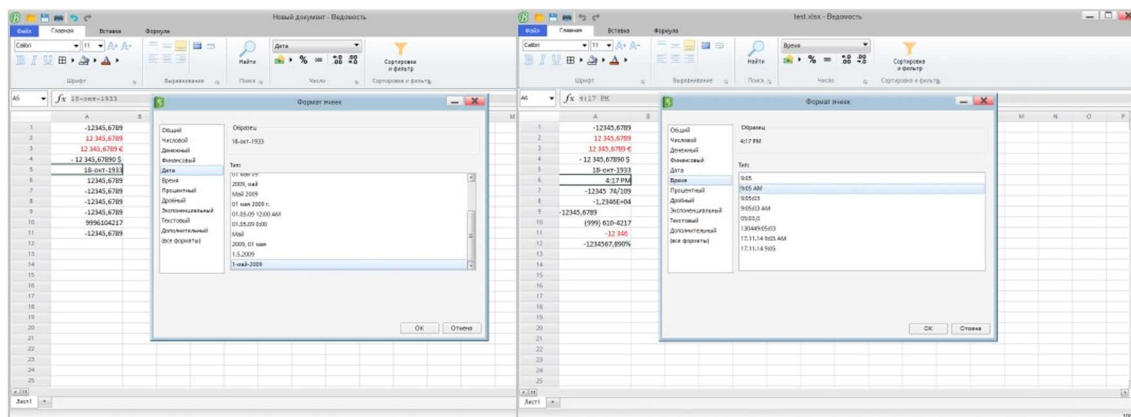


На рисунке изображен результат применения формата «Денежный»

На рисунке изображен результат применения формата «Финансовый»

16

Формат «Дата» и «Время»

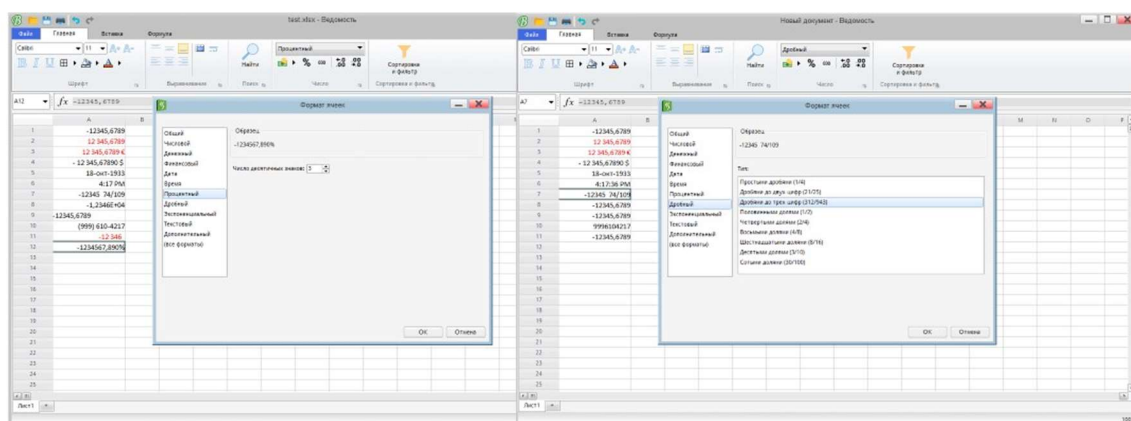


На рисунке изображен результат применения формата «Дата»

На рисунке изображен результат применения формата «Время»

17

Формат «Процентный» и «Дробный»

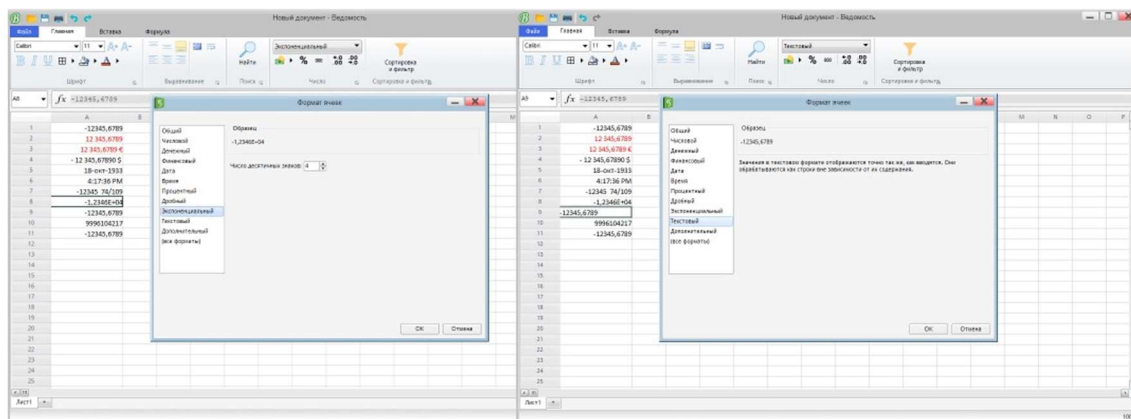


На рисунке изображен результат применения формата «Процентный»

На рисунке изображен результат применения формата «Дробный»

18

Формат «Экспоненциальный» и «Текстовый»

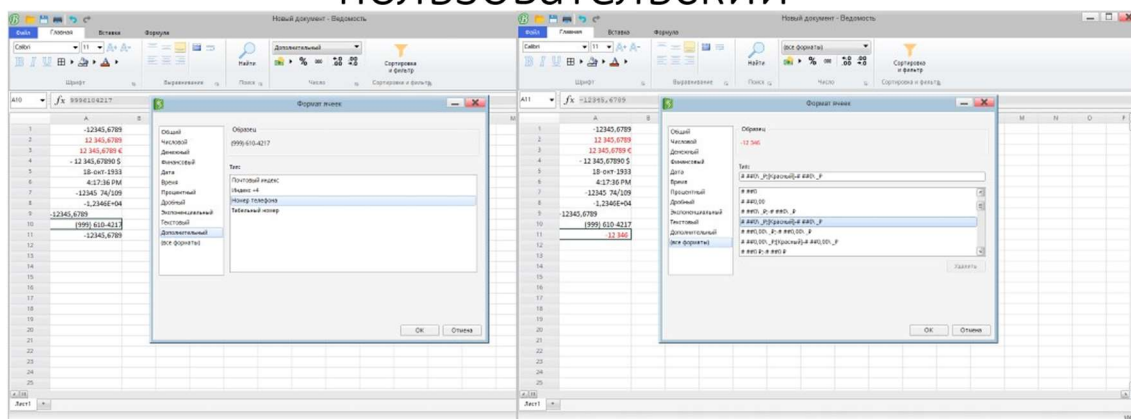


На рисунке изображен результат применения формата «Экспоненциальный»

На рисунке изображен результат применения формата «Текстовый»

19

Формат «Дополнительный» и пользовательский

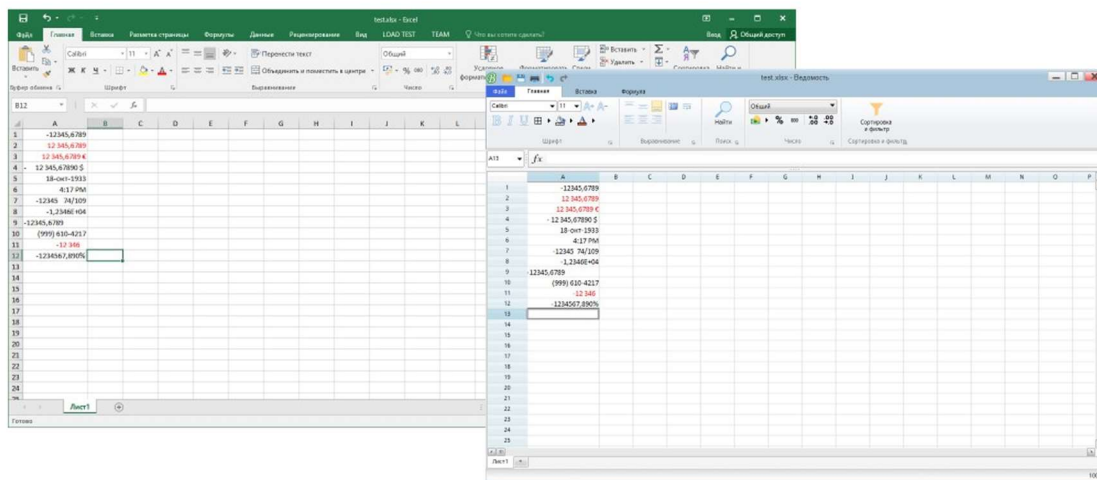


На рисунке изображен результат применения формата «Дополнительный»

На рисунке изображен результат применения пользовательского формата

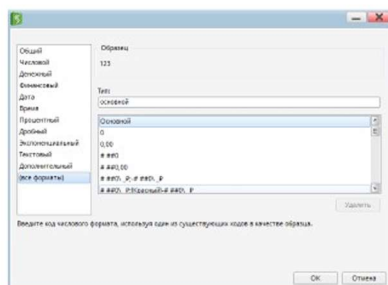
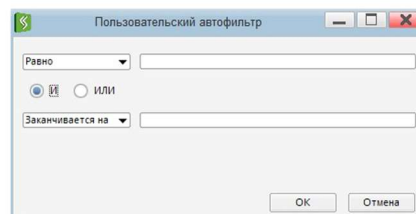
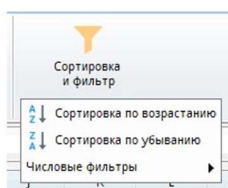
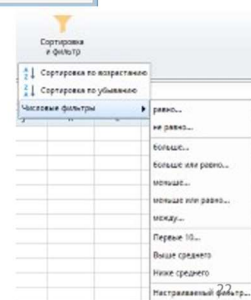
20

Сравнение результата с Microsoft Excel

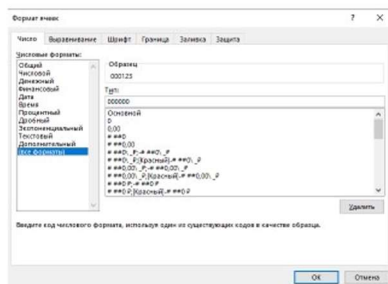
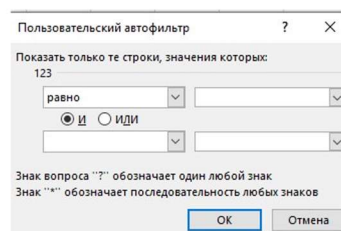
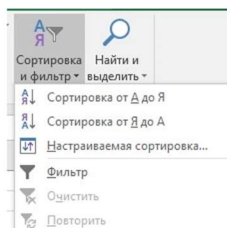
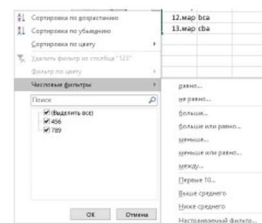


21

Разработанные модули

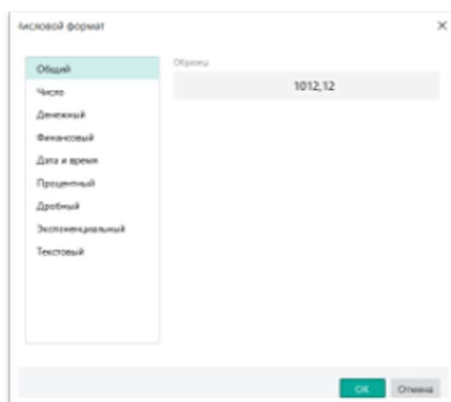
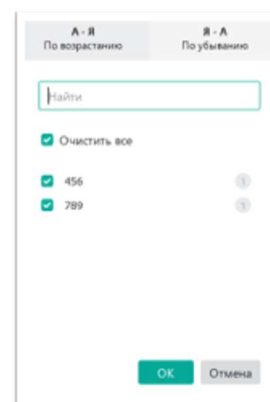


Модули из Microsoft Excel



23

Модули из Мой Офис Таблица



Окно пользовательского автофильтра и списка с фильтрами для каждого типа данных отсутствуют, так как данный функционал не реализован

24

Сравнительные характеристики

Критерии	Табличный процессор		
	Ведомость	Microsoft Excel	Мой Офис «Таблица»
Поддержка ввода пользовательского формата данных	+	+	-
Количество стандартных форматов данных	11	11	9
Поддержка сортировки различных типов данных	+	+	+
Уникальные фильтры для различных типов данных	+	+	Только ручная выборка
Поддержка комбинирования двух фильтров	+	+	-

25

Заключение

Поставленные для выполнения ВКР задачи были полностью решены. Были разработаны библиотеки форматирования, фильтрации и сортировки данных для табличного процессора. Модуль форматирования данных был интегрирован в редактор электронных таблиц «Ведомость», разработкой которого занимается ООО НТП «Криптософт».

В дальнейшем разработанные программные средства можно улучшить, путем добавления поддержки различных языков в модуль форматирования данных, реализовать больший набор стандартных форматов и более широкий набор фильтров.

26

Спасибо за внимание!

					ПГУ 2.090301.39.001 ПЗ	Лист
						159
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень принятых сокращений

IDE (Integrated development environment) – интегрированная среда разработки
TFS (Team Foundation Server) – многофункциональное решение, объединяющее в себе систему управления версиями, построение отчетов, сбор данных, отслеживание изменений по проекту

UI (User Interface) – пользовательский интерфейс

XML (eXtensible Markup Language) – расширяемый язык разметки

ОС – операционная система

ПО – программное обеспечение

ПК – персональный компьютер

ПФ – проверка функционала

НТП – научно-техническое предприятие

					ПГУ 2.090301.39.001 ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

Содержание

Реферат	2
Введение.....	6
1 Программные средства форматирования, фильтрации и сортировки данных для табличного процессора	7
1.1 Анализ предметной области	7
1.1.1 Постановка задачи	7
1.1.2 Анализ аналогов ПО	9
1.1.3 Анализ задачи.....	11
1.1.4 Структура кодов числовых форматов в табличных редакторах.....	12
1.1.5 Стандарт <i>Office Open XML</i>	13
1.1.6 Выбор инструментов разработки	16
1.1.7 Дополнительные плагины и технологии	18
1.2 Реализация программных средств.....	18
1.2.1 Описание архитектуры.....	18
1.2.2 Разработка библиотеки форматирования данных	23
1.2.3 Разработка библиотеки сортировки и фильтрации данных	26
1.2.4 Разработка графического интерфейса	27
1.3 Анализ результата	32
1.3.1 Тестирование и отладка	32
1.3.2 Тестирование в табличном процессоре «Ведомость».....	48
2 Безопасность жизнедеятельности.....	59
2.1 Анализ потенциальных опасностей и вредностей, возникающих при работе с ПЭВМ и способы их минимизации	59
2.2 Классификация и категорирование помещения, в котором производится разработка.....	59
2.2.1 Опасность поражения электрическим током	59
2.2.2 Пожарная безопасность.....	61
2.3 Санитарно-гигиенические требования к производству	62

2.4 Расчет пожарной нагрузки в помещениях категории В1-В4.....	63
Заключение	66
Список использованных источников	67
Приложение А. <i>UML</i> -диаграммы	68
Приложение Б. Листинг программы	76
Приложение В. Презентация.....	145

Введение

В последнее время актуальность импортозамещения заметно выросла. Для обеспечения устойчивости экономики страны необходимо прежде всего разрабатывать отечественные *IT*-продукты. Научно-техническое предприятие ООО НТП «Криптософт» около 20 лет занимается разработкой защищённой операционной системы *QP* ОС и предназначенных для нее прикладных программ. Важной целью разработки отечественной ОС является создание защищенной и безопасной системы.

Одним из разрабатываемых на предприятии продуктов является офисный пакет «*QP* Грамота». В пакет входят редактор текстовых документов, редактор электронных таблиц, менеджер презентаций и клиент электронной почты. При разработке особое внимание уделяется стабильности и быстродействию продукта.

Редактор текстовых документов предназначен для создания, редактирования, просмотра и форматирования текстовых статей, документов. Имеет широкий функционал: от набора и верстки текста до проверки орфографии, вставки в текст графического материала, печати текста.

Менеджер презентаций используется для подготовки и просмотра презентаций. Чаще всего материалы, подготовленные с его помощью, предназначены для отображения на большом экране – через проектор или телевизионный экран.

Редактор электронных таблиц позволяет создавать сложные электронные таблицы, а также производить тяжелые вычисления. С помощью него также можно проанализировать полученные данные, применяя удобные встроенные инструменты. Важными функциональными характеристиками табличного редактора является поддержка сортировки, фильтрации и форматирования данных.

Целью данной ВКР является разработка модулей форматирования, фильтрации и сортировки данных для табличного редактора.

1 Программные средства форматирования, фильтрации и сортировки данных для табличного процессора

1.1 Анализ предметной области

1.1.1 Постановка задачи

Прежде всего, стоит отметить, что модули разрабатываются для табличного процессора «Ведомость», применяемого в отечественной операционной системе *QP ОС*, разработкой которой занимается научно-техническое предприятие «Криптософт».

QP ОС позиционируется, как защищенная система, при разработке системы особое внимание было уделено безопасности системы и контролю доступа к данным. Для нее с 2017 года разрабатывается собственный офисный пакет приложений, в который входят: редакторы текстовых документов, электронных таблиц и презентаций, медиаплеер.

Офисные программы поддерживают, как новые форматы *Open XML* (.docx, .xlsx, .pptx), так и старые форматы (.doc, .xls, .ppt), их поддержка осуществляется за счет встроенного конвертера, который выполняет свою основную функцию при открытии документа, имеющего старый формат.

Разрабатываемые офисные программы могут работать как под управлением *Windows*, так и в *QP ОС*.

Главным мотивом разработки табличного процессора является создание полностью отечественного продукта, не уступающего, в некоторых моментах даже превосходящего, привычного большинству пользователей табличного процессора *Microsoft Excel*. При этом необходимо сохранить привычный пользователю интерфейс, которыми обладают все имеющиеся на данный момент пакеты офисных программ, например *Microsoft Office*, *Libre Office*, *Open Office*, Мой Офис.

На данный момент «Ведомость» уже поддерживает основной функционал табличного процессора:

					ПГУ 2.090301.39.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

- открытие документа в формате *XLSX* или *XLS*;
- сохранение документа в формате *XLSX*;
- создание таблиц, путем управления строками и столбцами (изменения высоты/ширины, кол-ва и тп.);
- управление атрибутами ячеек (цвет фона, цвет текста, цвет/ширина границ);
- редактирование текста ячеек, задание шрифта, цвета, др. атрибутов редактируемого текста;
- выравнивание текста (левый край, центр, правый край, по ширине);
- возможность добавления/удаления строк и столбцов;
- поддержка объединения/разъединения ячеек;
- отображение картинок в ячейках;
- поддержка буфера обмена.

В рамках выполнения квалификационной работы была поставлена задача разработать программные средства для табличного процессора, которые должны реализовать следующие функции:

- форматирование данных в таблице, как в соответствии с форматом, созданным пользователем, так и с заранее заданными форматами;
- сортировка данных по критерию, задаваемому пользователем;
- фильтрация данных на основании выбранных условий.

Задачи, которые должны быть решены при реализации:

- разработка эффективного алгоритма разбиения и структурирования кода формата, введенного пользователем, на токены;
- реализация алгоритма применения введенного формата к выбранному массиву данных;
- разработка эффективного алгоритма сортировки данных;
- разработка эффективного алгоритма фильтрации данных;
- разработка графического интерфейса пользователя;
- тестирование разработанного ПО.

1.1.2 Анализ аналогов ПО

Наиболее популярными программными продуктами, включающими в себя аналог разрабатываемых программных средств, являются *Microsoft Excel*, *Open Office*, *Libre Office*. Главным недостатком каждого из них является то, что все они разработаны за рубежом. Из отечественных аналогов можно отметить лишь МойОфис.

Отличительной особенностью, разрабатываемого НТП «Криптософт» продукта, является полностью уникальная архитектура проекта и код, написанный с нуля, без использования *Open-Source* технологий. При этом реализована полная поддержка чтения и записи файлов согласно стандарту *ЕСМА-376*.

Каждый из представленных аналогов содержит модуль форматирования, фильтрации и сортировки данных. Для совместимости с ними в ходе разработки, будет реализована поддержка аналогичных кодов форматов.

Самый большой набор числовых форматов реализован в продукте *Microsoft Excel*, в остальных отсутствуют несколько форматов.

В *Excel* заданы следующие стандартные числовые форматы:

- общий;
- числовой;
- денежный;
- финансовый;
- дата;
- время;
- процентный;
- дробный;
- экспоненциальный;
- текстовый;
- дополнительный.

Некоторых из них содержат определенные настройки, а остальные заранее созданные типы. Дополнительно в *Microsoft Excel* реализован числовой формат «Пользовательский», в последнем пользователь сам вводит код формата. Стоит отметить, что «Мой Офис Таблица» не предоставляет пользователю эти форматы, что является его существенным недостатком.

Эталонным модулем форматирования данных обладает продукт *Microsoft Excel*. Но, он является зарубежной разработкой. Таким образом, есть смысл произвести сравнение с единственным отечественным аналогом «Мой Офис Таблица». По сравнению с *Excel*, модуль форматирования данных содержит меньшее количество настроек для различных форматов. Но главным его недостатком является невозможность ввода пользовательского кода формата. Также «Мой Офис Таблица» не обладает модулем фильтрации данных по различным условиям. В ходе разработки модуля эти критерии будут учтены и реализованы.

В *Microsoft Excel* содержится огромное количество фильтров для следующих типов данных: текстовый, числовой и дата. Для текстовых данных реализованы следующие фильтры: «Равно», «Не равно», «Начинается с», «Заканчивается на», «Содержит», «Не содержит». Для числовых: «Равно», «Не равно», «Больше», «Больше или равно», «Меньше», «Меньше или равно», «Между», «Первые 10», «Выше среднего», «Ниже среднего». Для даты реализовано огромное количество фильтров, позволяющих делать выборку данных за определенный период или интервал. Также существует возможность комбинировать любые два из фильтров с помощью операций «И», «ИЛИ».

Отечественный аналог «Мой Офис Таблица» имеет возможность фильтрации только по возрастанию или убыванию. В ходе разработки модуля фильтрации будет реализована поддержка всех фильтров из продукта компании *Microsoft*, а также их комбинирование.

Во всех рассмотренных выше редакторах электронных таблиц интерфейсы инструментов форматирования, фильтрации и сортировки данных схожи. Для

того, чтобы пользователю было привычно работать в табличном процессоре «Ведомость», будет реализован схожий интерфейс с аналогами интерфейса.

В конечном итоге, разрабатываемые модули не должны ничем уступать аналогичным модулям в *Microsoft Excel*, а также превосходить в функционале отечественные аналоги. Так же в заключении стоит отметить, что пакет офисных программ «Мой Офис» не является полностью отечественным продуктом, так как некоторый функционал был заимствован из западных разработок.

1.1.3 Анализ задачи

Разрабатываемые программные модули должны поддерживать форматирование, сортировку и фильтрацию данных. При этом для форматирования могут использоваться, как заранее заданные, так и введенные пользователем в виде кода форматы.

Код формата — это унифицированная последовательность, составленная из следующих основных частей (токенов), который могут комбинироваться в зависимости от желания пользователя:

- литерал;
- цифра;
- текстовый заполнитель (*placeholder*) (#, *, ?);
- экспонента (e+, e-);
- основной формат (*general*);
- часть даты (y, m, d, h, m, s, am/pm);
- продолжительность времени ([h], [m], [s]);
- цвет ([Красный]);
- условие (>=100).

Для начала должна производиться валидации формата, если она прошла успешно, то необходимо приступить к разбору последовательности. На следующем этапе требуется определить к какому типу принадлежит введенный

код (основной, дробный, экспоненциальный и т.д.) и выбрать подходящий алгоритм форматирования данных.

Для выполнения сортировки, необходимо определить тип данных (текст, числовой, дата, время и т.д.) и в зависимости от него выбрать алгоритмы сортировки. Аналогично и для фильтрации: в зависимости от типа данных, пользователю будет предложен определенный набор условий для осуществления этой операции.

1.1.4 Структура кодов числовых форматов в табличных редакторах

С помощью числовых форматов можно изменять внешний вид чисел, даты или времени, при этом фактически не меняя числа. Числовой формат не влияет на значение ячейки, которое табличный процессор использует для вычислений и записывает в *XML*. Реальное значение содержится в строке формул.

При создании пользовательского формата можно указывать до четырех разделов. Они записываются в следующем порядке:

- формат для положительных чисел (1);
- формат для отрицательных чисел (2);
- формат для нулевых значений (3);
- формат для текстовых данных (4).

При этом разделы кода разделяются знаком «;». На рисунке 1 изображен пример записи пользовательского кода формата [1].

Рисунок 1 - Пользовательский код формата

Если в коде указан не полный набор разделов, то применяются следующие правила определения принадлежности раздела к конкретным данным: если указан только один раздел, то он будет использоваться для всех чисел. Если указаны два раздела, первый раздел будет использоваться для положительных чисел и нулей, а второй для отрицательных чисел. Также

пользователь имеет возможность пропустить какой-либо раздел, для этого необходимо добавить точку с запятой.

1.1.5 Стандарт *Office Open XML*

ECMA-376 определяет семейство *XML* схем, вместе называемых *Office Open XML*, которые определяют правила для обработки текстовых документов, электронных таблиц и презентаций, а также упаковку этих документов, в соответствии с определенными схемами [2].

Главная цель создания данного стандарта – стандартизировать структуру всех документов, что позволит воспроизводить их одинаково различными офисными приложениями.

Стандарт *Open XML* включает в себя несколько более детальных стандартов. Одним из них является стандарт *OPC (Open Packaging Convention)*, который описывает структуру файла, взаимосвязь его частей, наличие различных типов данных в документе, а также, при необходимости, цифровую подпись документа.

Помимо *OPC* используются следующие стандарты:

- *WordprocessingML*. Описывает разметку текстовых документов;
- *SpreadSheetML*. Отвечает за структуру электронных таблиц;
- *PresentationML*. Описывает структуру презентаций;
- *DrawingML*. Указывает структуру графиков, диаграмм и некоторых графических объектов, также может описывать формулы, выражения и метаданные документа.

С точки зрения пользователя любой документ, соответствующий стандарту *OpenXML*, представляет собой *ZIP*-архив, содержащий в себе структурированные *XML* файлы, файлы ресурсов документа и файлы, содержащие в себе отношения между составными частями документа (*Relationships*).

Каждый из стандартных форматов данных имеет свой собственный *ID*, который никак не хранится в архиве документа. Коды форматов, которые вводит пользователь хранятся в файла *styles.xml*, который располагается по

					ПГУ 2.090301.39.001 ПЗ	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

пути *table.xlsx/xl/styles.xml*. За хранение таких кодов отвечает элемент *numFmts*, который имеет атрибут *count*, отвечающий за количество пользовательских форматов [3].

Элемент *numFmts* содержит в себе множество элементов с тегом *numFmt*. В свою очередь, *numFmt* содержит следующие атрибуты:

– *numFmtID*. Определяет уникальный ID формата, введенного пользователем;

– *formatCode*. Определяет код формата, введенный пользователем.

Ниже приведен пример хранения в XML созданного пользовательского числового формата.

```
<numFmts count="1">
  <numFmt numFmtId="164" formatCode="##,##0;\-##,##0"/>
</numFmts>
```

При задании форматов создается элемент *cellXfs* с атрибутом *count*, определяющим количество примененных форматов. Внутри себя этот элемент содержит множество элементов *xf*, имеющих атрибуты определяющие параметры этого стиля.

Данный элемент имеет следующий вид.

```
<cellXfs count="3">
  <xf numFmtId="0" fontId="0" fillId="0" borderId="0"
xfId="0"/>
  <xf numFmtId="9" fontId="0" fillId="0" borderId="0"
xfId="1" applyFont="1"/>
  <xf numFmtId="164" fontId="0" fillId="0" borderId="0"
xfId="0" applyNumberFormat="1"/>
</cellXfs>
```

Здесь как раз и определяется ID числового формата, установленного для стиля, за это отвечает атрибут *numFmtId*. Идентификатор формата 0 принадлежит стандартному формату, 9 – процентному, который является стандартным, 164 – это как раз введенный пользователем формат, XML элемент которого приведен выше.

Принадлежность определенного формата ячейке определяется в файле *sheet.xml*, располагающегося по пути *table.xlsx/xl/worksheets/sheet1.xml*. Внутри

него содержится элемент *sheetData*, внутри которого содержится множество элементов *row*, отвечающих за строки, внутри которых в свою очередь содержатся элементы *c* (*cell*). Атрибут *s* у элемента *c* как раз и отвечает за номер стиля, который определен в файле *styles.xml*, установленный ячейке. Стиль в свою очередь содержит атрибут с кодом формата (*numFmtId*).

Ниже приведен пример элемента *sheetData*.

```
<sheetData>
  <row r="1" spans="1:2" x14ac:dyDescent="0.3">
    <c r="A1" s="1">
      <v>767878</v>
    </c>
    <c r="B1" s="1">
      <v>965936954</v>
    </c>
  </row>
  <row r="2" spans="1:2" x14ac:dyDescent="0.3">
    <c r="A2" s="2">
      <v>423423</v>
    </c>
    <c r="B2" s="2">
      <v>654654</v>
    </c>
  </row>
  <row r="3" spans="1:2" x14ac:dyDescent="0.3">
    <c r="A3" s="1">
      <v>123</v>
    </c>
    <c r="B3" t="s">
      <v>0</v>
    </c>
  </row>
</sheetData>
```

Стиль, применяемый к ячейке, зависит от атрибута *s*:

- *s*=1 – стиль, где установлен числовой формат «процентный» с идентификатором 9;
- *s*=2 – стиль, где установлен пользовательский числовой формат с идентификатором 164.

Атрибут *t="s"* означает, что применяется стандартный стиль.

В ходе интеграции библиотеки форматирования данных в «Ведомость» все перечисленные выше моменты будут учтены и будет реализована полная

поддержка стандарта *Office Open XML*.

1.1.6 Выбор инструментов разработки

1.1.6.1 Язык программирования

В качестве языка программирования был выбран *C#*. Он является объектно-ориентированным и строго типизированным языком. Также *C#* – это один из мощных и востребованных языков программирования на рынке. Его синтаксис схож с *Java* и *C++*. Основными преимуществами *C#* являются: высокая скорость работы, надежность, простота в освоении, автоматическая сборка мусора и многое другое. На платформе *.Net* можно реализовывать любое приложение от консольных до оконных *WinForms* программ, веб-сайтов с помощью *ASP.NET*, мобильных и облачных приложений. Также стоит отметить, что разработка офисного пакета «*QP Грамота*» ведется преимущественно на языке *C#*, поэтому целесообразно выбрать именно его для разработки описанных ранее модулей.

1.1.6.2 Среда разработки

Выбор среды разработки пал на продукт *Microsoft – Visual Studio 2019*. Официальная, бесплатная и самая функциональная *IDE* для разработки на *C#*. Является самой распространенной среди *C#* разработчиков. *Visual Studio* полностью совместима с *TFS*, имеет огромное количество встроенных библиотек и инструментов, поддерживающих компиляцию, отладку, профилирование, юнит-тестирование. Также стоит отметить, что *Visual Studio* поддерживает подключение сторонних плагинов.

1.1.6.3 Платформа *.Net Framework*

.Net Framework – программная платформа, выпущенная компанией *Microsoft*. Первый релиз состоялся в 2002 году. Основой платформы является общезыковая среда исполнения *Common Language Runtime (CLR)*, которая подходит для различных языков программирования. [4]

Можно выделить следующие отличительные черты данной платформы:

– поддержка нескольких языков. Благодаря тому, что в основе платформы лежит *CLR*, *.Net Framework* поддерживает несколько языков

					ПГУ 2.090301.39.001 ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

программирования: *C#, VB.NET, C++, F#*, а также различные диалекты других языков, привязанные к *.NET*;

- кроссплатформенность. Используя различные технологии платформы, разработчик может создавать приложения для самых разных операционных систем: *Windows, Linux, MacOS, Android*;

- мощная библиотека классов. *.NET* представляет единую библиотеку классов для всех поддерживаемых языков программирования;

- разнообразие технологий. Стек технологий платформы *.NET* – огромен, отличительными из них являются: *ASP.NET, Windows Forms, WPF, WinUI, Entity Framework, ADO.NET* и другие;

- производительность. Согласно огромному количеству тестов, приложения, построенные на *.NET* отличаются высокой производительностью.

В ходе реализации проекта использовалась платформа *.Net Framework 4.5.2* для обеспечения совместимости с *QP ОС*.

1.1.6.4 Платформа пользовательского интерфейса

Для реализации пользовательского интерфейса с целью тестирования на ОС *Windows* было принято использовать платформу *Windows Forms*. Также стоит отметить, что для интеграции в табличный редактор «Ведомость» реализовывались аналогичные графические интерфейсы на основе собственной платформы, разработанной научно-техническим предприятием «Криптософт».

Windows Forms – это платформа пользовательского интерфейса для создания классических интерфейсов *Windows*, включенная в *.NET Framework*. Прежде всего она является объектно-ориентированной оболочкой над *Win32 API* [5].

Как и большинство аналогичных платформ, *WinForms* управляется событиями. Основным элементом является форма, которая содержит различные элементы управления (кнопки, текстовые поля, надписи и т.д.), с которыми пользователь напрямую взаимодействует. В ответ на взаимодействие эти элементы вызывают события, которые выполняются программой.

В *Windows Forms* можно разрабатывать сложные графические интерфейсы, которые легко обновлять и развертывать. Также с ними удобно работать, как в автономно режиме, так и в сети. Приложения *Windows Forms* могут получать доступ к файловой системе и оборудованию компьютера.

Существенным преимуществом является глубокая интеграция со средой разработки *Visual Studio*. Данная *IDE* содержит конструктор графических интерфейсов, что существенно упрощает реализацию приложения.

1.1.7 Дополнительные плагины и технологии

В ходе разработки применялся плагин *ReSharper*. Это одна из наиболее популярных разработок компании *JetBrains*, специально разработанный на основе партнерских отношений с *Microsoft*. Выбранный плагин позволяет повысить производительность, качество и чистоту кода.

Для разработки библиотеки фильтрации и сортировки использовалась технология *LINQ*, а конкретно провайдер *LINQ To Objects*. Данная технология позволяет выполнять запросы к коллекциям, реализующим интерфейс *IEnumerable*, что позволяет существенно повысить читаемость и производительность в написании кода. Также *LINQ to objects* обладает схожей производительностью по отношению к «ручному» коду [6].

1.2 Реализация программных средств

1.2.1 Описание архитектуры

Было принято решение разделить разрабатываемые программные средства на две библиотеки классов:

- библиотека сортировки и фильтрации данных;
- библиотека форматирования данных.

При построении архитектуры первой библиотеки было принято решение создать несколько вспомогательных перечислений и словарей с типами фильтров, это необходимо для сокращения количества кода в основных модулях и улучшения его читаемости. Так же для каждого типа фильтра был создан собственный класс со статическими функциями фильтрации. Для

передачи информации из графического интерфейса в библиотеку был создан класс «*FilterData*», содержащий в себе значения и условия для фильтрации, тип первого и второго фильтра, а также операцию (и, или), выполняемую над ними, это позволяет повысить читаемость кода. Главным модулем является класс «*Filter*», туда и передается объект, описанный выше. Основная его цель – выполнить фильтрацию данных.

В главном методе проектируемого модуля определяется тип переданных значений и фильтра, который затем применяется. Для определения типа используются описанные ранее словари, содержащие в себе название фильтра и элемент перечисления либо делегат, инкапсулирующий нужный метод фильтрации.

На рисунке 2 приведена схема взаимодействия модулей программы.

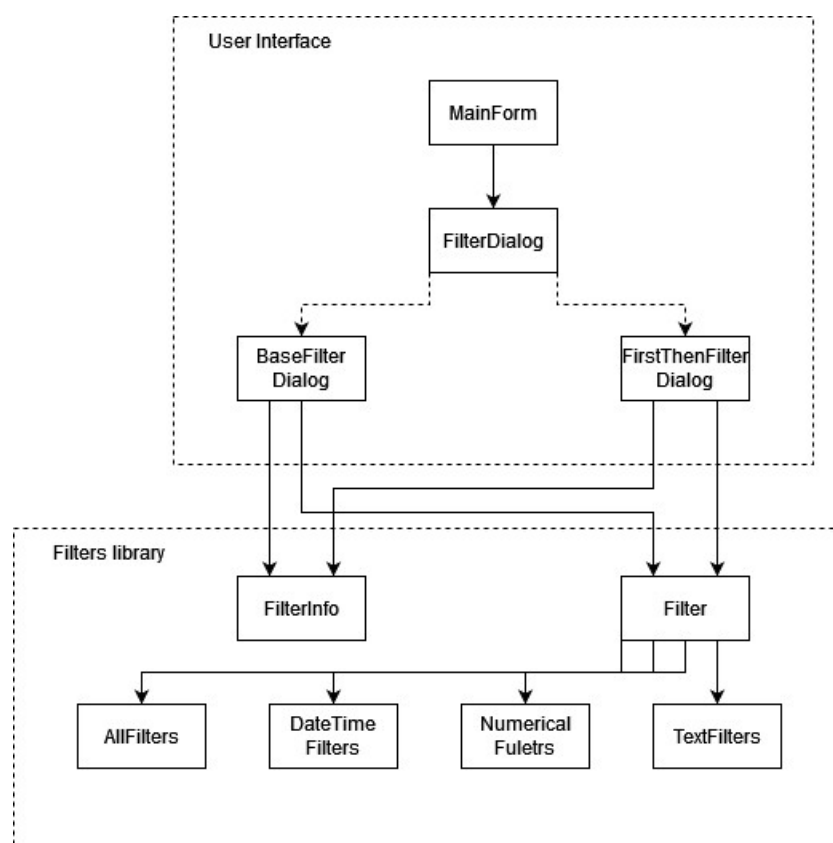


Рисунок 2 - Схема взаимодействия программы фильтрации данных

Графический интерфейс содержит 4 модуля. *MainForm* – это главное окно программы. Для задания фильтров было реализовано 2 диалога:

BaseFilterDialog, содержащий основные фильтры, *FirstThenFilterDialog*, содержащий фильтры «Первые 10...». Каждый из них унаследован от *FilterDialog*, содержащий в себе поле с массивом результирующим массивом индексов начального массива значений.

Передача данных из графического интерфейса на сторону библиотеки происходит с помощью вспомогательного класса *FilterArgs*, который содержит в себе исходные значения, название первого фильтра, значение первого фильтра, название второго фильтра, значение второго фильтра и операцию, выполняемую над результатами двух фильтров, при этом обязательными аргументами являются только первые два. Созданный объект передается в метод *ApplyFilter()*, находящийся в классе *Executor*. В нем определяется тип фильтра с помощью вспомогательного класса *AllFitlers*, содержащего перечисления, названия фильтров, а также делегаты, инкапсулирующие методы фильтрации. Они в свою очередь содержатся в классах *DateTimeFilters*, *NumericalFilters*, *TextFilters*.

Вторую библиотеку условно можно разделить на две части:

- парсер кода формата;
- модуль, форматирующий данные.

В ходе проектирования архитектуры парсера было принято создать несколько объектов-секций для каждого типа данных, на которые будет делиться код формата. Секции в свою очередь разбиваются на токены различных типов. Разбиение кода формата на секции будет производиться путем разделения строки по символу «;». Затем каждая секция будет считываться посимвольно для разбиения на токены, параллельно будет проверяться ее валидность, так как не все токены можно комбинировать в определенных разделах. Конечный результатом работы парсера - объект типа «*NumberFormat*», который содержит в себе список секций. На его основе будет производиться форматированные данных. Разбиение формата на мелкие части позволит структурировать написанный код и упростить процесс форматирования данных.

Модуль форматирования принимает на вход формируемое значение и созданный ранее объект типа «*Format*». На первом этапе определяется применяемая к данным секция. Затем, в зависимости от ее типа, будет вызван соответствующий метод для форматирования, где перебираются токены и постепенно собирается готовый результат.

На рисунке 3 приведена схема взаимодействия модулей программы.

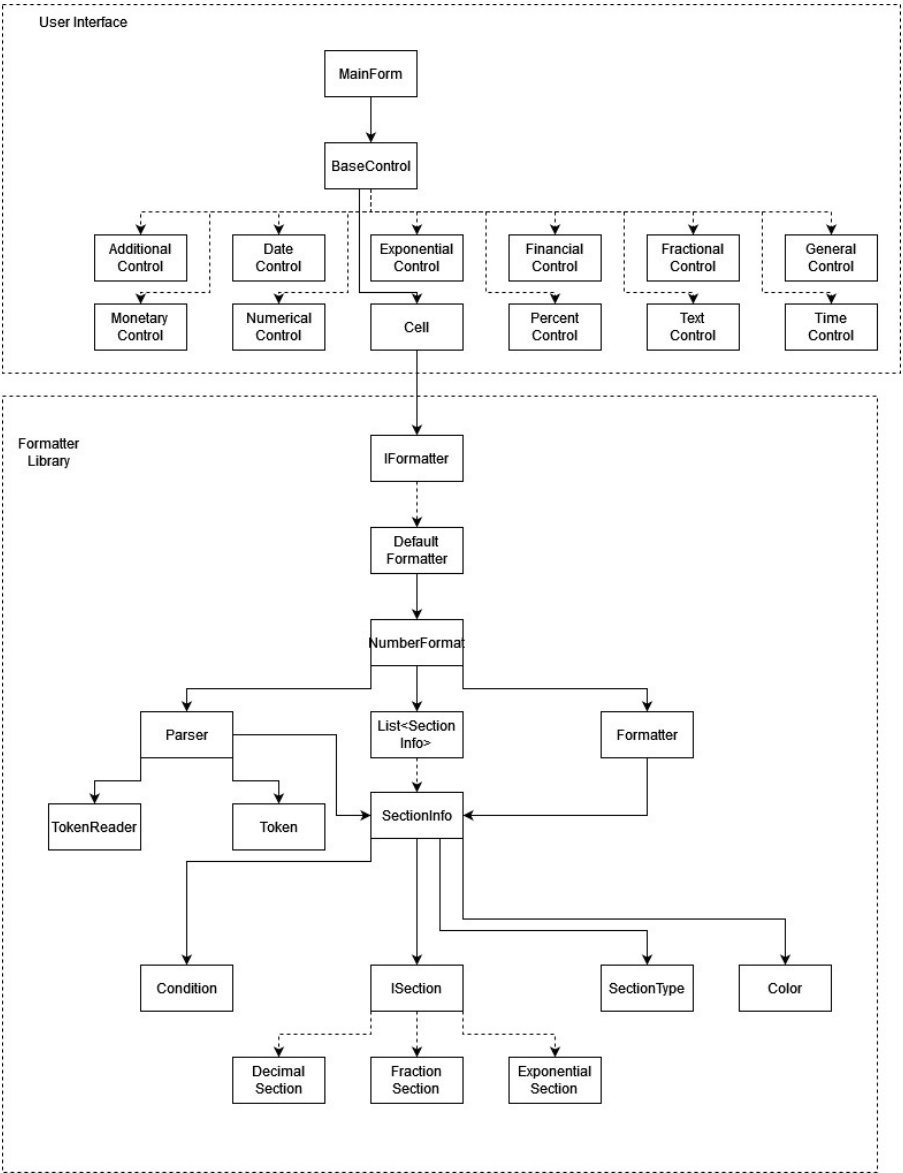


Рисунок 3 - Схема взаимодействия модулей программы форматирования данных

Графический интерфейс содержит 14 модулей. *MainForm* – главное окно программы. Для настройки форматов реализовано 11 пользовательских

элементов управления: *AdditionalControl*, *MonetaryControl*, *DateControl*, *NumericalControl*, *ExponentialControl*, *FinancialControl*, *PercentControl*, *FractionalControl*, *TextControl*, *TimeControl*. Все они содержат уникальный набор элементов графического интерфейса, для настройки соответствующего формата. Каждый из них унаследован от класса *BaseControl*, содержащий в себе метод *FormatValue()*, отвечающий за форматирование значения, свойства *Type*, *FormatCode* и *Cell*. Также в базовом классе содержится делегат *ValueHandler*, принимающий один параметра типа *Cell*, событие *ChangeFormattedValue*, которое представляет делегат *ValueHandler*, и метод *PerformChangeValue()*, вызывающий событие. Это необходимо для форматирования данных сразу после ввода их в соответствующее поле, без необходимости дополнительных действий от пользователя.

Форматирование значение происходит с помощью вызова метода *Format()* у класса *DefaultFormatter*, реализующего интерфейс *IFormatter*, хранящегося в виде свойства у объекта *Cell*. *DefaultFormatter* содержит свойство *FormatCode* типа *string* и объект типа *NumberFormat*, содержащий в себе «разобранный» по секциям формат, за что отвечает класс *Parser*, и метод *Format()*, который определяет нужную секцию и обращается к классу *Formatter* для получения результата. Разобранный формат хранится в виде списка объектов типа *SectionInfo*, который содержит тип и индекс секции, цвет, условие, саму секцию и токены типов «общий», «текст», «дата», «длительность». *Parser* в свою очередь в процессе работы создает объект типа *TokenReader*, который считывает все токены из строки, тип которых затем определяется с помощью класса «*Token*», что в свою очередь позволяет определить тип секции.

На рисунке 4 приведена *UML*-диаграмма вариантов использования разрабатываемых модулей.

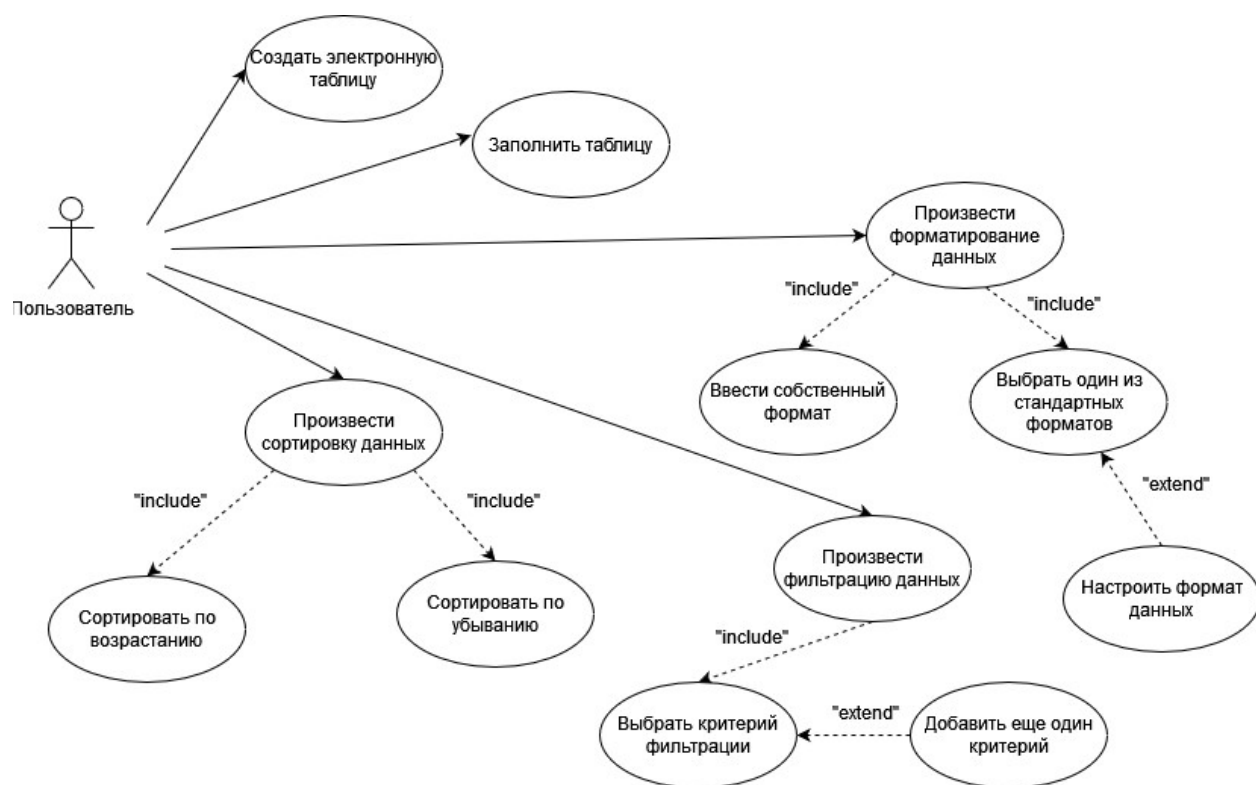


Рисунок 4 - UML-диаграмма вариантов использования

1.2.2 Разработка библиотеки форматирования данных

Условно, разрабатываемый модуль, можно разделить на две части: парсер и часть, отвечающая за форматирование данных.

Взаимодействие с графическим интерфейсом происходит с помощью класса *DefaultFormatter*, реализующий интерфейс *IFormatter*.

Он содержит в себе следующие свойства:

- *Code* – код формата;
- *FontColor* – цвет текста, который необходимо применить к отформатированному значению.

Также класс содержит поле типа *NumberFormat* – это разобранный на секции формат.

Код интерфейса *IFormatter*.

```

public interface IFormatter
{
    string Code { get; }
    Color FontColor { get; set; }
    string Format(string cellValue);
}
  
```

Код класса *DefaultFormatter*:

```
public class DefaultFormatter: IFormatter
{
    public string Code { get; }
    public Color FontColor { get; set; }

    private readonly NumberFormat _format;

    public DefaultFormatter(string formatCode, bool
isRuCode = false)
    {
        Code = formatCode;
        _format = new NumberFormat(formatCode, isRuCode);
    }

    public string Format(string cellValue)
    {
        var res = _format.Format(cellValue,
CultureInfo.CurrentCulture);
        FontColor = res.Color;
        return res.Text;
    }
}
```

Объект типа *DefaultFormatter* планируется хранить как свойство у ячейки, сделано это для реализации возможности кэширования объекта *NumberFormat*.

Для форматирования значение необходимо создать экземпляр класса *DefaultFormatter*, передав в него код применяемого формата. В свою очередь, в его конструкторе будет создан экземпляр класса *NumberFormat*. В конце необходимо вызывать метод *Format*, передав в него форматируемое значение. На выходе пользователь получит строку с отформатированным значением.

Как указывалось ранее, класс *NubmerFormat* содержит одну из секций формата, реализующей интерфейс *ISection*. Парсер секций вызывается в конструкторе данного класса, на выходе получается список объектов типа *Section*.

Класс *Section* содержит в себе несколько свойств:

- *SectionIndex* – индекс секции;
- *Type* – тип секции (элемент перечисления);
- *Color* – цвет, содержащийся в секции. По умолчанию – черный;

– *Condition* – условие, содержащиеся в секции. Реализован в виде отдельного класса, имеющего свойства *Op* (операция) и *Value* (значение). Имеет метод *Apply*, возвращающий результат применения условия. По умолчанию - true;

– одну из дополнительных секций: *Number*, *Exponential*, *Fraction*;

Для разбора секций был разработан класс *TokenReader*, содержащий методы посимвольного чтения строки формата для каждого типа токена.

После прочтения всех токенов, на основании полученных токенов, определяется тип формата, к которому принадлежит переданный в парсер код и и возвращается объект типа *Section*.

При вызове метода *Format* у объекта типа *NumberFormat* первым делом на основе переданного значения происходит определение применяемой секции. Затем у класса *Formatter* вызывается метод *Format*, куда передается значение и секция, в нем происходит форматирование значения по алгоритмам, разработанным для каждого типа формата.

На рисунке 5 приведена UML-диаграмма классов разработанной библиотеки.

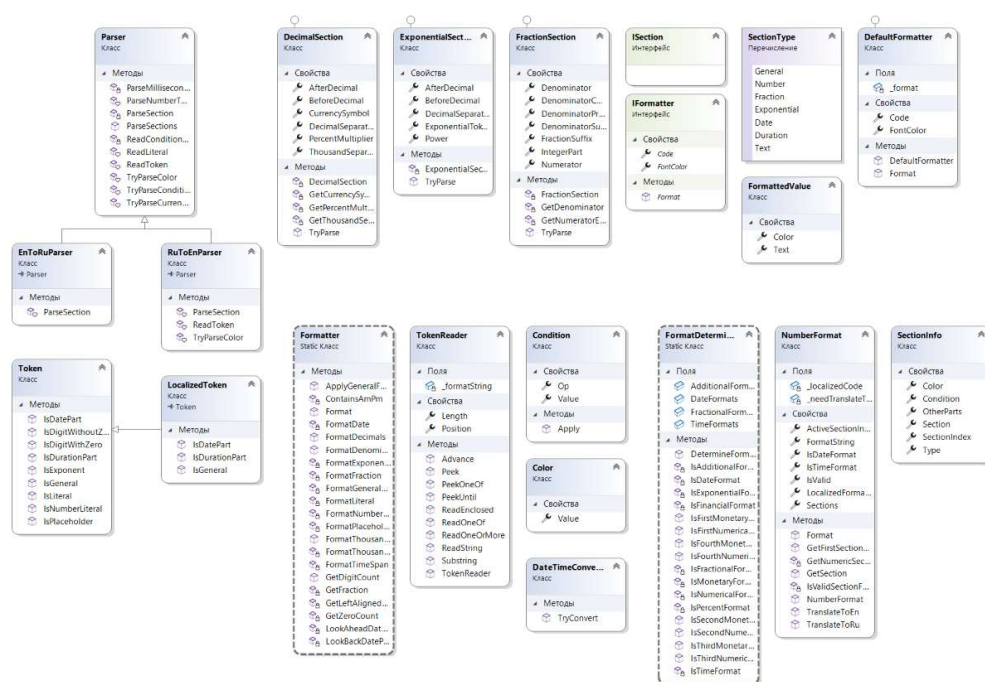


Рисунок 5 - UML диаграмма классов библиотеки форматирования данных

1.2.3 Разработка библиотеки сортировки и фильтрации данных

Разрабатываемый модуль позволяет применять различные фильтры к табличным данным и производить их сортировку.

Основным классом является – *Executer*. Он содержит в себе метод *ApplyFilter()*, применяющий фильтр и метод *Sort()*, выполняющий сортировку.

Вспомогательным классом для передачи информации между интерфейсом и библиотекой является класс *FilterArgs*, содержащий в себе следующие поля:

- *Values* – начальные значения;
- *NameOfFirstFilter* – название первого фильтра;
- *ValueOfFirstFilter* – значение первого фильтра;
- *NameOfSecondFilter* – название второго фильтра;
- *ValueOfSecondFilter* – значение второго фильтра;
- *FilterOp* – операция выполняемая над фильтрами.

Обязательными из них являются только первые два поля.

В метод *ApplyFilter()* передается объект типа *FilterArgs*, затем с помощью созданных ранее словарей и перечислений, определяется к какому типу принадлежит фильтр, после чего вызывается соответствующий метод фильтрации.

Классы *DateTimeFilters*, *NumericalFilters*, *TextFilters* содержат алгоритмы фильтрации для каждого из возможных типов данных. Возвращаемое значение – массив индексов начального массива данных.

На рисунке 6 приведена *UML*-диаграмма классов разработанной библиотеки.

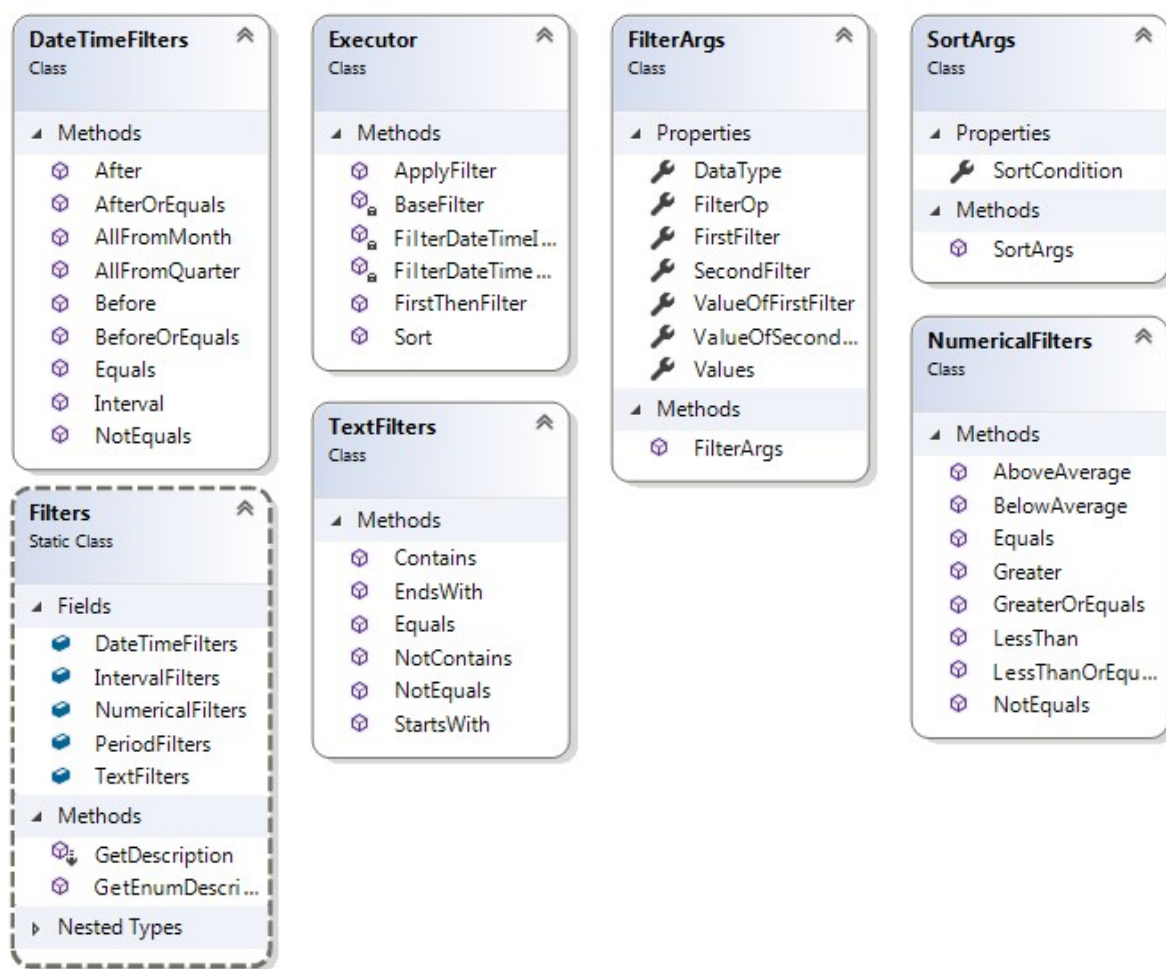


Рисунок 6 - UML-диаграмма классов библиотеки фильтрации и сортировки данных

1.2.4 Разработка графического интерфейса

Для реализации графического интерфейса использовалась платформа *Windows Forms*. В ходе разработки было реализовано два *UI*:

- графический интерфейс для тестирования библиотеки форматирования данных;
- графический интерфейс для тестирования библиотеки сортировки и фильтрации данных.

При их создании не преследовалась цель создать привлекательный и удобный для пользователя интерфейс. Главная цель – это демонстрации работы разработанных модулей. В дальнейшем графический интерфейс будет

разработан с использованием собственных технологий для применения в реальном табличном процессоре.

Основные элементы взаимодействия с программой – это три текстовых поля и список, в котором выбирается формат. В ходе разработки данного интерфейса были созданы 11 пользовательских элементов управления, для каждого типа формата, унаследованных от класса *BaseControl*. В каждом элементе содержатся соответствующие для него элементы управления, позволяющие настроить представление числа.

Форматирование введенного числа происходит после потери фокуса текстового поля, куда вводится начальное значение. Результат отображается в текстовом поле с подписью «Конвертированное число». Текущий код формата отображается в текстовом поле «*Format Code*», куда пользователь также может ввести собственный формат.

На рисунке 7 приведена *UML* диаграмма классов разработанного графического интерфейса.

Графический интерфейс библиотеки сортировки и фильтрации более прост в реализации, так как не требует множества элементов управления и привязки к большому числу событий. Интерфейс довольно прост: в текстовом поле слева вводится начальный массив, а в текстовом поле справа отображается результат сортировки или применения фильтра. Между ними есть кнопка, по нажатию на которую, программа определит тип введенных данных (текст, число, дата) и отобразит для пользователя возможные варианты сортировки и доступные фильтры.

На рисунке 8 представлен графический интерфейс, разработанный для библиотеки форматирования данных.

На рисунке 9 изображено главное окно интерфейса программы, разработанного для проверки работоспособности библиотеки.

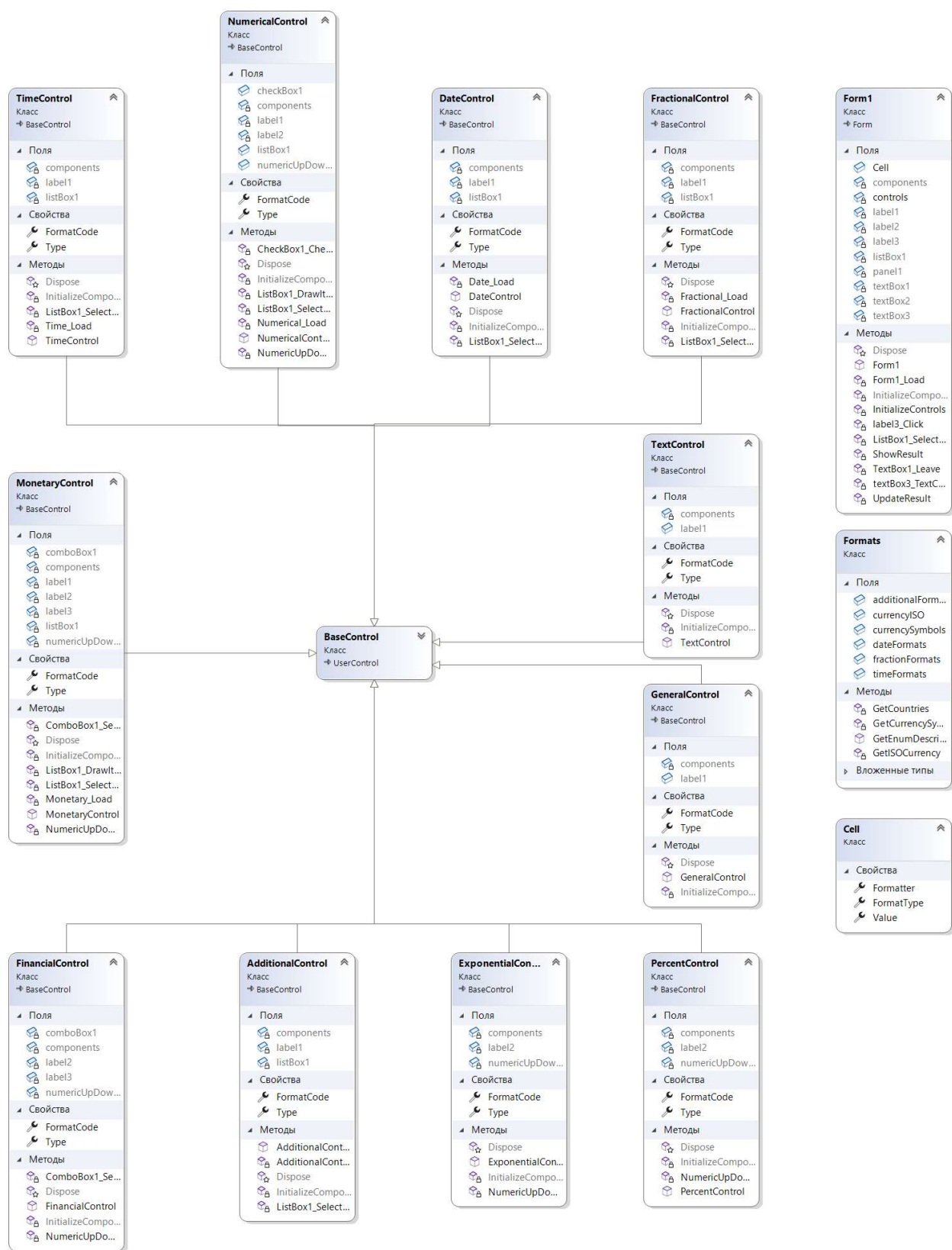


Рисунок 7 - UML диаграмма классов графического интерфейса библиотеки форматирования данных

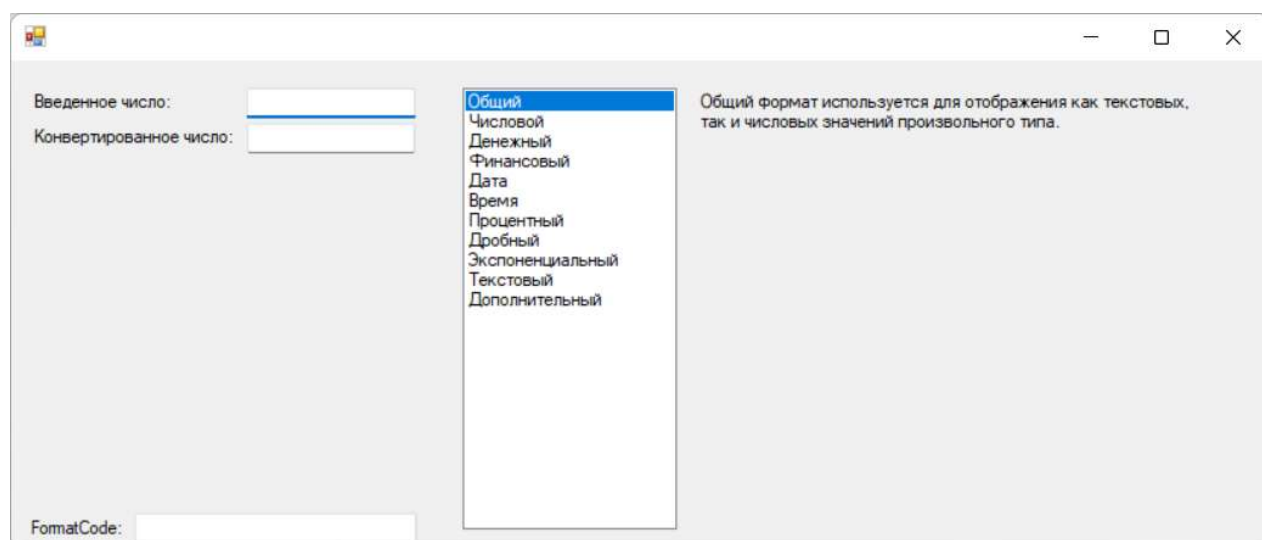


Рисунок 8 - Интерфейс, разработанный для библиотеки форматирования данных

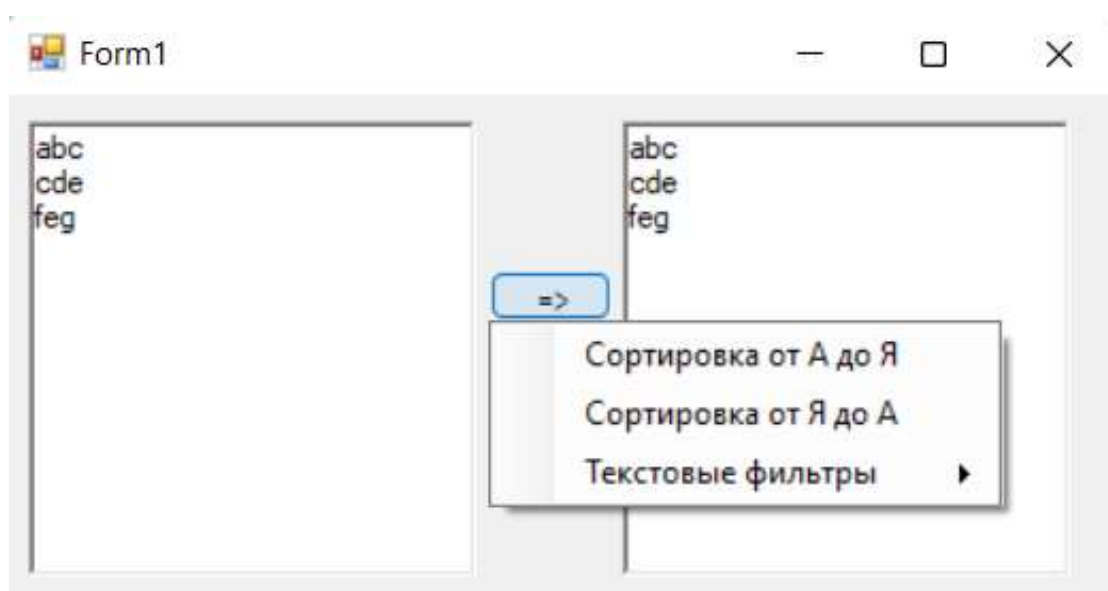


Рисунок 9 - Интерфейс, разработанный для библиотеки сортировки и фильтрации данных

Также для применения комбинированных фильтров была разработана отдельная форма, где выбирается первый и второй фильтр, критерии для них, а также логическая операция, выполняемая над результатами фильтрации.

На рисунке 10 изображено окно выбора фильтров.

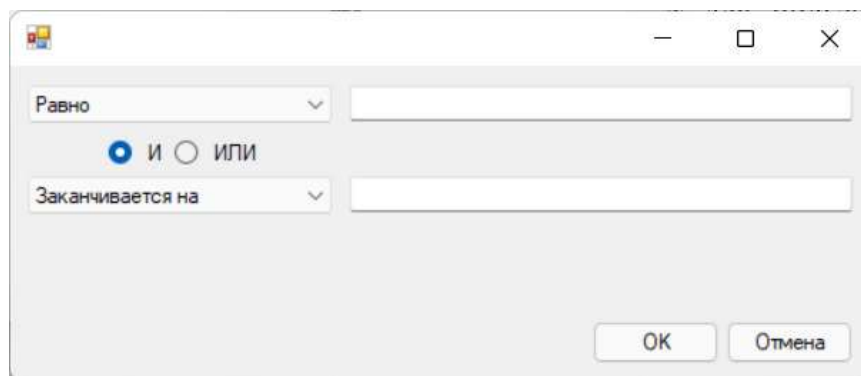


Рисунок 10 - Окно применения нескольких фильтров

На рисунке 11 приведена UML диаграмма классов разработанного графического интерфейса.

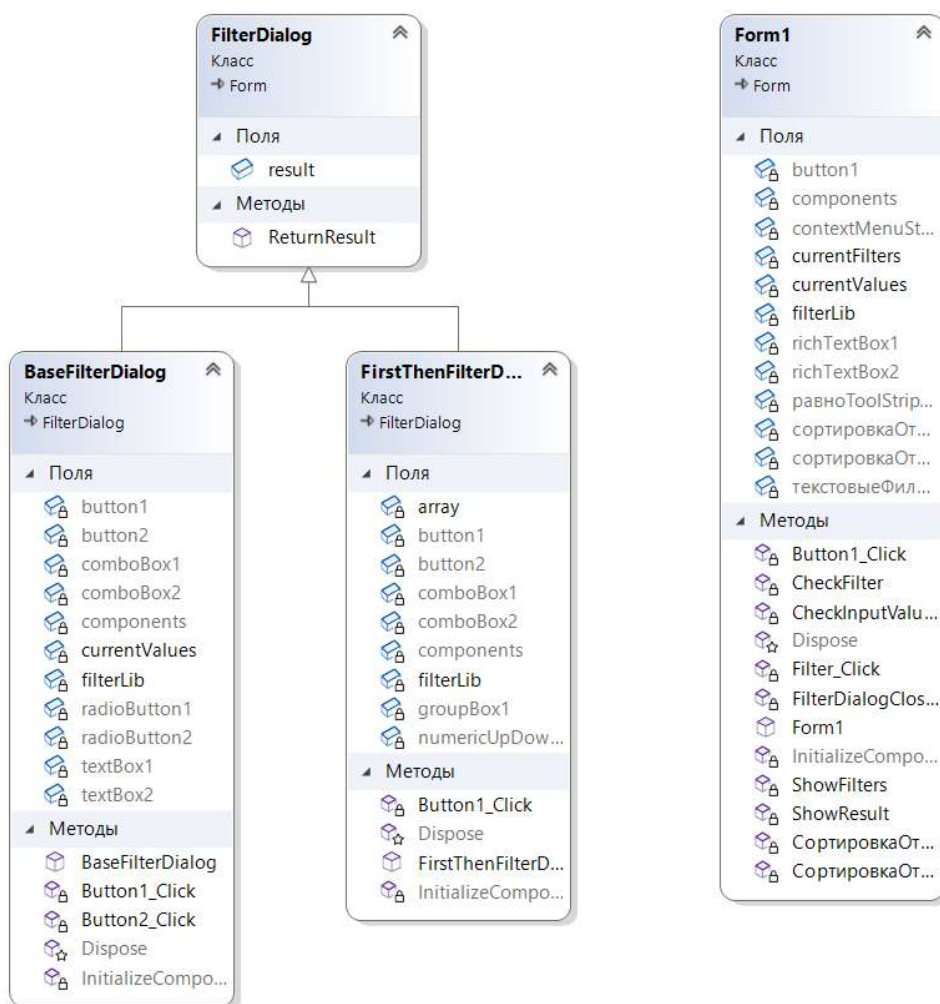


Рисунок 11 - UML диаграмма классов графического интерфейса библиотеки фильтрации и сортировки данных

1.3 Анализ результата

1.3.1 Тестирование и отладка

В ходе разработки программы было произведено функциональное тестирование, в процессе которого было проверено:

– иблиотека форматирования данных. Применение следующих типов форматов:

- 1) «общий»;
- 2) «числовой»;
- 3) «денежный»;
- 4) «финансовый»;
- 5) «дата»;
- 6) «время»;
- 7) «процентный»;
- 8) «дробный»;
- 9) «экспоненциальный»;
- 10) «текстовый»;
- 11) «дополнительный»;
- 12) введенный вручную пользователем;

– библиотека сортировки и фильтрации данных:

- 1) сортировка текстовых данных;
- 2) сортировка числовых данных;
- 3) сортировка даты и времени;
- 4) фильтрация текстовых данных;
- 5) фильтрация числовых данных;
- 6) фильтрация даты и времени;
- 7) комбинированные фильтры;

Функциональное тестирование выявило проблемы, связанные с отличием в форматировании данных по сравнению с аналогами, ошибки, возникающие

при вводе некорректного пользовательского кода формата, некорректное применение комбинированных фильтров, все они были успешно исправлены.

1.3.1.1 Тестирование библиотеки форматирования данных

Далее приведена таблица 1, содержащая набор тестов, в которых описывается идея теста, входная информация, процедура выполнения, ожидаемый и полученный результаты. После таблицы для каждой проверки прилагается снимок экрана, на котором изображен результат выполнения теста.

Таблица 1 - Тестирование библиотеки форматирования данных

Проверяется: Библиотека форматирования данных					
Документ: ВКР. Пункт: 1					
Дата создания: 26.05.2022				Дата модификации 26.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.1.1	Проверка корректности и работы применения формата «Общий»	Форматируемое значение «123456,789»	Ввести в текстовое поле форматируемое значение, выбрать в списке формат «Общий»	В текстовом поле с результатом появится значение «123456,789»	Совпадает с ожидаемым
ПФ.1.2	Проверка корректности и работы применения формата «Числовой»	Форматируемое значение «-123456,789», выбрано 2 знака после запятой, установлен разделитель разрядов, формат отрицательных второй (Красный без знака)»	Ввести в текстовое поле форматируемое значение, выбрать в списке формат «Числовой», установить 2 знака после запятой и разделитель разрядов, выбрать второй формат	В текстовом поле с результатом появится значение «123 456,79». Цвет текста - красный	Совпадает с ожидаемым

Проверяется: Библиотека форматирования данных

Документ: ВКР. Пункт: 1

Дата создания: 26.05.2022

Дата модификации

26.05.2022

№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
			отрицательных чисел		
ПФ.1.3	Проверка корректности и работы применения формата «Денежный»	Форматируемое значение «123456,789», выбрана валюта «Российский рубль», формат отрицательных чисел без изменений, число знаков после запятой - 0	Ввести в текстовое поле форматируемое значение, выбрать в списке формат «Денежный», выбрать валюту «Российский рубль»	В текстовом поле с результатом появится значение «123 457 Р».	Совпадает с ожидаемым
ПФ.1.4	Проверка корректности и работы применения формата «Финансовый»	Форматируемое значение «123456,789», выбрана валюта «Российский рубль», число знаков после запятой - 2	Ввести в текстовое поле форматируемое значение, выбрать в списке формат «Финансовый», выбрать валюту «Российский рубль», установить число знаков после запятой - 2	В текстовом поле с результатом появится значение «123 456,79 Р».	Совпадает с ожидаемым
ПФ.1.5	Проверка корректности и работы применения формата «Дата»	Форматируемое значение «08.03.2022», выбран тип представления даты «2012, март»	Ввести в текстовое поле форматируемое значение, выбрать в списке тип представления	В текстовом поле с результатом появится значение «2022, Март»	Совпадает с ожидаемым

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

34

Проверяется: Библиотека форматирования данных

Документ: ВКР. Пункт: 1

Дата создания: 26.05.2022

Дата модификации

26.05.2022

№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
			даты «2012, март»		
ПФ.1.6	Проверка корректности и работы применения формата «Время»	Форматируемое значение «17:30», выбран тип представления времени «1:30 PM»	Ввести в текстовое поле форматируемое значение, выбрать в списке тип представления времени «1:30 PM»	В текстовом поле с результатом появится значение «5:30 PM»	Совпадает с ожидаемым
ПФ.1.7	Проверка корректности и работы применения формата «Процентный»	Форматируемое значение «123456,789», количество знаков после запятой - 0	Ввести в текстовое поле форматируемое значение, установить количество знаков после запятой - 0	В текстовом поле с результатом появится значение «12345679%»	Совпадает с ожидаемым
ПФ.1.8	Проверка корректности и работы применения формата «Дробный»	Форматируемое значение «123456,789», вид представления дроби – «Шестнадцатыми долями»	Ввести в текстовое поле форматируемое значение, установить вид представления дроби – «Шестнадцатым и долями»	В текстовом поле с результатом появится значение «123456 13/16»	Совпадает с ожидаемым
ПФ.1.9	Проверка корректности и работы применения формата	Форматируемое значение «123456,789», количество знаков после запятой - 5	Ввести в текстовое поле форматируемое значение, установить	В текстовом поле с результатом появится значение	Совпадает с ожидаемым

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

35

Проверяется: Библиотека форматирования данных					
Документ: ВКР. Пункт: 1					
Дата создания: 26.05.2022				Дата модификации 26.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
	«Экспоненциальный»		количество знаков после запятой - 5	«1,23457E+05»	
ПФ.1.1 0	Проверка корректности работы применения формата «Текстовый»	Форматируемое значение «abcdefg»	Ввести в текстовое поле форматируемое значение	В текстовом поле с результатом появится значение «abcdefg»	Совпадает с ожидаемым
ПФ.1.1 1	Проверка корректности работы применения формата «Дополнительный»	Форматируемое значение «1234567890», тип формата «Номер телефона»	Ввести в текстовое поле форматируемое значение, установить тип формата «Номер телефона»	В текстовом поле с результатом появится значение «(123) 456-7890»	Совпадает с ожидаемым
ПФ.1.1 2	Проверка корректности работы применения формата, введенного пользователем	Форматируемое значение «12345,6», код формата «[Красный][>100] #.0»	Ввести в текстовое поле форматируемое значение, в текстовое поле с кодом формата ввести код формата	В текстовом поле с результатом появится значение «12345,6». Цвет текста - красный	Совпадает с ожидаемым

Исходя из полученных результатов, можно сделать вывод, что разработанная библиотека форматирования данных работает корректно и отвечает всем поставленным требованиям. Далее приведены снимки экранов, подтверждающие корректное выполнение каждого из тестов. Каждый из них имеет подпись с номером теста, к которому он относится.

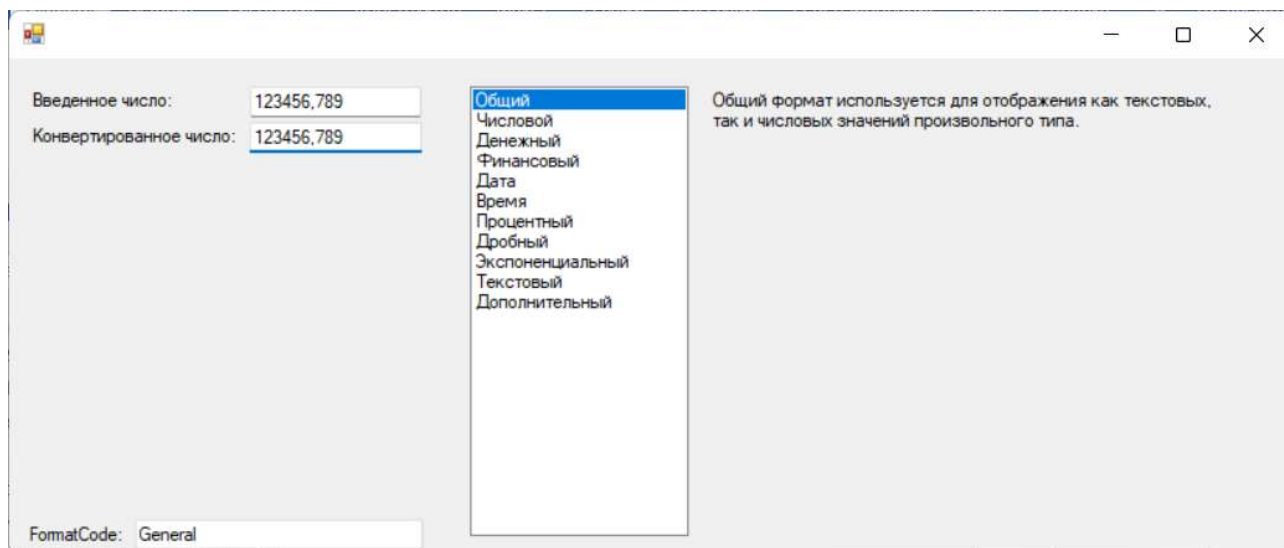


Рисунок 12 - ПФ.1.1

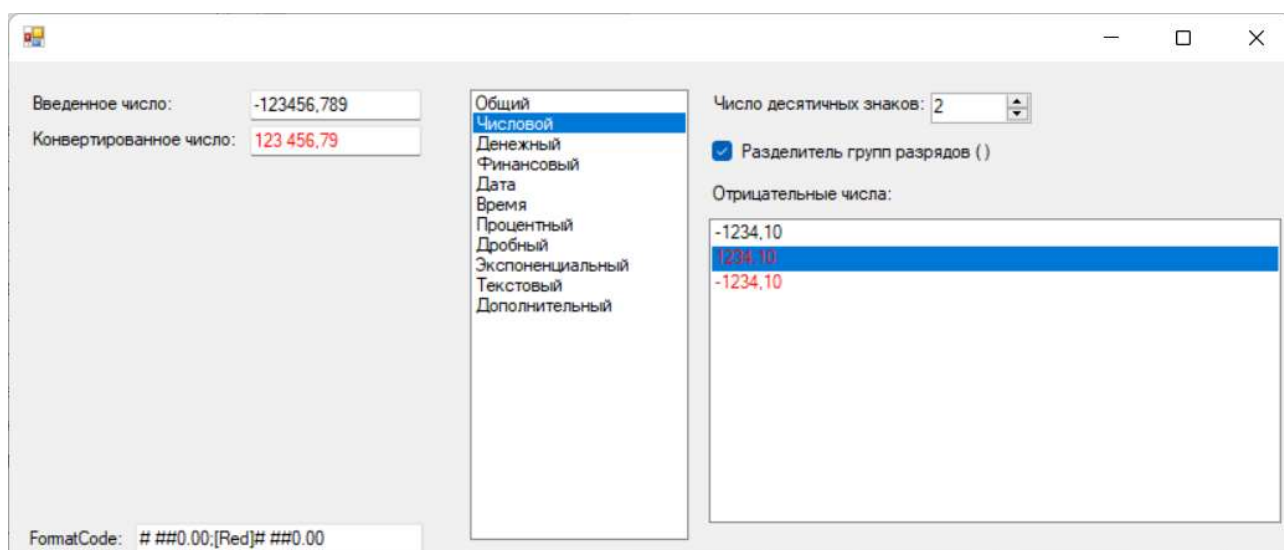


Рисунок 13 - ПФ.1.2

Введенное число:	123456,789	Общий Числовой Денежный Финансовый Дата Время Процентный Дробный Экспоненциальный Текстовый Дополнительный	Число десятичных знаков:	0
Конвертированное число:	123 457 P		Обозначение:	Russian Ruble
FormatCode: ##0 P		Отрицательные числа: -1 234,10 1 234,10 -1 234,10		

Рисунок 14 - ПФ.1.3

Введенное число:	123456,789	Общий Числовой Денежный Финансовый Дата Время Процентный Дробный Экспоненциальный Текстовый Дополнительный	Число десятичных знаков:	2
Конвертированное число:	123 456,79 P		Обозначение:	Russian Ruble
FormatCode: _* ##0.00 P_:_* ##0.00 P_:_* "				

Рисунок 15 - ПФ.1.4

Введенное число:	08.03.2022	Общий Числовой Денежный Финансовый Дата Время Процентный Дробный Экспоненциальный Текстовый Дополнительный	Тип:
Конвертированное число:	2022, Март		2012-03-1 1.3 1.3.12 1.03.12 1 мар 1 мар 12 01 мар 12 2012, март Март 2012 01 марта 2012 г. 01.03.12 1:30 PM 01.03.12 13:30 Март 2012, 14 марта 1.3.2012 1-мар-2012
FormatCode: yyyy, MMMM			

Рисунок 16 - ПФ.1.5

Введенное число:	17:30	Общий Числовой Денежный Финансовый Дата Время Процентный Дробный Экспоненциальный Текстовый Дополнительный	Тип:
Конвертированное число:	5:30 PM		13:30 1:30 PM 13:30:55 1:30:55 PM 30:55.2 37:30:55 14.03.12 1:30 PM 14.03.12 13:30
FormatCode: h:mm am/pm			

Рисунок 17 - ПФ.1.6

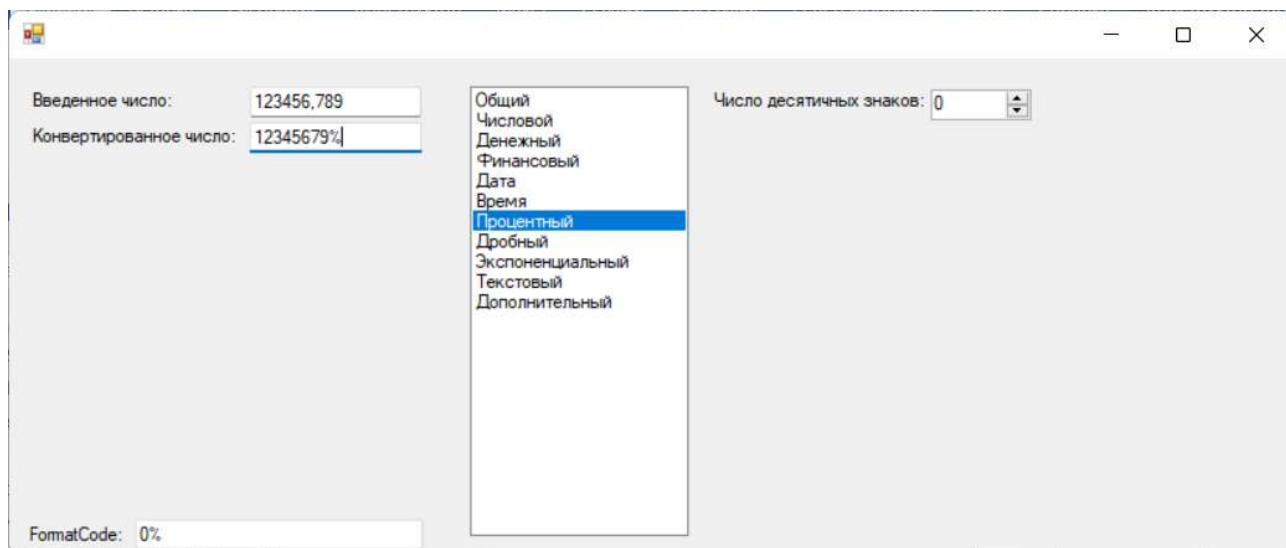


Рисунок 18 - ПФ.1.7

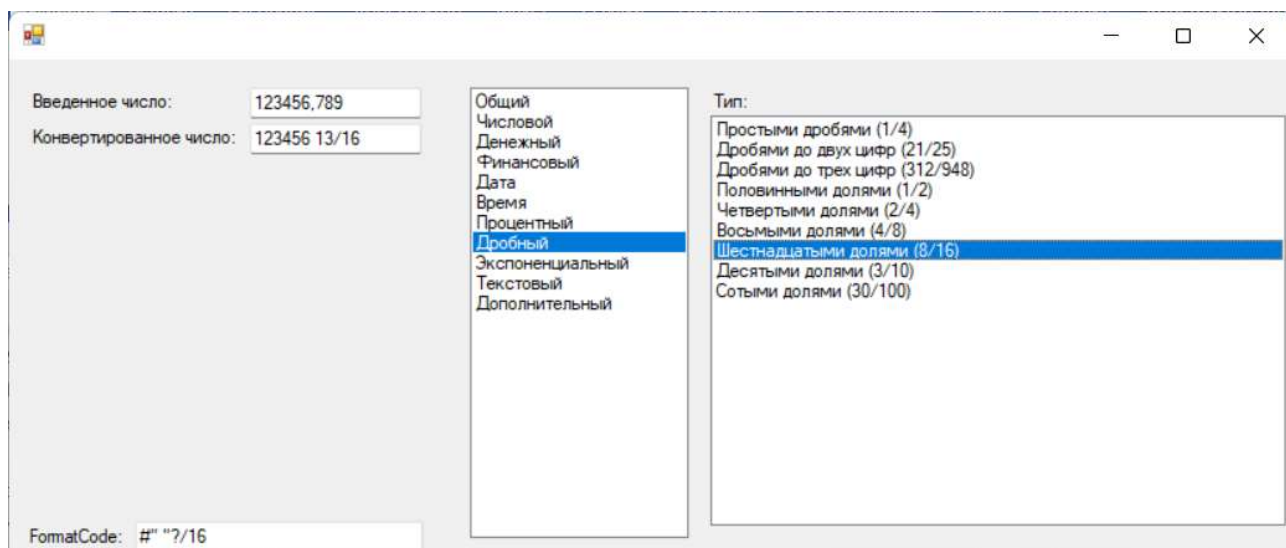


Рисунок 19 - ПФ.1.8

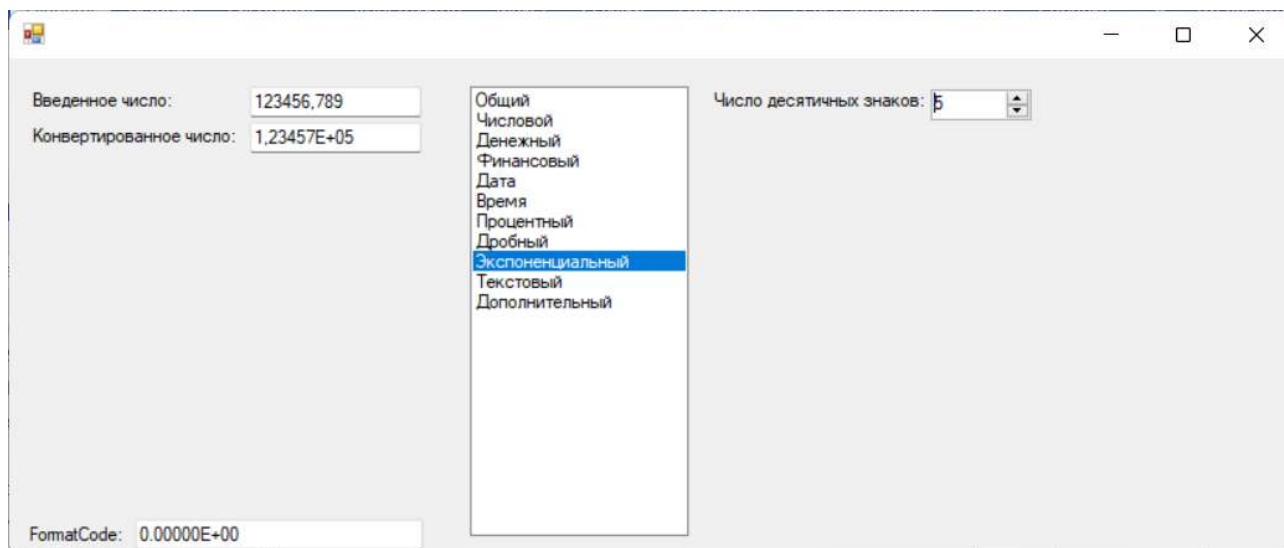


Рисунок 20 - ПФ.1.9

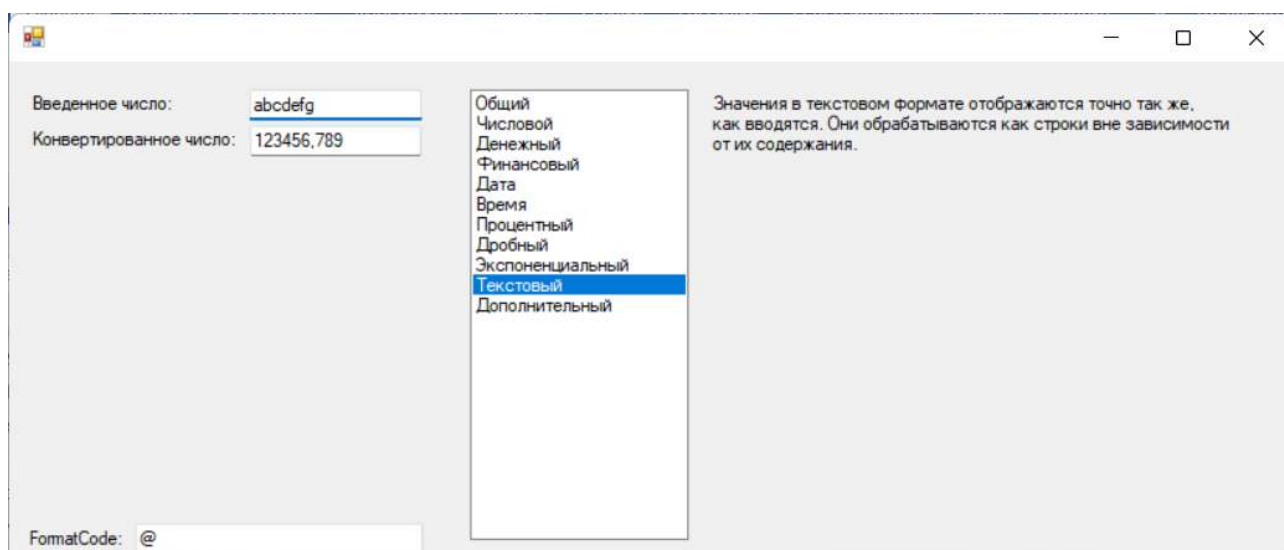


Рисунок 21 - ПФ1.10

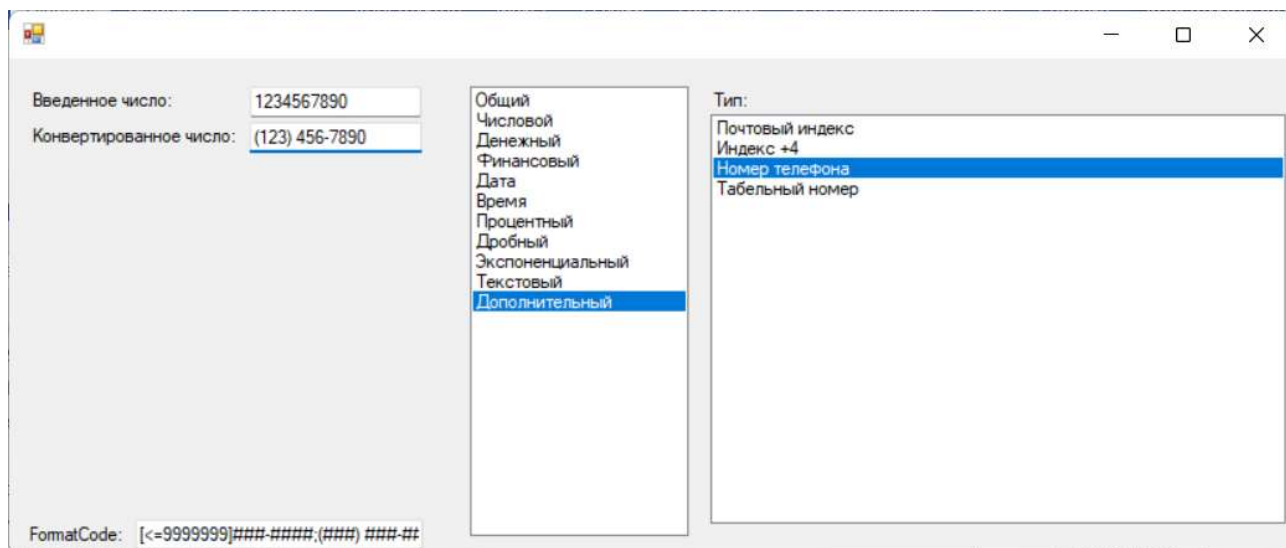


Рисунок 22 - ПФ1.11

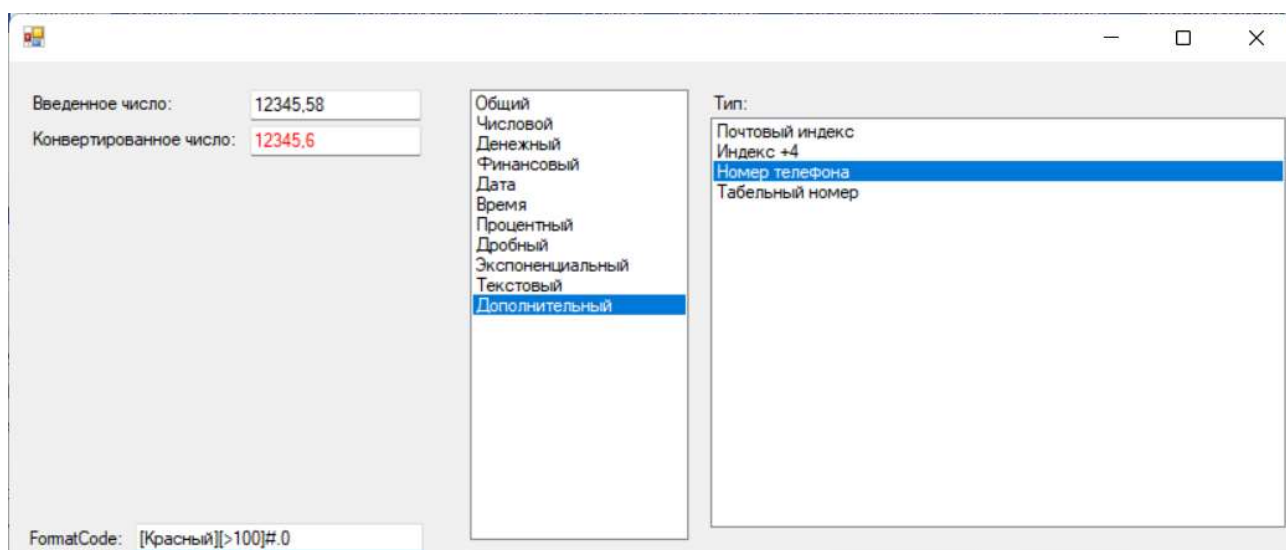


Рисунок 23 - ПФ.1.12

1.3.1.2 Тестирование библиотеки сортировки и фильтрации данных

Далее приведена таблица, содержащая набор тестов, в которых описывается идея теста, входная информация, процедура выполнения, ожидаемый и полученный результаты. После таблицы для каждой проверки прилагается снимок экрана, на котором изображен результат выполнения теста.

Таблица 2 - Тестирование библиотеки сортировки и фильтрации данных

Проверяется: Библиотека сортировки и фильтрации данных Документ: ВКР. Пункт: 2					
Дата создания: 27.05.2022				Дата модификации 27.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.2.1	Сортировка текстовых данных	Массив текстовых данных def abc zax efg klm	Ввести в текстовое поле массив, выбрать сортировку от «А до Я»	В текстовом поле с результатом появится текст «abc def efg klm zax»	Совпадает с ожидаемым
ПФ.2.2	Сортировка числовых данных	Массив числовых данных 543 123 432 999 875	Ввести в текстовое поле массив, выбрать сортировку по убыванию	В текстовом поле с результатом появится текст «999 875 543 432 123»	Совпадает с ожидаемым
ПФ.2.3	Сортировка даты и времени	Массив значений даты 09.12.2000 12.12.1998 08.03.2022 08.02.2022 01.01.2000	Ввести в текстовое поле массив, выбрать сортировку «От старых к новым»	В текстовом поле с результатом появится текст «12.12.1998 01.01.2000 09.12.2000 08.02.2022 08.03.2022»	Совпадает с ожидаемым
ПФ.2.4	Фильтрация текстовых данных	Массив текстовых	Ввести в текстовое поле массив,	В текстовом поле с	Совпадает с ожидаемым

Проверяется: Библиотека сортировки и фильтрации данных

Документ: ВКР. Пункт: 2

Дата создания: 27.05.2022				Дата модификации 27.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
		данных abc def aef jkl qwe anm	выбрать «Начинается С», в текстовом поле напротив названия фильтра ввести букву «а», нажать «ОК»	результатом появится текст «abc aef anm»	
ПФ.2.5	Фильтрация числовых данных	Массив числовых данных 123 8 54 987 342 11 17	Ввести в текстовое поле массив, выбрать фильтр «Больше», напротив названия фильтра ввести число «100», нажать «ОК»	В текстовом поле с результатом появится текст «123 987 342»	Совпадает с ожидаемым
ПФ.2.6	Фильтрация даты и времени	Массив значений даты 09.12.2000 12.12.1998 08.03.2022 08.02.2022 01.01.2000	Ввести в текстовое поле массив, выбрать фильтр «В этом году»	В текстовом поле с результатом появится текст «08.03.2022 08.02.2022»	Совпадает с ожидаемым
ПФ.2.7	Комбинированные фильтры	Массив значений даты 09.12.2000 12.12.1998 08.03.2022 08.02.2022 01.01.2000	Ввести в текстовое поле массив, выбрать фильтр «После», напротив названия фильтра ввести «02.01.2000», установить радио-	В текстовом поле с результатом появится текст «09.12.2000 08.02.2022»	Совпадает с ожидаемым

Проверяется: Библиотека сортировки и фильтрации данных					
Документ: ВКР. Пункт: 2					
Дата создания: 27.05.2022				Дата модификации 27.05.2022	
№ теста	Идея теста	Входная информация	Процедура выполнения	Ожидаемый результат	Полученный результат
			кнопку «И», выбрать второй фильтр «До», напротив названия фильтра ввести «01.03.2022», нажать «ОК»		

Исходя из полученных результатов, можно сделать вывод, что разработанная библиотека сортировки и фильтрации данных работает корректно и отвечает всем поставленным требованиям. Далее приведены рисунки, подтверждающие корректное выполнение каждого из тестов. Каждый из них имеет подпись с номером теста, к которому он относится.

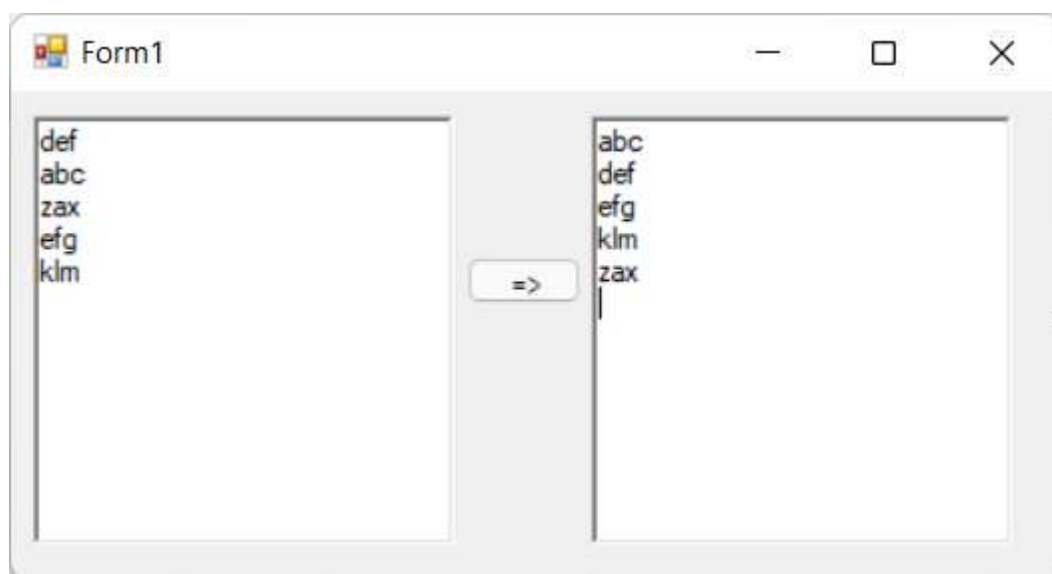


Рисунок 24 - ПФ.2.1

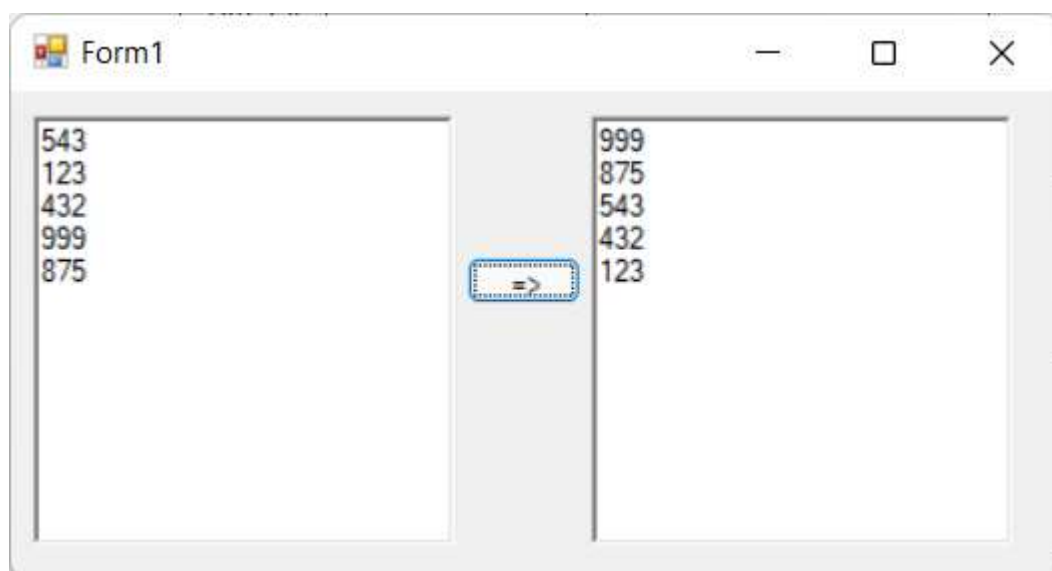


Рисунок 25 - ПФ.2.2

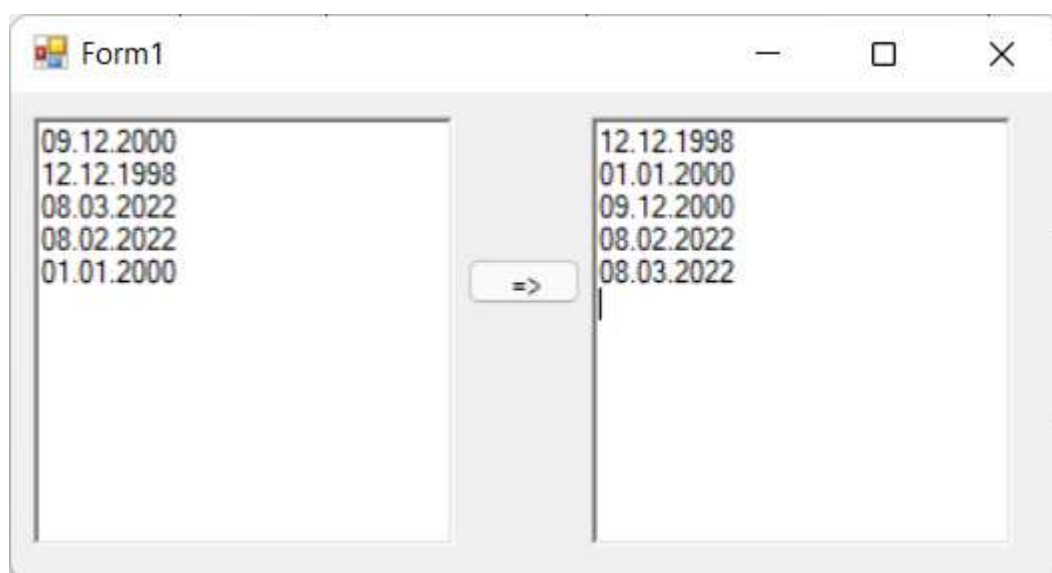


Рисунок 26 - ПФ.2.3

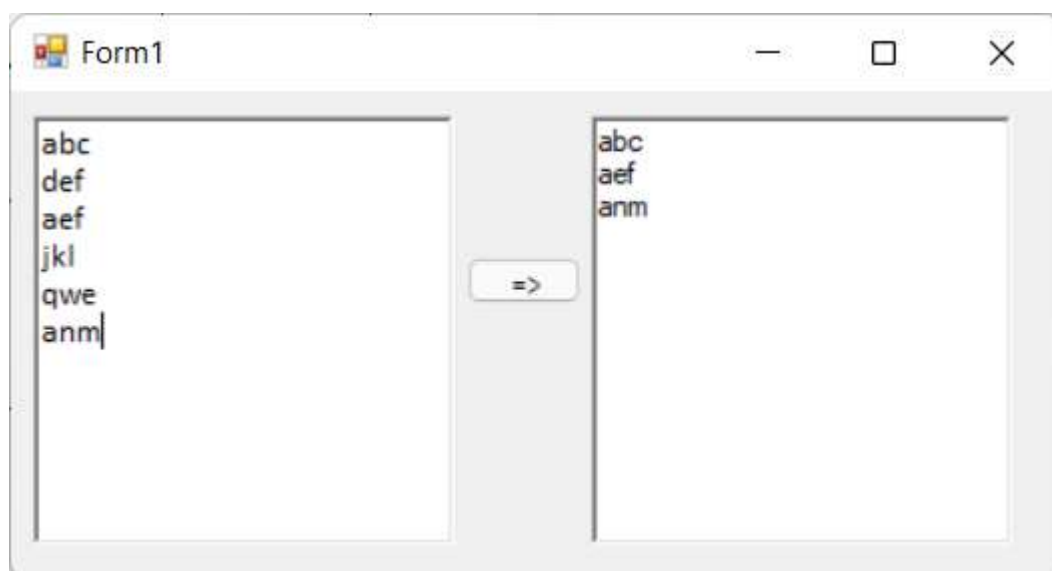


Рисунок 27 - ПФ.2.4

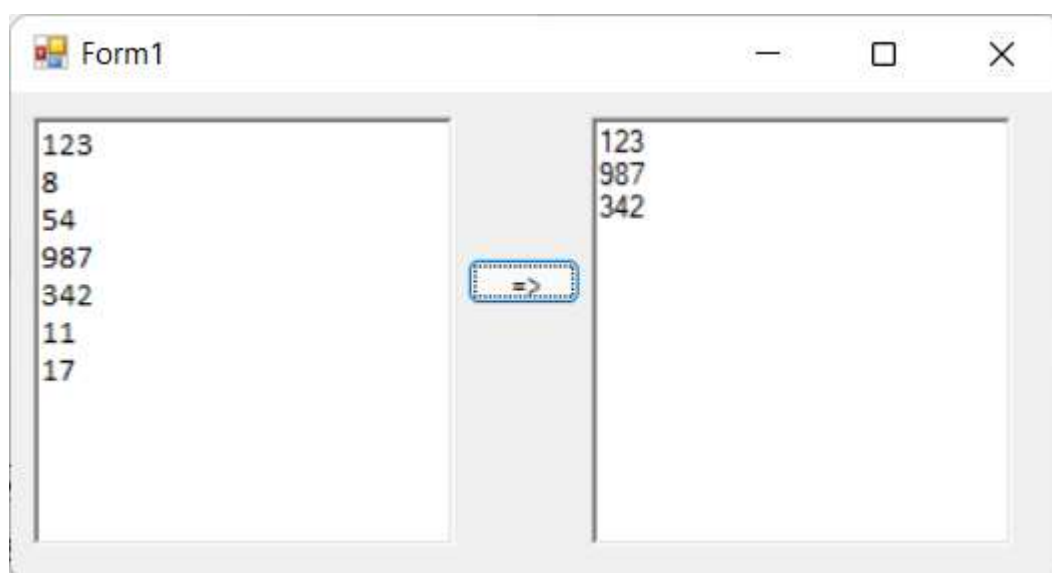


Рисунок 28 - ПФ.2.5

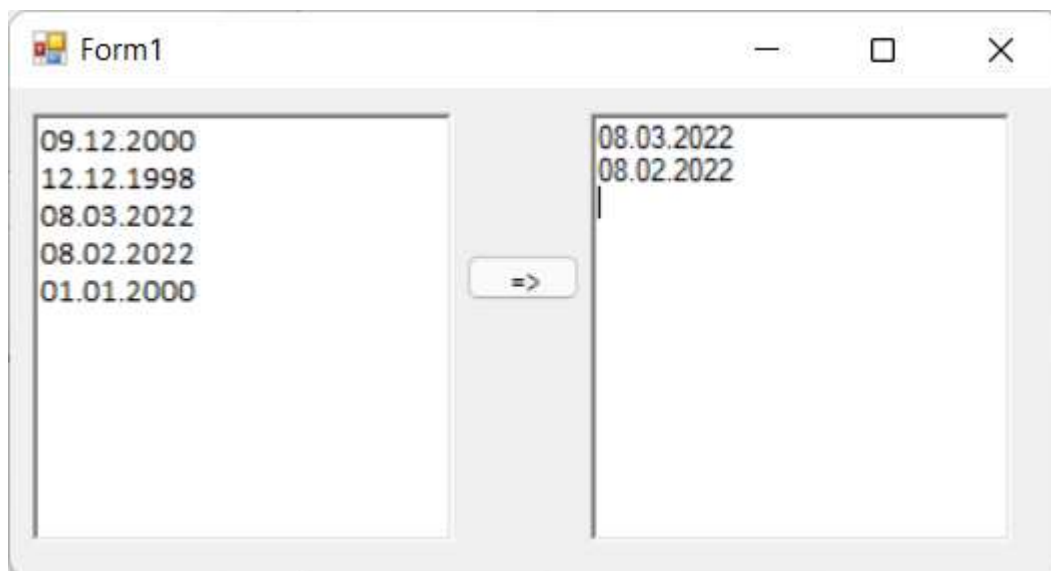


Рисунок 29 - ПФ.2.6

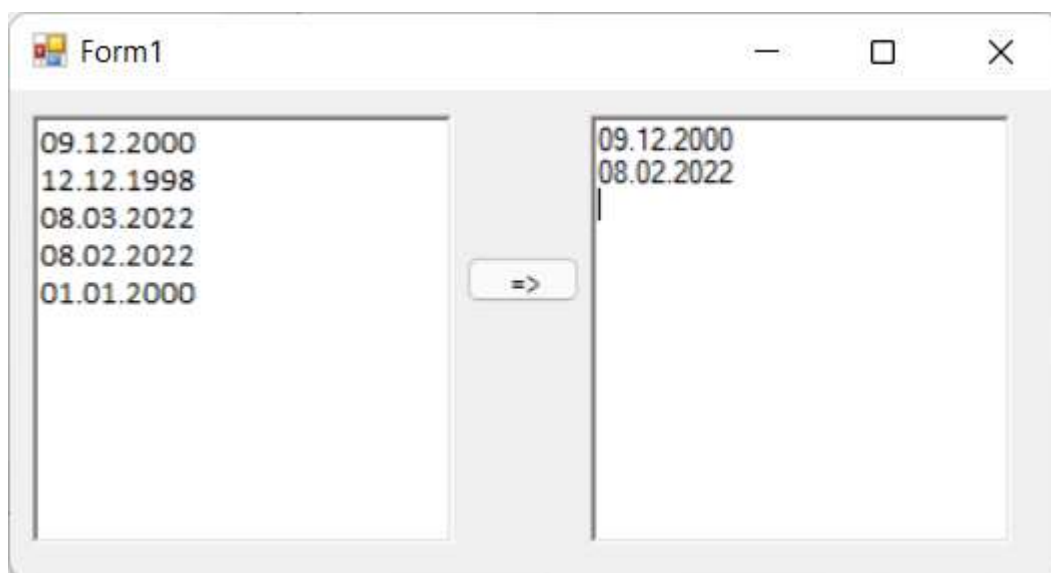


Рисунок 30 - ПФ.2.7

1.3.2 Тестирование в табличном процессоре «Ведомость»

Главным итогом выполнения ВКР, является интеграция модуля форматирования данных в табличный процессор «Ведомость», разрабатываемый научно-техническим предприятием «Криптософт».

Разработанный модуль соответствует всем поставленным требованиям и ничем не уступает по функционалу аналогам. Было принято решение оставить

графический интерфейс неизменным для удобства работы пользователей. Форматирование данных также происходит полностью корректно.

По итогам интеграции удалось добиться полной совместимости со стандартном *Office Open Xml*, что означает корректное считывание таблиц, созданных в других табличных редакторах, а также считывание ими таблиц созданных в Ведомости.

Далее представлены снимки экрана из табличного процессора, разрабатываемого НТП «Криптософт».

На рисунке 31 изображено главное окно табличного процессора, в красной рамке выделены элементы управления форматом данных.

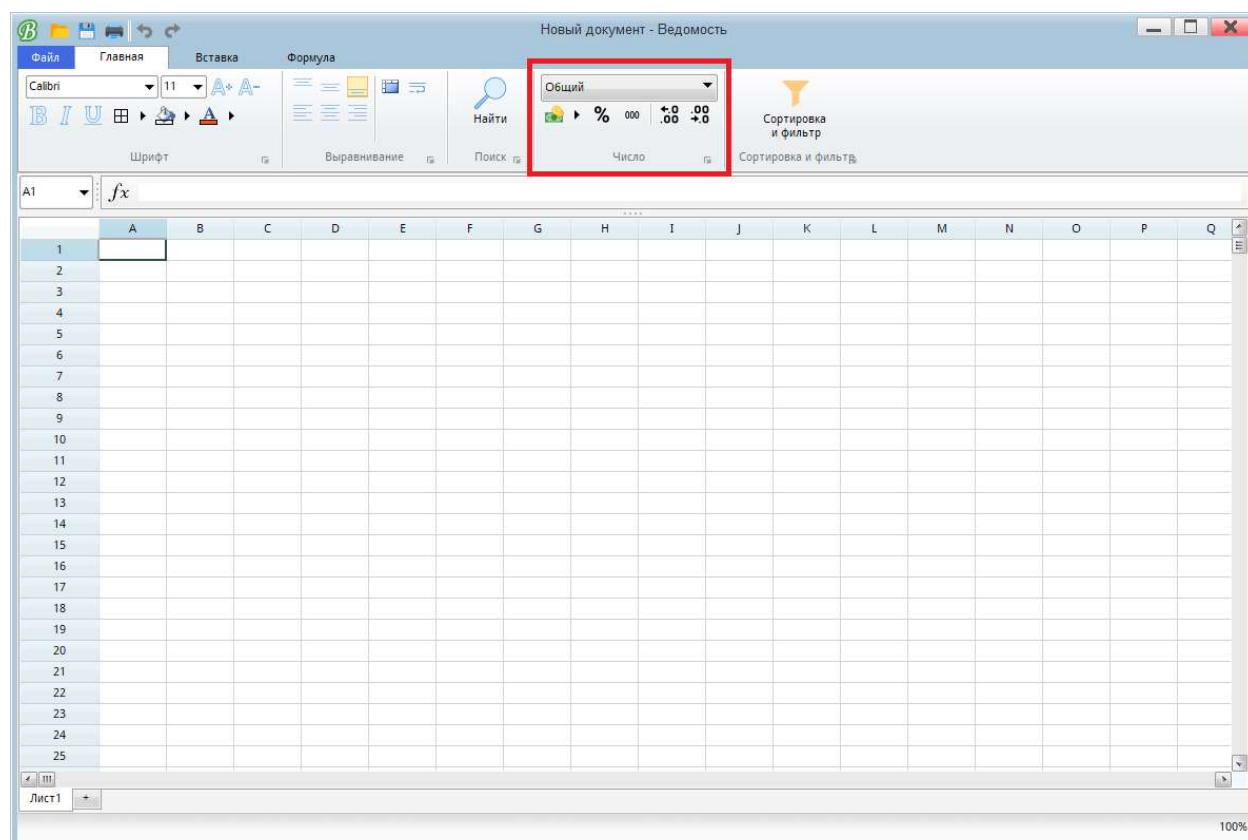


Рисунок 31 - Главное окно табличного процессора «Ведомость»

В выпадающем списке пользователь может выбрать один из форматов данных, не открывая окно форматирования. На рисунке 32 приведена структура данного списка:

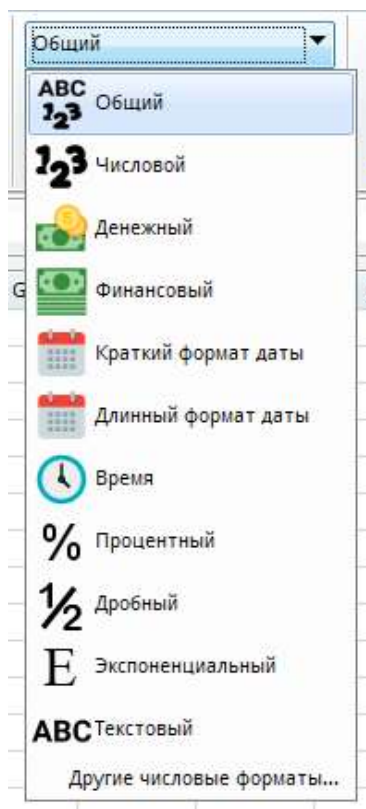


Рисунок 32 - Выпадающий список, содержащий форматы

Кнопки, находящиеся под выпадающим списком, отвечают за следующий функционал:

- выбор формата денежной единицы;
- отображение значение в процентах, отображение значения с разделителем групп разрядов;
- отображение более точного значения за счет увеличения числа знаков после запятой;
- отображение менее точного значения за счет уменьшения числа знаков после запятой.

На снимках экрана ниже приведены примеры форматирования данных для каждого из форматов.

На рисунке 33 изображен результат применения формата типа «Общий» к ячейке A1.

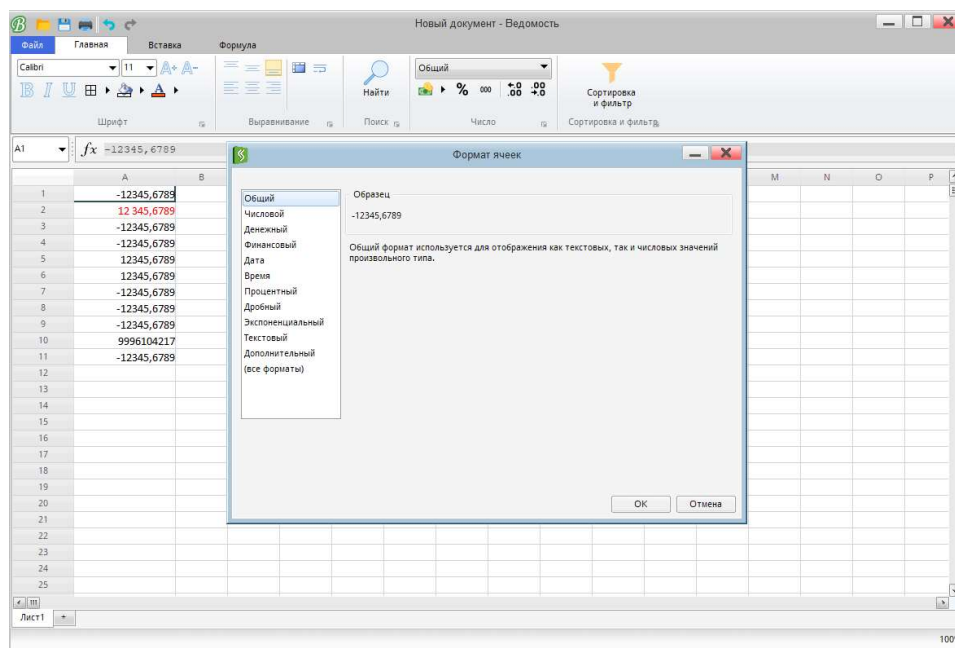


Рисунок 33 - Ведомость. Формат «Общий»

На рисунке 34 изображен результат применения к ячейке A2 формата типа «Числовой» с количеством десятичных знаков 4, разделителем и форматом для отрицательных чисел с красным цветом без знака «-».

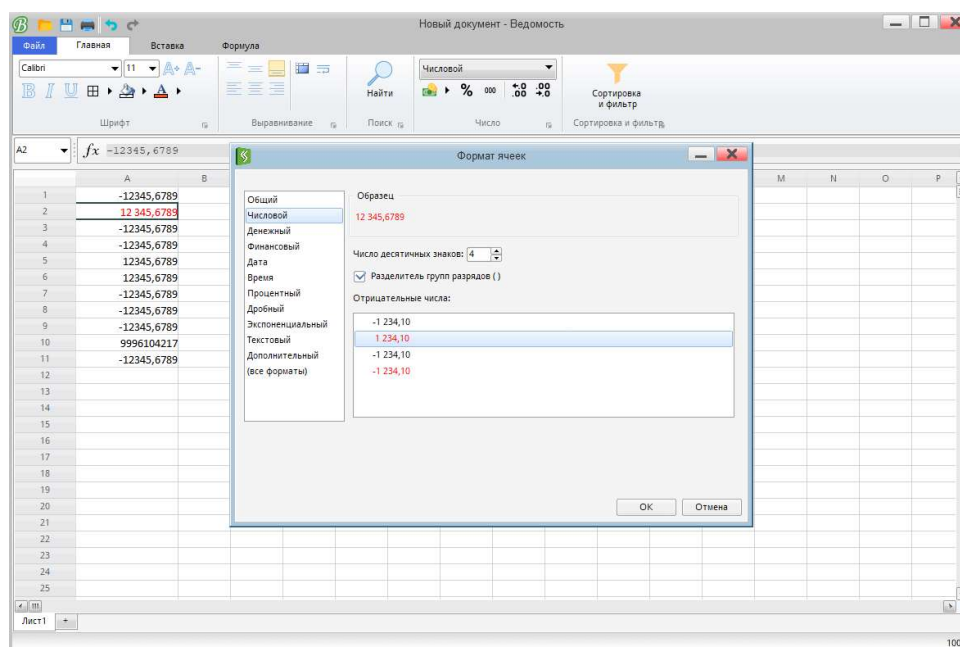


Рисунок 34 - Ведомость. Формат «Числовой»

На рисунке 35 изображен результат применения к ячейке A3 формата типа «Денежный» с количеством десятичных знаков 4, валютой евро и

форматом для отрицательных чисел с красным цветом без знака «-».

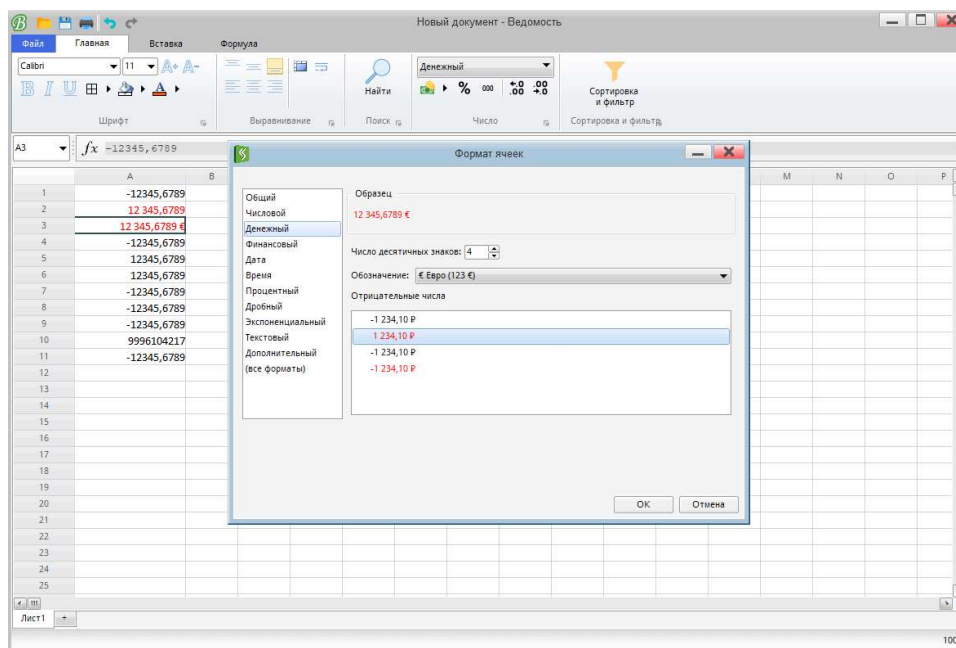


Рисунок 35 - Ведомость. Формат «Денежный»

На рисунке 36 изображен результат применения к ячейке A4 формата типа «Финансовый» с количеством десятичных знаков 5 и валютой доллар.

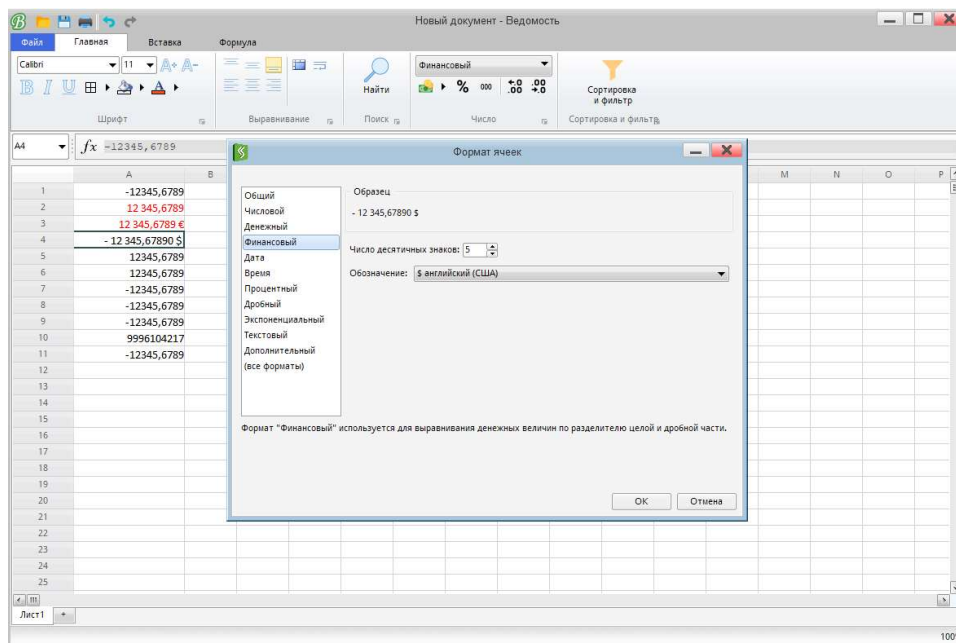


Рисунок 36 - Ведомость. Формат «Финансовый»

На рисунке 37 изображен результат применения к ячейке A5 формата типа «Дата» вида «ДД-МММ-ГГГГ».

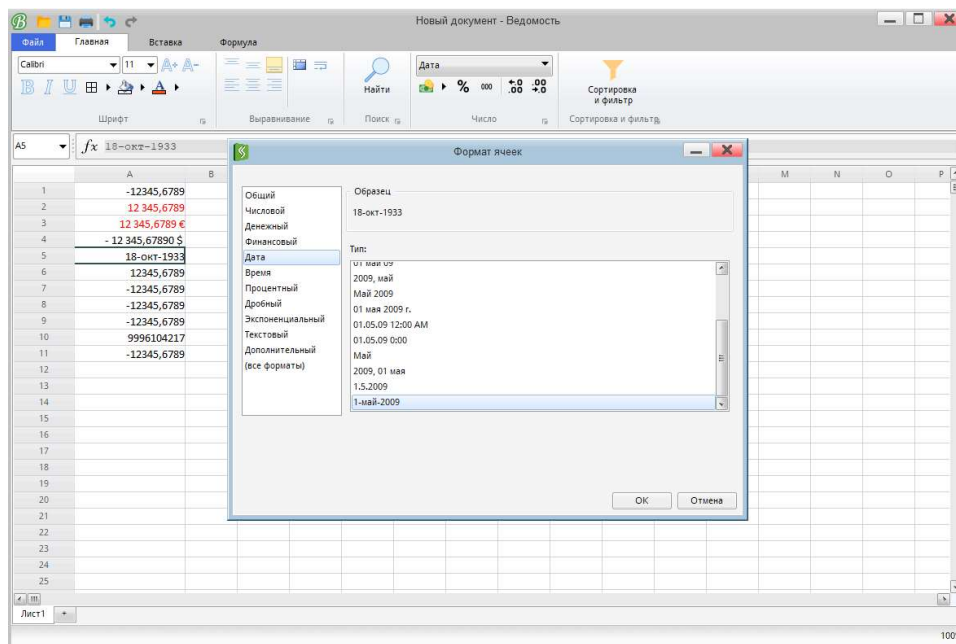


Рисунок 37 - Ведомость. Формат «Дата»

На рисунке 38 изображен результат применения к ячейке A6 формата типа «Время» вида «hh:mm AM/PM».

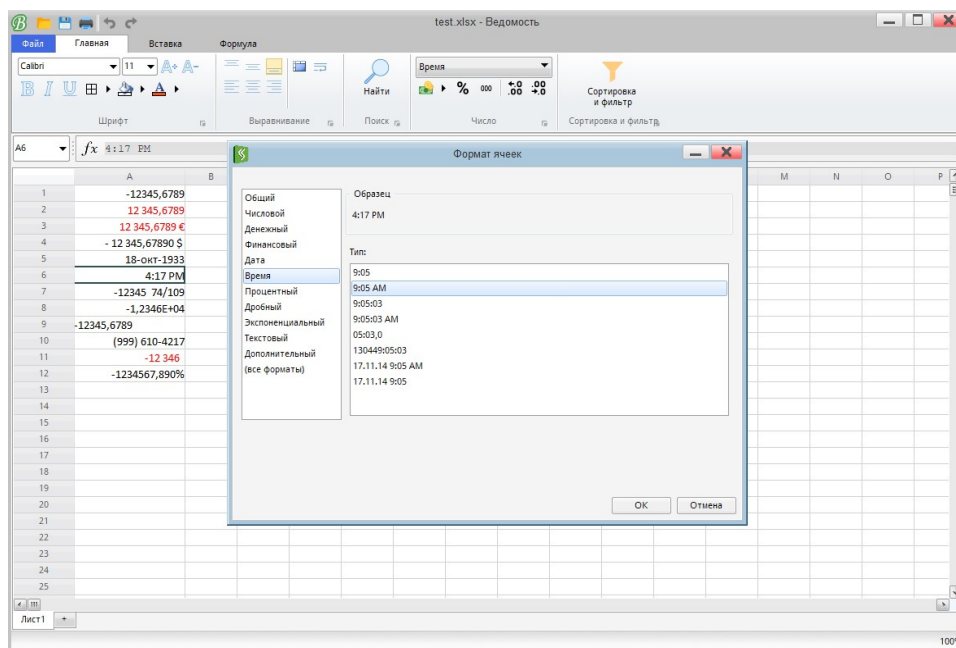


Рисунок 38 - Ведомость. Формат «Время»

На рисунке 39 изображен результат применения к ячейке A12 формата типа «Процентный» с количеством десятичных знаков 3.

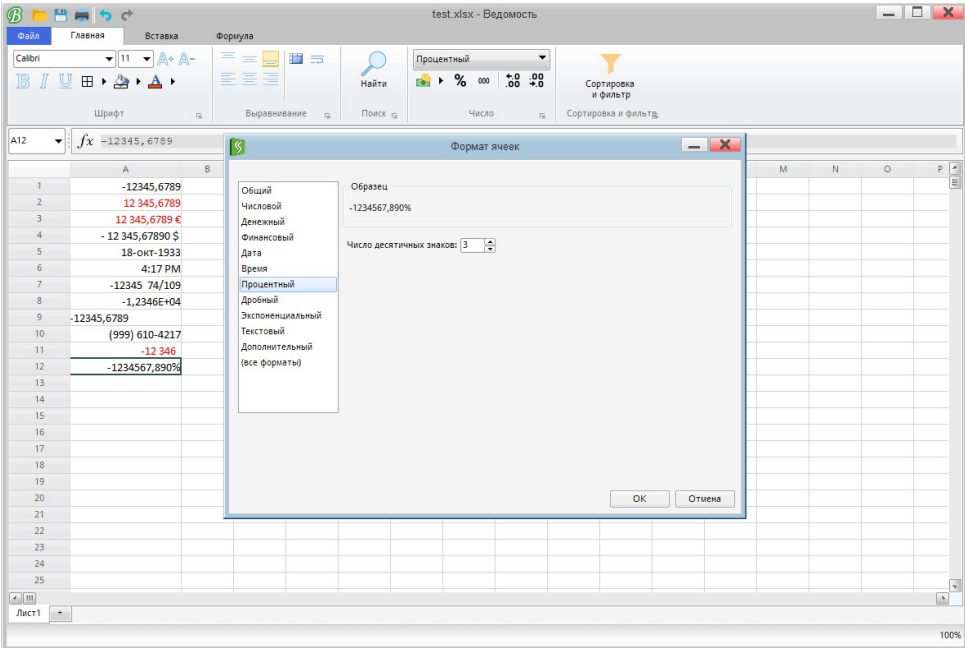


Рисунок 39 - Ведомость. Формат «Процентный»

На рисунке 40 изображен результат применения к ячейке A7 формата типа «Дробный» вида «Дробями до трех цифр».

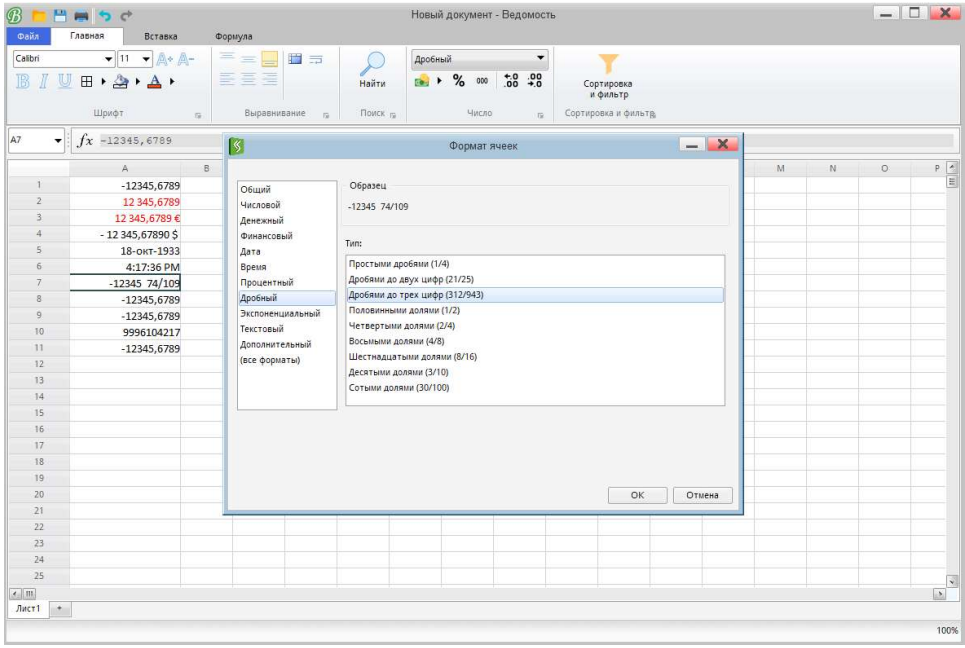


Рисунок 40 - Ведомость. Формат «Дробный»

На рисунке 41 изображен результат применения к ячейке A8 формата типа «Экспоненциальный» с количеством десятичных знаков 4.

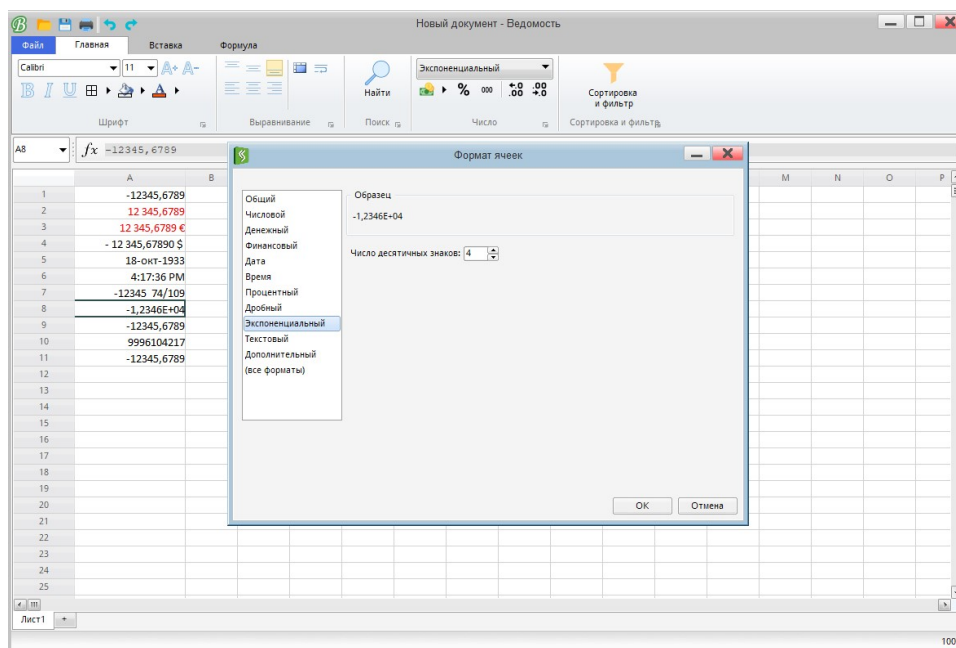


Рисунок 41 - Ведомость. Формат «Экспоненциальный»

На рисунке 42 изображен результат применения к ячейке A9 формата типа «Текстовый».

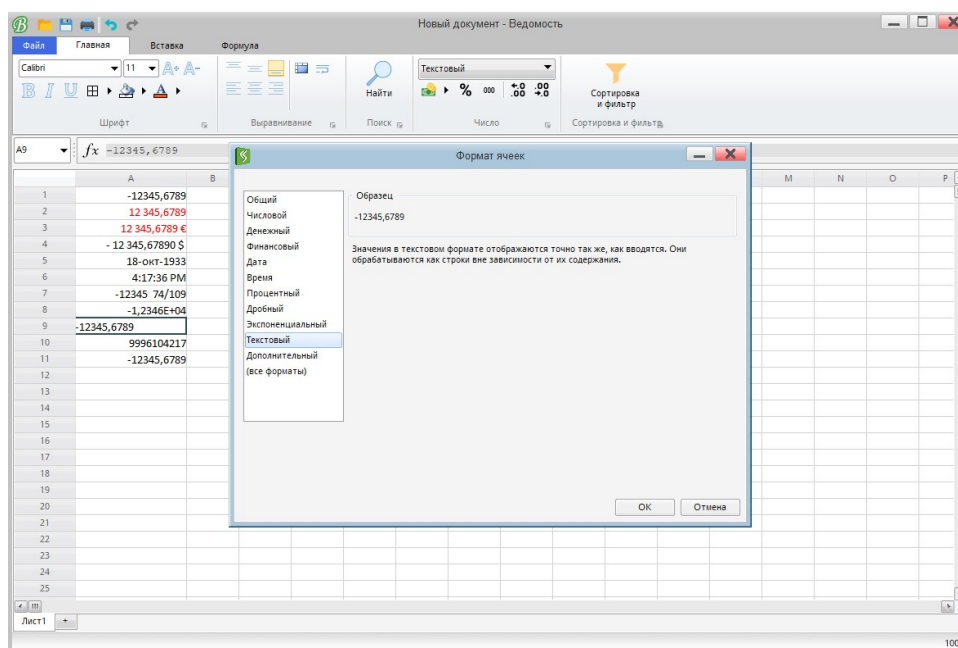


Рисунок 42 - Ведомость. Формат «Текстовый»

На рисунке 43 изображен результат применения к ячейке A10 формата типа «Дополнительный» вида «Номер телефона».

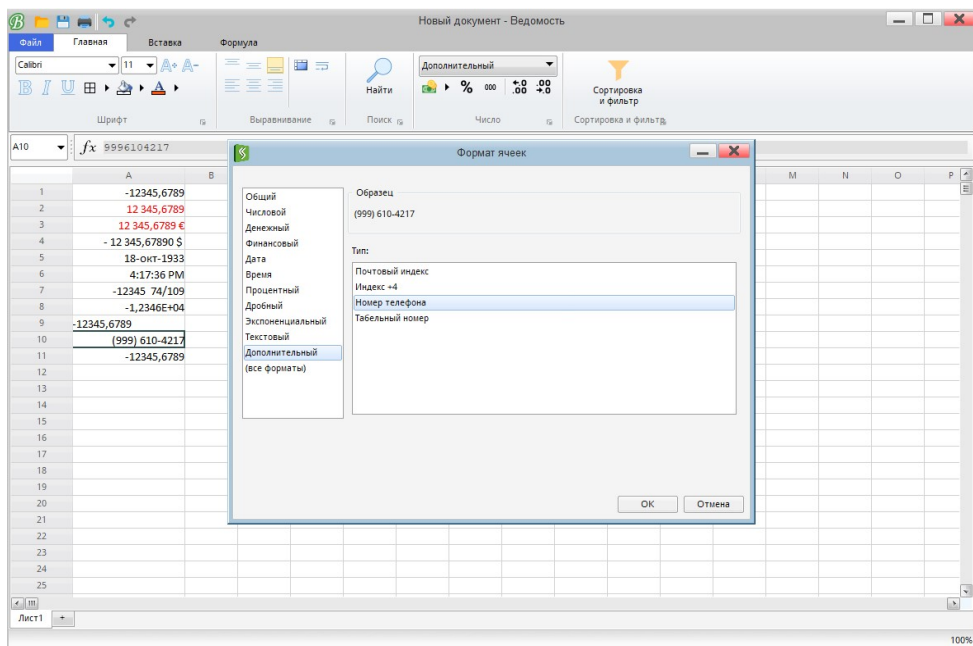


Рисунок 43 - Ведомость. Формат «Дополнительный»

На рисунке 44 изображен результат применения к ячейке A11 пользовательского кода формата.

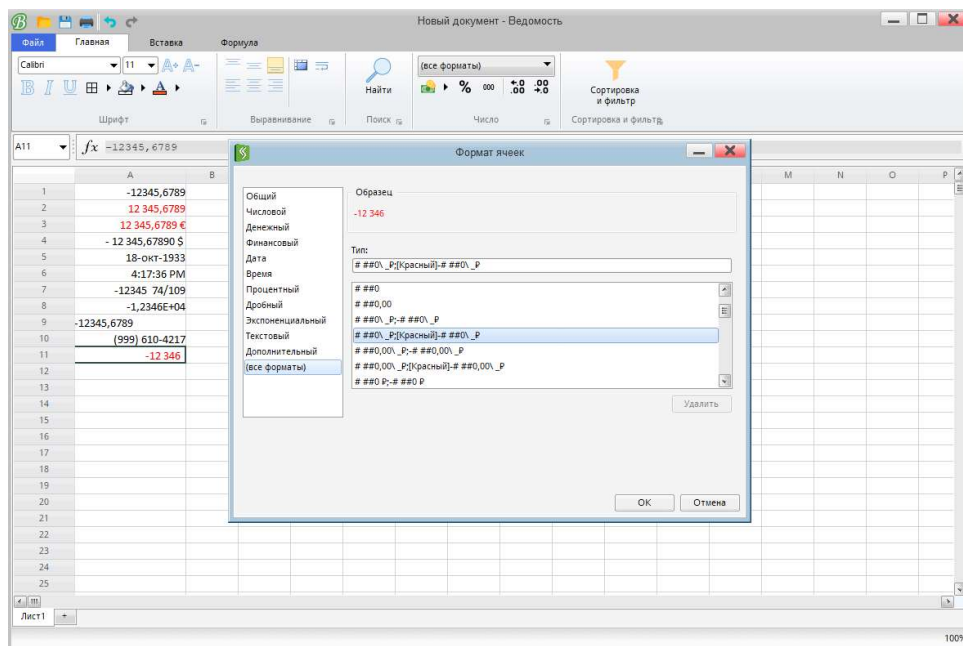


Рисунок 44 - Ведомость. Пользовательский формат

Конечный документ после применения всех форматов к диапазону ячеек A1-A13 имеет вид, изображенный на рисунке 45.

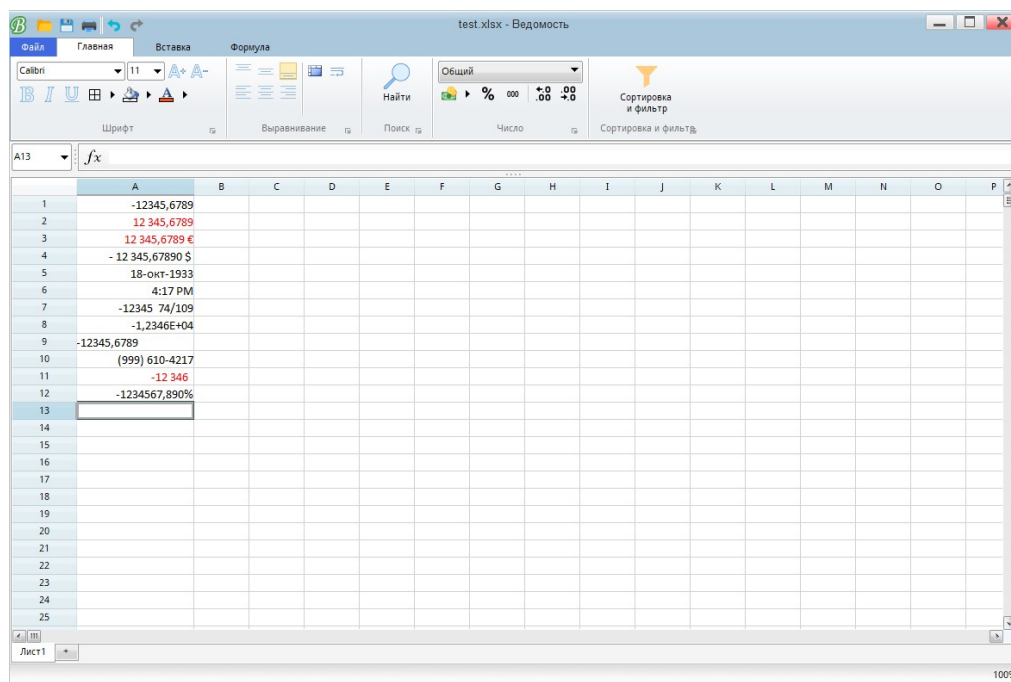


Рисунок 45 - Ведомость. Тестовый документ

Затем созданный документ был сохранен и открыт с помощью табличного редактора «Microsoft Excel». На рисунке 46 изображен открытый через него файл.

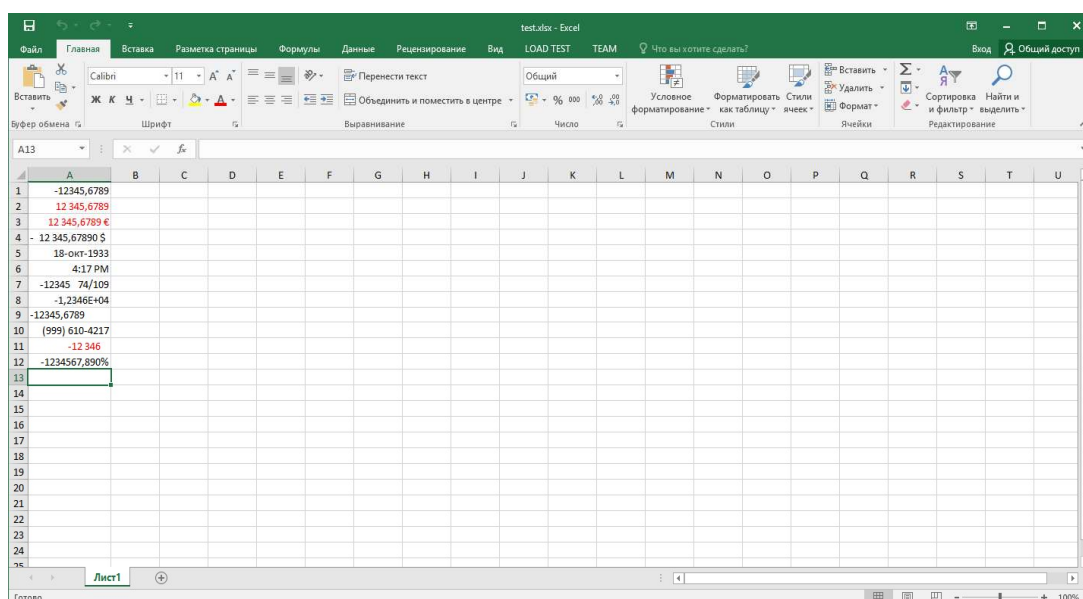


Рисунок 46 - Microsoft Excel. Тестовый файл

Сравнив результат из табличного редактора «Ведомость» и «Microsoft Excel», можно сделать вывод, что форматирование данных полностью идентично, что означает корректную работу разработанного модуля.

					ПГУ 2.090301.39.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

2 Безопасность жизнедеятельности

2.1 Анализ потенциальных опасностей и вредностей, возникающих при работе с ПЭВМ и способы их минимизации

В настоящее время персональные компьютеры стали неотъемлемой частью жизни почти всех людей во многих сферах жизнедеятельности. Но при этом нельзя отрицать их негативное взаимодействие на здоровье человека. Чтобы минимизировать вред, пользователю ПК необходимо соблюдать соответствующие меры безопасности. Основными из них являются: правильное позиционирование рабочего места, планировать режим работы за ПЭВМ, производить контроль освещенности и уровня шума помещения.

Далее в данном разделе ВКР будут рассматриваться требования по организации рабочего процесса для пользователей вычислительных машин, участвующих в разработке программного обеспечения, а также способы минимизации воздействия вредных факторов на здоровье человека.

2.2 Классификация и категорирование помещения, в котором производится разработка

Далее будет рассмотрена классификация помещений по уровню электробезопасности и пожарной безопасности.

2.2.1 Опасность поражения электрическим током

Использование электрического оборудования необходимо на любом производстве, где ведется разработка программного обеспечения. Однако оно может быть опасным для здоровья работников, если условия в помещении не соответствуют необходимым требованиям. Именно поэтому следует применять современные технологии в сфере безопасности жизнедеятельности, обеспечивающие максимальную защиту здоровью персонала.

С точки зрения электробезопасности выделяют три вида помещений:

– безопасные;

- с повышенной опасностью;
- особо опасные.

Чтобы определить условия, влияющие на опасность электрического оборудования для человека и возможность поражения током, следует разобраться в критериях, определяющих степень опасности помещения.

В безопасных помещения возможность поражения человека током сводится к минимуму.

Безопасные помещения должны соответствовать следующим критериям:

- температура воздуха составляет ниже 30°C, оптимальная температура должна быть от 18°C до 20°C;
- отсутствие токопроводящей пыли;
- процент заполнения площади предметами из металла должен составлять менее 20%;
- полы должны быть выполнены из материала, не проводящего ток;
- влажность воздуха не должна превышать 60%;
- помещение должно хорошо проветриваться.

В помещениях с повышенной опасностью вероятность поражения током выше, чем в безопасных поэтому в них персонал должен быть особенно осторожен при выполнении должностных обязанностей.

Для помещений с повышенной опасностью можно выделить следующие критерии:

- температура воздуха превышает 30°C;
- наличие токопроводящей пыли;
- наличие большого количества металлического оборудования;
- полы выполнены из токопроводящего материала;
- высокая влажность воздуха (70-80%);
- плохо проветриваемое помещение;

В особо опасных помещениях вероятность поражения током высокая, поэтому помимо техники безопасности, руководство должно обеспечить персонал специальным обмундированием, способным их защитить.

К особо опасным помещениям относятся следующие критерии:

- температура воздуха превышает 30-35°C;
- пожароопасные помещения, в которых производятся либо хранятся легковоспламеняющиеся вещества или материалы;
- взрывоопасные помещения;
- помещения с едкими парами или газами;
- влажность воздуха составляет 100%.

2.2.2 Пожарная безопасность

Применение электрического оборудования повышает не только риск поражения электрическим током, но и опасность возникновения возгорания или взрыва.

Помещения по степени пожарной и взрывопожарной опасности разделяют на 5 категорий:

- категория «А», объекты с повышенной пожарной активностью. К ней относятся помещения, в которых могут храниться опасные смеси и легко воспламеняющиеся составы с температурой возгорания не более 28°C;
- категория «Б», взрывопожароопасные объекты. К таким помещениям относятся комнаты, в которых хранится все то же самое, только с температурой возгорания более 28°C;
- категории «В1-В4». В нее включаются объекты, в которых разрешается хранить слабо горючие и горючие жидкости, а также твердые вещества, которые при определенных условиях поддерживают свободное горение;
- категория «Г». Допускается применять и содержать смеси и вещества, которые не поддерживают горение. При этом смеси разрешается хранить в разогретом или расплавленном состоянии. В процессе их обработки может

выделяться значительное количество искровых образований, которые в свою очередь провоцируют очаги возгорания;

– категория «Д». К ней принято относить помещения с пониженной пожарной опасностью. В них разрешено хранить негорючие материалы в не разогретом состоянии.

2.3 Санитарно-гигиенические требования к производству

С целью обеспечения благоприятных условий труда для персонала, руководство производства должно соблюдать установленные государственными органами санитарные нормы.

Прежде всего, производственные помещения должны быть теплыми, светлыми и сухими. Выделяют следующие основные требования:

- на каждого рабочего должно быть выделено не менее 15 м³;
- высота дверей и проходов должна составлять не менее 2 м;
- высота помещений не менее 3,2 м;
- расстояние от пола до выступающих конструктивных элементов – не менее 2,6 м;
- ширина дверей должна быть более 0,8 м;
- ширина коридоров должна быть более 1,4 м.

При проектировании помещения, необходимо расположить окна таким образом, чтобы освещалось каждое рабочее место. При этом каждое окно должно открываться. Световые фонари делают из армированного стекла, при применении простого стекла, необходимо под фонарями подвесить металлические сетки.

Температура в помещении чаще всего должна составлять 18-21°C, при относительной влажности 60-40% и скорости воздуха до 0,2 м/с.

Также руководство производства должно обеспечить каждое помещения питьевой водой температурой не менее 8°C и не более 20°C. Периодичность замены воды – 1 сутки.

					ПГУ 2.090301.39.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

Большую роль для сохранения здоровья человека на производстве играет освещенность рабочих мест. Для максимально эффективного использования естественного освещения необходимо очищать окна от пыли и грязи, а также правильно расставлять оборудование.

Помещения с повышенным уровнем шума должны быть обозначены соответствующими знаками. Также руководство обязано осуществлять контроль уровня шума на всех рабочих местах и определять правила безопасной работы в шумных условиях.

2.4 Расчет пожарной нагрузки в помещениях категории В1-В4

Расчет пожарной нагрузки в помещениях категории В1-В4 регулируется сводом правил 12.13130.2009 от 01.05.2009.

Пожарная нагрузка Q , МДж, определяется по формуле, изображенной ниже.

$$Q = \sum_{i=1}^n G_i + Q_{Hi}^p, \quad (1)$$

где G_i – количество i -го материала пожарной нагрузки, кг;

Q_{hi}^p – низшая теплота сгорания i -го материала пожарной нагрузки, МДж·кг⁻¹,

n – количество материалов.

Удельная пожарная нагрузка g , МДж·м⁻² определяется из соотношения $g = Q/S$, где S – площадь размещения пожарной нагрузки, м² (но не менее 10 м²).

Далее приведена таблица соответствия категорий В1-В4 к величине пожарной нагрузки.

Таблица 3 - Категории пожарной опасности помещения

№	Категория помещения	Удельная пожарная нагрузка g на участке, МДж·м ⁻²
1	В1	Более 2200
2	В2	1401-2200

№	Категория помещения	Удельная пожарная нагрузка g на участке, МДж·м ⁻²
3	B3	181-1400
4	B4	1-180

Если при определении категорий B2 или B3 количество пожарной нагрузки Q , определенное по формуле (1), отвечает неравенству $Q \geq 0,64 \cdot g_i \cdot H^2$, то помещение будет относиться к категориям B1 или B2 соответственно.

Ниже приведен список предметов рабочего помещения, их количество, вес, материал из которого они сделаны, значения низшей теплоты сгорания материалов.

Таблица 4 - Предметы рабочего помещения

№	Предмет	Количество	Вес одной единицы, кг.	Материал (с низшей теплотой сгорания)	Значение низшей теплоты сгорания, МДж/кг
1	Стол	5	25	Древесина	13,8
2	Стул	7	3	Древесина	13,8
3	Линолеум	20 м ²	3	Линолеум на тканной основе	20,292
4	Компьютер	5	15	Полистирол	39
5	Шкаф	2	40	Древесина	13,8
6	Рама оконная	3	35	Поливинилхлорид	14
7	Тумба	6	10	Древесина	13,8

Далее необходимо произвести расчет пожарной нагрузки:

$$Q = (5 \cdot 25 + 7 \cdot 3 + 2 \cdot 40 + 6 \cdot 10) \cdot 13,8 + 20 \cdot 3 \cdot 20,292 + 5 \cdot 15 \cdot 39 + 3 \cdot 35 \cdot 14 = 9559,32 \text{ МДж.}$$

Тогда удельная пожарная нагрузка равна:

					ПГУ 2.090301.39.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

$$g = Q/S = 9559,32 / 20 = 477,966 \text{ МДж} \cdot \text{м}^{-2}$$

Полученное значение соответствует категории В3.

Далее необходимо проверить неравенство

$$Q >= 0,64 * g_t * H^2.$$

$$Q = 9559,32.$$

$$0,64 * g_t * H^2 = 0,64 * 477,966 * 3^2 = 2753,08416.$$

$$9553,32 > 2753,08416.$$

Неравенство является верным, из этого можно сделать вывод, что помещение принадлежит к категории пожарной опасности В2. Так как в помещении находится компьютерное оборудование, то для пожаротушения следует применять огнетушитель углекислотный марки ОУ-5.

					ПГУ 2.090301.39.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

Поставленные для выполнения ВКР задачи были полностью решены. Были разработаны библиотеки форматирования, фильтрации и сортировки данных для табличного процессора. Также для тестирования и демонстрации работы был реализован графический интерфейс. Разработанный модуль форматирования данных был интегрирован в редактор электронных таблиц «Ведомость», разработкой которого занимается ООО НТП «Криптософт».

В ходе выполнения работы были улучшены навыки разработки на языке C# и разработки приложений *WinForms*. Также был получен новый опыт в объектно-ориентированном программировании и проектировании архитектур.

Были получены новые знания об основах разработки офисных приложений и стандарте *Open Office XML*, а также проектировании пользовательских элементов управления и применении их в графическом интерфейсе.

В дальнейшем разработанные программные средства можно улучшить, путем добавления поддержки различных языков в модуль форматирования данных, реализовать большой набор стандартных форматов и более широкий набор фильтров.

Список использованных источников

1. Коды числовых форматов: [Электронный ресурс]. URL: <https://support.microsoft.com/ru-ru/office/коды-числовых-форматов-5026bbd6-04bc-48cd-bf33-80f18b4eae68>.
2. ISO/IEC 29500-1:2016. Information technology — Document description and processing languages — Office Open XML FileFormats — Part 1: Fundamentals and Markup Language Reference, 2016.
3. Spreadsheet Styles: [Электронный ресурс]. URL: <http://officeopenxml.com/SSstyles.php>.
4. Документация по .NET Framework: [Электронный ресурс]. URL: <https://docs.microsoft.com/en-US/dotnet/framework>.
5. Руководство по рабочему столу .NET для Windows Forms: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/?view=netframeworkdesktop-4.8>.
6. Документация по C#: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>.

Приложение А

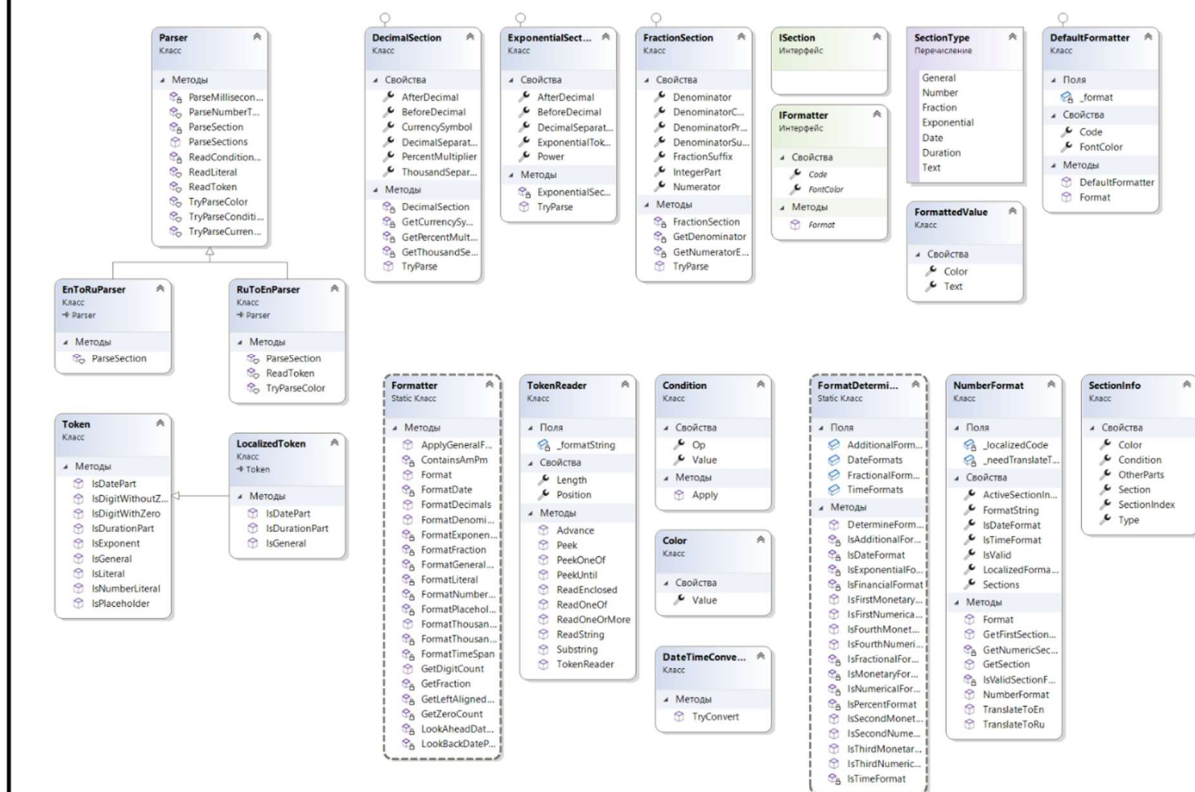
(Обязательное)

Программные средства форматирования, фильтрации и сортировки данных для
табличного процессора

UML-диаграммы

					ПГУ 2.090301.39.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 - Библиотека форматирования данных. Диаграмма классов



ПГУ 1.09.03.01.39.002 Д11				
Изм.	Лист	№ докум.	Подпись	Дата
Разраб.	Стреловой Д.Д.	Дорошенко И.Н.		
Провер.				
Т. контр.				
Реценз.				
Н. контр.	Белов А.С.			
Утверд.	Митрохин М.А.			
Программные средства форматирования, фильтрации и сортировки данных для табличного процессора. УМЛ-диаграмма классов библиотеки форматирования данных				
Лист		Масштаб	Лист 1	
Лист 1		Листов 1	ФБТ, зр. 18882	

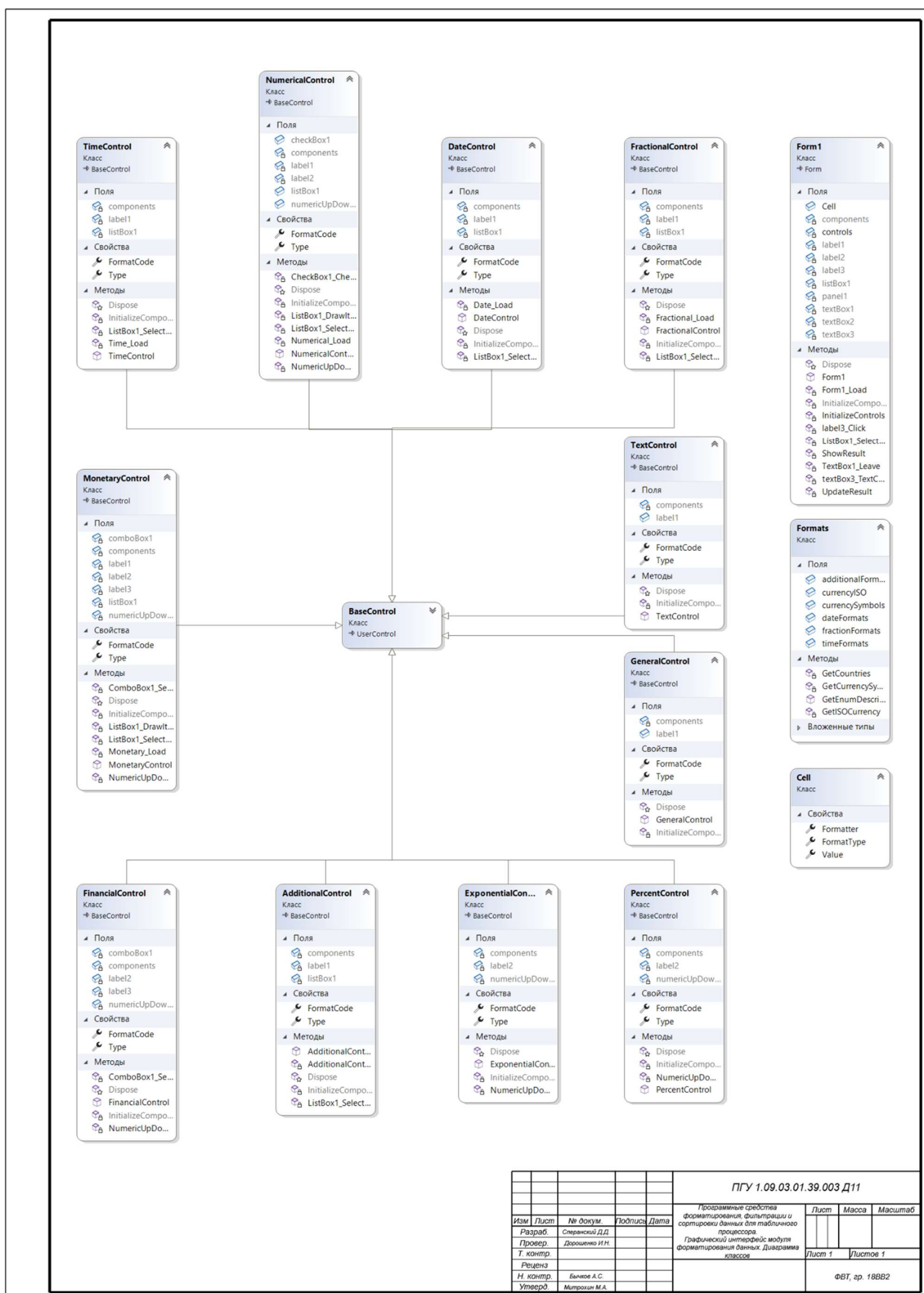
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

69

Приложение А.2 - Графический интерфейс модуля форматирования данных. Диаграмма классов



					ПГУ 1.09.03.01.39.003 Д11				
					Программные средства форматирования, фильтрации и сортировки данных для табличного процессора				
					Графический интерфейс модуля форматирования данных. Диаграмма классов				
Изм.	Лист	№ докум.	Подпись	Дата			Лист	Масштаб	Масштаб
Разраб.		Степанов Д.Д.					Лист 1	Листов 1	
Провер.		Дорошенко И.И.							
Т. контр.									
Реценз.									
Н. контр.		Белые А.С.							
Утверд.		Митрохин М.А.							
					ФБТ, зр. 18882				

Изм.	Лист	№ докум.	Подпись	Дата

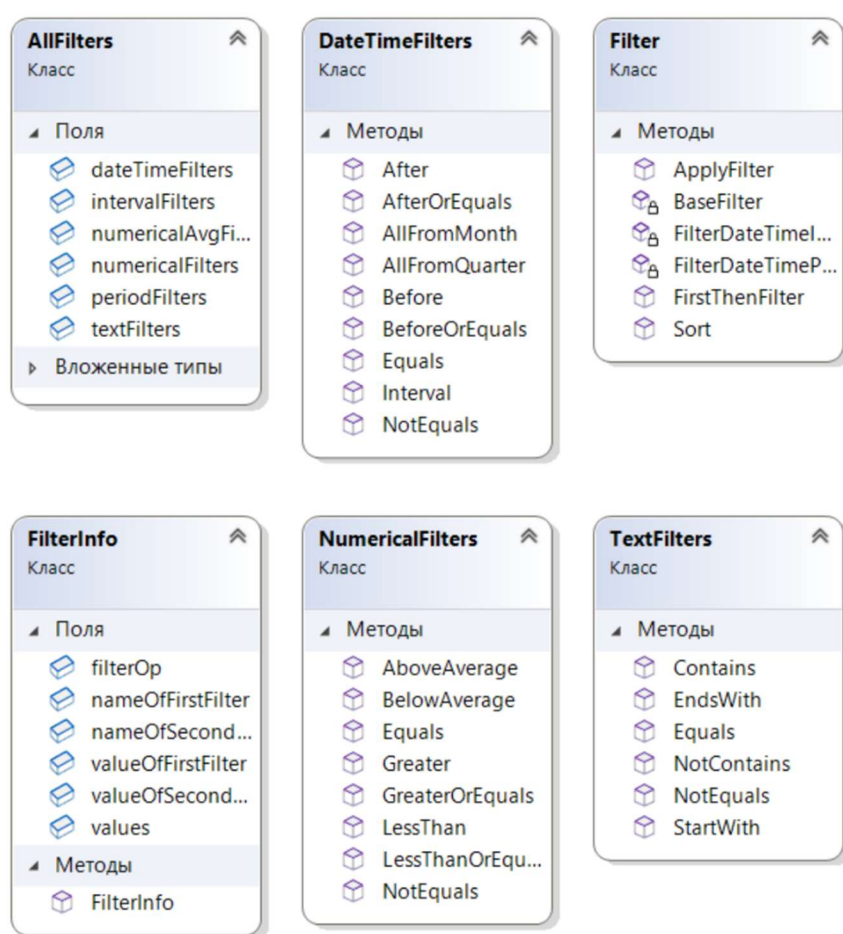
ПГУ 2.090301.39.001 ПЗ

Лист

70

Приложение А.3 - Библиотека фильтрации и сортировки данных.

Диаграмма классов



					ПГУ 1.09.03.01.39.004 Д11			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства форматирования, фильтрации и сортировки данных для табличного процессора. Библиотека фильтрации и сортировки данных. Диаграмма классов	Лист	Масштаб	Масштаб
Разраб.		Степанов Д.Д.						
Провер.		Дорошенко И.Н.						
Т. контр.						Лист 1	Листов 1	
Реценз.								
Н. контр.		Белов А.С.						
Утверд.		Митрохин М.А.						ФБТ, гр. 18882

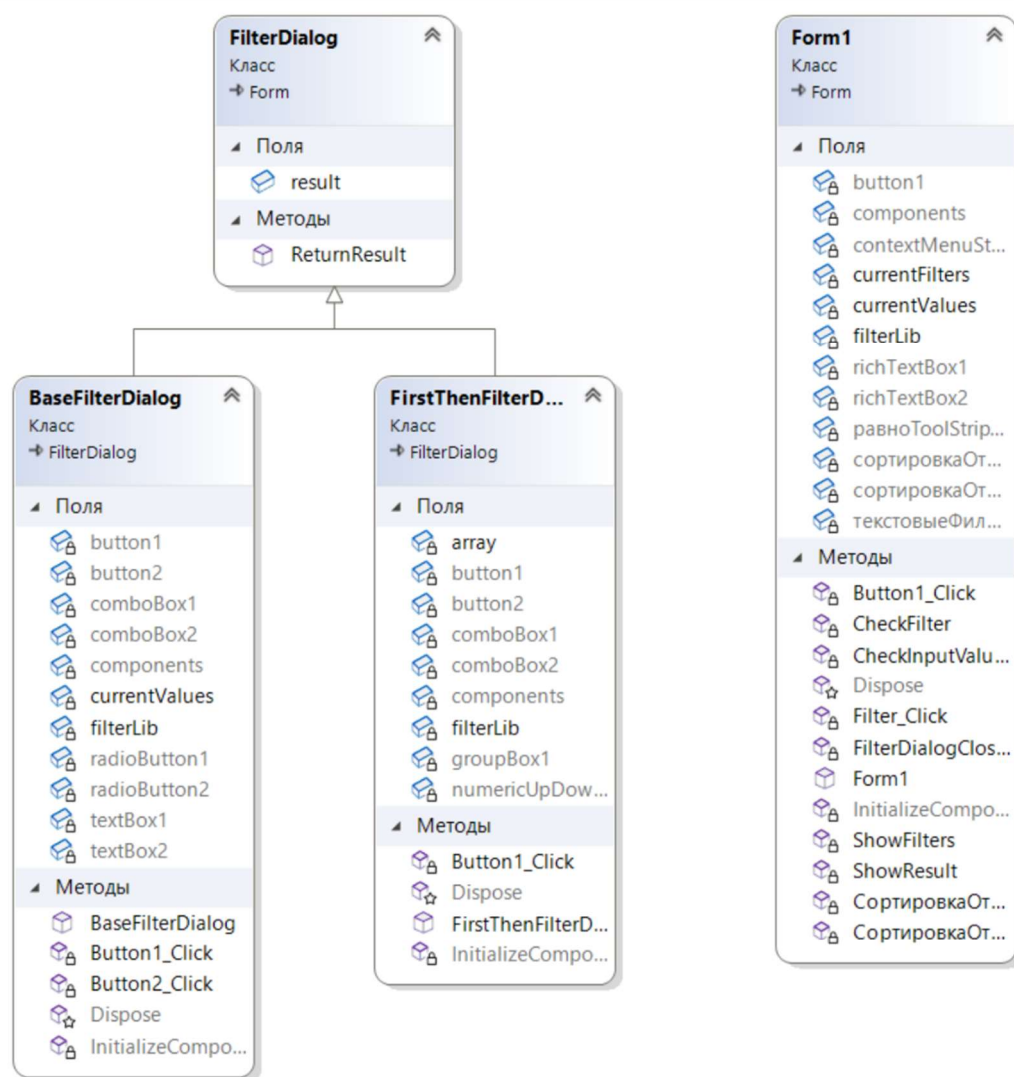
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

71

Приложение А.4 - Графический интерфейс модуля фильтрации и сортировки данных. Диаграмма классов



					ПГУ 1.09.03.01.39.005 Д11			
					Программные средства форматирования, фильтрации и сортировки данных для табличного процессора			
Изм.	Лист	№ докум.	Подпись	Дата	Лист			
Разраб.		Степанов Д.Д.			Масса			
Провер.		Дерюченко И.Н.			Масштаб			
Т. контр.					Лист 1			
Реценз.					Листов 1			
Н. контр.		Беляков А.С.			ФБТ, зр. 18882			
Утверд.		Митрохин М.А.						

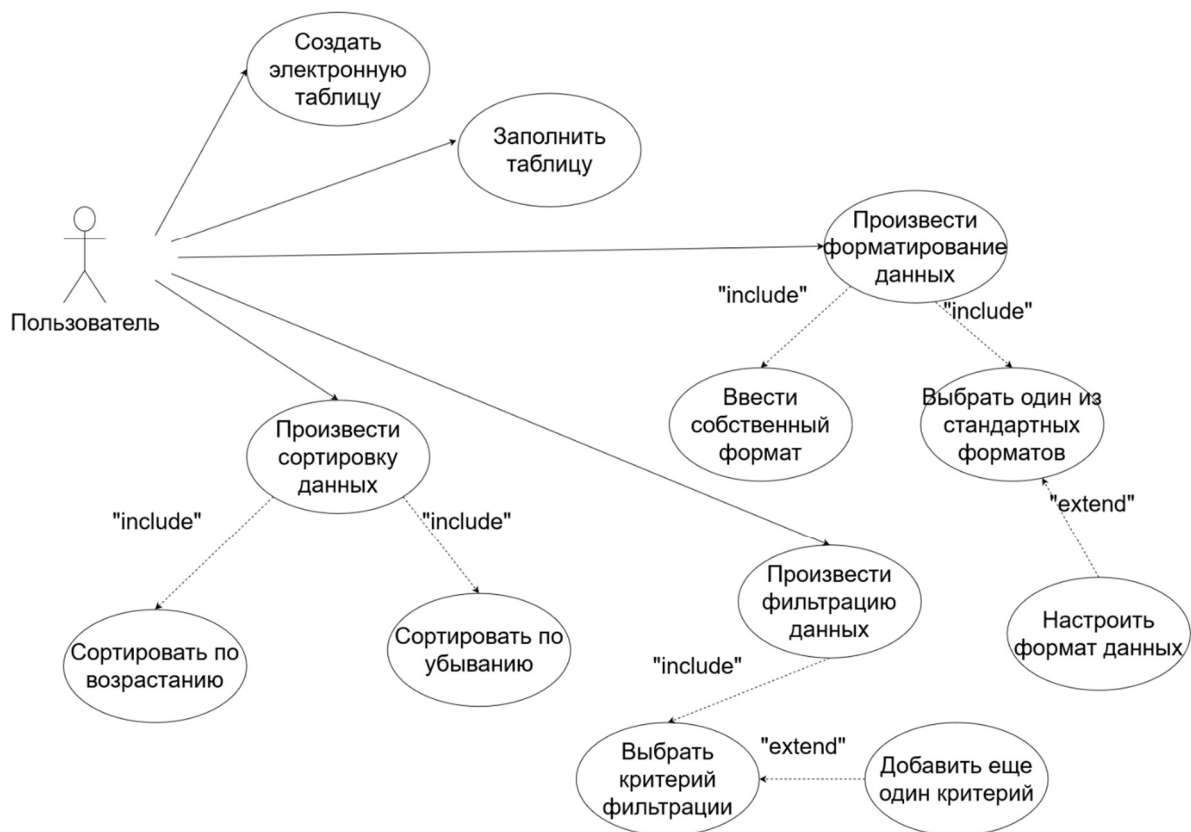
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

72

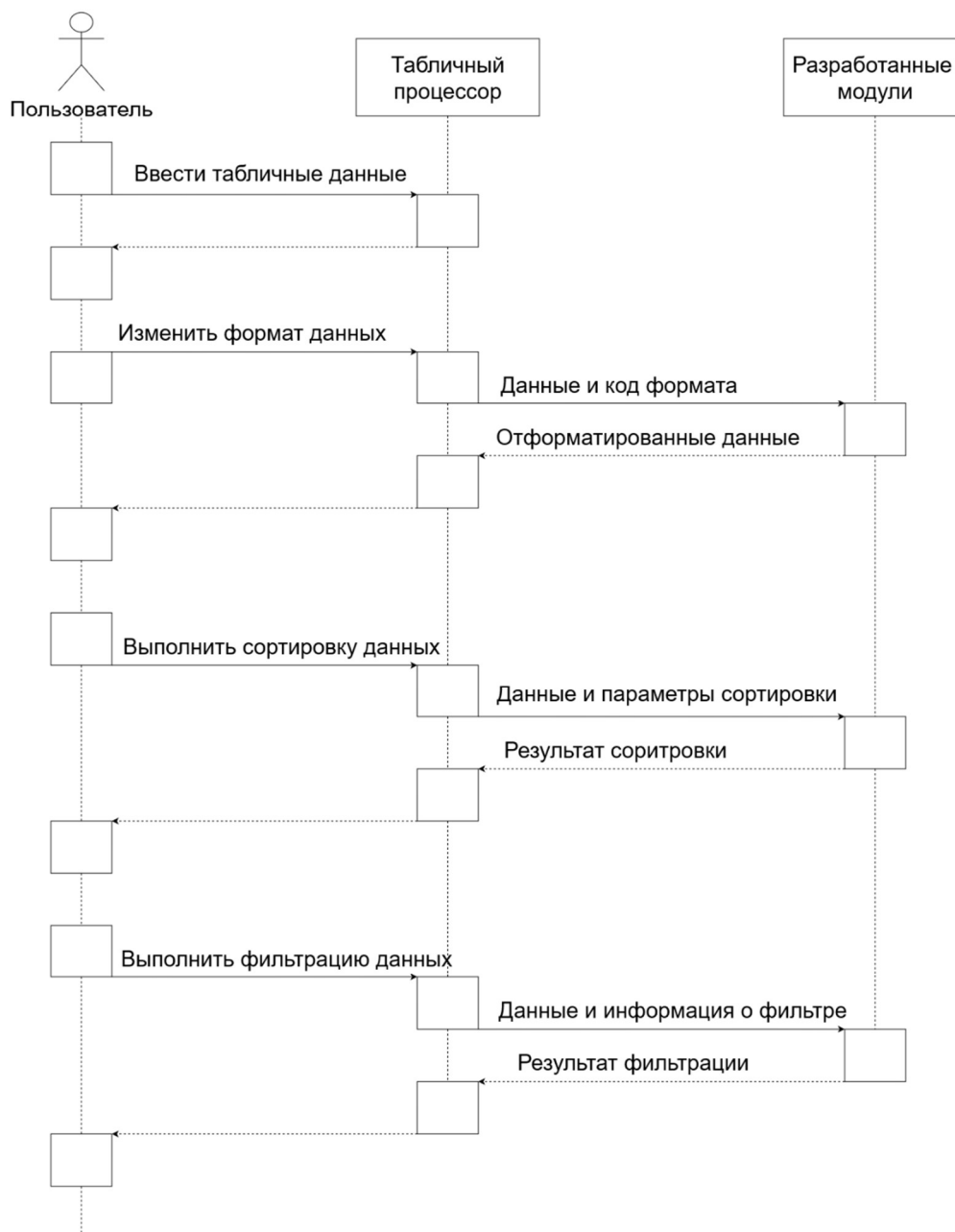
Приложение А.5 - Диаграмма прецедентов



					ПГУ 1.09.03.01.39.006 Д13			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства форматирования, фильтрации и сортировки данных для табличного процессора. Диаграмма прецедентов	Лист	Масса	Масштаб
Разраб.		Степанов Д.Д.						
Провер.		Дерюгина И.И.				Лист 1	Листов 1	
Т. контр.								
Реценз.								
Н. контр.		Беляков А.С.						
Утверд.		Митрохин М.А.						ФБТ, зр. 188В2

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.6 - Диаграмма последовательностей



					ПГУ 1.09.03.01.39.007 Д14			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства форматирования, фильтрации и сортировки данных для табличного процессора. Диаграмма последовательностей	Лист	Масштаб	Масштаб
Разраб.		Степанов Д.Д.						
Провер.		Дерюченко И.Н.						
Т. контр.						Лист 1	Листов 1	
Реценз.								
Н. контр.		Беляков А.С.						
Утверд.		Митрохин М.А.						ФБТ, зр. 188В2

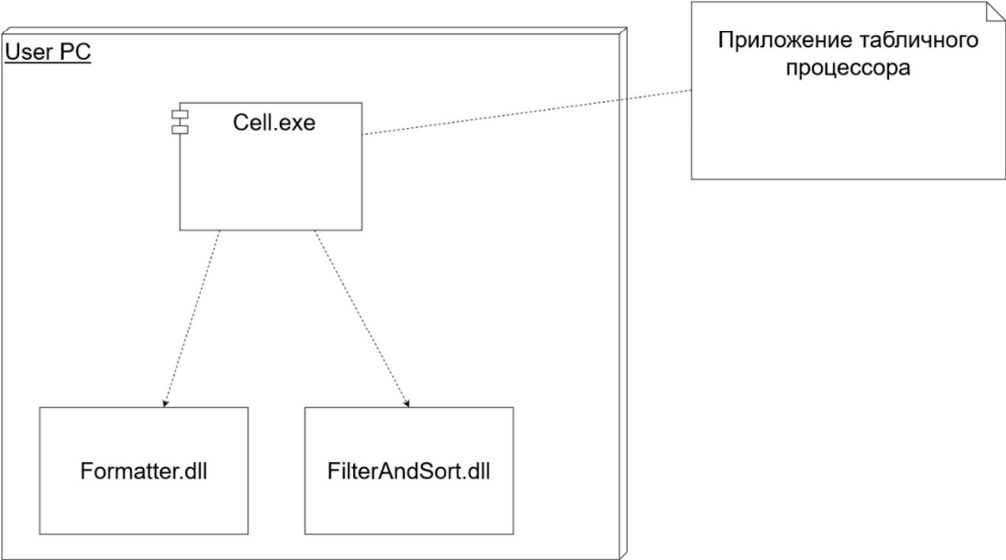
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

74

Приложение А.7 - Диаграмма развертывания



						ПГУ 1.09.03.01.39.008 Д19			
Изм.	Лист	№ докум.	Подпись	Дата		Программные средства форматирования, фильтрации и сортировки данных для табличного процессора. Диаграмма развертывания	Лист	Масса	Масштаб
Разраб.		Степанов Д.Д.							
Провер.		Дерюченко И.Н.							
Т. контр.							Лист 1	Листов 1	
Реценз.									
Н. контр.		Беляков А.С.							
Утверд.		Митрохин М.А.							ФБТ, зр. 188В2

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(Обязательное)

Программные средства форматирования, фильтрации и сортировки данных для
табличного процессора

Листинг программы

					ПГУ 2.090301.39.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.1 - Модуль форматирования данных

Файл Color.cs

```
namespace FormatterLib
{
    public class Color
    {
        public string Value { get; set; }
    }
}
```

Файл Condition.cs

```
namespace FormatterLib
{
    public class Condition
    {
        public string Op { get; set; }
        public double Value { get; set; }
        public bool Apply(double lhs)
        {
            switch (Op)
            {
                case "<":
                    return lhs < Value;
                case "<=":
                    return lhs <= Value;
                case ">":
                    return lhs > Value;
                case ">=":
                    return lhs >= Value;
                case "<>":
                    return lhs != Value;
                case "=":
                    return lhs == Value;
            }
            return false;
        }
    }
}
```

Файл DateTimeConverter.cs

```
using System;
namespace FormatterLib
{
    public class DateTimeConverter
    {
    }
```



```

        public static bool TryConvert(object value, out
DateTime result)
        {
            switch (value)
            {
                case double doubleValue:
                    result =
DateTime.FromOADate(doubleValue);
                    return true;
                case int intValue:
                    result = DateTime.FromOADate(intValue);
                    return true;
                case short shortValue:
                    result = DateTime.FromOADate(shortValue);
                    return true;
                case DateTime dateTimeValue:
                    result = dateTimeValue;
                    return true;
                case string stringValue:
                    if (DateTime.TryParse(stringValue, out
result))
                    {
                        return true;
                    }
                    result = DateTime.MinValue;
                    return false;
                default:
                    result = DateTime.MinValue;
                    return false;
            }
        }
    }
}

```

Файл DecimalSection.cs

```

using System.Collections.Generic;
using System.Linq;

namespace FormatterLib
{
    public class DecimalSection : ISection
    {
        public List<string> BeforeDecimal { get; }
        public bool DecimalSeparator { get; }
        public List<string> AfterDecimal { get; }
        public double PercentMultiplier { get; }
        public string CurrencySymbol { get; }
        public bool ThousandSeparator { get; }
    }
}

```

```

        private DecimalSection(List<string> tokens,
List<string> beforeDecimal, bool decimalSeparator, List<string>
afterDecimal)
        {
            BeforeDecimal = beforeDecimal;
            DecimalSeparator = decimalSeparator;
            AfterDecimal = afterDecimal;
            PercentMultiplier = GetPercentMultiplier(tokens);
            ThousandSeparator = GetThousandSeparator(tokens);
            CurrencySymbol = GetCurrencySymbol(tokens);
        }

        private double
GetPercentMultiplier(IEnumerable<string> tokens)
        {
            return tokens.Any(token => token == "%") ? 100 :
1;
        }

        private string GetCurrencySymbol(List<string> tokens)
        {
            foreach (var token in tokens.Where(token => token
== "\"$\"" || token == "\"€\"" || token == "\"£\"" || token ==
 "\"₽\""))
            {
                return token.Replace("\"", "");
            }

            return string.Empty;
        }

        private bool
GetThousandSeparator(IReadOnlyList<string> tokens)
        {
            var hasLastPlaceholder = false;

            for (var j = 0; j < tokens.Count; j++)
            {
                var tokenIndex = tokens.Count - 1 - j;
                var token = tokens[tokenIndex];

                if (hasLastPlaceholder)
                {
                    if (token != ",") continue;

                    return true;
                }
            }

            return false;
        }

```

```

        public static bool TryParse(List<string> tokens, out
ISection format)
        {
            if (Parser.ParseNumberTokens(tokens, 0, out var
beforeDecimal, out var decimalSeparator, out var afterDecimal) ==
tokens.Count)
            {
                format = new DecimalSection(tokens,
beforeDecimal, decimalSeparator, afterDecimal);
                return true;
            }

            format = null;
            return false;
        }
    }
}

```

Файл DefaultFormatter.cs

```

using System.Globalization;

namespace FormatterLib
{
    public class DefaultFormatter : IFormatter
    {
        public string Code { get; }
        public Color FontColor { get; set; }

        private readonly NumberFormat _format;
        public DefaultFormatter(string formatCode, bool
isRuCode = false)
        {
            Code = formatCode;
            _format = new NumberFormat(formatCode, isRuCode);
        }

        public string Format(string cellValue)
        {
            var res = _format.Format(cellValue,
CultureInfo.CurrentCulture);
            FontColor = res.Color;
            return res.Text;
        }
    }
}

```

Файл EnToRuParser.cs

```

using System.Collections.Generic;

```

```

namespace FormatterLib
{
    internal class EnToRuParser : Parser
    {
        internal static List<string> ParseSection(TokenReader
reader, out bool syntaxError)
        {
            string token;
            var tokens = new List<string>();
            var hasHoursTokens = false;
            var lastDateToken = '\0';

            while ((token = ReadToken(reader, out
syntaxError)) != null)
            {

                if (token == ";")
                    break;

                Token.IsPlaceholder(token);

                if (Token.IsDatePart(token))
                {
                    var translatedToken = string.Empty;
                    Token.IsDurationPart(token);

                    foreach (var c in token.ToLower())
                    {
                        switch (c)
                        {
                            case 'y':
                                translatedToken += "г";
                                break;
                            case 'm':
                                if (hasHoursTokens)
                                {
                                    translatedToken += "М";
                                }
                                else
                                {
                                    translatedToken += "М";
                                }
                                break;
                            case 'd':
                                translatedToken += "д";
                                break;
                            case 'h':
                                translatedToken += "ч";

                                hasHoursTokens = true;
                                break;
                            case 's':
                                translatedToken += "с";

```

					ПГУ 2.090301.39.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

```

        if (lastDateToken == 'm')
        {
            for (int i = tokens.Count
- 1; i >= 0; i--)
            {
                if
                (!tokens[i].Contains("M")) continue;

                tokens[i] =
tokens[i].Replace("M", "М");

                break;
            }
        }
        break;
    default:
        translatedToken += c;
        break;
    }

    lastDateToken = c;

}
tokens.Add(translatedToken);
}
else if (Token.IsGeneral(token))
{
    token =
token.ToLower().Replace("general", "основной");
    tokens.Add(token);
}
else if (token == "@")
{
    tokens.Add(token);
}
else if (token.StartsWith("["))
{
    var expression = token.Substring(1,
token.Length - 2);
    if (TryParseCondition(expression, out var
parseCondition))
tokens.Add($"[{parseCondition.Op}{parseCondition.Value}]");

    else if (TryParseColor(expression, out
var parseColor))
    {
        var localizedColor = string.Empty;
        switch (parseColor.Value.ToLower())
        {
            case "black":
                localizedColor = "Черный";

```

```

        break;
    case "blue":
        localizedColor = "Синий";
        break;
    case "cyan":
        localizedColor = "Голубой";
        break;
    case "green":
        localizedColor = "Зеленый";
        break;
    case "magenta":
        localizedColor =
"Фиолетовый";
        break;
    case "red":
        localizedColor = "Красный";
        break;
    case "white":
        localizedColor = "Белый";
        break;
    case "yellow":
        localizedColor = "Желтый";
        break;
    }
    tokens.Add($"[{localizedColor}]");
}

else if
(TryParseCurrencySymbol(expression, out var parseCurrencySymbol))
    tokens.Add("\"" + parseCurrencySymbol
+ "\"");
    }
else
{
    tokens.Add(token);
}
}

if (syntaxError || tokens.Count == 0)
{
    return null;
}

return tokens;
}
}
}

```

Файл ExponentialSection.cs

```
using System.Collections.Generic;
```

					ПГУ 2.090301.39.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

namespace FormatterLib
{
    public class ExponentialSection : ISection
    {
        public List<string> BeforeDecimal { get; }
        public bool DecimalSeparator { get; }
        public List<string> AfterDecimal { get; }
        public string ExponentialToken { get; }
        public List<string> Power { get; }

        private ExponentialSection(List<string>
beforeDecimal, bool decimalSeparator, List<string> afterDecimal,
string exponentialToken, List<string> power)
        {
            BeforeDecimal = beforeDecimal;
            DecimalSeparator = decimalSeparator;
            AfterDecimal = afterDecimal;
            ExponentialToken = exponentialToken;
            Power = power;
        }

        public static bool TryParse(List<string> tokens, out
ExponentialSection format)
        {
            var position = Parser.ParseNumberTokens(tokens,
0, out var beforeDecimal, out var decimalSeparator, out var
afterDecimal);

            if (position != 0 && position < tokens.Count &&
Token.IsExponent(tokens[position]))
            {
                var exponentialToken = tokens[position];
                position++;
                format = new
ExponentialSection(beforeDecimal, decimalSeparator, afterDecimal,
exponentialToken, tokens.GetRange(position, tokens.Count -
position));

                return true;
            }

            format = null;
            return false;
        }
    }
}

```

Файл FormatDeterminant.cs

```

using System.Linq;
using System.Text.RegularExpressions;

```

					ПГУ 2.090301.39.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

```

namespace FormatterLib
{
    public static class FormatDeterminant
    {
        public static readonly string[] DateFormats =
        {
            "d/m;@", "d/m/yy;@", "dd/mm/yy;@", "d\\ mmm;@",
            "d\\ mmm\\ yy;@", "dd\\ mmm\\ yy;@", "yyyy\\,\\ mmmm;@", "mmmm\\
            yyyy;@",
            "dd\\ mmmm\\ yyyy\\ \\r\\.;@", "dd/mm/yy\\ h:mm\\
            AM/PM;@", "dd/mm/yy\\ h:mm;@", "mmmm;@", "yyyy\\,\\ dd\\ mmmm;@",
            "d/m/yyyy;@", "d\\-mmm\\-yyyy;@"
        };

        public static readonly string[] TimeFormats =
        {
            "h:mm;@", "h:mm\\ AM/PM;@", "h:mm:ss;@",
            "h:mm:ss\\ AM/PM;@", "mm:ss.0;@", "[h]:mm:ss;@",
            "dd/mm/yy\\ h:mm\\ AM/PM;@", "dd/mm/yy\\ h:mm;@",
        };

        public static readonly string[] FractionalFormats =
        {
            "#\" \"?/?", "#\" \"??/??", "#\" \"???/???",
            "#\" \"?/2", "#\" \"?/4", "#\" \"?/8", "#\" \"??/16", "#\"
            \"?/10", "#\" \"??/100\";
        };

        public static readonly string[] AdditionalFormats =
        {
            "000000", "00000\\-0000", "[<=9999999]###\\-
            ####;\\(###\\)\\ ###\\-####", "0000\";
        };

        public static int DetermineFormat(string formatCode)
        {
            if (formatCode == "General")
            {
                return 0;
            }

            if (IsNumericalFormat(formatCode))
            {
                return 1;
            }

            if (IsMonetaryFormat(formatCode))
            {
                return 2;
            }

            if (IsFinancialFormat(formatCode))
            {
                return 3;
            }
        }
    }
}

```



```

        if (IsDateFormat(formatCode))
        {
            return 4;
        }

        if (IsTimeFormat(formatCode))
        {
            return 5;
        }

        if (IsPercentFormat(formatCode))
        {
            return 6;
        }

        if (IsFractionalFormat(formatCode))
        {
            return 7;
        }

        if (IsExponentialFormat(formatCode))
        {
            return 8;
        }

        if (formatCode == "@")
        {
            return 9;
        }

        if (IsAdditionalFormat(formatCode))
        {
            return 10;
        }

        return 11;
    }

    private static bool IsNumericalFormat(string
formatCode)
    {
        return IsFirstNumericalFormat(formatCode) ||
IsSecondNumericalFormat(formatCode) ||
                IsThirdNumericalFormat(formatCode) ||
IsFourthNumericalFormat(formatCode);
    }

    private static bool IsMonetaryFormat(string
formatCode)
    {
        return IsFirstMonetaryFormat(formatCode) ||
IsSecondMonetaryFormat(formatCode) ||

```

```

        IsThirdMonetaryFormat(formatCode) ||
IsFourthMonetaryFormat(formatCode);
    }

    private static bool IsFinancialFormat(string
formatCode)
    {
        return new Regex("^_-\*\ #,##0\\.0*(\\|\\|\\|)?
(\\|)?_?.(\\|)?_-(\\|\\|\\|)?-\\*\ #,##0.0*(\\|\\|\\|)? (\\|)?_?.(\\|)?_-(\\|\\|\\|)?
\\|"-\\|(\\|\\|\\|)?*(\\|\\|\\|)? (\\|)?_?.(\\|)?_-(\\|\\|\\|)?_@_-$").IsMatch(formatCode);
    }

    private static bool IsDateFormat(string formatCode)
    {
        return DateFormats.Contains(new
Regex("\\[\\.\\*\\|\\]").Replace(formatCode, string.Empty));
    }

    private static bool IsTimeFormat(string formatCode)
    {
        return TimeFormats.Contains(new
Regex("\\[\\|\\$.\\*\\|\\]").Replace(formatCode, string.Empty));
    }

    private static bool IsPercentFormat(string
formatCode)
    {
        return new
Regex("^0\\.0*%$").IsMatch(formatCode);
    }

    private static bool IsFractionalFormat(string
formatCode)
    {
        return FractionalFormats.Contains(formatCode);
    }

    private static bool IsExponentialFormat(string
formatCode)
    {
        return new Regex("^0\\.0*E\\|\\|+00$",
RegexOptions.IgnoreCase).IsMatch(formatCode);
    }

    private static bool IsAdditionalFormat(string
formatCode)
    {
        return AdditionalFormats.Contains(formatCode);
    }

    public static bool IsFirstNumericalFormat(string
formatCode)
    {

```

```

        return new
Regex("(#,#)?0\\.0*$").IsMatch(formatCode);
    }

    public static bool IsSecondNumericalFormat(string
formatCode)
    {
        return new
Regex("(#,#)?0\\.0*;\[Red\\](#,#)?0\\.0*$",
RegexOptions.IgnoreCase).IsMatch(formatCode);
    }

    public static bool IsThirdNumericalFormat(string
formatCode)
    {
        return new Regex("(#,#)?0\\.0*_ ?;\[Red\\]?(#,#)?0\\.0*\[Red\\]? ?$").IsMatch(formatCode);
    }

    public static bool IsFourthNumericalFormat(string
formatCode)
    {
        return new Regex("(#,#)?0\\.0*_ ?;\[Red\\]\[Red\\]?(#,#)?0\\.0*\[Red\\]? ?$",
RegexOptions.IgnoreCase).IsMatch(formatCode);
    }

    public static bool IsFirstMonetaryFormat(string
formatCode)
    {
        return new Regex("^#,#0\\.0*\[Red\\]?_?(\\")?_?.(\\")?$").IsMatch(formatCode);
    }

    public static bool IsSecondMonetaryFormat(string
formatCode)
    {
        return new Regex("^#,#0\\.0*\[Red\\]?_?(\\")?_?.(\\")?;\[Red\\]#,#0\\.0*\[Red\\]?_?(\\")?_?.(\\")?$",
RegexOptions.IgnoreCase).IsMatch(formatCode);
    }

    public static bool IsThirdMonetaryFormat(string
formatCode)
    {
        return new Regex("^#,#0\\.0*\[Red\\]?_?(\\")?_?.(\\")?;\[Red\\]#,#0\\.0*\[Red\\]?_?(\\")?_?.(\\")?$").IsMatch(formatCode);
    }

    public static bool IsFourthMonetaryFormat(string
formatCode)
    {

```

```

        return new Regex("^#,##0\\.0*\\\\\\?
(\\\")?_?.(\\\")?;\\\\[Red\\\\]\\\\\\\\?-#,##0\\.0*\\\\\\\\?
(\\\")?_?.(\\\")?").IsMatch(formatCode);
    }

}

```

Файл FormattedValue.cs

```

namespace FormatterLib
{
    public class FormattedValue
    {
        public string Text { get; set; }
        public Color Color { get; set; }
    }
}

```

Файл Formatter.cs

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
namespace FormatterLib
{
    public static class Formatter
    {
        public static string Format(object value, SectionInfo
sectionInfo, CultureInfo culture)
        {
            switch (sectionInfo.Type)
            {
                case SectionType.Number:
                    var number = Convert.ToDouble(value,
culture);
                    if ((sectionInfo.SectionIndex == 0 &&
sectionInfo.Condition != null) || sectionInfo.SectionIndex == 1)
                        number = Math.Abs(number);
                    if (!(sectionInfo.Section is
DecimalSection decimalSection))
                    {
                        throw new Exception("Invalid
Section");
                    }
                    return FormatNumber(number,
decimalSection, culture);
            }
        }
    }
}

```

```

        case SectionType.Date:
            if (DateTimeConverter.TryConvert(value,
out var conevrtdDateTime))
            {
                return FormatDate(conevrtdDateTime,
sectionInfo.OtherParts, culture);
            }
            else
            {
                throw new FormatException("Unexpected
date value");
            }
        case SectionType.Duration:
            if (value is TimeSpan ts)
            {
                return FormatTimeSpan(ts,
sectionInfo.OtherParts);
            }
            else
            {
                var d = Convert.ToDouble(value);
                return
FormatTimeSpan(TimeSpan.FromDays(d), sectionInfo.OtherParts);
            }
        case SectionType.General:
        case SectionType.Text:
            return
FormatGeneralText(ApplyGeneralFormat(value, culture),
sectionInfo.OtherParts);
        case SectionType.Exponential:
            return
FormatExponential(Convert.ToDouble(value, culture), sectionInfo,
culture);
        case SectionType.Fraction:
            return
FormatFraction(Convert.ToDouble(value, culture), sectionInfo,
culture);
        default:
            throw new
InvalidOperationException("Unknown number format sectionInfo");
    }
}
private static string FormatGeneralText(string text,
List<string> tokens)
{
    var result = new StringBuilder();
    foreach (var token in tokens)
    {
        if (Token.IsGeneral(token) || token == "@")
        {
            result.Append(text);
        }
        else
    }
}

```

```

        {
            FormatLiteral(token, result);
        }
    }
    return result.ToString();
}
private static string FormatTimeSpan(TimeSpan
timeSpan, IReadOnlyList<string> tokens)
{
    var result = new StringBuilder();
    var containsMilliseconds = false;
    for (var i = tokens.Count - 1; i >= 0; i--)
    {
        if (!tokens[i].StartsWith(".0")) continue;
        containsMilliseconds = true;
        break;
    }
    foreach (var token in tokens)
    {
        if (token.StartsWith("m",
StringComparison.OrdinalIgnoreCase))
        {
            var value = timeSpan.Minutes;
            var digits = token.Length;
            result.Append(value.ToString("D" +
digits));
        }
        else if (token.StartsWith("s",
StringComparison.OrdinalIgnoreCase))
        {
            var formatMs = containsMilliseconds ? 0 :
timeSpan.Milliseconds / 1000D;
            var value =
(int)Math.Round(timeSpan.Seconds + formatMs, 0,
MidpointRounding.AwayFromZero);
            var digits = token.Length;
            result.Append(value.ToString("D" +
digits));
        }
        else if (token.StartsWith("[h",
StringComparison.OrdinalIgnoreCase))
        {
            var value = (int)timeSpan.TotalHours;
            var digits = token.Length - 2;
            result.Append(value.ToString("D" +
digits));
            timeSpan = new TimeSpan(0, 0,
Math.Abs(timeSpan.Minutes), Math.Abs(timeSpan.Seconds),
Math.Abs(timeSpan.Milliseconds));
        }
        else if (token.StartsWith("[m",
StringComparison.OrdinalIgnoreCase))
        {

```

```

        var value = (int)timeSpan.TotalMinutes;
        var digits = token.Length - 2;
        result.Append(value.ToString("D" +
digits));
        TimeSpan = new TimeSpan(0, 0, 0,
Math.Abs(timeSpan.Seconds), Math.Abs(timeSpan.Milliseconds));
    }
    else if (token.StartsWith("[s",
StringComparison.OrdinalIgnoreCase))
    {
        var value = (int)timeSpan.TotalSeconds;
        var digits = token.Length - 2;
        result.Append(value.ToString("D" +
digits));
        TimeSpan = new TimeSpan(0, 0, 0, 0,
Math.Abs(timeSpan.Milliseconds));
    }
    else if (token.StartsWith(".0"))
    {
        var value = timeSpan.Milliseconds;
        var digits = token.Length - 1;
        result.Append("." + value.ToString("D" +
digits));
    }
    else
    {
        FormatLiteral(token, result);
    }
}
return result.ToString();
}
private static string FormatDate(DateTime date,
IEnumerable<string> tokens, CultureInfo culture)
{
    var containsAmPm = ContainsAmPm(tokens);
    var result = new StringBuilder();
    for (var i = 0; i < tokens.Count; i++)
    {
        var token = tokens[i];
        if (token.StartsWith("y",
StringComparison.OrdinalIgnoreCase))
        {
            // year
            var digits = token.Length;
            if (digits < 2)
                digits = 2;
            if (digits == 3)
                digits = 4;
            var year = date.Year;
            if (digits == 2)
                year = year % 100;
            result.Append(year.ToString("D" +
digits));

```

```

        }
        else if (token.StartsWith("m",
StringComparison.OrdinalIgnoreCase))
        {
            if (LookBackDatePart(tokens, i - 1, "h")
|| LookAheadDatePart(tokens, i + 1, "s"))
            {
                var digits = token.Length;

result.Append(date.Minute.ToString("D" + digits));
            }
            else
            {
                var digits = token.Length;
                switch (digits)
                {
                    case 3:

result.Append(culture.DateTimeFormat.AbbreviatedMonthNames[date.Mo
nth - 1]);

                    break;
                    case 4:

result.Append(culture.DateTimeFormat.MonthNames[date.Month - 1]);
                    break;
                    case 5:

result.Append(culture.DateTimeFormat.MonthNames[date.Month -
1][0]);

                    break;
                    default:

result.Append(date.Month.ToString("D" + digits));
                    break;

                }
            }
        }
        else if (token.StartsWith("d",
StringComparison.OrdinalIgnoreCase))
        {
            var digits = token.Length;
            switch (digits)
            {
                case 3:

result.Append(culture.DateTimeFormat.AbbreviatedDayNames[(int)date
.DayOfWeek]);

                    break;
                    case 4:

result.Append(culture.DateTimeFormat.DayNames[(int)date.DayOfWeek]
);

                    break;

```



```

        default:

result.Append(date.Day.ToString("D" + digits));
        break;
    }
    }
    else if (token.StartsWith("h",
StringComparison.OrdinalIgnoreCase))
    {
        var digits = token.Length;
        result.Append(containsAmPm
            ? ((date.Hour + 11) % 12 +
1).ToString("D" + digits)
            : date.Hour.ToString("D" + digits));
    }
    else if (token.StartsWith("s",
StringComparison.OrdinalIgnoreCase))
    {
        var digits = token.Length;
        result.Append(date.Second.ToString("D" +
digits));
    }
    else if (token.StartsWith("g",
StringComparison.OrdinalIgnoreCase))
    {
        var era =
culture.DateTimeFormat.Calendar.GetEra(date);
        var digits = token.Length;
        result.Append(digits < 3
            ?
culture.DateTimeFormat.GetAbbreviatedEraName(era)
            :
culture.DateTimeFormat.GetEraName(era));
    }
    else if (string.Compare(token, "am/pm",
StringComparison.OrdinalIgnoreCase) == 0)
    {
        result.Append(date.ToString("tt",
CultureInfo.InvariantCulture).ToUpperInvariant());
    }
    else if (string.Compare(token, "a/p",
StringComparison.OrdinalIgnoreCase) == 0)
    {
        var ampm = date.ToString("%t",
CultureInfo.InvariantCulture);
        result.Append(char.IsUpper(token[0]) ?
ampm.ToUpperInvariant() : ampm.ToLowerInvariant());
    }
    else if (token.StartsWith(".0"))
    {
        var value = date.Millisecond;
        var digits = token.Length - 1;

```

```

        result.Append(".") + value.ToString("D" +
digits));
    }
    else switch (token)
    {
        case "/":

result.Append(culture.DateTimeFormat.DateSeparator);

        break;
        case ",":
        {
            while (i < tokens.Count - 1
&& tokens[i + 1] == ",")
            {
                i++;
            }

            result.Append(",");
            break;
        }
        default:
            FormatLiteral(token, result);
            break;
    }
}
return result.ToString();
}
private static bool
LookAheadDatePart(IReadOnlyList<string> tokens, int fromIndex,
string startsWith)
{
    for (var i = fromIndex; i < tokens.Count; i++)
    {
        var token = tokens[i];
        if (token.StartsWith(startsWith,
StringComparison.OrdinalIgnoreCase))
            return true;
        if (Token.IsDatePart(token))
            return false;
    }
    return false;
}
private static bool
LookBackDatePart(IReadOnlyList<string> tokens, int fromIndex,
string startsWith)
{
    for (var i = fromIndex; i >= 0; i--)
    {
        var token = tokens[i];
        if (token.StartsWith(startsWith,
StringComparison.OrdinalIgnoreCase))
            return true;

```

```

        if (Token.IsDatePart(token))
            return false;
    }
    return false;
}
private static bool ContainsAmPm(IEnumerable<string>
tokens)
{
    foreach (var token in tokens)
    {
        if (string.Compare(token, "am/pm",
StringComparison.OrdinalIgnoreCase) == 0)
        {
            return true;
        }
        if (string.Compare(token, "a/p",
StringComparison.OrdinalIgnoreCase) == 0)
        {
            return true;
        }
    }
    return false;
}
private static string FormatNumber(double value,
DecimalSection format, CultureInfo culture)
{
    var thousandSeparator = format.ThousandSeparator;
    value *= format.PercentMultiplier;
    var result = new StringBuilder();
    FormatNumber(value, format.BeforeDecimal,
format.DecimalSeparator, format.AfterDecimal, thousandSeparator,
culture, result);
    return result.ToString();
}
private static void FormatNumber(double value,
List<string> beforeDecimal, bool decimalSeparator,
IReadOnlyList<string> afterDecimal, bool thousandSeparator,
CultureInfo culture, StringBuilder result)
{
    var significantDigits = 0;
    if (afterDecimal != null)
        significantDigits =
GetDigitCount(afterDecimal);
    var valueString = Math.Abs(value).ToString("F" +
significantDigits, CultureInfo.InvariantCulture);
    var valueStrings = valueString.Split('.');
    var thousandsString = valueStrings[0];
    var decimalString = valueStrings.Length > 1 ?
valueStrings[1].TrimEnd('0') : "";
    if (value < 0)
    {
        result.Append("-");
    }
}

```

```

        if (beforeDecimal != null)
        {
            FormatThousands(thousandsString,
thousandSeparator, false, beforeDecimal, culture, result);
        }
        if (afterDecimal == null) return;
        if (decimalSeparator)
        {
            result.Append(culture.NumberFormat.NumberDecimalSeparator);
        }
        FormatDecimals(decimalString, afterDecimal,
result);
    }
    public static void FormatThousands(string
valueString, bool thousandSeparator, bool significantZero,
IEnumerable<string> tokens, CultureInfo culture, StringBuilder
result)
    {
        var significant = false;
        var formatDigits = GetDigitCount(tokens);
        valueString = valueString.PadLeft(formatDigits,
'0');

        var tokenIndex = 0;
        for (; tokenIndex < tokens.Count; tokenIndex++)
        {
            var token = tokens[tokenIndex];
            if (Token.IsPlaceholder(token))
                break;
            else
                FormatLiteral(token, result);
        }
        var digitIndex = 0;
        for (; digitIndex < (valueString.Length -
formatDigits); digitIndex++)
        {
            significant = true;
            result.Append(valueString[digitIndex]);
            if (thousandSeparator)
                FormatThousandSeparator(valueString,
digitIndex, culture, result);
        }
        for (; tokenIndex < tokens.Count; ++tokenIndex)
        {
            var token = tokens[tokenIndex];
            if (Token.IsPlaceholder(token))
            {
                var c = valueString[digitIndex];
                if (c != '0' || (significantZero &&
digitIndex == valueString.Length - 1)) significant = true;
                FormatPlaceholder(token, c, significant,
result);
            }
        }
    }

```

```

        if (thousandSeparator && (significant ||
token.Equals("0")))
            FormatThousandSeparator(valueString,
digitIndex, culture, result);
            digitIndex++;
        }
        else
        {
            FormatLiteral(token, result);
        }
    }
}

private static void FormatThousandSeparator(string
valueString, int digit, CultureInfo culture, StringBuilder result)
{
    var positionInTens = valueString.Length - 1 -
digit;
    if (positionInTens > 0 && (positionInTens % 3) ==
0)
    {
        result.Append(culture.NumberFormat.NumberGroupSeparator);
    }
}

public static void FormatDecimals(string valueString,
IEnumerable<string> tokens, StringBuilder result)
{
    var unpaddedDigits = valueString.Length;
    var formatDigits = GetDigitCount(tokens);
    valueString = valueString.PadRight(formatDigits,
'0');

    var valueIndex = 0;
    foreach (var token in tokens)
    {
        if (Token.IsPlaceholder(token))
        {
            var c = valueString[valueIndex];
            var significant = valueIndex <
unpaddedDigits;
            FormatPlaceholder(token, c, significant,
result);
            valueIndex++;
        }
        else
        {
            FormatLiteral(token, result);
        }
    }
}

private static string FormatExponential(double value,
SectionInfo format, CultureInfo culture)
{

```

```

        if (!(format.Section is ExponentialSection
exponentialSection))
        {
            throw new Exception("Invalid section");
        }
        var baseDigits = 0;
        if (exponentialSection.BeforeDecimal != null)
        {
            baseDigits =
GetDigitCount(exponentialSection.BeforeDecimal);
        }
        var exponent =
(int)Math.Floor(Math.Log10(Math.Abs(value)));
        var mantissa = value / Math.Pow(10, exponent);
        var shift = Math.Abs(exponent) % baseDigits;
        if (shift > 0)
        {
            if (exponent < 0)
                shift = (baseDigits - shift);
            mantissa *= Math.Pow(10, shift);
            exponent -= shift;
        }
        var result = new StringBuilder();
        FormatNumber(mantissa,
exponentialSection.BeforeDecimal,
exponentialSection.DecimalSeparator,
exponentialSection.AfterDecimal, false, culture, result);

result.Append(exponentialSection.ExponentialToken[0]);
        if (exponentialSection.ExponentialToken[1] == '+'
&& exponent >= 0)
        {
            result.Append("+");
        }
        else if (exponent < 0)
        {
            result.Append("-");
        }

        FormatThousands(Math.Abs(exponent).ToString(CultureInfo.InvariantCulture), false, false, exponentialSection.Power, culture, result);
        return result.ToString();
    }

    private static string FormatFraction(double value,
SectionInfo format, CultureInfo culture)
    {
        if (!(format.Section is FractionSection
fractionSection))
        {
            throw new Exception("Invalid section");
        }
        var integral = 0;
        int numerator, denominator;

```

```

var sign = value < 0;
if (fractionSection.IntegerPart != null)
{
    integral = (int)Math.Truncate(value);
    value = Math.Abs(value - integral);
}
if (fractionSection.DenominatorConstant != 0)
{
    denominator =
fractionSection.DenominatorConstant;
    var rr = Math.Round(value * denominator);
    var b = Math.Floor(rr / denominator);
    numerator = (int)(rr - b * denominator);
}
else
{
    var denominatorDigits =
Math.Min(GetDigitCount(fractionSection.Denominator), 7);
    GetFraction(value, (int)Math.Pow(10,
denominatorDigits) - 1, out numerator, out denominator);
}
    var numeratorZeros =
GetZeroCount(fractionSection.Numerator);
    var hideFraction = (fractionSection.IntegerPart
!= null && numerator == 0 && numeratorZeros == 0);
    var result = new StringBuilder();
    if (sign)
        result.Append("-");
    if (fractionSection.IntegerPart != null)

FormatThousands(Math.Abs(integral).ToString("F0",
CultureInfo.InvariantCulture), false, hideFraction,
fractionSection.IntegerPart, culture, result);
        var numeratorString =
Math.Abs(numerator).ToString("F0", CultureInfo.InvariantCulture);
        var denominatorString =
denominator.ToString("F0", CultureInfo.InvariantCulture);
        var fraction = new StringBuilder();
        FormatThousands(numeratorString, false, true,
fractionSection.Numerator, culture, fraction);
        fraction.Append("/");
        if (fractionSection.DenominatorPrefix != null)
            FormatThousands("", false, false,
fractionSection.DenominatorPrefix, culture, fraction);
        if (fractionSection.DenominatorConstant != 0)

fraction.Append(fractionSection.DenominatorConstant.ToString());
        else
            FormatDenominator(denominatorString,
fractionSection.Denominator, fraction);
        if (fractionSection.DenominatorSuffix != null)
            FormatThousands("", false, false,
fractionSection.DenominatorSuffix, culture, fraction);

```

```

        result.Append(hideFraction ? new string(' ',
fraction.ToString().Length) : fraction.ToString());
        if (fractionSection.FractionSuffix != null)
            FormatThousands("", false, false,
fractionSection.FractionSuffix, culture, result);
        return result.ToString();
    }
    // Adapted from ssf.js 'frac()' helper
    private static void GetFraction(double x, int D, out
int nom, out int den)
    {
        var sgn = x < 0 ? -1 : 1;
        var B = x * sgn;
        var P_2 = 0.0;
        var P_1 = 1.0;
        var P = 0.0;
        var Q_2 = 1.0;
        var Q_1 = 0.0;
        var Q = 0.0;
        var A = Math.Floor(B);
        while (Q_1 < D)
        {
            A = Math.Floor(B);
            P = A * P_1 + P_2;
            Q = A * Q_1 + Q_2;
            if ((B - A) < 0.00000005) break;
            B = 1 / (B - A);
            P_2 = P_1; P_1 = P;
            Q_2 = Q_1; Q_1 = Q;
        }
        if (Q > D) { if (Q_1 > D) { Q = Q_2; P = P_2; }
else { Q = Q_1; P = P_1; } }
        nom = (int)(sgn * P);
        den = (int)Q;
    }
    public static void FormatDenominator(string
valueString, List<string> tokens, StringBuilder result)
    {
        var formatDigits = GetDigitCount(tokens);
        valueString = valueString.PadLeft(formatDigits,
'0');

        var significant = false;
        var valueIndex = 0;
        foreach (var token in tokens)
        {
            char c;
            if (valueIndex < valueString.Length)
            {
                c = GetLeftAlignedValueDigit(token,
valueString, valueIndex, significant, out valueIndex);
                if (c != '0')
                    significant = true;
            }
        }
    }

```



```

        else
        {
            c = '0';
            significant = false;
        }
        FormatPlaceholder(token, c, significant,
result);
    }
}

private static char GetLeftAlignedValueDigit(string
token, string valueString, int startIndex, bool significant, out
int valueIndex)
{
    char c;
    valueIndex = startIndex;
    if (valueIndex < valueString.Length)
    {
        c = valueString[valueIndex];
        valueIndex++;
        if (c != '0')
            significant = true;
        if (token != "?" || significant) return c;
        while (valueIndex < valueString.Length)
        {
            c = valueString[valueIndex];
            valueIndex++;
            if (c == '0') continue;
            break;
        }
    }
    else
    {
        c = '0';
    }
    return c;
}

private static void FormatPlaceholder(string token,
char c, bool significant, StringBuilder result)
{
    switch (token)
    {
        case "0" when significant:
            result.Append(c);
            break;
        case "0":
            result.Append("0");
            break;
        case "#":
            if (significant)
                result.Append(c);
            break;
        case "?" when significant:
            result.Append(c);

```

```

        break;
        case "?":
            result.Append(" ");
            break;
    }
}
public static int GetDigitCount(ReadOnlyList<string>
tokens)
{
    return tokens.Count(token =>
Token.IsPlaceholder(token));
}
private static int GetZeroCount(IEnumerable<string>
tokens)
{
    return tokens.Count(token => token == "0");
}
private static void FormatLiteral(string token,
StringBuilder result)
{
    var literal = string.Empty;
    if (token == ",")
    {
    }
    else switch (token.Length)
    {
        case 2 when (token[0] == '*' || token[0]
== '\\'):
            literal = token[1].ToString();
            break;
        case 2 when token[0] == '_':
            literal = " ";
            break;
        default:
            literal = token.StartsWith("\\") ?
token.Substring(1, token.Length - 2) : token;
            break;
    }
    result.Append(literal);
}
public static string ApplyGeneralFormat(object value,
IFormatProvider provider)
{
    switch (value)
    {
        case double d:
            return d.ToString("G15", provider);
        default:
            return Convert.ToString(value, provider);
    }
}
}
}

```

Файл FractionSection.cs

```
using System.Collections.Generic;

namespace FormatterLib
{
    public class FractionSection : ISection
    {
        public List<string> IntegerPart { get; }
        public List<string> Numerator { get; }
        public List<string> DenominatorPrefix { get; }
        public List<string> Denominator { get; }
        public int DenominatorConstant { get; }
        public List<string> DenominatorSuffix { get; }
        public List<string> FractionSuffix { get; }

        private FractionSection(List<string> numTokens,
            List<string> denomTokens, bool hasDenominatorConstant, int
            denominatorIndex, bool hasDenominatorPlaceholder)
        {
            var numEndIndex = GetNumeratorEndIndex(numTokens,
            out var hasIntegerPart);
            var denomEndIndex = GetDenominator(denomTokens,
            hasDenominatorConstant, hasDenominatorPlaceholder, out var
            constant);

            var fractionSuffixIndex = denomTokens.Count;
            while (fractionSuffixIndex > denomEndIndex)
            {
                if
            (Token.IsPlaceholder(denomTokens[fractionSuffixIndex - 1]))
                {
                    break;
                }
                fractionSuffixIndex--;
            }

            IntegerPart = hasIntegerPart ?
            numTokens.GetRange(0, numEndIndex) : null;
            Numerator = hasIntegerPart ?
            numTokens.GetRange(numEndIndex, numTokens.Count - numEndIndex) :
            numTokens;
            DenominatorPrefix = denominatorIndex > 0 ?
            denomTokens.GetRange(0, denominatorIndex) : null;
            Denominator =
            denomTokens.GetRange(denominatorIndex, denomEndIndex -
            denominatorIndex);
            DenominatorConstant = hasDenominatorConstant ?
            int.Parse(constant) : 0;
        }
    }
}
```

```

        DenominatorSuffix = denomEndIndex <
fractionSuffixIndex ? denomTokens.GetRange(denomEndIndex,
fractionSuffixIndex - denomEndIndex) : null;
        FractionSuffix = fractionSuffixIndex <
denomTokens.Count ? denomTokens.GetRange(fractionSuffixIndex,
denomTokens.Count - fractionSuffixIndex) : null;
    }

    private int GetNumeratorEndIndex(List<string> tokens,
out bool hasIntegerPart)
    {
        var hasPlaceholder = false;
        var hasSpace = false;
        var numeratorIndex = -1;
        var index = tokens.Count - 1;

        hasIntegerPart = false;
        while (index >= 0)
        {
            if (Token.IsPlaceholder(tokens[index]))
            {
                hasPlaceholder = true;

                if (hasSpace)
                {
                    hasIntegerPart = true;
                    break;
                }
            }
            else
            {
                if (hasPlaceholder && !hasSpace)
                {
                    hasSpace = true;
                    numeratorIndex = index + 1;
                }
            }
            index--;
        }

        return numeratorIndex;
    }

    private int GetDenominator(List<string> tokens, bool
hasConstant, bool hasPlaceholder, out string constant)
    {
        var index = 0;

        constant = string.Empty;
        while (index < tokens.Count)
        {

```

```

        if (hasConstant &&
Token.IsDigitWithZero(tokens[index]))
        {
            constant += tokens[index];
        }
        else if (!(hasPlaceholder &&
Token.IsPlaceholder(tokens[index])))
        {
            break;
        }
        index++;
    }

    return index;
}

public static bool TryParse(List<string> tokens, out
FractionSection format)
{
    format = null;
    var fractionSplitterIndex = tokens.IndexOf("/");
    if (fractionSplitterIndex == -1) return false;

    var numTokens = tokens.GetRange(0,
fractionSplitterIndex);
    fractionSplitterIndex++;
    var denomTokens =
tokens.GetRange(fractionSplitterIndex, tokens.Count -
fractionSplitterIndex);

    var hasDenominatorPlaceholder = false;
    var hasDenominatorConstant = false;
    var denominatorIndex = 0;

    while (denominatorIndex < denomTokens.Count)
    {
        var token = denomTokens[denominatorIndex];
        if (Token.IsPlaceholder(token))
        {
            hasDenominatorPlaceholder = true;
            break;
        }

        if (Token.IsDigitWithoutZero(token))
        {
            hasDenominatorConstant = true;
            break;
        }
        denominatorIndex++;
    }

    if (!hasDenominatorPlaceholder &&
!hasDenominatorConstant)

```

```

        {
            return false;
        }

        format = new FractionSection(numTokens,
denomTokens, hasDenominatorConstant, denominatorIndex,
hasDenominatorPlaceholder);

        return true;
    }
}

```

Файл IFormatter.cs

```

namespace FormatterLib
{
    public interface IFormatter
    {
        string Code { get; }
        Color FontColor { get; set; }
        string Format(string cellValue);
    }
}

```

Файл ISection.cs

```

namespace FormatterLib
{
    public interface ISection
    {
    }
}

```

Файл LocalizedToken.cs

```

using System;

namespace FormatterLib
{
    internal class LocalizedToken : Token
    {
        public new static bool IsGeneral(string token)
        {
            return string.Compare(token, "основной",
StringComparison.OrdinalIgnoreCase) == 0;
        }

        public new static bool IsDatePart(string token)
        {

```

```

        return
            token.StartsWith("r",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("m",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("д",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("с",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("ч",
StringComparison.OrdinalIgnoreCase) ||
            (token.StartsWith("g",
StringComparison.OrdinalIgnoreCase) && !IsGeneral(token)) ||
            string.Compare(token, "am/pm",
StringComparison.OrdinalIgnoreCase) == 0 ||
            string.Compare(token, "a/p",
StringComparison.OrdinalIgnoreCase) == 0 ||
            IsDurationPart(token);
    }

    public new static bool IsDurationPart(string token)
    {
        return
            token.StartsWith("[ч",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("[м",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("[с",
StringComparison.OrdinalIgnoreCase);
    }
}
}

```

Файл NumberFormat.cs

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;

namespace FormatterLib
{
    public class NumberFormat
    {
        private string _localizedCode;
        private readonly bool _needTranslateToEn;

        public bool IsValid { get; }
        public string FormatString { get; private set; }
        public List<SectionInfo> Sections { get; private set; }
    }
}

```

```

set; }

        public SectionInfo ActiveSectionInfo { get; private
        public bool IsDateFormat { get; }
        public bool IsTimeFormat { get; }

        public string LocalizedFormatString
        {
            get
            {
                return _localizedCode;
            }
            set
            {
                _localizedCode = value;
                if (_needTranslateToEn)
                {
                    FormatString = TranslateToEn(value);
                }
            }
        }

        public NumberFormat(string formatString, bool
isRuCode = false)
        {
            if (isRuCode)
            {
                _needTranslateToEn = true;
                LocalizedFormatString = formatString;
            }
            else
            {
                _needTranslateToEn = false;
                FormatString = formatString;
                LocalizedFormatString =
TranslateToRu(formatString);
            }

            var sections = Parser.ParseSections(FormatString,
out bool syntaxError);

            IsValid = !syntaxError;

            if (IsValid)
            {
                Sections = sections;
                IsDateFormat =
GetFirstSectionOfType(SectionType.Date) != null;
                IsTimeFormat =
GetFirstSectionOfType(SectionType.Duration) != null;
            }
            else
            {

```



```

        Sections = new List<SectionInfo>();
    }
}

    public FormattedValue Format(object value,
CultureInfo culture)
    {
        if (value == null)
        {
            return new FormattedValue { Color = null,
Text = null };
        }

        if (double.TryParse(value.ToString(), out var
dValue))
        {
            value = dValue;
        }
        else if (DateTime.TryParse(value.ToString(), out
var dtValue))
        {
            value = dtValue;
        }

        var section = GetSection(value);
        ActiveSectionInfo = section;

        if (section == null)
            return new FormattedValue()
            {
                Text =
Formatter.ApplyGeneralFormat(value, culture),
                Color = null
            };

        try
        {
            return new FormattedValue()
            {
                Text = Formatter.Format(value, section,
culture),
                Color = section.Color
            };
        }
        catch (Exception)
        {
            return new FormattedValue()
            {
                Text =
Formatter.ApplyGeneralFormat(value, culture),
                Color = null
            };
        }
    }
}

```

					ПГУ 2.090301.39.001 ПЗ	Лист
						110
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

    public static string TranslateToEn(string formatCode)
    {
        var translatedCode = string.Empty;
        var tokenizer = new TokenReader(formatCode);
        while (true)
        {
            var tokens =
RuToEnParser.ParseSection(tokenizer, out var sectionSyntaxError);

            if (sectionSyntaxError)
            {
                return null;
            }
            if (tokens == null)
            {
                translatedCode =
translatedCode.Remove(translatedCode.Length - 1);
                break;
            }

            translatedCode =
tokens.Aggregate(translatedCode, (current, token) => current +
token);

            translatedCode += ';';
        }

        return translatedCode;
    }

    public static string TranslateToRu(string formatCode)
    {
        var translatedCode = string.Empty;
        var tokenizer = new TokenReader(formatCode);
        while (true)
        {
            var tokens =
EnToRuParser.ParseSection(tokenizer, out var sectionSyntaxError);

            if (sectionSyntaxError)
            {
                return null;
            }

            if (tokens == null)
            {
                translatedCode =
translatedCode.Remove(translatedCode.Length - 1);
                break;
            }

```

```

        translatedCode =
tokens.Aggregate(translatedCode, (current, token) => current +
token);

        translatedCode += ' ';
    }

    return translatedCode;
}

public SectionInfo GetSection(object value)
{
    switch (value)
    {
        case double d:
            return GetNumericSection(d);
        case int i:
            return GetNumericSection(i);
        case short s:
            return GetNumericSection(s);
        case string _:
            return Sections.Count >= 4 ? Sections[3]
: null;

        case DateTime _:
            return
GetFirstSectionOfType(SectionType.Date);
        case TimeSpan ts:
            return GetNumericSection(ts.TotalDays);
        default:
            return null;
    }
}

public SectionInfo GetFirstSectionOfType(SectionType
type)
{
    return Sections.FirstOrDefault(section =>
section.Type == type);
}

private bool IsValidSectionForValue(int sectionIndex,
double value)
{
    if (Sections[sectionIndex].Condition != null)
    {
        if
(Sections[sectionIndex].Condition.Apply(value))
        {
            return true;
        }
    }

    return false;
}

```

```

    }

    private SectionInfo GetNumericSection(double value)
    {
        if (Sections.Count < 1)
        {
            return null;
        }

        if (IsValidSectionForValue(0, value) ||
Sections.Count == 1 || (Sections.Count == 2 && value >= 0) ||
(Sections.Count >= 2 && value > 0))
        {
            return Sections[0];
        }

        if (Sections.Count < 2)
        {
            return null;
        }

        if (IsValidSectionForValue(1, value) || value < 0
|| (Sections.Count == 2 && Sections[0].Condition != null))
        {
            return Sections[1];
        }

        return Sections.Count < 3 ? null : Sections[2];
    }
}

```

Файл Parser.cs

```

using System.Collections.Generic;
using System.Globalization;

namespace FormatterLib
{
    internal class Parser
    {
        public static List<SectionInfo> ParseSections(string
formatString, out bool syntaxError)
        {
            var tokenReader = new TokenReader(formatString);
            var sections = new List<SectionInfo>();
            syntaxError = false;
            while (true)
            {
                var section = ParseSection(tokenReader,
sections.Count, out var sectionSyntaxError);

```

```

        if (sectionSyntaxError)
            syntaxError = true;

        if (section == null)
            break;

        sections.Add(section);
    }

    return sections;
}

private static SectionInfo ParseSection(TokenReader
reader, int index, out bool syntaxError)
{
    var hasDateParts = false;
    var hasDurationParts = false;
    var hasGeneralPart = false;
    var hasTextPart = false;
    var hasPlaceholders = false;
    Condition condition = null;
    Color color = null;
    string token;
    var tokens = new List<string>();

    while ((token = ReadToken(reader, out
syntaxError)) != null)
    {
        if (token == ";")
            break;

        hasPlaceholders |=
Token.IsPlaceholder(token);

        if (Token.IsDatePart(token))
        {
            hasDateParts = true;
            hasDurationParts |=
Token.IsDurationPart(token);
            tokens.Add(token);
        }
        else if (Token.IsGeneral(token))
        {
            hasGeneralPart = true;
            tokens.Add(token);
        }
        else if (token == "@")
        {
            hasTextPart = true;
            tokens.Add(token);
        }
        else if (token.StartsWith("["))
        {

```

```

        token.Length - 2);
        if (TryParseCondition(expression, out var
        parseCondition))
            condition = parseCondition;
        else if (TryParseColor(expression, out
        var parseColor))
            color = parseColor;
        else if
        (TryParseCurrencySymbol(expression, out var parseCurrencySymbol))
            tokens.Add("\"" + parseCurrencySymbol
+ "\"");
    }
    else
    {
        tokens.Add(token);
    }
}

if (syntaxError || tokens.Count == 0)
{
    return null;
}

if (
hasTextPart)) ||
    (hasDateParts && (hasGeneralPart ||
    (hasGeneralPart && (hasTextPart)))
    {
        syntaxError = true;
        return null;
    }

ISection section = null;
SectionType type;
List<string> generalTextDateDuration = null;

if (hasDateParts)
{
    type = hasDurationParts ?
SectionType.Duration : SectionType.Date;

    ParseMilliseconds(tokens, out
generalTextDateDuration);
}
else if (hasGeneralPart)
{
    type = SectionType.General;
    generalTextDateDuration = tokens;
}
else if (hasTextPart || !hasPlaceholders)
{
    type = SectionType.Text;

```

```

        generalTextDateDuration = tokens;
    }
    else if (FractionSection.TryParse(tokens, out var
fraction))
    {
        type = SectionType.Fraction;
        section = fraction;
    }
    else if (DecimalSection.TryParse(tokens, out var
number))
    {
        type = SectionType.Number;
        section = number;
    }
    else if (ExponentialSection.TryParse(tokens, out
var exponential))
    {
        type = SectionType.Exponential;
        section = exponential;
    }

    else
    {
        syntaxError = true;
        return null;
    }

    return new SectionInfo()
    {
        Type = type,
        SectionIndex = index,
        Color = color,
        Condition = condition,
        Section = section,
        OtherParts = generalTextDateDuration
    };
}

internal static int ParseNumberTokens(List<string>
tokens, int startPosition, out List<string> beforeDecimal, out
bool decimalSeparator, out List<string> afterDecimal)
{
    beforeDecimal = null;
    afterDecimal = null;
    decimalSeparator = false;

    var remainder = new List<string>();
    var index = 0;

    for (index = 0; index < tokens.Count; ++index)
    {
        var token = tokens[index];
        if (token == "." && beforeDecimal == null)

```

```

        {
            decimalSeparator = true;
            beforeDecimal = tokens.GetRange(0,
index);

            remainder = new List<string>();
        }
        else if (Token.IsNumberLiteral(token))
        {
            remainder.Add(token);
        }
        else if (token.StartsWith("["))
        {
        }
        else
        {
            break;
        }
    }

    if (remainder.Count <= 0) return index;

    if (beforeDecimal != null)
    {
        afterDecimal = remainder;
    }
    else
    {
        beforeDecimal = remainder;
    }

    return index;
}

private static void
ParseMilliseconds(IReadOnlyList<string> tokens, out List<string>
result)
{
    result = new List<string>();
    for (var i = 0; i < tokens.Count; i++)
    {
        var token = tokens[i];
        if (token == ".")
        {
            var zeros = 0;
            while (i + 1 < tokens.Count && tokens[i +
1] == "0")
            {
                i++;
                zeros++;
            }

```



```

        if (zeros > 0)
            result.Add("." + new string('0',
zeros));

        else
            result.Add(".");
    }
    else
    {
        result.Add(token);
    }
}

internal static string ReadToken(TokenReader reader,
out bool syntaxError)
{
    var offset = reader.Position;
    if (
        ReadLiteral(reader) ||
        reader.ReadEnclosed('[', ']') ||

        reader.ReadOneOf("#?;!&%+-
$€£₽0123456789{ } ( ) ; / . @ ") ||
        reader.ReadString("e+", true) ||
        reader.ReadString("e-", true) ||
        reader.ReadString("General", true) ||

        reader.ReadString("am/pm", true) ||
        reader.ReadString("a/p", true) ||
        reader.ReadOneOrMore('y') ||
        reader.ReadOneOrMore('Y') ||
        reader.ReadOneOrMore('m') ||
        reader.ReadOneOrMore('M') ||
        reader.ReadOneOrMore('d') ||
        reader.ReadOneOrMore('D') ||
        reader.ReadOneOrMore('h') ||
        reader.ReadOneOrMore('H') ||
        reader.ReadOneOrMore('s') ||
        reader.ReadOneOrMore('S') ||
        reader.ReadOneOrMore('g') ||
        reader.ReadOneOrMore('G'))
    {
        syntaxError = false;
        var length = reader.Position - offset;
        return reader.Substring(offset, length);
    }

    syntaxError = reader.Position < reader.Length;
    if (syntaxError)
    {
        syntaxError = true;
    }
    return null;
}

```

```

    }

    internal static bool ReadLiteral(TokenReader reader)
    {
        if (reader.Peek() == '\\\\' || reader.Peek() == '*'
|| reader.Peek() == '_')
        {
            reader.Advance(2);
            return true;
        }
        else if (reader.ReadEnclosed("'", "''))
        {
            return true;
        }

        return false;
    }

    internal static bool TryParseCondition(string token,
out Condition result)
    {
        var tokenizer = new TokenReader(token);

        if (tokenizer.ReadString("<=") ||
            tokenizer.ReadString("<>") ||
            tokenizer.ReadString("<") ||
            tokenizer.ReadString(">=") ||
            tokenizer.ReadString(">") ||
            tokenizer.ReadString("="))
        {
            var conditionPosition = tokenizer.Position;
            var op = tokenizer.Substring(0,
conditionPosition);

            if (ReadConditionValue(tokenizer))
            {
                var valueString =
tokenizer.Substring(conditionPosition, tokenizer.Position -
conditionPosition);

                result = new Condition()
                {
                    Op = op,
                    Value = double.Parse(valueString,
CultureInfo.InvariantCulture)
                };
                return true;
            }

            result = null;
            return false;
        }
    }
}

```

```

private static bool ReadConditionValue(TokenReader
tokenReader)
{
    tokenReader.ReadString("-");
    while (tokenReader.ReadOneOf("0123456789"))
    {
    }

    if (tokenReader.ReadString("."))
    {
        while (tokenReader.ReadOneOf("0123456789"))
        {
        }
    }

    if (!tokenReader.ReadString("e+", true) &&
!tokenReader.ReadString("e-", true)) return true;

    if (tokenReader.ReadOneOf("0123456789"))
    {
        while (tokenReader.ReadOneOf("0123456789"))
        {
        }
    }
    else
    {
        return false;
    }

    return true;
}

internal static bool TryParseColor(string token, out
Color color)
{
    var tokenizer = new TokenReader(token);
    if (
        tokenizer.ReadString("black", true) ||
        tokenizer.ReadString("blue", true) ||
        tokenizer.ReadString("cyan", true) ||
        tokenizer.ReadString("green", true) ||
        tokenizer.ReadString("magenta", true) ||
        tokenizer.ReadString("red", true) ||
        tokenizer.ReadString("white", true) ||
        tokenizer.ReadString("yellow", true))
    {
        color = new Color
        {
            Value = tokenizer.Substring(0,
tokenizer.Position)
        };
        return true;
    }
}

```

```

    }

    color = null;
    return false;
}

internal static bool TryParseCurrencySymbol(string
token, out string currencySymbol)
{
    if (string.IsNullOrEmpty(token)
        || !token.StartsWith("$"))
    {
        currencySymbol = null;
        return false;
    }

    currencySymbol = token.Contains("-") ?
token.Substring(1, token.IndexOf('-') - 1) : token.Substring(1);

    return true;
}
}
}

```

Файл RuToEnParser.cs

```

using System.Collections.Generic;

namespace FormatterLib
{
    internal class RuToEnParser : Parser
    {
        internal static List<string> ParseSection(TokenReader
reader, out bool syntaxError)
        {
            string token;
            var tokens = new List<string>();

            while ((token = ReadToken(reader, out
syntaxError)) != null)
            {
                if (token == ";"")
                    break;

                Token.IsPlaceholder(token);

                if (LocalizedToken.IsDatePart(token))
                {
                    var translatedToken = string.Empty;
                    LocalizedToken.IsDurationPart(token);

```

```

        foreach (var c in token.ToLower())
        {
            switch (c)
            {
                case 'г':
                    translatedToken += "y";
                    break;
                case 'м':
                    translatedToken += "m";
                    break;
                case 'д':
                    translatedToken += "d";
                    break;
                case 'ч':
                    translatedToken += "h";
                    break;
                case 'с':
                    translatedToken += "s";
                    break;
                default:
                    translatedToken += c;
                    break;
            }

            tokens.Add(translatedToken);
        }
    }
    else if (LocalizedToken.IsGeneral(token))
    {
        token =
token.ToLower().Replace("основной", "general");
        tokens.Add(token);
    }
    else if (token == "@")
    {
        tokens.Add(token);
    }
    else if (token.StartsWith("[")
token.Length - 2);
        if (TryParseCondition(expression, out var
parseCondition))
        tokens.Add($"[{parseCondition.Op}{parseCondition.Value}]");

        else if (TryParseColor(expression, out
var parseColor))
        {
            var localizedColor = string.Empty;
            switch (parseColor.Value.ToLower())
            {

```

```

        case "черный":
            localizedColor = "black";
            break;
        case "синий":
            localizedColor = "blue";
            break;
        case "голубой":
            localizedColor = "cyan";
            break;
        case "зеленый":
            localizedColor = "green";
            break;
        case "фиолетовый":
            localizedColor = "magenta";
            break;
        case "красный":
            localizedColor = "red";
            break;
        case "белый":
            localizedColor = "white";
            break;
        case "желтый":
            localizedColor = "yellow";
            break;
    }
    tokens.Add($"[{localizedColor}]");
}
else if
(TryParseCurrencySymbol(expression, out var parseCurrencySymbol))
    tokens.Add("\"" + parseCurrencySymbol
+ "\"");
    }
    else
    {
        tokens.Add(token);
    }
}

if (syntaxError || tokens.Count == 0)
{
    return null;
}

return tokens;
}

internal new static string ReadToken(TokenReader
reader, out bool syntaxError)
{
    var offset = reader.Position;
    if (
        ReadLiteral(reader) ||
        reader.ReadEnclosed('[', ']') ||

```

```

        reader.ReadOneOf("#?;!&%+-
$€£0123456789{}() :;/.@# ") ||
        reader.ReadString("e+", true) ||
        reader.ReadString("e-", true) ||
        reader.ReadString("Основной", true) ||

        reader.ReadString("am/pm", true) ||
        reader.ReadString("a/p", true) ||
        reader.ReadOneOrMore('r') ||
        reader.ReadOneOrMore('Г') ||
        reader.ReadOneOrMore('м') ||
        reader.ReadOneOrMore('М') ||
        reader.ReadOneOrMore('д') ||
        reader.ReadOneOrMore('Д') ||
        reader.ReadOneOrMore('ч') ||
        reader.ReadOneOrMore('Ч') ||
        reader.ReadOneOrMore('с') ||
        reader.ReadOneOrMore('С') ||
        reader.ReadOneOrMore('g') ||
        reader.ReadOneOrMore('G'))
    {
        syntaxError = false;
        var length = reader.Position - offset;
        return reader.Substring(offset, length);
    }

    syntaxError = reader.Position < reader.Length;
    return null;
}

internal new static bool TryParseColor(string token,
out Color color)
{
    var tokenizer = new TokenReader(token);
    if (
        tokenizer.ReadString("черный", true) ||
        tokenizer.ReadString("синий", true) ||
        tokenizer.ReadString("голубой", true) ||
        tokenizer.ReadString("зеленый", true) ||
        tokenizer.ReadString("фиолетовый", true) ||
        tokenizer.ReadString("красный", true) ||
        tokenizer.ReadString("белый", true) ||
        tokenizer.ReadString("желтый", true))
    {
        color = new Color()
        {
            Value = tokenizer.Substring(0,
tokenizer.Position)
        };
        return true;
    }
}

```

```

        color = null;
        return false;
    }
}

```

Файл SectionInfo.cs

```

using System.Collections.Generic;
namespace FormatterLib
{
    public class SectionInfo
    {
        public int SectionIndex { get; set; }
        public SectionType Type { get; set; }
        public Color Color { get; set; }
        public Condition Condition { get; set; }
        public ISection Section { get; set; }
        // General, Text, Date, Duration Parts
        public List<string> OtherParts { get; set; }
    }
}

```

Файл SectionType.cs

```

namespace FormatterLib
{
    public enum SectionType
    {
        General,
        Number,
        Fraction,
        Exponential,
        Date,
        Duration,
        Text,
    }
}

```

Файл Token.cs

```

using System;
namespace FormatterLib
{
    internal class Token
    {
        public static bool IsExponent(string token)
        {
            return

```



```

        (string.Compare(token, "e+",
StringComparison.OrdinalIgnoreCase) == 0) ||
        (string.Compare(token, "e-",
StringComparison.OrdinalIgnoreCase) == 0);
    }
    public static bool IsLiteral(string token)
    {
        return
            token.StartsWith(" ") ||
            token.StartsWith("\\"") ||
            token.StartsWith("\"") ||
            token.StartsWith("*") ||
            token == "," ||
            token == "!" ||
            token == "&" ||
            token == "%" ||
            token == "+" ||
            token == "-" ||
            token == "$" ||
            token == "€" ||
            token == "£" ||
            token == "₽" ||
            token == "1" ||
            token == "2" ||
            token == "3" ||
            token == "4" ||
            token == "5" ||
            token == "6" ||
            token == "7" ||
            token == "8" ||
            token == "9" ||
            token == "{" ||
            token == "}" ||
            token == "(" ||
            token == ")" ||
            token == " ";
    }
    public static bool IsNumberLiteral(string token)
    {
        return
            IsPlaceholder(token) ||
            IsLiteral(token) ||
            token == ".";
    }
    public static bool IsPlaceholder(string token)
    {
        return token == "0" || token == "#" || token ==
"?";
    }
    public static bool IsGeneral(string token)
    {
        return string.Compare(token, "general",
StringComparison.OrdinalIgnoreCase) == 0;
    }

```

```

    }
    public static bool IsDatePart(string token)
    {
        return
            token.StartsWith("y",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("m",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("d",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("s",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("h",
StringComparison.OrdinalIgnoreCase) ||
            (token.StartsWith("g",
StringComparison.OrdinalIgnoreCase) && !IsGeneral(token)) ||
            string.Compare(token, "am/pm",
StringComparison.OrdinalIgnoreCase) == 0 ||
            string.Compare(token, "a/p",
StringComparison.OrdinalIgnoreCase) == 0 ||
            IsDurationPart(token);
    }
    public static bool IsDurationPart(string token)
    {
        return
            token.StartsWith("[h",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("[m",
StringComparison.OrdinalIgnoreCase) ||
            token.StartsWith("[s",
StringComparison.OrdinalIgnoreCase);
    }
    public static bool IsDigitWithZero(string token)
    {
        return token == "0" || IsDigitWithoutZero(token);
    }
    public static bool IsDigitWithoutZero(string token)
    {
        switch (token)
        {
            case "1":
            case "2":
            case "3":
            case "4":
            case "5":
            case "6":
            case "7":
            case "8":
            case "9":
                return true;
            default:
                return false;
        }
    }

```

```

    }
}
}

```

Файл TokenReader.cs

```

using System;
using System.Linq;
namespace FormatterLib
{
    internal class TokenReader
    {
        private readonly string _formatString;
        public TokenReader(string fmt)
        {
            _formatString = fmt;
        }
        public int Position { get; private set; }
        public int Length => _formatString?.Length ?? 0;
        public string Substring(int startIndex, int length)
        {
            return _formatString.Substring(startIndex,
length);
        }
        public int Peek(int offset = 0)
        {
            if (Position + offset >= Length)
                return -1;
            return _formatString[Position + offset];
        }
        public int PeekUntil(int startOffset, int until)
        {
            var offset = startOffset;
            while (true)
            {
                var c = Peek(offset++);
                if (c == -1)
                    break;
                if (c == until)
                    return offset - startOffset;
            }
            return 0;
        }
        public bool PeekOneOf(int offset, string s)
        {
            return s.Any(c => Peek(offset) == c);
        }
        public void Advance(int characters = 1)
        {
            Position = Math.Min(Position + characters,
_formatString.Length);
        }
    }
}

```

```

    }
    public bool ReadOneOrMore(int c)
    {
        if (Peek() != c)
            return false;
        while (Peek() == c)
            Advance();
        return true;
    }
    public bool ReadOneOf(string s)
    {
        if (!PeekOneOf(0, s)) return false;
        Advance();
        return true;
    }
    public bool ReadString(string s, bool ignoreCase =
false)
    {
        if (Position + s.Length > Length)
            return false;
        for (var i = 0; i < s.Length; i++)
        {
            var c1 = s[i];
            var c2 = (char)Peek(i);
            if (ignoreCase)
            {
                if (char.ToLower(c1) != char.ToLower(c2))
return false;
            }
            else
            {
                if (c1 != c2) return false;
            }
        }
        Advance(s.Length);
        return true;
    }
    public bool ReadEnclosed(char open, char close)
    {
        if (Peek() != open) return false;
        var length = PeekUntil(1, close);
        if (length <= 0) return false;
        Advance(1 + length);
        return true;
    }
}
}

```

Приложение Б.2 - Модуль сортировки и фильтрации данных

Файл AllFilters.cs

					ПГУ 2.090301.39.001 ПЗ	Лист
						129
Изм.	Лист	№ докум.	Подпись	Дата		

```

using System;
using System.Collections.Generic;

namespace FiltersLib
{
    public class AllFilters
    {
        public enum FilterOp
        {
            AND,
            OR
        }

        public enum MonthsNames
        {
            January = 1,
            February,
            March,
            April,
            May,
            June,
            July,
            August,
            September,
            October,
            November,
            December
        }

        public enum QuartersNames
        {
            FirstQuarter = 1,
            SecondQuarter,
            ThirdQuarter,
            FourthQuarter,
        }

        public enum Days
        {
            Yesterday = -1,
            Today,
            Tomorrow
        }

        public enum Weeks
        {
            LastWeek = -7,
            ThisWeek = 0,
            NextWeek = 7
        }

        public enum Months
        {

```

```

        LastMonth = -1,
        ThisMonth,
        NextMonth
    }

    public enum Quarters
    {
        LastQuarter = -1,
        ThisQuarter,
        NextQuarter
    }

    public enum Years
    {
        LastYear = -1,
        ThisYear,
        NextYear,
        FromBeginningYear
    }

    public static readonly Dictionary<string,
Func<string[], string, int[]>> textFilters =
        new Dictionary<string, Func<string[], string,
int[]>>()
    {
        { "Равно", TextFilters.Equals },
        { "Не равно", TextFilters.NotEquals },
        { "Начинается с", TextFilters.StartsWith },
        { "Заканчивается на", TextFilters.EndsWith },
        { "Содержит", TextFilters.Contains },
        { "Не содержит", TextFilters.NotContains }
    };

    public static readonly Dictionary<string,
Func<double[], double, int[]>> numericalFilters =
        new Dictionary<string, Func<double[], double,
int[]>>()
    {
        { "Равно", NumericalFilters.Equals },
        { "Не равно", NumericalFilters.NotEquals },
        { "Больше", NumericalFilters.Greater },
        { "Больше или равно",
NumericalFilters.GreaterOrEquals },
        { "Меньше", NumericalFilters.LessThan},
        { "Меньше или равно",
NumericalFilters.LessThanOrEquals },
        { "Между", null },
        { "Первые 10", null },
    };

    public static readonly Dictionary<string,
Func<double[], int[]>> numericalAvgFilters =
        new Dictionary<string, Func<double[], int[]>>()

```

```

        {
            { "Выше среднего",
NumericalFilters.AboveAverage },
            { "Ниже среднего",
NumericalFilters.BelowAverage }
        };

        public static readonly Dictionary<string,
Func<DateTime[], DateTime, int[]>> dateTimeFilters =
            new Dictionary<string, Func<DateTime[], DateTime,
int[]>>()
        {
            { "Равно", DateTimeFilters.Equals },
            { "Не равно", DateTimeFilters.NotEquals },
            { "После", DateTimeFilters.After },
            { "После или равно",
DateTimeFilters.AfterOrEquals },
            { "До", DateTimeFilters.Before },
            { "До или равно", DateTimeFilters.BeforeOrEquals
},
        };

        public static readonly Dictionary<string, object>
intervalFilters = new Dictionary<string, object>()
        {
            {"Завтра", Days.Tomorrow},
            {"Сегодня", Days.Today},
            {"Вчера", Days.Yesterday},
            {"На следующей неделе", Weeks.NextWeek},
            {"На этой неделе", Weeks.ThisWeek},
            {"На прошлой неделе", Weeks.LastWeek},
            {"В следующем месяце", Months.NextMonth},
            {"В этом месяце", Months.ThisMonth},
            {"В прошлом месяце", Months.LastMonth},
            {"В следующем квартале", Quarters.NextQuarter},
            {"В этом квартале", Quarters.ThisQuarter},
            {"В прошлом квартале", Quarters.LastQuarter},
            {"В следующем году", Years.NextYear},
            {"В этом году", Years.ThisYear},
            {"В прошлом году", Years.LastYear},
            {"С начала года", Years.FromBeginningYear},
            {"Все даты за период", null}
        };

        public static readonly Dictionary<string, object>
periodFilters = new Dictionary<string, object> {
            { "Квартал 1", QuartersNames.FirstQuarter },
            { "Квартал 2", QuartersNames.SecondQuarter },
            { "Квартал 3", QuartersNames.ThirdQuarter },
            { "Квартал 4", QuartersNames.FourthQuarter },
            { "Январь", MonthsNames.January },
            { "Февраль", MonthsNames.February },
            { "Март", MonthsNames.March },

```

```

        { "Апрель", MonthsNames.April },
        { "Май", MonthsNames.May },
        { "Июнь", MonthsNames.June },
        { "Июль", MonthsNames.July },
        { "Август", MonthsNames.August },
        { "Сентябрь", MonthsNames.September },
        { "Октябрь", MonthsNames.October },
        { "Ноябрь", MonthsNames.November },
        { "Декабрь", MonthsNames.December }
    };
}
}

```

Файл DateTimeFilters.cs

```

using System;
using System.Linq;

namespace FiltersLib
{
    class DateTimeFilters
    {
        public static int[] Equals(DateTime[] array, DateTime
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => (x.v > param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] NotEquals(DateTime[] array,
DateTime param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => !(x.v.Equals(param)))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] After(DateTime[] array, DateTime
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => (x.v > param))
                        .Select(x => x.i)
                        .ToArray();
        }
    }
}

```



```

        public static int[] AfterOrEquals(DateTime[] array,
DateTime param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v >= param))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] Before(DateTime[] array, DateTime
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v <= param))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] BeforeOrEquals(DateTime[] array,
DateTime param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v <= param))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] Interval(DateTime[] array,
DateTime param1, DateTime param2)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v >= param1) && (x.v <
param2))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] AllFromQuarter(DateTime[] array,
int quarterNum)
        {
            int firstMonthOfQuarter = 1 + (quarterNum - 1) *
3;
            int lastMonthOfQuarter = 1 + (quarterNum * 3);

            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v.Month >=
firstMonthOfQuarter) && (x.v.Month < lastMonthOfQuarter))
                .Select(x => x.i)

```

```

        .ToArray();
    }

    public static int[] AllFromMonth(DateTime[] array,
int month)
    {
        return array.Select((value, index) => new { v =
value, i = index })
            .Where(x => x.v.Month == month)
            .Select(x => x.i)
            .ToArray();
    }
}

```

Файл Filter.cs

```

using System;
using System.Linq;

namespace FiltersLib
{
    public class Executor
    {
        public int[] ApplyFilter(FilterArgs FilterArgs)
        {
            string filterName = FilterArgs.nameOfFirstFilter;
            if
(AllFilters.numericalAvgFilters.ContainsKey(filterName))
            {
                double[] doubleArray =
FilterArgs.values.Cast<double>().ToArray();

                return
AllFilters.numericalAvgFilters[FilterArgs.nameOfFirstFilter].Invok
e(doubleArray);
            }
            else if
(AllFilters.periodFilters.Keys.Contains(filterName))
            {
                return FilterDateTimePeriod(FilterArgs);
            }
            else if
(AllFilters.intervalFilters.Keys.Contains(FilterArgs.nameOfFirstFi
lter))
            {
                return FilterDateTimeInterval(FilterArgs);
            }
            else
            {
                return BaseFilter(FilterArgs);
            }
        }
    }
}

```

```

    }

    private int[] BaseFilter(FilterArgs FilterArgs)
    {
        bool useSecondFilter =
FilterArgs.nameOfSecondFilter != null;
        int[] result1 = null;
        int[] result2 = null;

        Type t = FilterArgs.values[0].GetType();

        if (FilterArgs.nameOfFirstFilter != null)
        {
            if (t == typeof(string))
            {
                string[] stringArray =
FilterArgs.values.Cast<string>().ToArray();
                result1 =
AllFilters.textFilters[FilterArgs.nameOfFirstFilter]
                    .Invoke(stringArray,
FilterArgs.valueOfFirstFilter);

                if (useSecondFilter)
                {
                    result2 =
AllFilters.textFilters[FilterArgs.nameOfSecondFilter]
                        .Invoke(stringArray,
FilterArgs.valueOfSecondFilter);
                }
            }
            else if (t == typeof(double))
            {
                double[] doubleArray =
FilterArgs.values.Cast<double>().ToArray();

                if
(double.TryParse(FilterArgs.valueOfFirstFilter, out double
doubleValue))
                {
                    result1 =
AllFilters.numericalFilters[FilterArgs.nameOfFirstFilter]
                        .Invoke(doubleArray,
doubleValue);

                    if (useSecondFilter)
                    {
                        if
(double.TryParse(FilterArgs.valueOfSecondFilter, out double
doubleValue2))
                        {
                            result2 =
AllFilters.numericalFilters[FilterArgs.nameOfSecondFilter]

```

```

doubleValue2);

        .Invoke(doubleArray,
        }
        else result1 = null;
    }
}
else if (t == typeof(DateTime))
{
    DateTime[] dateTimeArray =
FilterArgs.values.Cast<DateTime>().ToArray();
    if
(DateTime.TryParse(FilterArgs.valueOfFirstFilter, out DateTime
dateTimeValue))
    {
        result1 =
AllFilters.dateTimeFilters[FilterArgs.nameOfFirstFilter]
.Invoke(dateTimeArray,
dateTimeValue);

        if (useSecondFilter)
        {
            if
(DateTime.TryParse(FilterArgs.valueOfSecondFilter, out DateTime
dateTimeValue2))
            {
                result2 =
AllFilters.dateTimeFilters[FilterArgs.nameOfSecondFilter]
.Invoke(dateTimeArray,
dateTimeValue2);
            }
            else result1 = null;
        }
    }
}

if (result1 != null && result2 != null)
{
    if (FilterArgs.filterOp ==
AllFilters.FilterOp.AND)
    {
        result1 =
result1.Intersect(result2).ToArray();
    }
    else if (FilterArgs.filterOp ==
AllFilters.FilterOp.OR)
    {
        result1 =
result1.Union(result2).ToArray();
    }
}

```

```

        return result1;
    }

    private int[] FilterDateTimeInterval(FilterArgs
FilterArgs)
    {
        int[] result = null;

        DateTime[] array =
FilterArgs.values.Cast<DateTime>().ToArray();
        DateTime today = DateTime.Today;
        DateTime startOfThisWeek = today.AddDays(-
(int)today.DayOfWeek + 1);
        DateTime startOfThisMonth = today.AddDays(-
today.Day + 1);
        DateTime startOfThisYear =
startOfThisMonth.AddMonths(-today.Month + 1);

        int currentQuarter =
(int)Math.Ceiling(today.Month / 3m);
        var selectedFilter =
AllFilters.intervalFilters[FilterArgs.nameOfFirstFilter];
        int num = (int)selectedFilter;

        if (selectedFilter.GetType() ==
typeof(AllFilters.Days))
        {
            result = DateTimeFilters.Interval(array,
today.AddDays(num), today.AddDays(num + 1));
        }
        else if (selectedFilter.GetType() ==
typeof(AllFilters.Weeks))
        {
            result = DateTimeFilters.Interval(array,
startOfThisWeek.AddDays(num), startOfThisWeek.AddDays(num + 7));
        }
        else if (selectedFilter.GetType() ==
typeof(AllFilters.Months))
        {
            result = DateTimeFilters.Interval(array,
startOfThisMonth.AddMonths(num), startOfThisMonth.AddMonths(num +
1));
        }
        else if (selectedFilter.GetType() ==
typeof(AllFilters.Quarters))
        {
            result = DateTimeFilters.Interval(array,
startOfThisYear.AddMonths((currentQuarter + num - 1) * 3),
startOfThisYear.AddMonths((currentQuarter + num) * 3));
        }
        else if (selectedFilter.GetType() ==
typeof(AllFilters.Years))
        {

```

```

        if (num == 2)
        {
            result = DateTimeFilters.Interval(array,
startOfThisYear, today);
        }
        else
        {
            result = DateTimeFilters.Interval(array,
startOfThisYear.AddYears(num), startOfThisMonth.AddYears(num +
1));
        }
    }

    return result;
}

private int[] FilterDateTimePeriod(FilterArgs
FilterArgs)
{
    int[] result = null;
    DateTime[] array =
FilterArgs.values.Cast<DateTime>().ToArray();
    var selectedFilter =
AllFilters.periodFilters[FilterArgs.nameOfFirstFilter];
    int num = (int)selectedFilter;

    if (selectedFilter.GetType() ==
typeof(AllFilters.MonthsNames))
    {
        result = DateTimeFilters.AllFromMonth(array,
num);
    }
    else if (selectedFilter.GetType() ==
typeof(AllFilters.QuartersNames))
    {
        result =
DateTimeFilters.AllFromQuarter(array, num);
    }

    return result;
}

/// <summary>
/// Фильтрует первые N чисел по заданному критерию.
/// </summary>
/// <param name="values">Значения</param>
/// <param name="showLargest">Первые N наибольших или
наименьших</param>
/// <param name="count">Количество чисел</param>
/// <param name="countIsPercent">Вид количества чисел
(процентный или числовой)</param>
/// <returns></returns>

```

```

        public int[] FirstThenFilter(double[] values, bool
showLargest, int count, bool countIsPercent)
        {
            int countOfElements = 0;

            if (!countIsPercent)
            {
                countOfElements = count;
            }

            else countOfElements =
(int)Math.Truncate((double)(count * values.Length / 100));

            if (countOfElements >= values.Length)
            {
                return null;
            }

            if (showLargest)
            {
                return values.Select((value, index) => new {
v = value, i = index })
                    .OrderByDescending(x => x.v)
                    .Select(x => x.i)
                    .Take(countOfElements)
                    .ToArray();
            }
            else
            {
                return values.Select((value, index) => new {
v = value, i = index })
                    .OrderBy(x => x.v)
                    .Select(x => x.i)
                    .Take(countOfElements)
                    .ToArray();
            }
        }

        public int[] Sort(object[] values, bool byDescending)
        {
            if (byDescending)
            {
                return values.Select((value, index) => new {
v = value, i = index })
                    .OrderByDescending(x => x.v)
                    .Select(x => x.i)
                    .ToArray();
            }
            else
            {
                return values.Select((value, index) => new {
v = value, i = index })

```

```

        .OrderBy(x => x.v)
        .Select(x => x.i)
        .ToArray();
    }
}
}

```

Файл FilterArgs.cs

```

namespace FiltersLib
{
    public class FilterArgs
    {
        public object[] values;
        public string nameOfFirstFilter;
        public string valueOfFirstFilter;
        public string nameOfSecondFilter;
        public string valueOfSecondFilter;
        public AllFilters.FilterOp filterOp;

        /// <param name="_values">Значения для
        фильтрации</param>
        /// <param name="_nameOfFirstFilter">Имя первого
        фильтра</param>
        /// <param name="_valueOfFirstFilter">Значение
        первого фильтра</param>
        /// <param name="_nameOfSecondFilter">Имя второго
        фильтра. По умолчанию null</param>
        /// <param name="_valueOfSecondFilter">Значение
        второго фильтра</param>
        /// <param name="_filterOp">Операция над
        фильтрами</param>
        public FilterArgs(object[] _values, string
        _nameOfFirstFilter, string _valueOfFirstFilter = null,
        string _nameOfSecondFilter = null, string
        _valueOfSecondFilter = null, AllFilters.FilterOp _filterOp = 0)
        {
            values = _values;
            nameOfFirstFilter = _nameOfFirstFilter;
            valueOfFirstFilter = _valueOfFirstFilter;
            nameOfSecondFilter = _nameOfSecondFilter;
            valueOfSecondFilter = _valueOfSecondFilter;
            filterOp = _filterOp;
        }
    }
}

```

Файл NumericalFilters.cs

```

using System.Linq;

```



```

namespace FiltersLib
{
    class NumericalFilters
    {
        public static int[] Equals(double[] array, double
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => x.v.Equals(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] NotEquals(double[] array, double
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => !(x.v.Equals(param)))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] Greater(double[] array, double
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => (x.v > param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] GreaterOrEquals(double[] array,
double param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => (x.v >= param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] LessThan(double[] array, double
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => (x.v < param))
                        .Select(x => x.i)
                        .ToArray();
        }
    }
}

```

```

        public static int[] LessThanOrEquals(double[] array,
double param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v <= param))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] AboveAverage(double[] array)
        {
            double avg = array.Average();

            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v > avg))
                .Select(x => x.i)
                .ToArray();
        }

        public static int[] BelowAverage(double[] array)
        {
            double avg = array.Average();

            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => (x.v < avg))
                .Select(x => x.i)
                .ToArray();
        }
    }
}

```

Файл TextFilters.cs

```

using System.Linq;

namespace FiltersLib
{
    class TextFilters
    {
        public static int[] Equals(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                .Where(x => x.v.Equals(param))
                .Select(x => x.i)
                .ToArray();
        }
    }
}

```

```

        public static int[] NotEquals(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => !x.v.Equals(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] StartWith(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => x.v.StartsWith(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] EndsWith(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => x.v.EndsWith(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] Contains(string[] array, string
param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => x.v.Contains(param))
                        .Select(x => x.i)
                        .ToArray();
        }

        public static int[] NotContains(string[] array,
string param)
        {
            return array.Select((value, index) => new { v =
value, i = index })
                        .Where(x => !x.v.Contains(param))
                        .Select(x => x.i)
                        .ToArray();
        }
    }
}

```

Приложение В

(Обязательное)

Программные средства форматирования, фильтрации и сортировки данных для
табличного процессора

Презентация

					ПГУ 2.090301.39.001 ПЗ	Лист
						145
Изм.	Лист	№ докум.	Подпись	Дата		

Программные средства форматирования, фильтрации и сортировки данных для табличного процессора

Автор работы

студент гр. 18BB2 Сперанский Дмитрий Денисович

Научный руководитель:

к.т.н., доцент, Дорошенко Иван Николаевич

Пенза, 2022

Цель и задачи работы

Цель работы – разработка программных средств для табличного процессора «Ведомость», позволяющих пользователю выполнять форматирование, фильтрацию и сортировку данных, а также проектирование для них графического интерфейса.

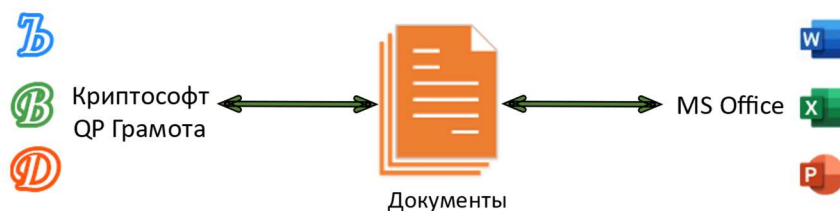
Объекты работы:

1. Модуль форматирования данных . Задачи:
 - Форматирование данных в табличном процессоре
 - Редактирование XML файлов в соответствии с полученным результатом
2. Модуль сортировки и фильтрации данных . Задачи:
 - Сортировка данных в табличном процессоре.
 - Фильтрация данных в табличном процессоре .
 - Редактирование XML файлов в соответствии с полученным результатом

2

Офисный пакет QR Грамота

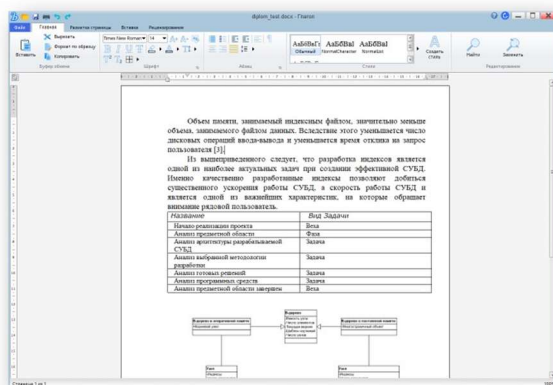
- Текстовый процессор Глагол ;
- Табличный процессор Ведомость ;
- Менеджер презентацией Доклад ;



3



Глагол



- Форматы: DOC, DOCX;
- Текст;
- Списки;
- Таблицы;
- Рисунки;
- Оглавление;
- Колонтитулы.

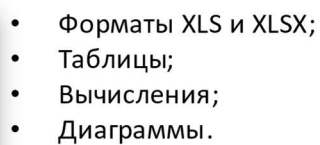
4

Изм.	Лист	№ докум.	Подпись	Дата

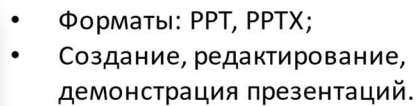
ПГУ 2.090301.39.001 ПЗ

Лист

147



5



6

Задачи модуля форматирования данных

Разрабатываемый программный модуль должен поддерживать форматирование данных согласно стандартным форматам и введенному пользователем кода формата.

Стандартные форматы:

- «Дата», «Время», «Дробный» и др.;
- Всего их 11 штук;
- Имеют базовый код, который может изменяться в зависимости от настроек пользователя.

Пример структуры кода формата приведен на рисунке ниже. Содержит 4 части:

- 1 – для положительных чисел;
- 2 – для отрицательных чисел;
- 3 – для нуля;
- 4 – для текста.


#,###.00_); [Red] (#,###.00); 0.00; "gross receipts for"@

7

Задачи модуля сортировки и фильтрации данных

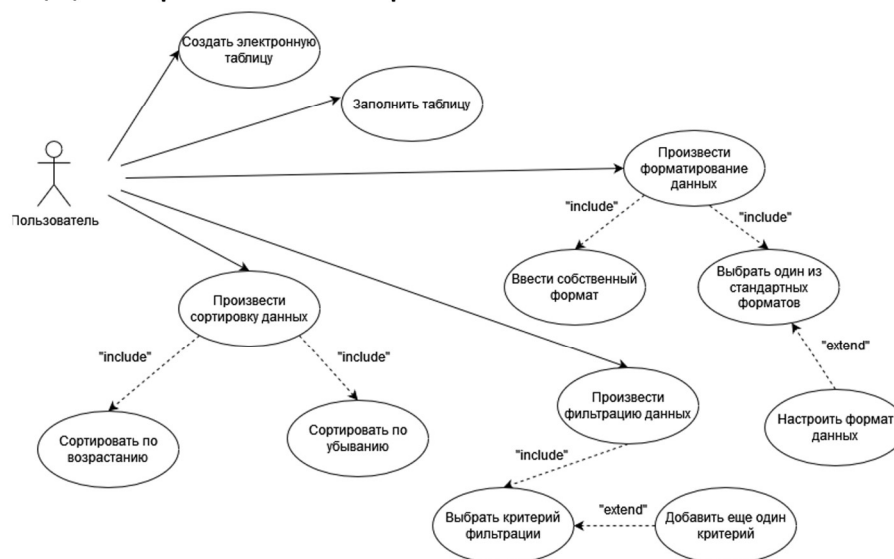
- Поддержка сортировки и фильтрации данных ;
- Поддержка всех фильтров , реализованных в других популярных табличных процессорах;
- Методы сортировки данных, также как и фильтры , должны зависеть от типа введенных пользователем данных.

Всего в табличных процессорах принято разделять три типа данных:

- Числовые;
- Текстовые;
- Дата.

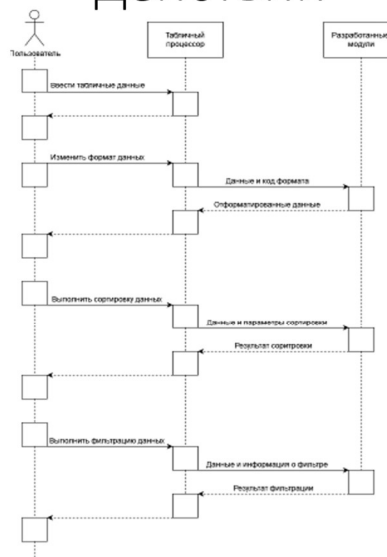
8

UML-диаграмма вариантов использования



9

UML-диаграмма последовательности действий



10

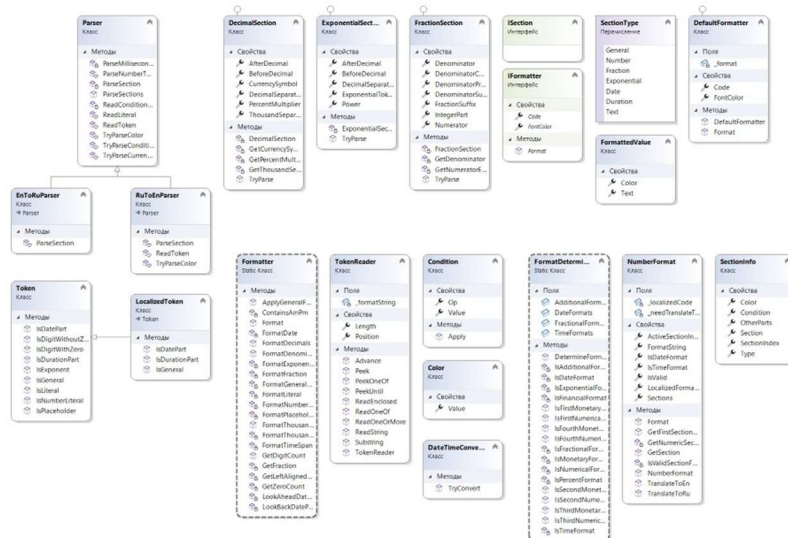
Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 2.090301.39.001 ПЗ

Лист

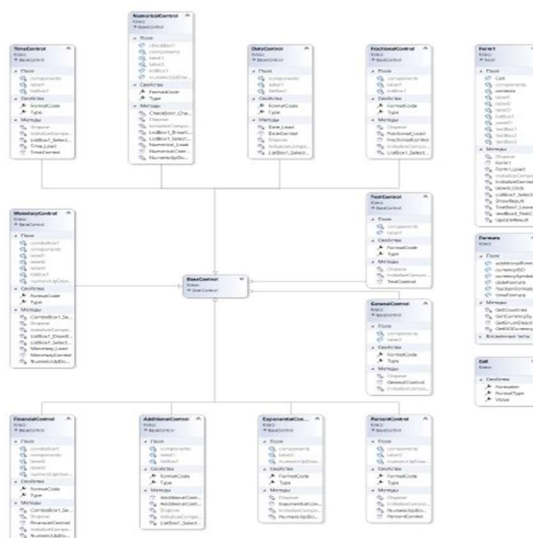
150

UML-диаграмма классов библиотеки форматирования данных



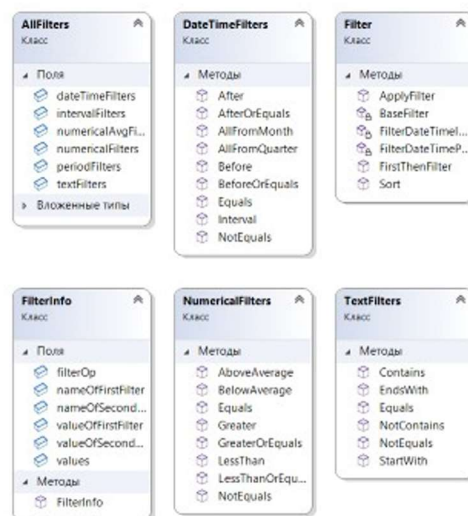
11

UML-диаграмма графического интерфейса модуля форматирования данных



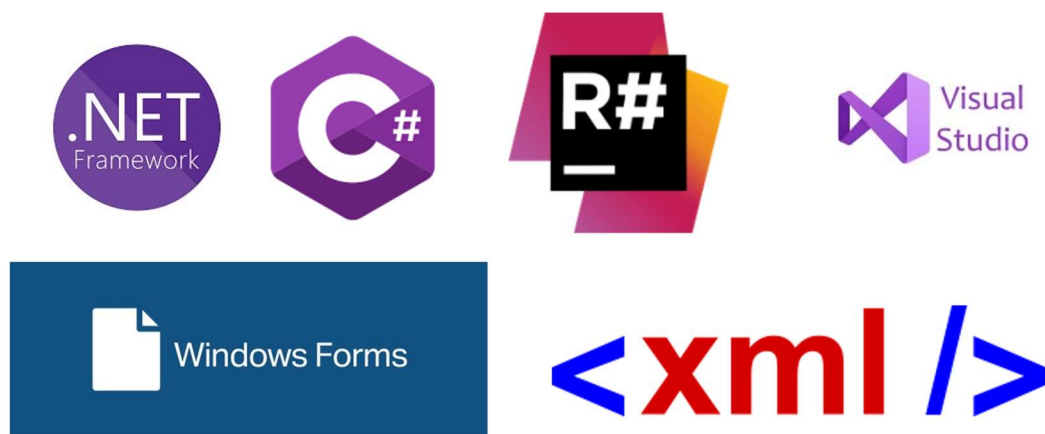
12

UML-диаграмма классов библиотеки фильтрации и сортировки данных



13

Используемые инструменты и технологии



14

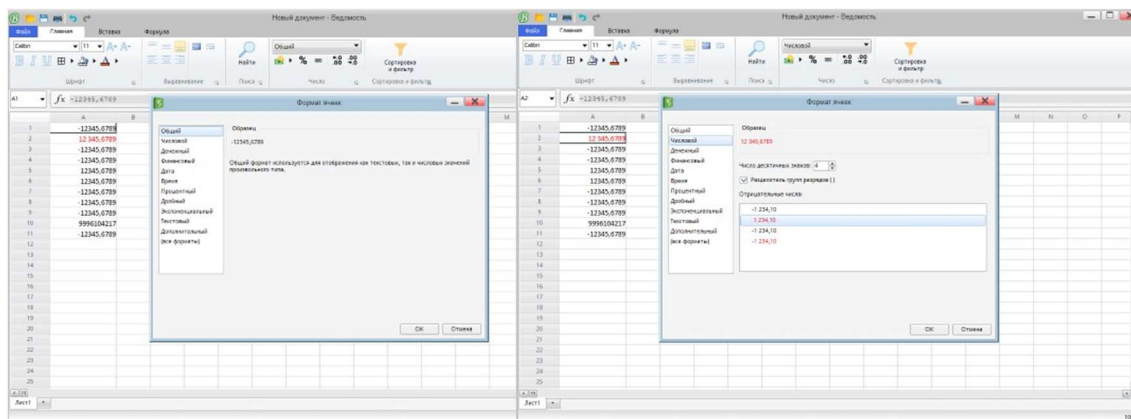
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.39.001 ПЗ

Лист

152

Формат «Общий» и «Числовой»

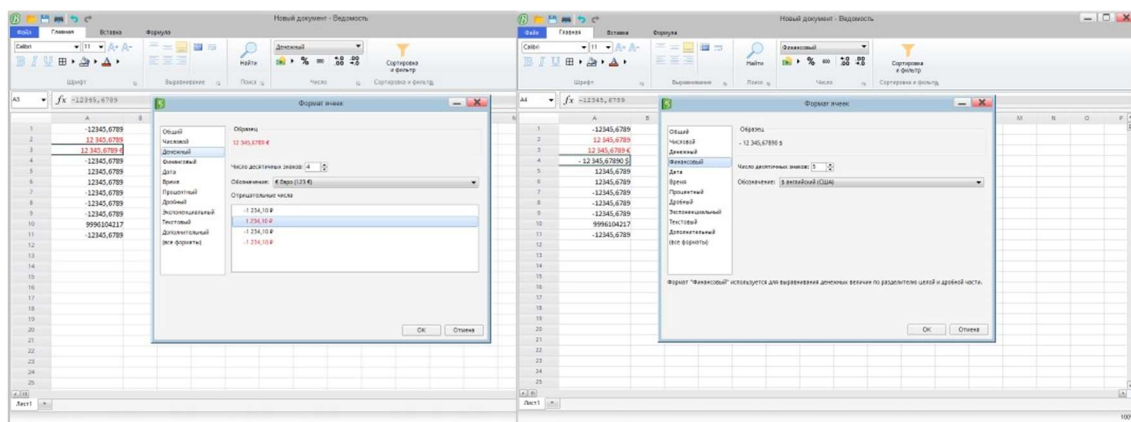


На рисунке изображен результат применения формата «Общий»

На рисунке изображен результат применения формата «Числовой»

15

Формат «Денежный» и «Финансовый»

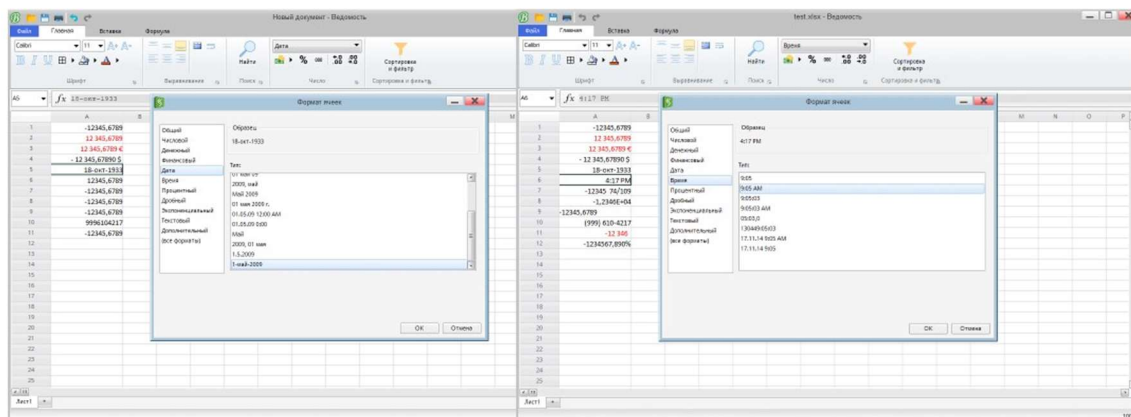


На рисунке изображен результат применения формата «Денежный»

На рисунке изображен результат применения формата «Финансовый»

16

Формат «Дата» и «Время»

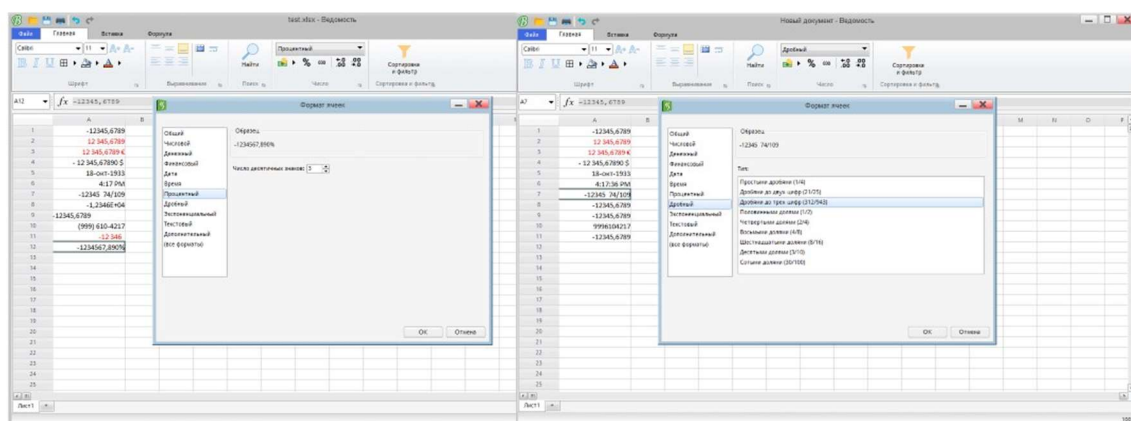


На рисунке изображен результат применения формата «Дата»

На рисунке изображен результат применения формата «Время»

17

Формат «Процентный» и «Дробный»

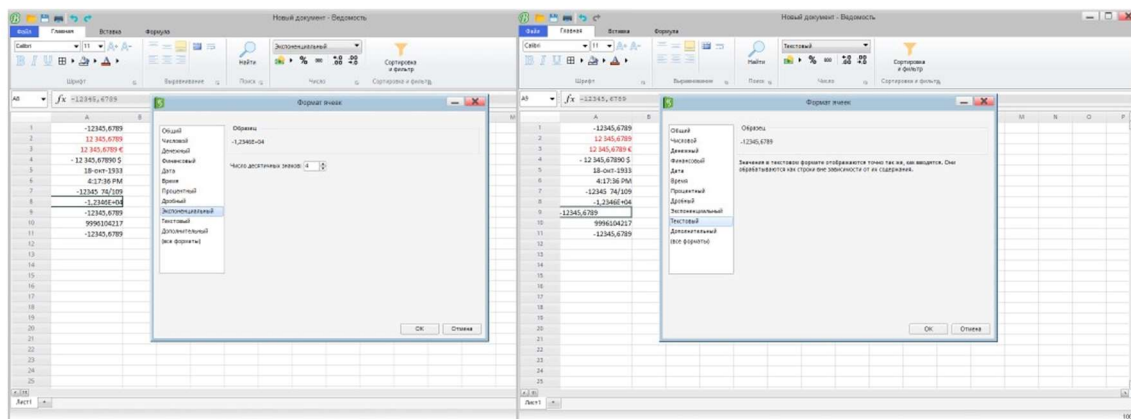


На рисунке изображен результат применения формата «Процентный»

На рисунке изображен результат применения формата «Дробный»

18

Формат «Экспоненциальный» и «Текстовый»

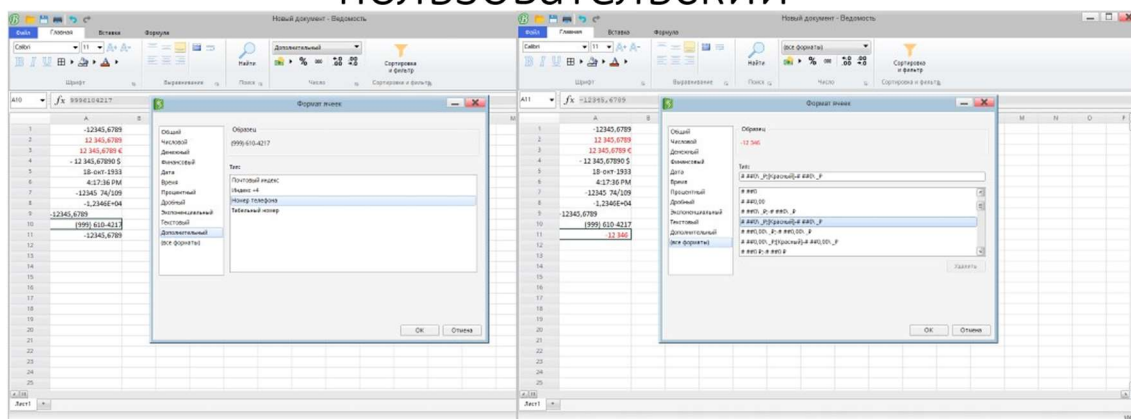


На рисунке изображен результат применения формата «Экспоненциальный»

На рисунке изображен результат применения формата «Текстовый»

19

Формат «Дополнительный» и пользовательский

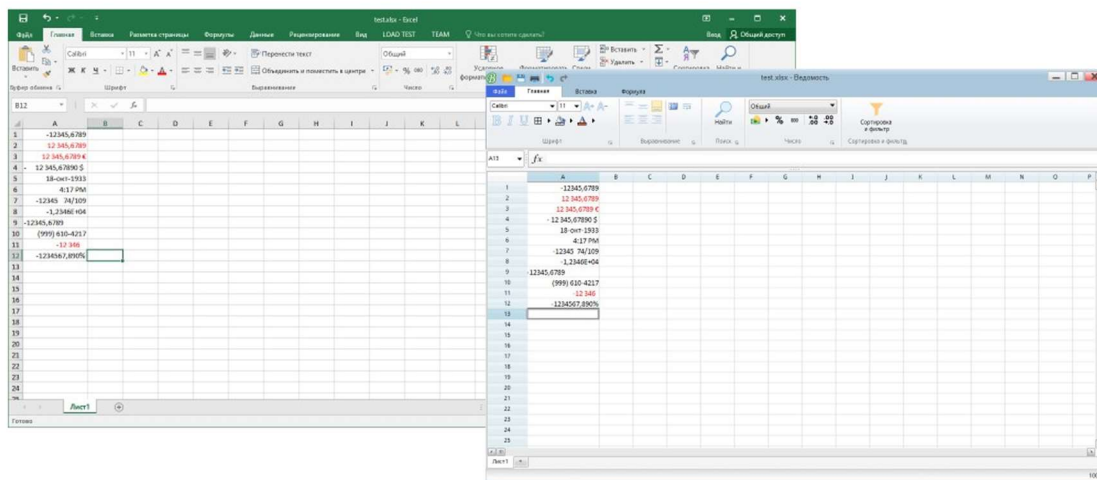


На рисунке изображен результат применения формата «Дополнительный»

На рисунке изображен результат применения пользовательского формата

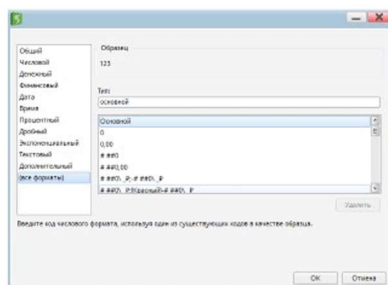
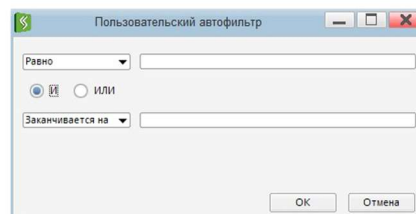
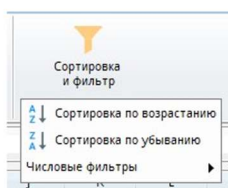
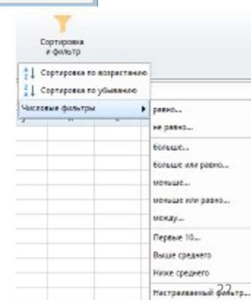
20

Сравнение результата с Microsoft Excel

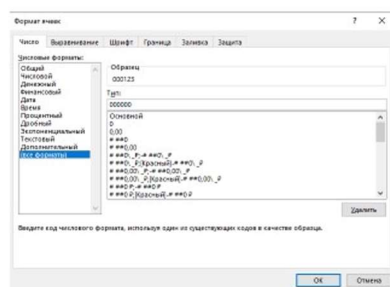
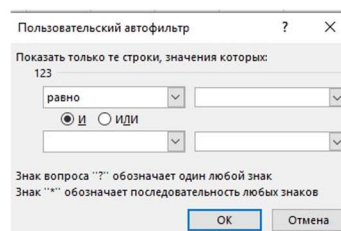
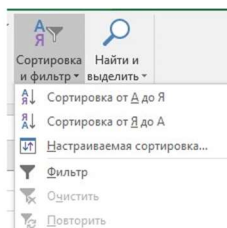
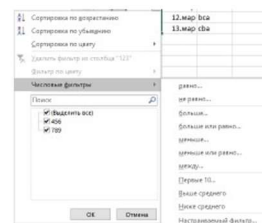


21

Разработанные модули

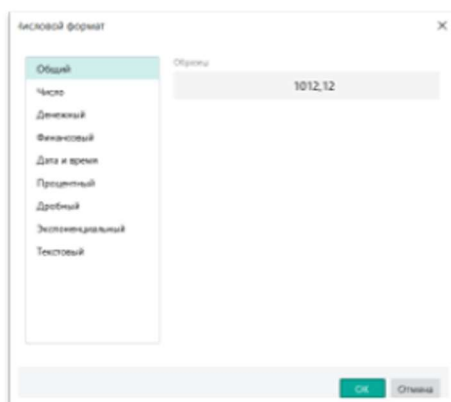
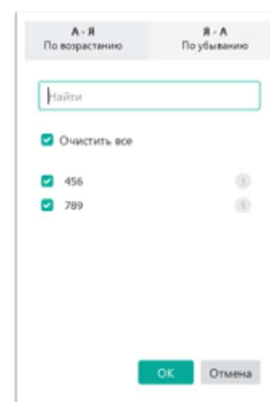


Модули из Microsoft Excel



23

Модули из Мой Офис Таблица



Окно пользовательского автофильтра и списка с фильтрами для каждого типа данных отсутствуют, так как данный функционал не реализован

24

Сравнительные характеристики

Критерии	Табличный процессор		
	Ведомость	Microsoft Excel	Мой Офис «Таблица»
Поддержка ввода пользовательского формата данных	+	+	-
Количество стандартных форматов данных	11	11	9
Поддержка сортировки различных типов данных	+	+	+
Уникальные фильтры для различных типов данных	+	+	Только ручная выборка
Поддержка комбинирования двух фильтров	+	+	-

25

Заключение

Поставленные для выполнения ВКР задачи были полностью решены. Были разработаны библиотеки форматирования, фильтрации и сортировки данных для табличного процессора. Модуль форматирования данных был интегрирован в редактор электронных таблиц «Ведомость», разработкой которого занимается ООО НТП «Криптософт».

В дальнейшем разработанные программные средства можно улучшить, путем добавления поддержки различных языков в модуль форматирования данных, реализовать больший набор стандартных форматов и более широкий набор фильтров.

26

Спасибо за внимание!

					ПГУ 2.090301.39.001 ПЗ	Лист
						159
Изм.	Лист	№ докум.	Подпись	Дата		