СОДЕРЖАНИЕ

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

ОС – Операционная система

ООП – Объектно-ориентированное программирование

DLL – Dynamic Link Library – Динамически подключаемые библиотеки

ПО – Программное обеспечение

ВВЕДЕНИЕ

Современные студенты часто сталкиваются с проблемой быстрого поиска различных терминов в научной и учебной литературе. Многие монографии и учебники содержат в себе предметные указатели, однако они бывают на столько объемными, что поиск термина по ним все равно занимает достаточно много времени.

Чтобы ускорить процесс поиска терминов, была создана программа, обеспечивающая быстрый и удобный поиск. Также программный продукт оснащен дополнительными средствами работы с предметными указателями, такие как добавление нового термина или изменение информации об имеющимся термине.

Данная программа будет полезна не только студентам, но и другим категориям лиц, так или иначе контактирующими с литературой специального назначения.

1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1.1 Введение

Составленное техническое задание по дисциплине “Структуры и алгоритмы обработки данных” является документом к курсовой работе, который отражает все этапы разработки программного продукта, а также процесс проектирования и выявление требований, предъявляемых конечному продукту.

1.1.1 Наименование программы

Название данного приложения четко отражает его суть и назначение: «Предметный указатель». Данное название целиком и полностью описывает функционал будущей программы.

1.1.2 Краткая характеристика области применения программы

Программа предназначена в первую очередь для использования учащимися школ для поиска номеров страниц учебной литературы, на которых упоминаются искомые термины.

1.2 Основания для разработки

Основанием для разработки программного продукта является курсовая работа по дисциплине "Структуры и алгоритмы обработки данных", предусмотренная учебным планом направления подготовки 09.03.04 "Программная инженерия" профиля "Разработка программных продуктов и проектирование информационных систем".

1.3 Назначение разработки

Приложение «Предметный указатель» позволит пользователям без труда, не перелистывая полностью учебную литературу, находить необходимые

термины и номера страниц, на которых они упоминаются. Приложение ориентированно на обучающихся школ, студентов, преподавателей и других социальных групп, использующих научную и учебную литературу.

1.4 Требования, предъявляемые к программе

Предметный указатель – это алфавитный список терминов документа с указанием страниц, на которых они упоминаются. Пример элемента указателя: {[заголовочный файл], 108, 189, 927}.

1.4.1 Требования к функциональным характеристикам программы

Приложение должно реализовать следующий функционал:

1) создание пустого предметного указателя;

2) добавление термина в предметный указатель;

3) редактирование элемента предметного указателя (добавление ссылки на страницу для заданного термина);

4) вывод элементов указателя в алфавитном порядке на экран;

5) вывод элементов указателя в алфавитном порядке в текстовый файл;

6) поиск элемента предметного указателя по термину.

Кроме того, в демонстрационной программе необходимо предусмотреть ввод списка терминов с клавиатуры, считывание текста, для которого осуществляется поиск терминов, из текстового файла и запись предметного указателя в другой текстовый файл перед завершением работы программы.

1.4.2 Требования к техническим средствам, необходимые при работе программы

Для обеспечения работоспособности приложения «Предметный указатель» на персональный компьютер пользователя должна быть установлена операционная система Windows (рекомендуется Windows 10). Также

необходимо наличие 22,3 МБ свободной памяти, компьютер должен быть оснащен графическим адаптером (видеокартой).

1.4.3 Требования к языкам программирования и среде разработки приложения

Для разработки программы «Предметный указатель» использовался язык программирования C++ и интегрированная среда разработки QT Creator. Для разработки графического интерфейса пользователя применялся механизм Qt.

1.4.4 Требования к информационным структурам на входе и выходе программного продукта

Входные данные могут представлять из себя как данные, передаваемые пользователем посредством ввода с клавиатуры (типа QString), так и данные, получаемые приложением из файла. Программа должна предоставлять возможность считывания данных их полей для ввода графического интерфейса пользователя.

Кроме того, выходные данные будут представлены пользователю в удобном для восприятия виде: в виде таблицы или списка (QTableWidget), в зависимости от задачи, решаемой программой в текущий момент времени.

1.5 Требования к программной документации

1) Пояснительная записка, оформленная в соответствии с локальными нормативными актами РТУ МИРЭА;

2) Проектная документация, составленная в соответствии с ГОСТ.

В процессе создания приложения вся проделанная работа документируется, должны быть сохранены все детали разработки, а также трудности, с которыми пришлось столкнуться. Всё вышеперечисленное должно быть отражено в пояснительной записке, которая прилагается проекту.

1.6 Этапы разработки

В «жизненный цикл» приложения включены следующие этапы:

1. Обзор способов организации данных и обоснование выбора структуры данных для эффективного выполнения операций: 01.10.2019 - 15.10.2019.

2. Разработка программы: 16.10.2019-30.11.2019.

3. Разработка программной документации: 01.12.2019-10.12.2019.

4. Оформление пояснительной записки: 11.12.2019-15.12.2019.

5. Защита курсовой работы: 16.12.2019-26.12.2019

Дальнейшее сопровождение программы после сдачи и защиты курсового проекта не предусмотрено.

2 ОБЗОР СПСОБОВ ОРГАНИЗАЦИИ ДАННЫХ И ОБОСНОВАНИЕ ВЫБОРА СТРУКТУРЫ ДАННЫХ ДЛЯ ЭФФЕКТИВНОГО ВЫПОЛНЕНИЯ ОПЕРАЦИЙ

Для наиболее эффективного выполнения операций необходимо выбрать подходящую структуру данных. Рассмотрим некоторые шаблоны контейнерных классов языка C++, определим их преимущества и недостатки.

Использование контейнеров позволит улучшить такие характеристики программы как надёжность, универсальность, переносимость. Кроме того, повысится быстрота разработки. Однако, использование контейнерных классов снижает быстродействие программы, но в данном случае этот параметр не является существенным, т.к. программа не будет предназначена для обработки большого количества данных.

Стандартная библиотека шаблонов (STL) содержит следующие основные типы структур данных: векторы, стеки, деки, списки и их разновидности, словари и множества.

Контейнеры можно разделить на ассоциативные и последовательные.

В ассоциативных контейнерах можно получать доступ по ключу. К таким структурам данных относятся map (словарь), set (множество), multiset (множество с дубликатами) и multimap (словарь с дубликатами).

Последовательные контейнеры позволяют хранить однотипные данные в виде непрерывной последовательности. К этому типу структур данных относятся например vector (вектор), list (список) и deque (дек).

Ниже рассмотрим подробнее некоторые структуры данных.

2.1 Последовательные контейнеры

2.1.1 Vector (вектор)

Вектор – это структура данных, эмулирующая динамический массив переменного размера, хранящий только однотипные элементы в одной области памяти. Элементы хранятся в порядке добавления и имеют свой индекс, начиная с нуля. В векторе можно добавлять и удалять элементы и с начала, и с конца.

Удаление элементов происходит наиболее эффективным способом, однако добавление нового элемента может повлечь за собой неэффективную работу программы из-за перераспределения памяти.

Вектор можно просматривать в обе стороны как с помощью цикла, так и с помощью итераторов или реверсивных итераторов. Работа с недействительными итераторами приведет к неопределенному поведению программы.

Таким образом, вектор – довольно удобная структура данных. Однако не все ее процессы работают эффективно, а это может привести к ошибкам во время работы программы.

2.1.2 Deque (дек, или двухсторонняя очередь)

Deque – структура данных, во многом похожая на vector, но устроенная более сложным способом. Доступ к элементам может осуществляться произвольно, добавление и удаление элементов производится с обоих концов дека, как и в векторе. Все основные действия (сравнение, копирование, присваивание и пр.) аналогичны операциям над вектором.

Отличие двухсторонней очереди от вектора заключается в том, что элементы дека могут храниться в разных областях памяти, а это значит, что добавление элементов происходит более эффективным способом, чем в векторе. Однако такое устройство замедляет операцию доступа к элементу, значит обход дека будет осуществляться медленнее чем обход вектора.

2.1.3 List (список)

List – шаблонный класс, реализованный в виде двусвязного списка, и, в отличие от предыдущих контейнеров, не предоставляет возможность произвольного доступа к элементам.

Списки поддерживают основные операции аналогичные операциям над деками (сравнение, присваивание, копирование, наличие итераторов). В данной структуре данных удаление и добавление элементов происходит эффективнее чем в векторе (процесс удаления и добавления основан на работе с указателями).

Недостатком контейнера list является необходимость просматривать последовательно все элементы с помощью итераторов для доступа к конкретному элементу.

2.2 Ассоциативные контейнеры

2.2.1 Map (словарь)

Map (словарь) – структура данных, построенная на основе пар значений (pair) и работающая по принципу *ключ – значение*. Ключи используются для идентификации ячеек, аналогично индексам в массиве или векторе. Однако ключи могут быть не только целыми числами, они могут принадлежать к любому другому типу данных, начиная от элементарных типов и заканчивая классами. Значение элемента можно изменить напрямую.

Операции обращения, добавления и исключения имеют логарифмическую сложность (т.е. выполняются за время равное log(n), где n – размер контейнера).

Архитектура данного контейнера такова, что элементы в нем всегда отсортированы в соответствии с функцией сравнения, т.е. пара, добавляемая в словарь, сразу попадет на «свое» место в списке. Таким образом, словарь позволяет освободиться от дополнительных сортировок. Еще одной особенностью map является невозможность добавлять элементы с одинаковыми ключами (в словаре не может быть дубликатов, все ключи уникальны).

Итак, map позволяет хранить пары, имеющие уникальные ключи, при этом все пары упорядочены по ключам, что очень удобно.

2.2.2 Multimap

Multimap – словарь с дубликатами, т.е. это словарь, позволяющий хранить элементы с одинаковыми ключами. Из-за этого для multimap определена операция доступа по индексу.

Элементы multimap отсортированы по значениям ключей в соответствии с заданной функцией сравнения. *Дубликаты* в этой структуре данных хранятся в том порядке, в котором их добавляли. При удалении элемента по ключу из словаря удаляются все элементы с этим ключом.

2.2.3 Set (множество)

Set (множество) – ассоциативный контейнер, построенный на основе списков, в котором значения элементов уникальны и служат как значения ключей, по которым данные сортируются в соответствии с функцией сравнения.

В данной структуре данных не предусмотрено изменение элемента-ключа напрямую, его придётся заменять новым (т.е. удаляется старый элемент, а на его место помещается новый). Кроме того нельзя обращаться к произвольным элементам множества, поэтому нужно использовать итераторы.

2.2.4 Multiset

Multiset – множество с дубликатами. Свойства этого контейнера аналогичны свойствам простого множества, за исключением уникальности ключей (множество с дубликатами поддерживает наличие одинаковых ключей в коллекции).

2.3 Вывод

Подробно рассмотрев особенности каждого контейнера, можно прийти к выводу, что наиболее удобной структурой данных для данного приложения будет контейнер map (словарь).

Словарь обеспечит эффективное добавление и удаление записей из предметного указателя. Кроме того, все записи автоматически будут упорядочены в алфавитном порядке по ключам, что позволит освободиться от сортировок перед выводом списка на экран или в файл. Еще одним преимуществом словаря в контексте данного приложения является поиск по ключу, что очень удобно для осуществления поиска в предметном указателе по термину.

3 ОПИСАНИЕ ПРОГРАММЫ

3.1 Общие сведения

В ходе выполнения курсовой работы было создано приложение, позволяющее работать с предметным указателем. Приложение содержит сами списки терминов, а также предоставляет инструменты для работы с ними, обозначенные требованиями технического задания.

3.1.1 Наименование программы

Название программы, "Предметный указатель", формировалось, исходя из ее назначения. Наименование описывает структуру данных и вид данных, над которой она призвана проводить вышеописанные операции.

3.1.2 Программное обеспечение, необходимое для правильного функционирования программы

Для корректного функционирования программы необходимо, чтобы на компьютер была установлена ОС Windows (рекомендуемая ОС – Windows 10, данная версия ОС гарантирует корректную работу всех функций), наличие 22.3 МБ свободной памяти (архив для загрузки содержит все необходимые DLL). Необходимо, чтобы компьютер был оснащен графическим адаптером (видеокартой) для отображения графического интерфейса приложения. Для начала работы программы достаточно загрузить архив, содержащий DLL и .exe файл, извлечь его и запустить .exe файл.

3.1.3 Язык программирования, на котором написана программа

Язык программирования для выполнения курсовой работы избирался исходя из требований, предъявленных к приложению, а также исходя из возможностей сред разработки, поддерживающих тот или иной язык программирования.

Таким образом, для написания программы, требования к которой определены техническим заданием, был выбран язык C++. Данный язык поддерживает такие парадигмы программирования как процедурное программирование и ООП. Кроме того, в C++ предусмотрены необходимые структуры данных, для удобной и эффективной разработки программы.

3.2 Функциональное назначение программы (классы решаемых задач и функциональные ограничения на применения)

Назначением данного программного продукта является помощь пользователю в обращении с предметными указателями к научной и учебной литературе, посредством представления сведений о терминах в удобном для восприятия виде и предоставления инструментов для работы с ними.

Программа даёт пользователю возможность работать в трех режимах: работа с существующими предметными указателями, создание нового и поиск по тексту.

В режиме работы с существующим указателем пользователь может просматривать список терминов, представленный в виде таблицы и отсортированный в алфавитном порядке, добавлять в указатель новые термины, изменять существующие термины, удалять их, а также сохранять список в текстовый файл.

Режим создания нового предметного указателя предусматривает создание пустого списка с последующим предложением открыть один из существующих предметных указателей, включая только что созданный.

Функция поиска по тексту дает пользователю возможность открывать текстовые файлы (.txt) и осуществлять в них поиск слов и их сочетаний.

Данная программа имеет ряд ограничений, на которые следует обратить внимание при ее эксплуатации. Программное обеспечение является свободно переносимым с одного устройства на другое, однако данное свойство не подразумевает возможность переноса уже сохраненных данных вместе с ПО.

Программа позволяет работать только с файлами форматов csv (хранение предметных указателей) и txt (запись указателя в текстовый файл, загрузка текста для поиска терминов).

3.3 Описание логической структуры программы

Логическая структура данной программы подразумевает использование двух различных парадигм программирования: объектно-ориентированное программирование и процедурное программирование.

ООП позволяет удобно представлять классы и данные, с которыми работают эти классы. Эта парадигма программирования очень удобна для создания графического интерфейса или отдельных механизмов работы с данными.

Вторая парадигма, процедурное программирование, применялась для определения различных вспомогательных функций и операций. Такой подход позволяет придать коду более простой для восприятия вид.

3.3.1 Алгоритмы, применяемые в программе

Ключевыми алгоритмами в программном продукте являются алгоритм поиска элементов по ключу в списке, в котором хранятся термины, алгоритм поиска заданных выражений в тексте, алгоритм для редактирования значения элемента с заданным ключом и пр.

Также важной частью приложения являются алгоритмы графической части приложения, проверки входных данных, алгоритмы представления структур данных в удобном для пользователя виде.

3.3.1.1 Начало работы приложения

При запуске программы открывается главное окно с выбором режима работы приложения. Пользователю доступно три режима: режим открытия

существующего предметного указателя, режим создания нового и режим поиска по загруженному из файла тексту.

3.3.1.2 Алгоритмы обработки существующего предметного указателя

При выборе пользователем режима работы с уже готовым списком терминов пользователю следует выбрать предметный указатель над которым ему необходимо провести те или иные операции.

Когда предметный указатель выбран, содержимое соответствующего файла загружается в программу. Каждый сегмент строки файла заносится в соответствующий элемент списка: термин заносится в ячейку ключа, а номера страниц в ячейку значения ключа. Список автоматически сортирует элементы по ключу в порядке возрастания (в алфавитном порядке). Такую структуру данных называют словарем.

Далее рассмотрим непосредственно алгоритмы работы с полученным списком.

3.3.1.2.1 Алгоритм поиска терминов

Данный алгоритм применятся для поиска термина в предметном указателе. В качестве аргумента он принимает указатель на словарь терминов.

Алгоритм получает от пользователя через графический интерфейс слово, по которому будет осуществляться поиск. Если пользователь не ввел слово (т.е. не оставил окошко для ввода искомого термина незаполненным) и нажал кнопку поиска, то приложение уведомит его о том, что нужно заполнить поле ввода термина для успешного осуществления поиска.

В противном случае, если алгоритм все же получил термин для поиска, он создает переменную flag логического типа, по которой в конце работы алгоритма можно будет судить об успешности поиска. Затем запускается цикл по словарю терминов. В данном цикле перебираются все ключи списка по порядку и сравниваются с введенным пользователем словом. В случае совпадения ключа

на текущей итерации с поисковым словом логическая переменная flag меняет свое значение на противоположное, а найденный элемент списка вносится в таблицу с результатами поиска.

После прохода цикла по списку оценивается значение переменной flag. Изменение ее значения говорит о том, что интересующий пользователя элемент словаря найден. Соответственно, если значение этой переменной не поменялось, значит искомого термина в предметном указателе нет. В таком случае пользователь увидит на экране сообщение о том, что заданный термин не найден. В ином случае, если значение flag поменялось на противоположное, на экран выводится таблица, содержащая одну строку (ключи в списке терминов уникальны) с результатом поиска. Для наглядного пояснения работы данного алгоритма ниже приведена его блок-схема (рисунок 1).

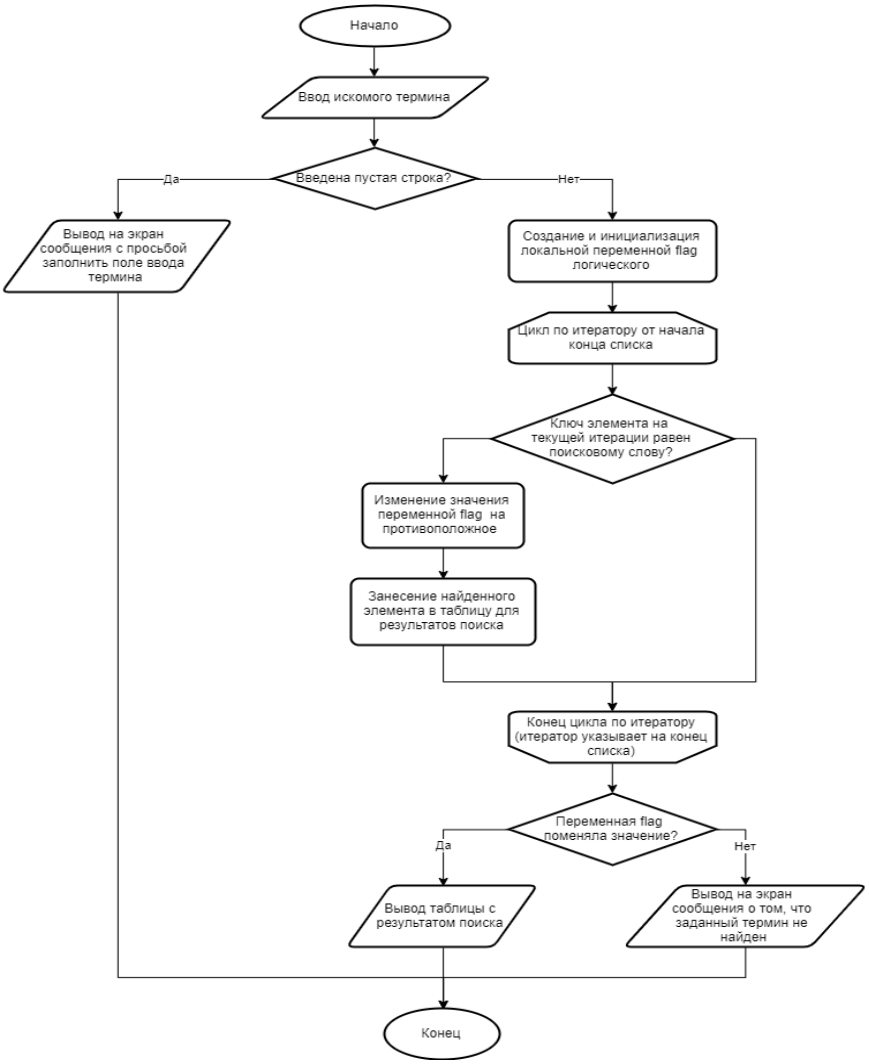

Рисунок 1 – Блок-схема алгоритма поиска элементов в предметном указателе по термину

3.3.1.2.2 Алгоритм редактирования записей предметного указателя

Данный алгоритм не предусматривает изменения самого термина, т.е. ключа элемента списка, а допускает лишь изменение значения по заданному ключу. Алгоритм получает от пользователя термин, для которого он хотел бы изменить перечень страниц, на которых он встречается.

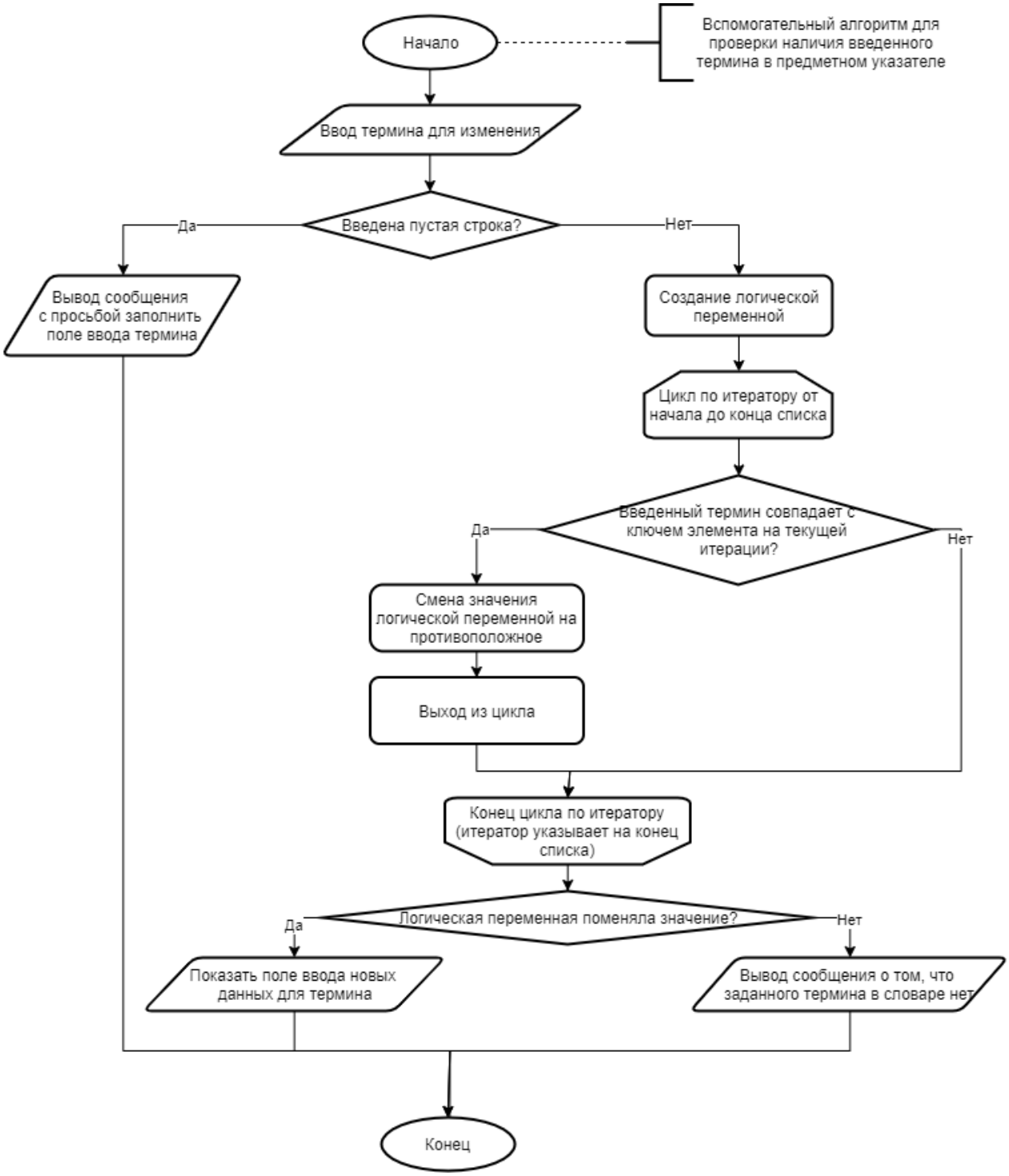

Рисунок 2 – Блок-схема вспомогательного алгоритма checkInput

Затем путем перебора всех ключей в списке и их сравнения с введенным словом устанавливается наличие требуемого термина в предметном указателе. Если нужный термин не найден в списке, выводится соответствующее сообщение. В противном случае открывается доступ к новому полю ввода, в которое пользователь должен ввести новые данные для термина. Все эти подзадачи выполняются вспомогательным методом checkInput (рисунок 2), задача которого проверить правильность вводимых пользователем данных и при необходимости проинформировать его о непригодности введенных им данных для работы или, наоборот, допустить пользователя к следующему этапу алгоритма.

После того как выше упомянутый метод предоставил возможность продолжить изменение перечня страниц для нужного термина, пользователь вводит новые данные о термине в открывшиеся для заполнения поля.

Если пользователь ничего не ввел, но нажал кнопку «Изменить», алгоритм выведет сообщение о том, что поля не заполнены. В ином случае алгоритм найдет в списке элемент с ключом, равным введенному, изменит значение по данному ключу на новые, предоставленные пользователем и выведет сообщение об успешном проведении операции. Ниже представлена схема общего алгоритма редактирования записей.

3.3.1.2.3 Алгоритм добавления нового термина в предметный указатель

Данный алгоритм позволяет добавить новый термин в список терминов. В случае, если данный термин уже есть в указателе, алгоритм предлагает воспользоваться функцией редактирования. Если же хотя бы одно поле для ввода данных не заполнено, на экране появляется сообщение с просьбой заполнить их.

После ввода термина, соответствующей информации о нем и нажатия кнопки добавления программа отправляет сигнал для записи нового термина в файл, где хранятся данные о терминах, для добавления в словарь. Затем обновляется таблица на экране (благодаря архитектуре выбранной структуры

данных новый термин занимает «свое» место в словаре согласно алфавитному порядку), а файл, в котором хранится информация о терминах, перезаписывается.

3.3.1.2.4 Алгоритм удаления заданного термина из словаря терминов

Данный алгоритм предназначен для удаления из предметного указателя термина, заданного пользователем. Пользователь должен заполнить поле, где необходимо указать термин, который он желает удалить из словаря.

Если пользователь не заполнил поле ввода и нажал кнопку «удалить», программа потребует заполнить поле. В ином случае, если поле все же заполнено, нажатие кнопки отправляет строку с термином в обработку.

Сначала аналогичным п. 3.3.1.2.1 (Алгоритм поиска терминов) способом проверяется, есть ли в списке элемент с таким ключом. Если в результате проверки установлено, что элемента с заданным ключом нет, то на экран выводится сообщение о том, что в предметном указателе нет термина, который пользователь желает удалить; а если искомый элемент списка нашелся, он удаляется из словаря с помощью встроенной функции, а файл, где записаны термины и информация о них, перезаписывается.

3.3.1.3 Алгоритм создания нового предметного указателя

Данный алгоритм создает пустой предметный указатель с автоматически формируемым названием.

Для создания нового предметного указателя пользователю предлагается заполнить форму, где нужно указать название монографии, ее автора и год издания. Первые два поля обязательны для заполнения. Если обязательные поля (помеченные «звездочкой» *) не заполнены, на экран выводится сообщение с требованием заполнить пустые поля.

В случае, когда поля заполнены, после нажатия кнопки «Создать» формируется строка, которая станет названием предметного указателя.

Например: Математика, 5 класс (Виленкин Н.Я., 2013). Затем открывается окно проводника для выбора директории, куда нужно сохранить предметный указатель. Путь к этой директории добавляется к имени файла, таким образом алгоритм получает полное название файла. Далее создается файл с только что полученным именем. После того, как новый предметный указатель создан, программа автоматически переходит в режим открытия существующих предметных указателей.

3.3.1.4 Алгоритм поиска по загруженному тексту

Данный алгоритм позволяет найти совпадения по загруженному из txt – файла текста. После нажатия кнопки поиска создается специальный объект, назначение которого заключается в поиске, выделении и подсчете совпадений с введенным выражением.

С помощью регулярного выражения находятся все совпадения разом. Каждому совпадению в тексте соответствует итератор. Затем в цикле по установленным итераторам нужные фрагменты текста выделяются голубым цветом, а счетчик совпадений увеличивается на единицу с каждой новой итерацией. Затем выводится сообщение со значением счетчика.

3.3.2 Структура программы с описанием составных частей, их функций и связей между ними

Для более удобной и иллюстративной работы с однотипными структурами данных в разработке данного программного продукта использовалась концепция ООП, позволяющая представить некоторые однотипные виды данных в виде классов, что делает обработку этих данных более рациональной и структурированной.

В данном приложении использовались как встроенные классы QT, так и классы, созданные в процессе разработки. Для реализации интерфейса были задействованы различные классы QT.

3.3.2.1 Класс MainWindow

Данный класс унаследован от класса QMainWindow, который представляет из себя главное окно приложения.

Ниже представлена UML-диаграмма (рисунок 3) и подробное описание данного класса.

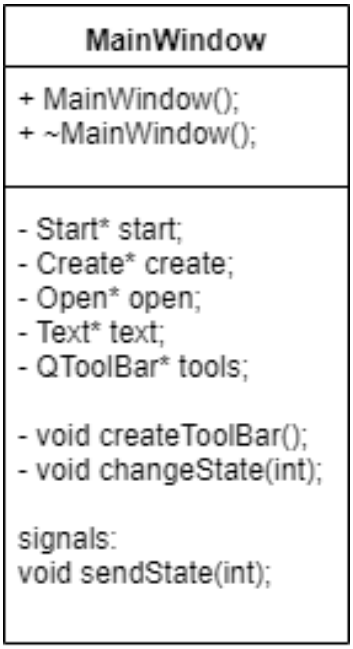

Рисунок 3 – UML-диаграмма класса MainWindow

Все поля скрыты от внешнего вмешательства модификатором доступа private.

Как уже было упомянуто выше, приложение поддерживает работу в нескольких режимах: начало (домашняя страница), создание нового предметного указателя, открытие уже существующего словаря и поиск по загруженному тексту.

Каждому из этих состояний соответствует целая константа, определенная в перечислении (enum) для удобства использования. Начальному состоянию соответствует элемент перечисления START (2), режиму открытия предметного указателя – OPEN (0), режиму создания пустого словаря – CREATE (1), и наконец, режиму поиска по тексту соответствует TEXT (3).

Также каждому состоянию соответствует поле данного класса:

1) Start* start – начальное состояние

2) Create* create – режим создания пустого предметного указателя

3) Open* open – режим открытия существующего предметного указателя

4) Text* text – режим поиска по тексту

Все выше описанное используется для реализации механизма переключений между состояниями.

За смену состояний отвечает метод данного класса void changeState(int). В качестве параметра он принимает целое число, которое определяет состояние, в которое необходимо перейти. Для этого и используется вышеописанное перечисление.

В зависимости от числа, поступившего в метод, исполняется определенной блок кода, который изменяет внешний вид главного окна в соответствии с нужным состоянием.

Ниже приведена блок-схема метода void changeState(int) (рисунок 4).

Все классы, использующиеся как типы полей в данном классе, будут описаны далее в следующих разделах.

Метод void createToolBar() создает панель инструментов, необходимых для работы с существующим предметным указателем. Данный метод вызывается только при переходе в режим работы с уже существующим предметным указателем.

Конструктор класса устанавливает форму, созданную в QT Designer, и предустанавливает состояние START, испуская сигнал sendState(int), который, в свою очередь, принимает метод changeState(int).

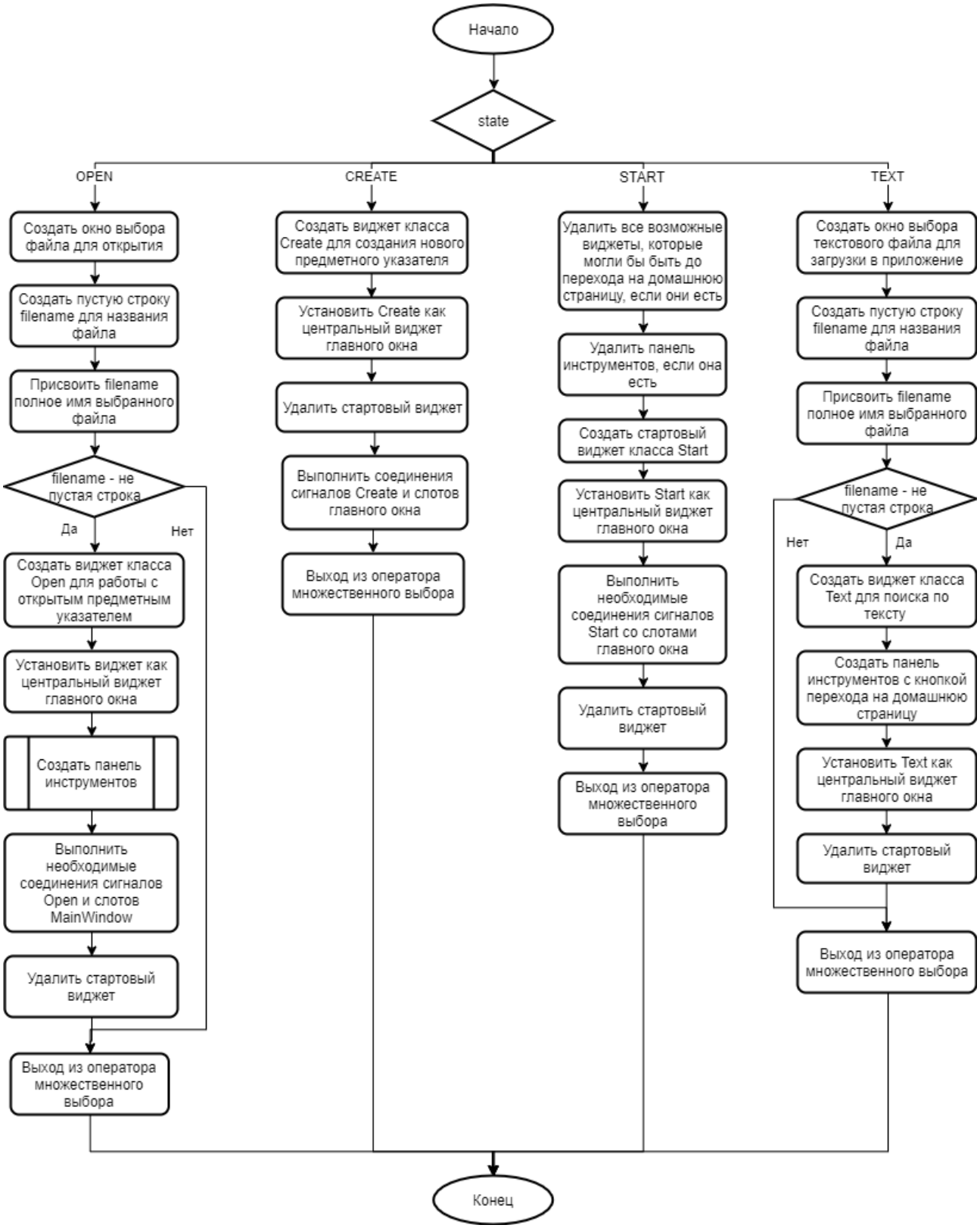

Рисунок 4 – Блок-схема метода changeState, реализующего смену режимов в приложении

3.3.2.2 Класс Start

Данный класс унаследован от встроенного класса QT QWidget и определяет внешний вид и функционал главного окна в начальном состоянии, когда программа только запущена или, когда пользователь перешел на домашнюю страницу. Далее рассмотрим диаграмму данного класса (рисунок 5).

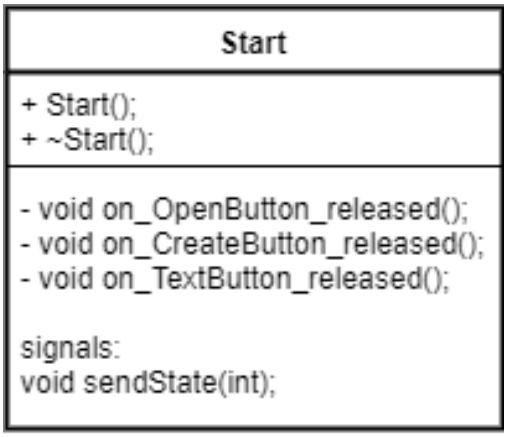

Рисунок 5 – UML-диаграмма класса Open

Конструктор класса Start устанавливает интерфейс, созданный с помощью QT Designer, и программно дополняет его. Графический интерфейс класса содержит три кнопки: «Открыть», «Создать» и «Загрузить текст». Нажатие каждой из этих кнопок запускает переход к другому состоянию. Т.е. при нажатии кнопки «Открыть» пользователь перейдет в режим открытия существующего предметного указателя, аналогичным образом работают и другие кнопки.

Поведение кнопок определяют слоты, обрабатывающие сигналы нажатия.

Кнопке «Открыть» соответствует слот void on_OpenButton_released(), кнопке «Создать» - void on_CreateButton_released(), кнопке «Загрузить текст» - void on_TextButton_released(). Эти методы отправляют сигналы sendState(int) с соответствующим значением состояния. Затем эти сигналы обрабатываются методом changeState главного окна (п. 3.3.2.1 Класс MainWindow).

3.3.2.3 Класс Open

Класс Open также унаследован от класса QWidget. Этот класс реализует работу с предметным указателем и предоставляет инструменты для этого.

Далее рассмотрим UML-диаграмму класса Open.

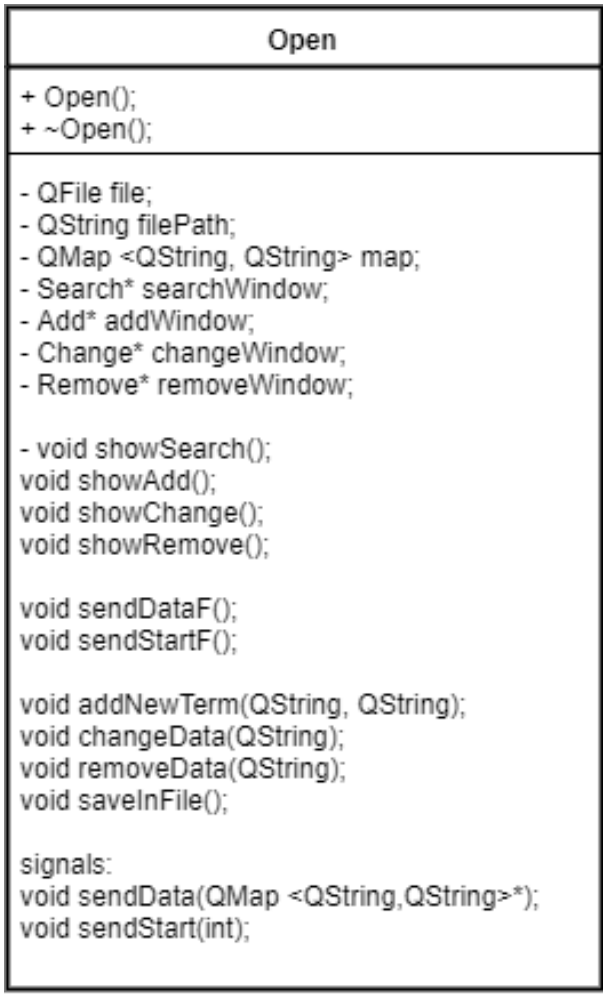

Рисунок 6 – UML-диаграмма класса Open

Рассмотрим поля данного класса:

1) QFile file – поле класса QFile, содержащее csv-файл, из которого считываются данные предметного указателя.

2) QString filePath – поле строкового типа, содержит полный путь к csv-файлу.

3) QMap <QString, QString> map – поле класса QMap. Описание данной структуры данных можно найти в п. 2.2.1. Поле используется для хранения и проведения различных операций над записями предметного указателя. Данные хранятся в виде пар «ключ-значение», где ключ – термин, а значение – строка с номерами страниц, где содержится данный термин.

4) Search* searchWindow – указатель на объект класса Search. Объект отвечает за осуществление поиска термина в предметном указателе.

5) Add* addWindow – отвечает за добавление нового термина в словарь.

6) Change* changeWindow – реализует функцию редактирования записей предметного указателя.

7) Remove* removeWindow – реализует удаление элемента словаря по заданному ключу.

Классы Search, Add, Change и Remove описаны в последующих разделах. В целом они представляют собой окна небольшого размера, которые выполняют вышеописанные функции.

Методы showSearch, showAdd, showChange и showRemove работают по одному принципу. Эти методы динамически создают объект соответствующего класса, выводят окно на экран и выполняют соединения слотов класса Open с сигналами полей вышеупомянутых классов.

Сигнал sendData(QMap<QString,QString>*) необходим для предоставления доступа другим классам к списку терминов для просмотра или изменения.

Сигнал sendStart(int) используется для перехода на домашнюю страницу и принимается слотом changeState(int) главного окна.

Методы addNewTerm(QString, QString), changeData(QString), removeData(QString), saveInFile() выполняют операции добавления нового термина, изменения существующего термина, удаления и записи в файл соответственно. Данные методы реагируют на сигналы, получаемые из классов Add, Search, Remove и Change, которые позволяют программе «общаться» с пользователем.

3.3.2.4 Класс Search

Класс унаследован от класса QWidget и представляет собой вспомогательное окно небольшого размера, реализующее функцию поиска термина по словарю.

Далее перейдем к рассмотрению UML-диаграммы (рисунок 7) данного класса.

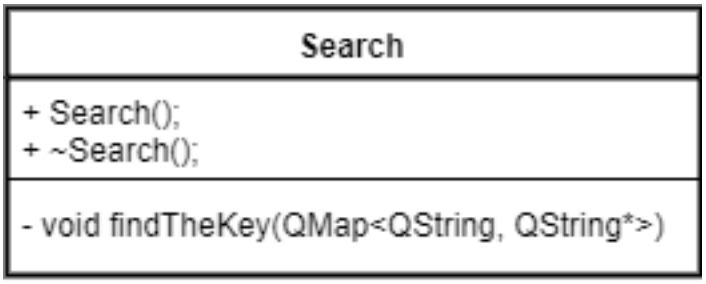

Рисунок 7 – UML-диаграмма класса Search

Экземпляр данного класса создается при нажатии кнопки «Поиск» на панели инструментов. После заполнения поля ввода и нажатия кнопки поиска в работу вступает алгоритм поиска по ключу, описанный в п. 3.3.1.2.1 (Алгоритм поиска терминов).

3.3.2.5 Класс Add

Данный класс реализует функцию добавления нового термина в предметный указатель. Также как и прошлые классы, он основан на классе QWidget и представляет из себя отдельное небольшое окошко. Ниже представлена UML-диаграмма данного класса.

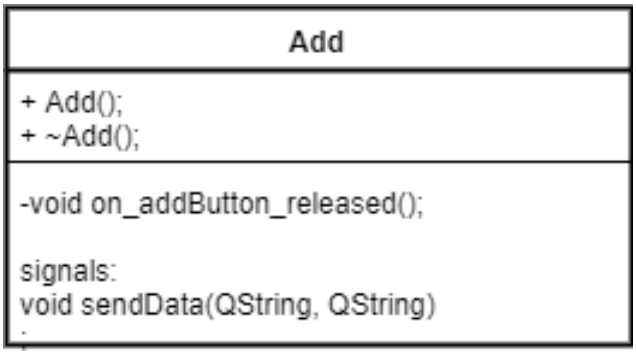

Рисунок 8 – UML-диаграмма класса Add

По нажатию на кнопку добавления термина вызывается метод void on_addButton_released(), который отправляет сигнал sendData(QString, QString) с данными введенными в поля формы класса Add.

Дальнейшая обработка сигнала проходит в соответствии с алгоритмом добавления нового термина, описанного в п. 3.3.1.2.3.

3.3.2.6 Класс Change

Этот класс реализует изменение информации термина, т.е. в паре «ключ-значение» можно поменять только значение.

Данный класс использует алгоритм, описанный в п. 3.3.1.2 (Алгоритм редактирования записей предметного указателя).

Ниже представлена UML-диаграмма данного класса.

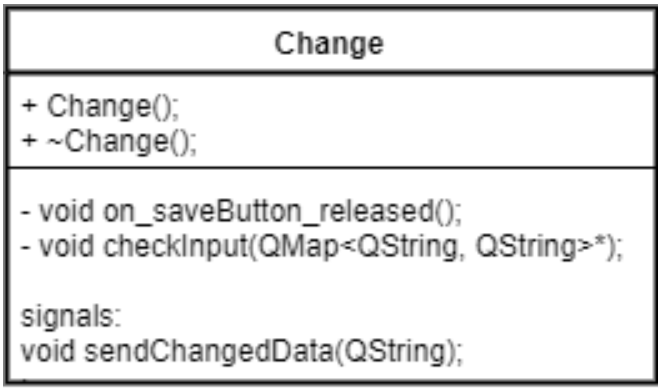

Рисунок 9 – UML-диаграмма класса Change

Сначала проверяется наличие введенного слова в предметном указателе с помощью метода chenckInput. Если проверка прошла успешно, то на форме появляется еще одна кнопка (кнопка сохранения изменений) и поле для ввода новых данных. После заполнения второго поля и после нажатия кнопки объект класса Change генерирует сигнал, который принимает Open, отвечающий за работу с предметным указателем в целом.

3.3.2.7 Класс Remove

Данный класс реализует удаление термина из предметного указателя. Алгоритм работы описан выше в п. 3.3.1.2.4 (Алгоритм удаления заданного термина из словаря терминов).

Рассмотрим UML-диаграмму данного класса.

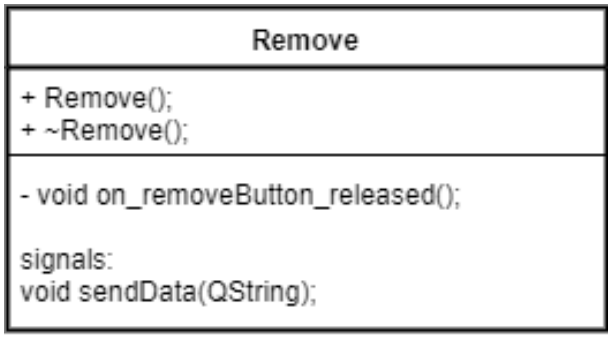

Рисунок 10 – UML-диаграмма класса Remove

on_removeButton_released() – слот, реагирующий на нажатие кнопки удаления. Данный метод отправляет слово введенное пользователем, генерируя сигнал sendData(QString).

Затем сигнал принимается слотом removeData(QString) класса Open, который удаляет данным термин непосредственно из самого списка.

3.3.2.8 Класс Create

Данный класс реализует возможность создания нового пустого предметного указателя. Рассмотрим устройство класса Create (рисунок 11).

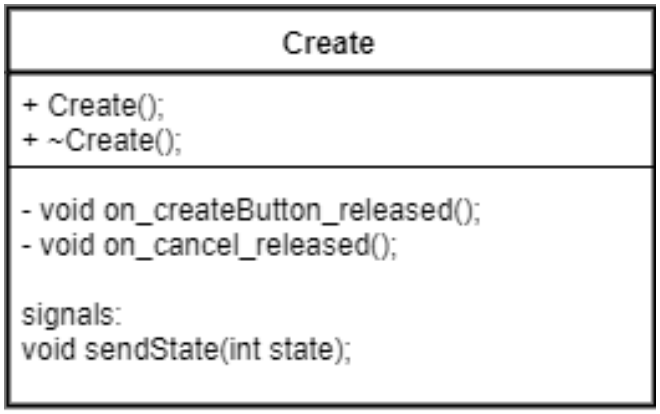

Рисунок 11 – UML-диаграмма класса Create

Принцип работы данного класса описан в разделе 3.3.1.3 (Алгоритм создания нового предметного указателя).

Метод void on_cancel_released() вызывается при нажатии кнопки «Отмена». Он генерирует сигнал sendState(int), который принимается слотом changeState главного окна. В результате программа возвращает пользователя на домашнюю страницу.

3.3.2.9 Класс Text и класс Light

Данные классы используются для загрузки текста из txt-файла и поиска по нему заданного слова или выражения. С помощью методов класса Light обнаруженные совпадения выделяются голубым цветов сразу в тексте.

Подробный алгоритм описан в п. 3.3.1.4 (Алгоритм поиска по загруженному тексту).

Ниже приведены UML-диаграммы классов Text (рисунок 12) и Light (рисунок 13).

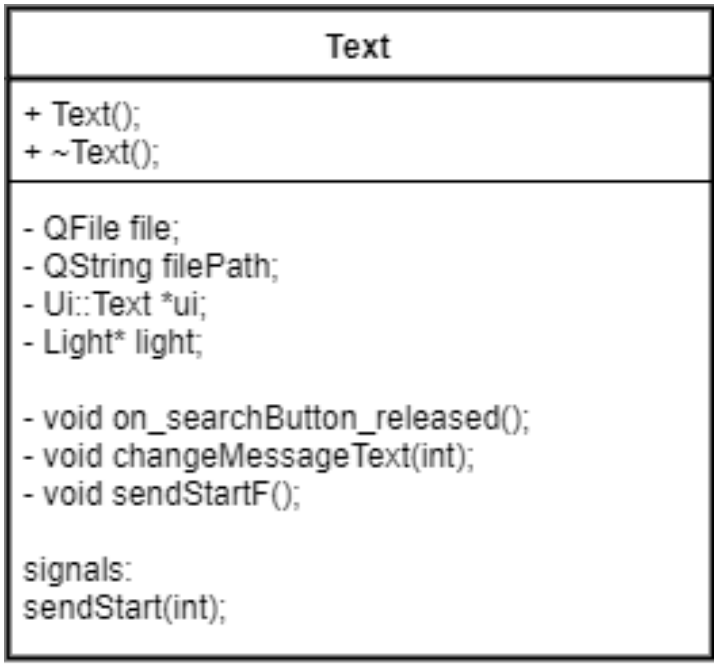

Рисунок 12 – UML-диаграмма класса Text

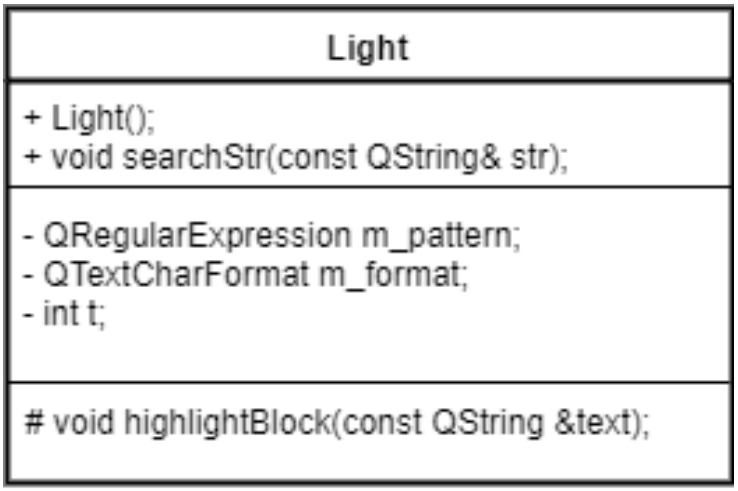

Рисунок 13 – UML-диаграмма класса Light

3.4 Технические средства, которые используются при работе программы

Для запуска программы необходимо запустить исполняемый файл, представленный пользователю в формате exe. В предыдущих разделах уже было отмечено, что для корректной работы приложения необходимо, чтобы на компьютер были установлены все необходимые DLL. Все нужные .dll-файлы находятся в одном архиве для скачивания.

В качестве статического способа хранения данных выступает файл формата «.csv». Пользователь имеет возможность открывать уже имеющиеся на компьютере файлы, редактировать их с помощью приложения, конвертировать их в формат txt, а также создавать новые файлы.

3.5 Вызов программы

При запуске приложения появляется главное окно, содержащее три кнопки для выбора одного из режимов работы: режим открытия существующего предметного указателя, режим создания нового и режим поиска по тексту.

3.5.1 Главное окно приложения, три основных режима

Как было уже сказано выше, программа имеет три режима работы. На рисунке 14 представлена главная (домашняя) страница приложения.

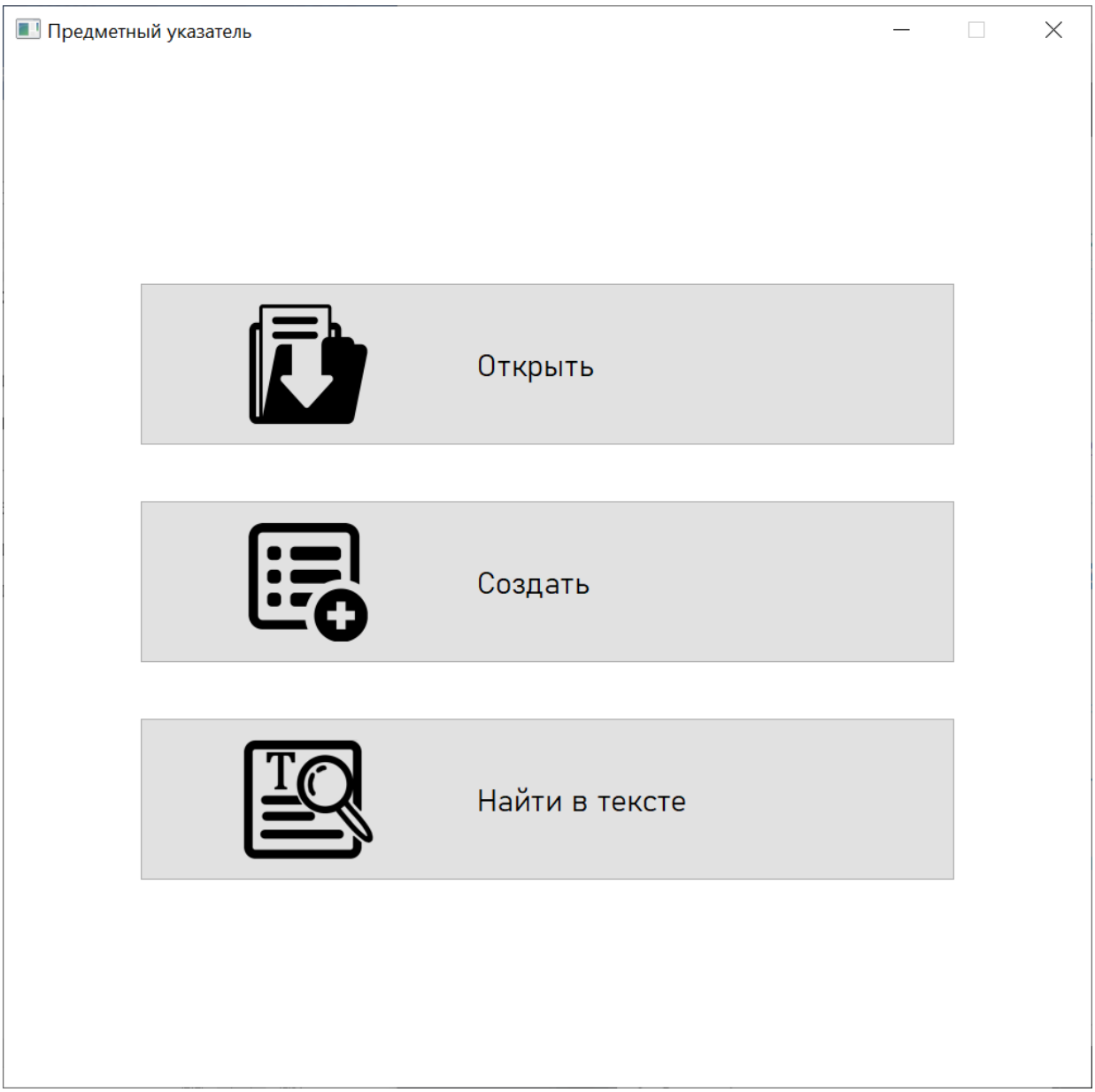

Рисунок 14 – Главная страница приложения

Страница первого режима, режима открытия существующего предметного указателя, включает в себя таблицу, в которой перечислены термины предметного указателя. В левой части окна располагается панель инструментов для работы с предметным указателем, а в верхней части указано название источника, по которому составлен указатель. Страница режима открытия существующего файла изображена на рисунке 15.

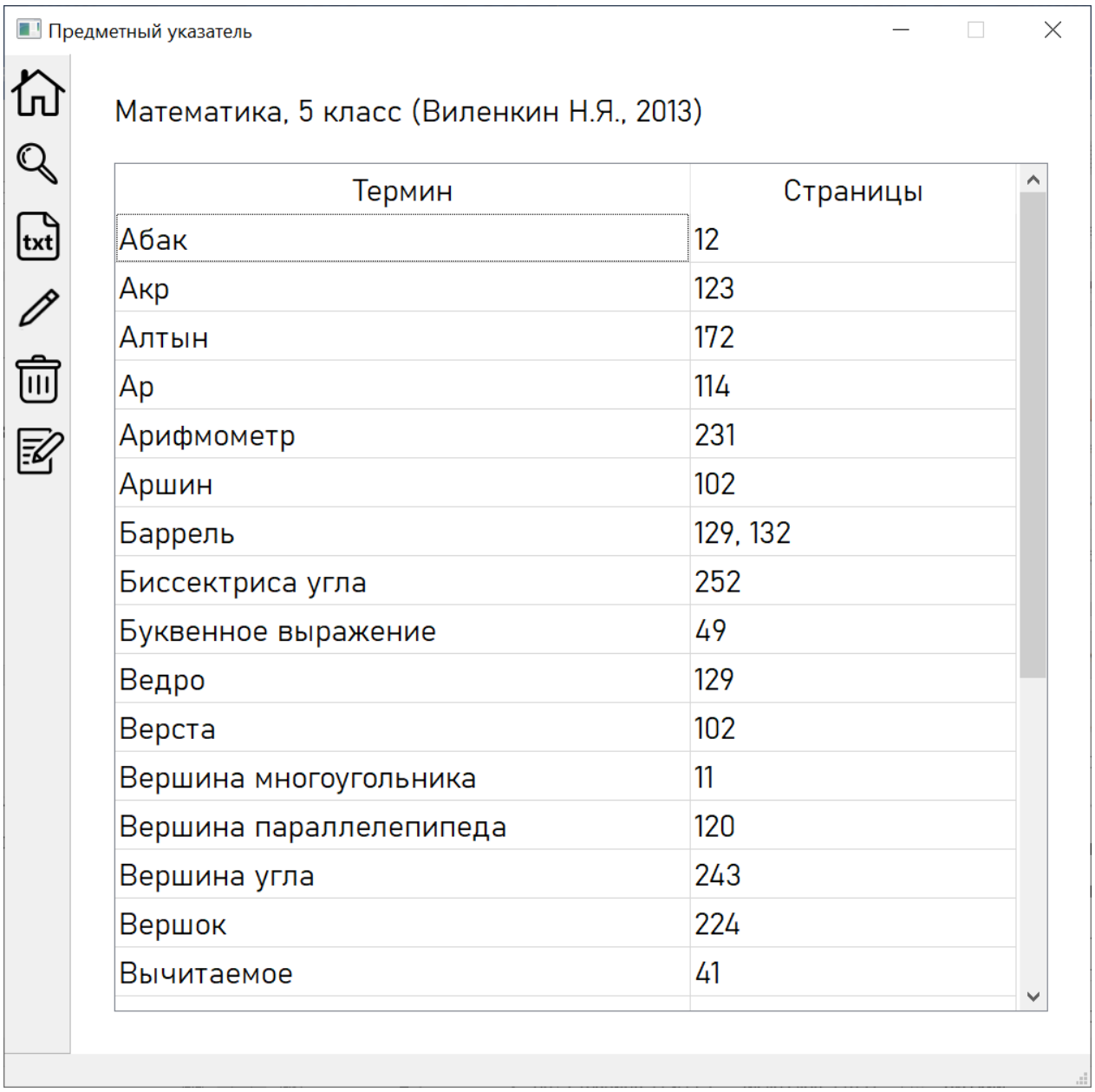

Рисунок 15 – Главная страница режима открытия предметного указателя

При переходе в режим создания нового предметного указателя появляется форма для заполнения. Форма содержит поля для ввода информации о названии монографии, ее авторе и годе издания. Также снизу страницы имеются две кнопки: «Создать» и «Отменить». Ниже, на рисунке 16 изображена страница создания пустого предметного указателя.

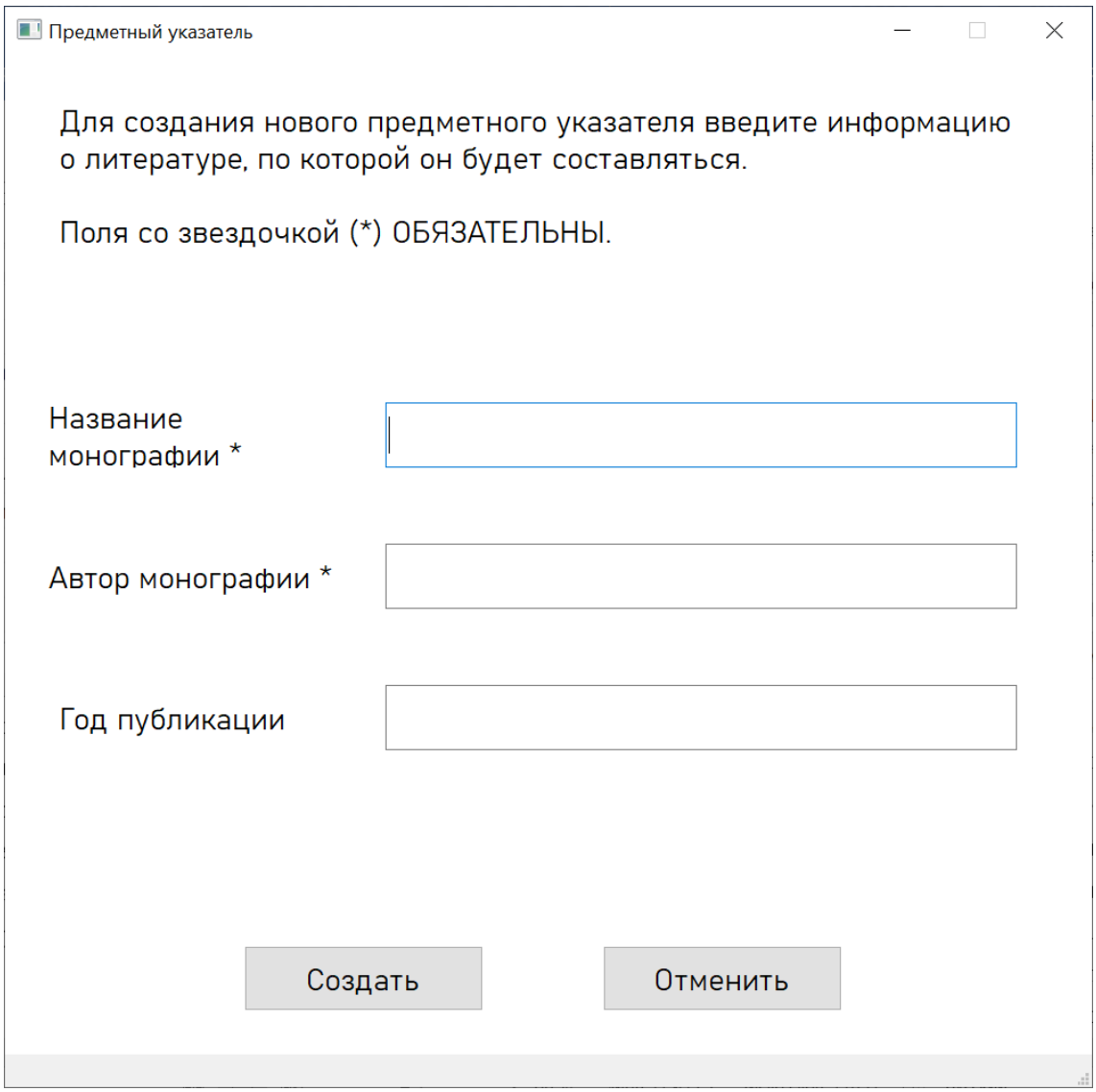

Рисунок 16 – Страница создания нового предметного указателя

При переходе в третий режим, режим поиска по загруженному тексту, появляется окно, содержащее следующие элементы: поле для ввода искомого слова, кнопка поиска, текстовое поле, где отображается загруженный текст, и кнопка перехода на домашнюю страницу.

Далее, представлен рисунок 16, на котором изображена страница поиска по тексту.

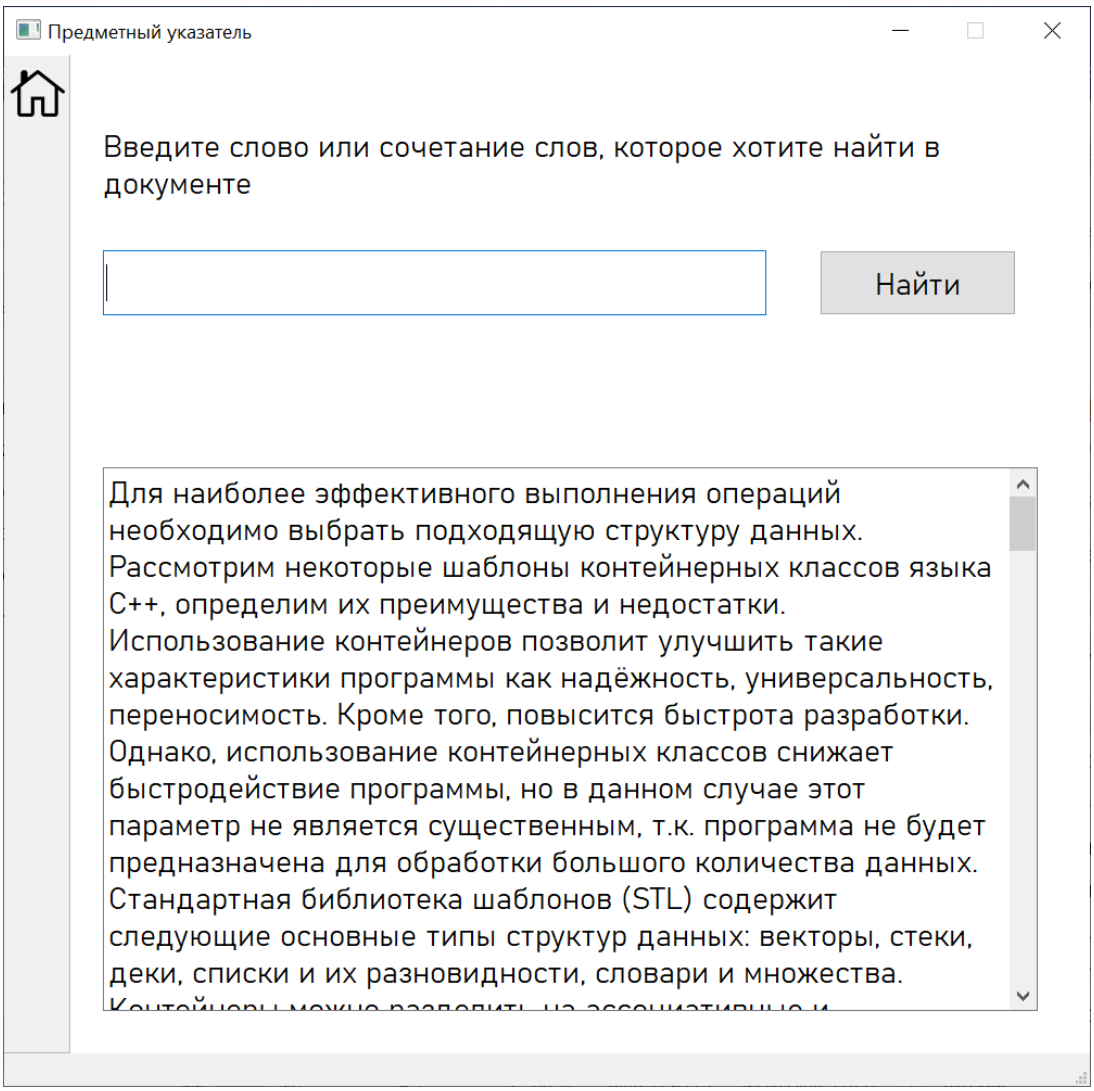

Рисунок 16 – страница поиска по загруженному тексту

3.5.2 Дополнительный окна приложения

Для перехода в режимы работы с существующим предметным указателем и поиска по тексту необходимо выбрать файл для загрузки в приложение. После нажатия соответствующих кнопок на главной странице, приложение откроет окно проводника для того, чтобы дать пользователю возможность выбрать нужный файл в файловой системе.

Для каждого из режимов установлен фильтр: при переходе в режим открытия предметного указателя отображаются только файлы формата csv, а для загрузки текста в приложение в проводнике отображаются только txt-файлы. Окна выбора изображены на рисунках 17 и 18.

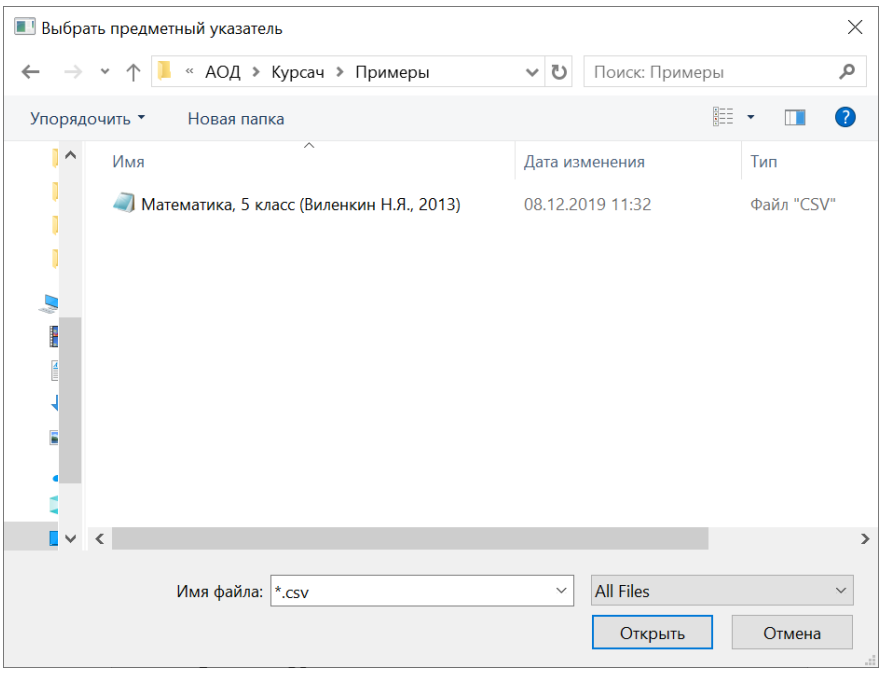

Рисунок 17 – Окно проводника при открытии предметного указателя

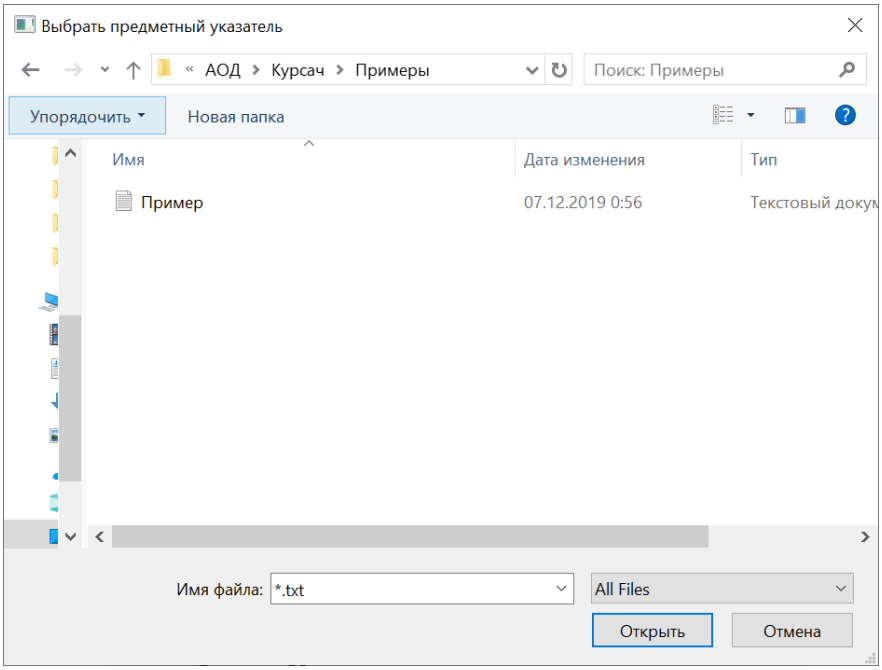

Рисунок 18 – Окно проводника при загрузке текста

3.5.3 Подокна режима работы с существующим предметным указателем

Как видно из рисунка 15, в данном режиме доступно пять инструментов: поиск, сохранение в файл, редактирование, удаление и добавление, соответственно.

Каждому инструменту соответствует свое подокно. Далее рассмотрим возможности каждого инструмента подробно.

3.5.3.1 Подокно поиска

Данное подокно предназначено для поиска терминов в предметрном указателе. На рисунке 19 видно, что окно имеет поле ввода и кнопку «Найти», а также верхняя часть окна содержит небольшую инструкцию.

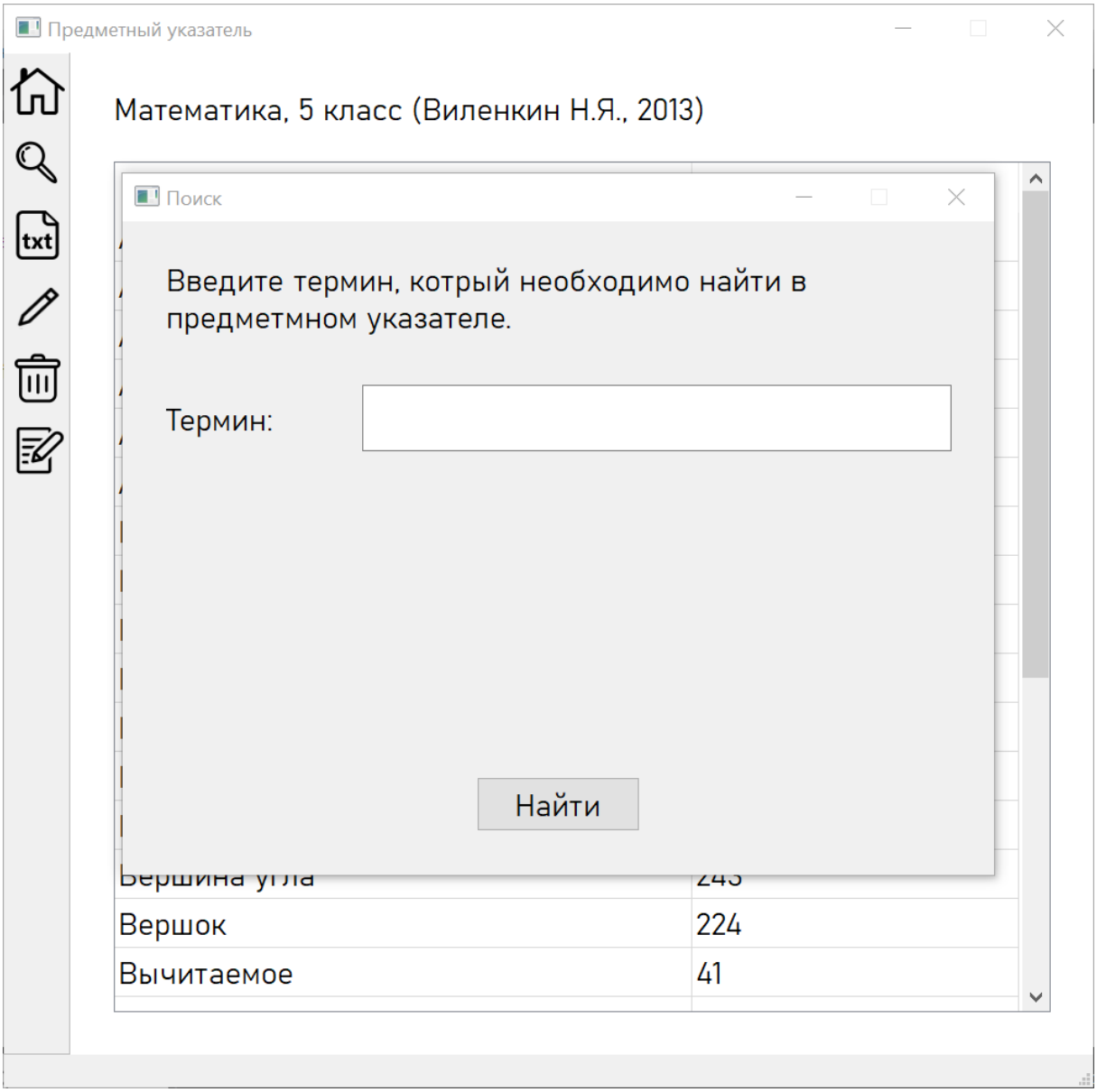

Рисунок 19 – Подокно поиска

Если пользователь нажал кнопку «Найти», не заполнив поле, приложение покажет сообщение с требованием заполнить поле (рисунок 20).

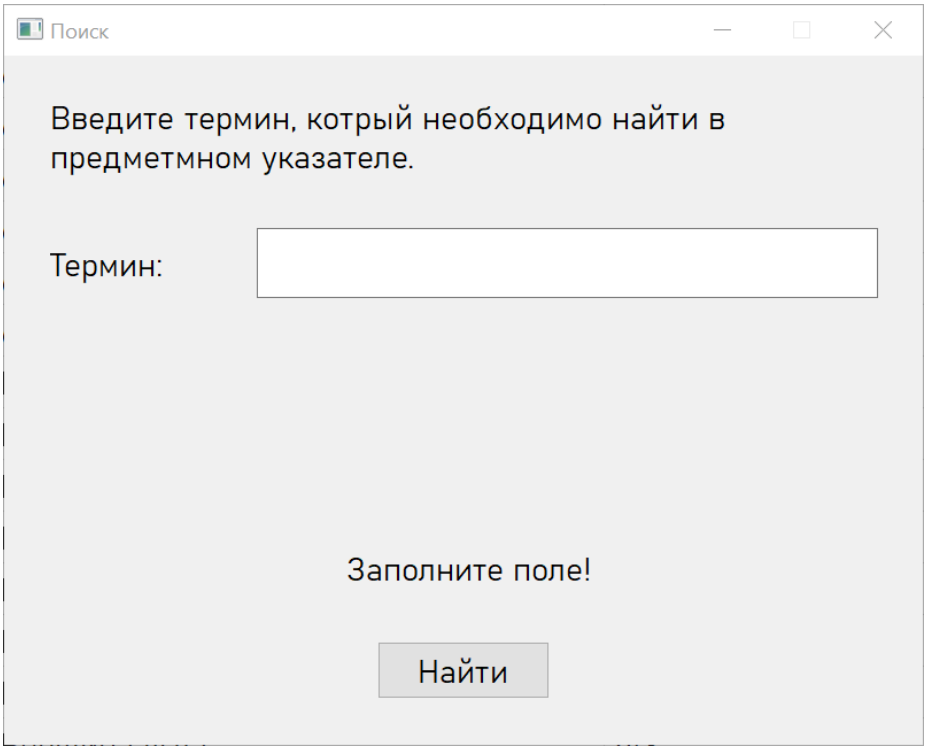

Рисунок 20 – Результат работы приложения при нажатии на кнопку поиска без заполнения поле ввода термина

Если пользователь ввел слово, и оно нашлось в предметном указателе, то в средней части окна появится таблица, состоящая из одной строки, которая содержит результат поиска. Пример работы программы приведен ниже, на рисунке 21.

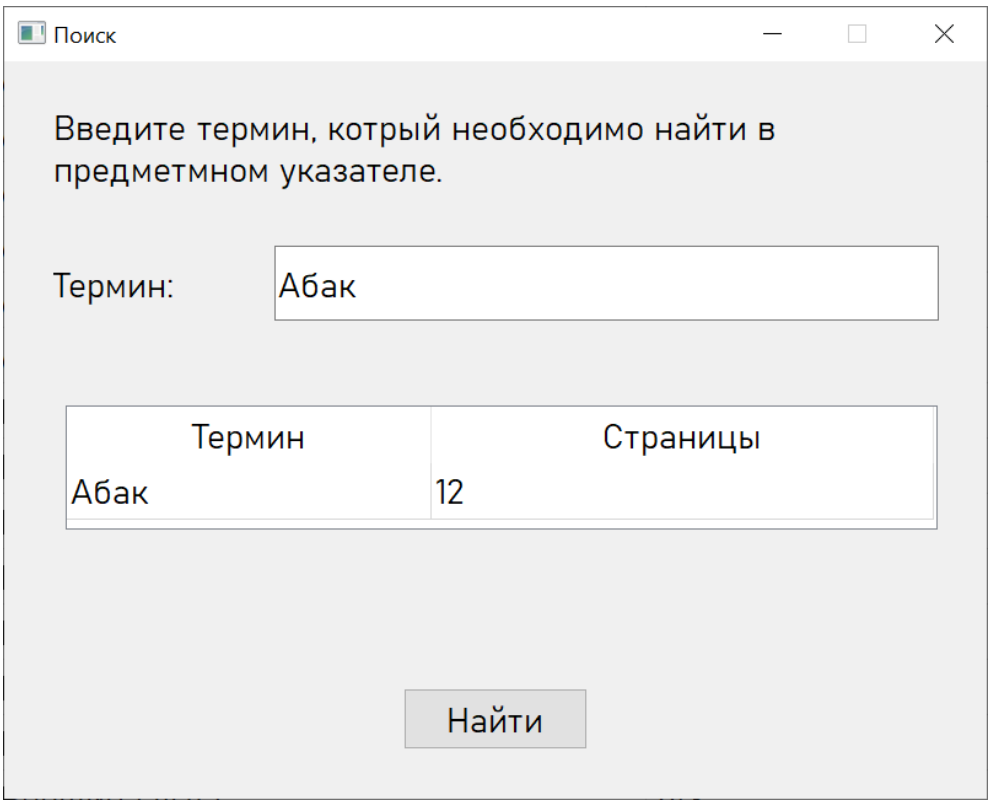

Рисунок 21 – Пример результата работы программы в случае, если введенный термин есть в предметном указателе

Если же введенный термин в словаре не был найден, то программа выведет сообщение о том, что термин не найден. Пример работы программы в такой ситуации можно также найти ниже, на рисунке 22.

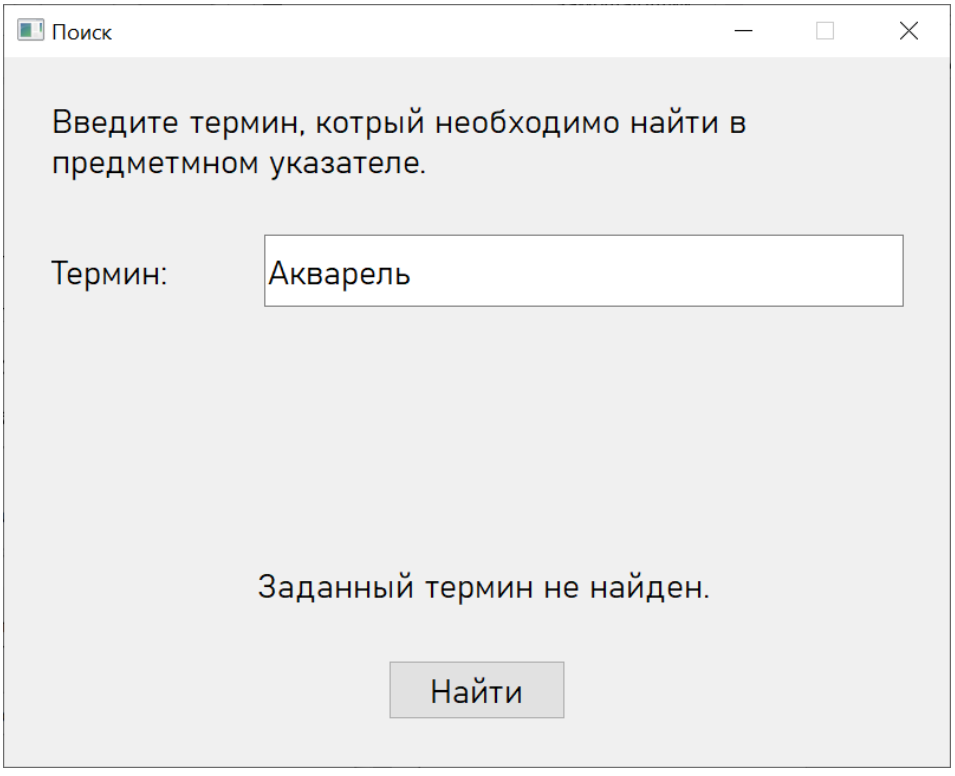

Рисунок 22 – Пример работы программы, в случае, если термин, заданный пользователем, не найден в предметном указателе

3.5.3.2 Подокна записи предметного указателя в текстовый файл

Первое подокно представляет из себя окно проводника для выбора директории для сохранения текстового файла (рисунок 23).

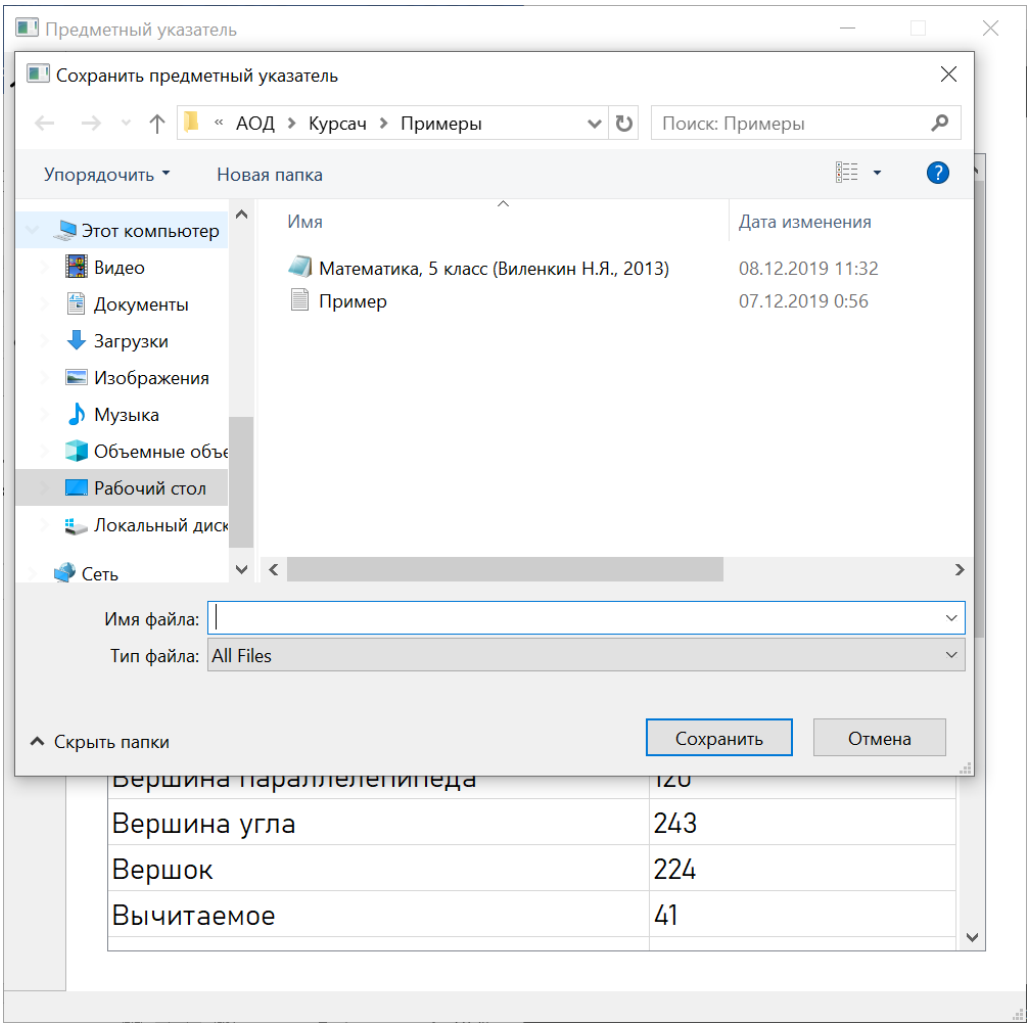

Рисунок 23 – Окно для выбора директории сохранения файла

Далее следует указать название, под которым необходимо сохранить файл, и нажать кнопку сохранить. После сохранения файла на экране появится небольшое окошко-сообщение, говорящее об успешности сохранения файла (рисунок 24).

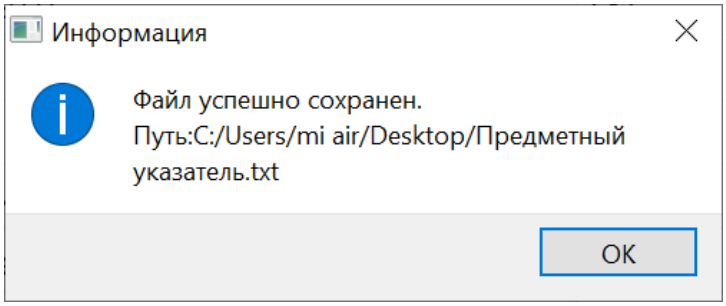

Рисунок 24 – Окошко с сообщением об успешности сохранения файла

3.5.3.3 Подокно редактирования записей предметного указателя

При нажатии соответствующего пункта на панели инструментов (кнопка с изображением ручки) появится окно (рисунок 25), содержащее короткую инструкцию, поле для ввода термина, который нудно отредактировать и кнопка «Изменить».

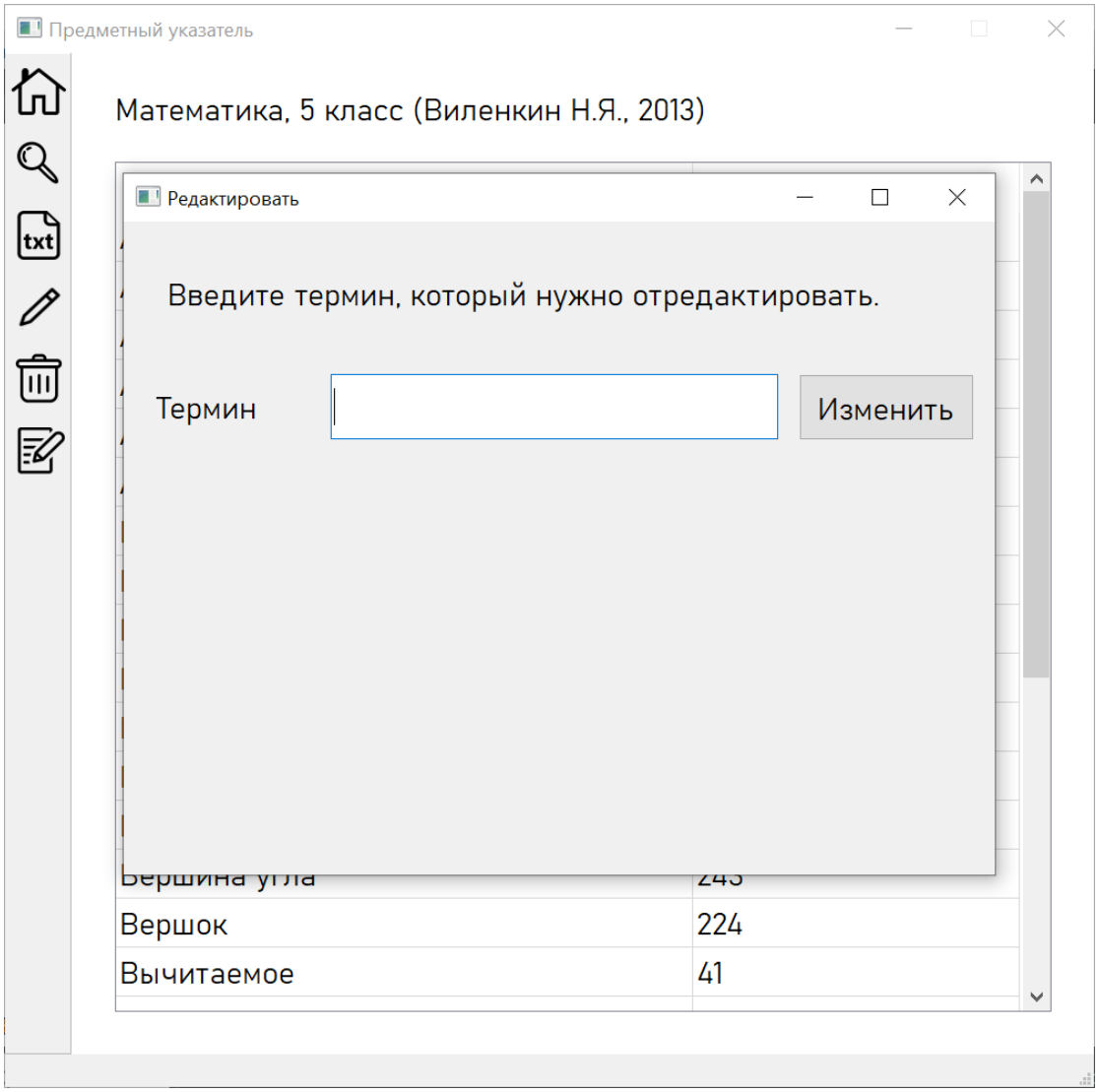

Рисунок 25 – Окно редактирования записей

Если, не заполнив поле, нажать кнопку «Изменить», то появится сообщение с требованием заполнить поле, аналогичное сообщению окна поиска. Затем, если введенный термин есть в предметном указателе, на форме появится дополнительное поле для введения новой информации (рисунок 26).

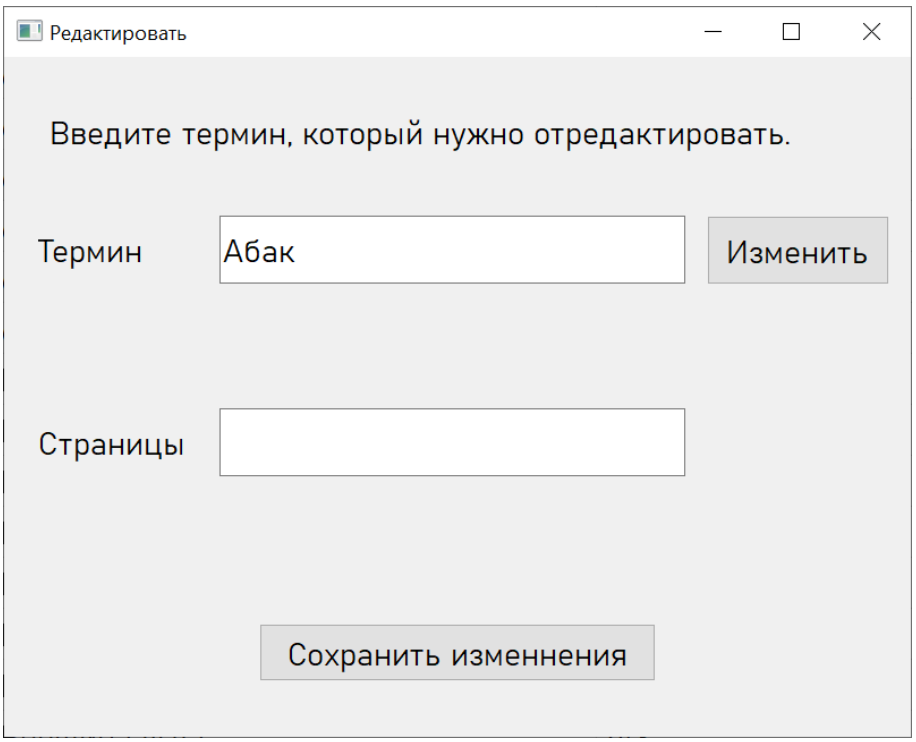

Рисунок 26 – Пример работы программы, в случае, если термин, заданный пользователем найден в предметном указателе

В противном случае, если термин не найден, на кране появится сообщение о том, что заданного термина нет в предметном указателе (рисунок 27).

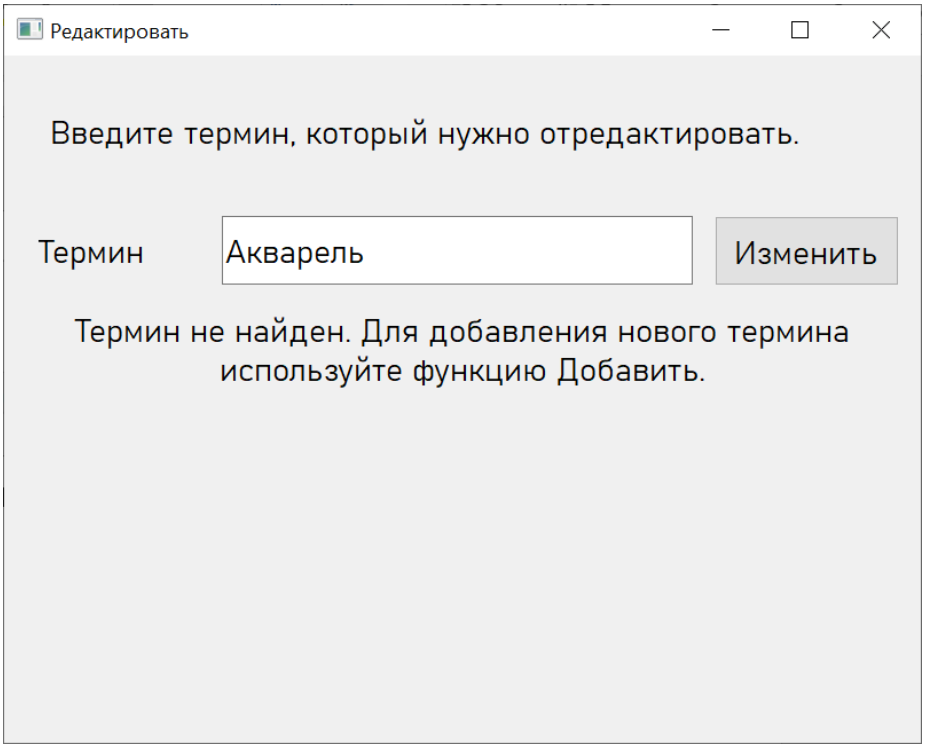

Рисунок 27 – Пример работы программы, в случае, если термин заданный пользователем не найден в предметном указателе

3.5.3.4 Подокно удаления записи

Данное подокно используется для удаления записей из предметного указателя. Оно, как же как и остальные подокна, содержит короткую инструкцию в верхней части, а также поле ввода термина и кнопку удаления. Ниже, на рисунке 28, приведено изображение данного подокна.

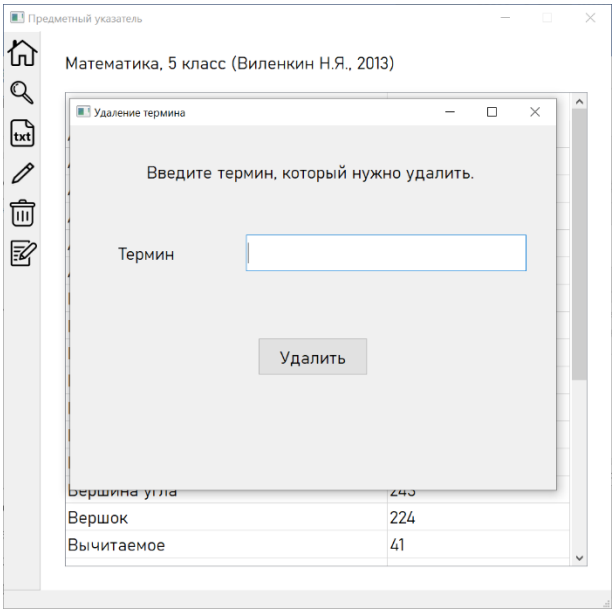

Рисунок 28 – Окно удаления терминов

Если нажать кнопку удаления прежде, чем поле «Термин» будет заполнено, появится сообщение, аналогичное сообщению в любом другом подокне, с требованием заполнить поле ввода. Также, если введенного термина нет в предметном указателе, появится сообщение об отсутствии термина в списке.

3.5.3.5 Подокно добавления нового термина

Данное подокно используется для добавления новых терминов в предметный указатель. Окно содержит краткую подсказку в его верхней части, два поля: первое – для указания термина, второе – для ввода номеров страниц, где упоминается этот термин, и кнопка сохранения.

Если хотя бы одно поле не заполнено, то информация считается неполной для добавления в предметный указатель, и на экране появляется сообщение с требованием заполнить поля. Далее, на рисунке 29, изображено подокно добавления нового термина.

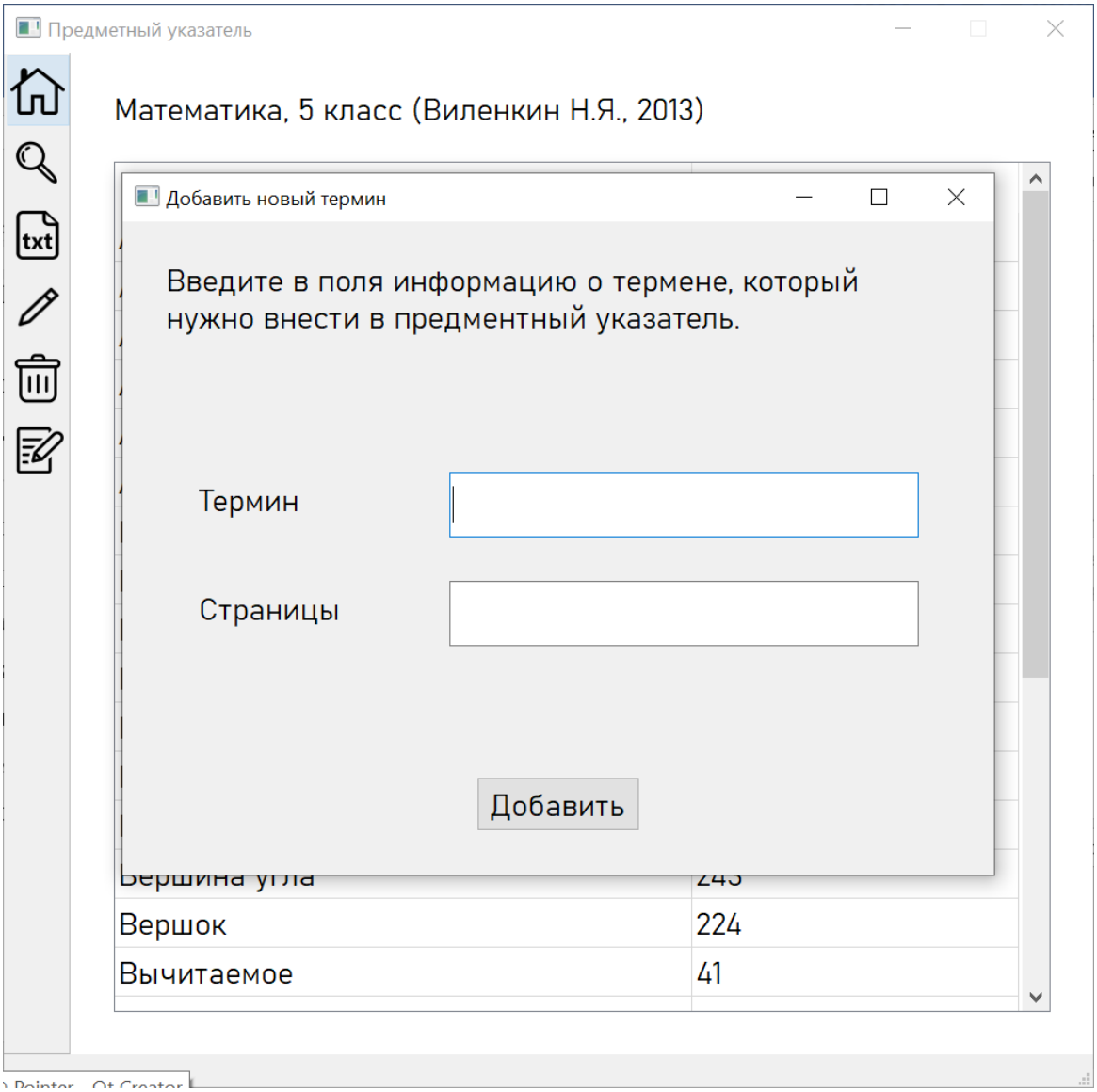

Рисунок 29 – Окно добавления нового термина

В случае же, если оба поля заполнены, термин добавляется в предметный указатель, и появляется сообщение об успешном добавлении нового термина (рисунок 30).

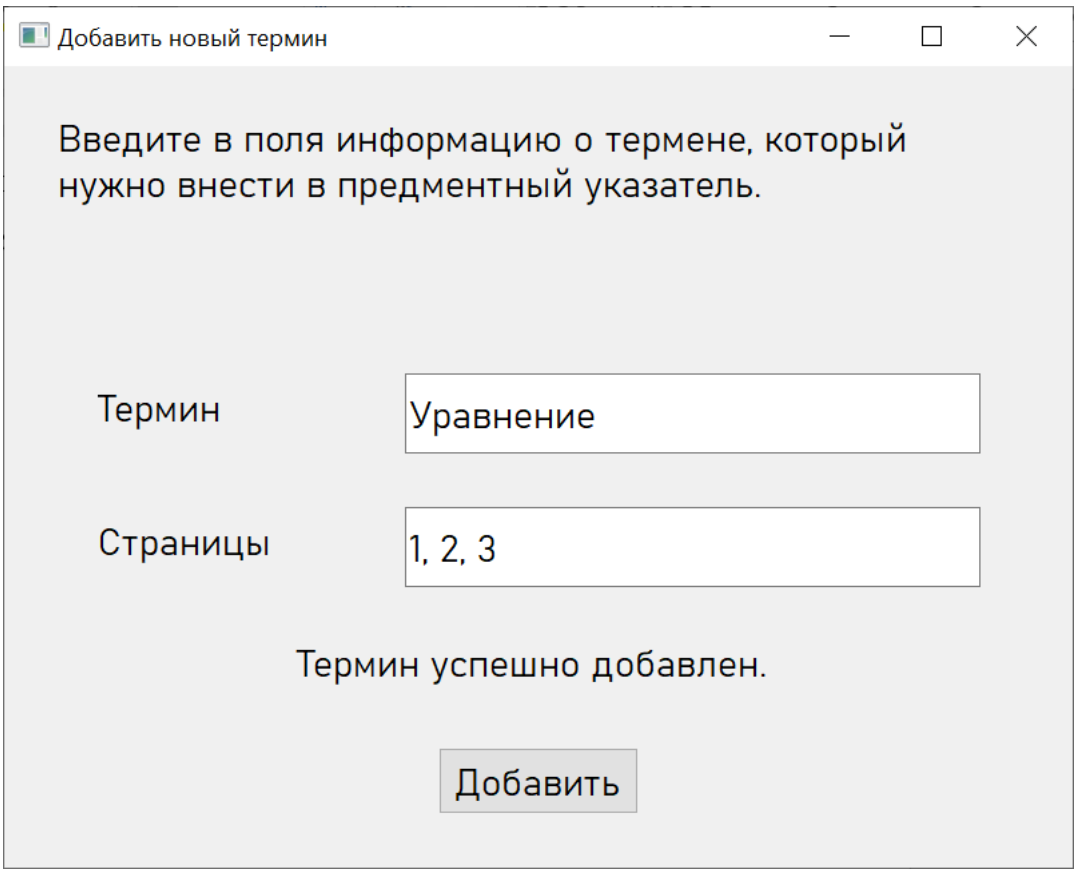

Рисунок 30 – Добавление термина прошло успешно

3.6 Входные данные приложения (организация и предварительная подготовка входных данных)

Для разных режимов работы приложения используются разные типы входных данных.

Для программы, работающей в режиме работы с существующим предметным указателем, входными данными является файл формата «.csv». В таком файле может хранится только один предметный указатель.

Если приложение запущено в режиме поиска по тексту, то входные данные представляются в виде файла формата «.txt».

3.7 Выходные данные

Выходными данными выступают любые преобразования проведенные над существующим предметным указателем или новые указатели, созданные с помощью приложения.

ЗАКЛЮЧЕНИЕ

В ходе разработки данного программного продукта были изучены многие структуры данных, используемые для хранения и обработки различных типов данных, их свойства, архитектура, преимущества и недостатки. Также были разобраны основные алгоритмы работы с этими структурами.

Кроме того, в рамках реализации графического интерфейса пользователя были получены навыки интеграции работы графических компонентов с различными алгоритмами обработки данных.

Основной задачей выполнения курсовой работы является создание программного продукта, ориентированного на хранение и удобное использование предметных указателей для различного рода литературы.

Данная задача выполнена в полном объеме, т.е. все требования к программному продукту удовлетворены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шлее, М. Qt 5.3. Профессиональное программирование на C++ / Макс Шлее.— СПб.: БХВ-Петербург, 2015.— 928 с.

2. Статья «О выборе структур данных для начинающих» [Электронный ресурс] – URL: https://m.habr.com/ru/post/339656/

3. Статья «MAP в C++: что это и как с этим работать» [Электронный ресурс] – URL: https://codelessons.ru/cplusplus/map-v-c-chto-eto-i-kak-s-etim-rabotat.html

4. Лафоре, Р. Объектно-ориентированное программирование в C++ / Роберт Лафоре.— 4-е изд., перераб. и доп.— М.: Питер, 2004.— 924 с.

5. Седжвик, Р. Алгоритмы на C++ / Роберт Седжвик.— М.: Издательский дом “Вильямс”, 2011.— 1056 с.

6. Васильев А. Н. Программирование на C++ в примерах и задачах / Алексей Васильев. – М: изд. «Э», 2018. – 361с.

7. Бьярне Страуструп. Программирование. Принципы и практика с использованием C++, испр. изд. : Пер. с англ. – М.: ООО «И.Д. Вильямс», 2011. – 1248 с.

8. Фаулер, М. Язык UML. Основы / Мартин Фаулер.— 3-е изд.— СПб.: Символ, 2005.— 192 с.

ПРИЛОЖЕНИЕ

Приложение А – Класс MainWindow

Приложение Б – Класс Open

Приложение В – Класс Create

Приложение Г – Класс Text

Приложение Д – Класс Add

Приложение Е – Класс Search

Приложение Ж – Класс Remove

Приложение И – Класс Change

Приложение К – Класс Start

Приложение Л – Класс Light

Приложение М – Главная функция, вспомогательные программные элементы

Приложение А

Класс MainWindow

Листинг А.1 – Заголовочный файл MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QToolBar>
#include "start.h"
#include "open.h"
#include "create.h"
#include "search.h"
#include "text.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;

    Start* start;
    Create* create;
    Open* open;
    Text* text;

    QToolBar* tools;

    void createToolBar();

private slots:
    void changeState(int);

signals:
    void sendStart(int);

};
#endif // MAINWINDOW_H
```

Листинг А.2 – Файл реализации MainWindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "functions.h"
#include <QObject>
#include <QToolBar>
#include <QIcon>
#include <QString>
#include <QPair>
#include <QFileDialog>

#include "start.h"
#include "open.h"
#include "create.h"
#include "search.h"

#include "ui_start.h"
#include "ui_open.h"
#include "ui_create.h"
#include "ui_search.h"

using namespace std;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    start = nullptr;
    open = nullptr;
    create = nullptr;
    text = nullptr;
    tools = nullptr;

    connect(this, SIGNAL(sendStart(int)), this, SLOT(changeState(int)));
    emit sendStart(START);
}

MainWindow::~MainWindow(){
    delete ui;
}


void MainWindow:: createToolBar(){ //создание панели инструментов
    tools =  new QToolBar("Инструменты");
    QSize size (50,60);

    QIcon home (":/images/icons/home.png");
    tools->addAction(home, "На начальную страницу",open, SLOT(sendStartF()));


    QIcon add (":/images/icons/add.png");
    QIcon file (":/images/icons/file.png");
    QIcon find (":/images/icons/find.png");
    QIcon change (":/images/icons/change.png");
    QIcon remove (":/images/icons/delete.png");
```

```
    tools->addAction(find, "Найти", open, SLOT(showSearch()));
    tools->addAction(file, "Сохранить в файл", open, SLOT(saveInFile()));
    tools->addAction(change, "Изменить", open, SLOT(showChange()));
    tools->addAction(remove, "Удалить термин", open,SLOT(showRemove()));
    tools->addAction(add,"Добавить термин", open, SLOT(showAdd()));

    addToolBar(Qt::LeftToolBarArea, tools);
    tools->setMovable(false);
    tools->setIconSize(size);
}

void MainWindow:: changeState(int state) { //функция изменяющяя внешний вид
окна
    switch (state) {

    case OPEN:
    {
        QFileDialog choice;
        QString filename = "";
        filename = choice.getOpenFileName(nullptr, "Выбрать предметный
указатель","*.csv");

        if(filename != ""){
            open = new Open (filename);
            setCentralWidget(open);
            open->show();

            createToolBar();
            connect(open, SIGNAL(sendStart(int)), this,
SLOT(changeState(int)));

            if(start != nullptr){
                delete start;
                start = nullptr;
            }
        }
        break;
    }

    case CREATE:
    {
        create = new Create();
        setCentralWidget(create);
        create->show();

        if(start != nullptr){
            delete start;
            start = nullptr;
        }

        connect (create, SIGNAL(sendState(int)), this,
SLOT(changeState(int)));
        break;
    }

    case START:
    {
        if (create != nullptr){
            delete create;
            create = nullptr;
        }
        if (open != nullptr){
            delete open;
```

```
            open = nullptr;
        }
        if (tools != nullptr){
            delete tools;
            tools = nullptr;
        }
        if (text != nullptr){
            delete text;
            text = nullptr;
        }
        start = new Start();
        setCentralWidget(start);
        start->show();

        connect(start, SIGNAL(sendState(int)), this, SLOT(changeState(int)));
        break;
    }

    case TEXT:
    {
        QFileDialog choice;

        QString filename = "";
        filename = choice.getOpenFileName(nullptr, "Выбрать предметный
указатель","*.txt");



        if(filename != ""){
            text = new Text (filename);

            tools =  new QToolBar("Инструменты");
            QSize size (50,60);
            QIcon home (":/images/icons/home.png");
            tools->addAction(home, "На начальную страницу",text,
SLOT(sendStartF()));
            addToolBar(Qt::LeftToolBarArea, tools);
            tools->setMovable(false);
            tools->setIconSize(size);

            text->show();
            setCentralWidget(text);
            connect(text, SIGNAL(sendStart(int)), this,
SLOT(changeState(int)));

            if(start != nullptr){
                delete start;
                start = nullptr;
            }
        }
            break;
    }
  }

}
```

Приложение Б

Класс Open

Листинг Б.1 – Заголовочный файл Open.h

```
#ifndef OPEN_H
#define OPEN_H

#include <QWidget>
#include <QMap>
#include <QFile>
#include "search.h"
#include "ui_search.h"
#include "add.h"
#include "ui_add.h"
#include "change.h"
#include "ui_change.h"
#include "remove.h"
#include "ui_remove.h"

namespace Ui {
class Open;
}

class Open : public QWidget
{
    Q_OBJECT

public:
    explicit Open(QString filePath, QWidget *parent = nullptr);
    ~Open();
    Ui::Open *ui;

private:
    QFile file;
    QString filePath;
    QMap <QString, QString> map;

    Search* searchWindow;
    Add* addWindow;
    Change* changeWindow;
    Remove* removeWindow;

signals:
    void sendData(QMap <QString,QString>*);
    void sendStart(int);

private slots:
    void showSearch();
    void showAdd();
    void showChange();
    void showRemove();
    void sendDataF();
    void sendStartF();
    void addNewTerm(QString, QString);
    void changeData(QString);
    void removeData(QString);
    void saveInFile();
};
#endif // OPEN_H
```

Листинг Б.2 – Файл реализации Open.cpp

```
#include "open.h"
#include "ui_open.h"
#include "search.h"
#include "ui_search.h"
#include "add.h"
#include "ui_add.h"
#include "change.h"
#include "ui_change.h"
#include"functions.h"
#include <QPalette>
#include <QColor>
#include <QTableWidget>
#include <QFile>
#include <QFileDialog>
#include <QMap>
#include <QString>
#include <QFileInfo>
#include <QMessageBox>

//конструктор
Open::Open(QString filePath, QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Open)
{
    ui->setupUi(this);

    QPalette window;
    window.setColor(QPalette::Window, Qt::white);
    setAutoFillBackground(true);
    setPalette(window);

    this->filePath = filePath;

    file.setFileName(filePath);
    file.open(QIODevice::ReadOnly);

    this->filePath = filePath;

    filePath.chop(4);
    while  (filePath.contains("/")){
        if (filePath.contains("/")) {
            filePath.remove(0, filePath.indexOf("/")+1);
        }
    }
    ui->book->setText(filePath);
    ui->list->setColumnWidth(0,530);
    ui->list->setColumnWidth(1, 300);

    QString newLine;
    QMap<QString, QString>::iterator it = map.begin();
    while (!file.atEnd()){ //пихаем в таблицу мап
        newLine = file.readLine();

        ui->list->setRowCount(ui->list->rowCount()+1);
        it = map.insert(newLine.section(';',0,0), newLine.section(';',1,1));

        QTableWidgetItem* book = new QTableWidgetItem(it.key());
        QTableWidgetItem* pages = new QTableWidgetItem(it.value());
        book->setFlags(Qt::ItemIsEnabled);
        pages->setFlags(Qt::ItemIsEnabled);
        ui->list->setItem(ui->list->rowCount()-1, 0, book);
        ui->list->setItem(ui->list->rowCount()-1, 1, pages);
```

```
    }
    file.close();
}

//деструктор
Open::~Open(){
    delete ui;
}

////////////////////////////////////////////////////////////////////////////
//открывает окно поиска
void Open::showSearch(){
    searchWindow = new Search();
    searchWindow->show();

    connect (searchWindow->ui->search, SIGNAL(released()), this, SLOT
(sendDataF()));
    connect(this, SIGNAL(sendData(QMap<QString, QString>*)), searchWindow,
SLOT(findTheKey(QMap<QString, QString>*)));
}


//открывает окно добавления термина
void Open:: showAdd(){
    addWindow = new Add();
    addWindow->show();

    connect(addWindow, SIGNAL(sendData(QString,QString)), this,
SLOT(addNewTerm(QString, QString)));

}

//открывает окно редактирования терминов
void Open:: showChange(){
    changeWindow = new Change();
    changeWindow->show();

    connect(changeWindow->ui->checkButton, SIGNAL(released()), this,
SLOT(sendDataF()));
    connect(this, SIGNAL(sendData(QMap<QString, QString>*)), changeWindow,
SLOT(checkInput(QMap<QString, QString>*)));

    connect (changeWindow, SIGNAL(sendChangedData(QString)), this,
SLOT(changeData(QString)));
}

//открывает окно удаления терминов
void Open:: showRemove(){
    removeWindow = new Remove();
    removeWindow->show();

    connect(removeWindow, SIGNAL(sendData(QString)), this,
SLOT(removeData(QString)));
}

////////////////////////////////////////////////////////////////////////////
//отправление map в функции
void Open:: sendDataF(){
    emit sendData(&map);
}

void Open:: sendStartF(){
    emit sendStart(START);
```

```
}


//добавления нового термина
void Open:: addNewTerm(QString term, QString pages){
    QFile file (filePath);
    file.open(QIODevice::WriteOnly);

    bool shit = false;

    for (QMap<QString,QString>::iterator it = map.begin(); it != map.end();
it++){
        if (it.key() == term){
            shit = true;
        }
    }

    if (shit == true){
        addWindow->ui->massage->setText("Такой термин уже есть в указателеле,"
                                        "используйте функцию редактирования
записей.");
        addWindow->ui->massage->show();
    }
    else{
            QString newLine = term + ";" + pages + ";\n";
            file.write(newLine.toUtf8());
            map.insert(term, pages);

            ui->list->setRowCount(0);

            for (QMap<QString, QString>::iterator it = map.begin(); it!=
map.end(); it++){
                ui->list->setRowCount(ui->list->rowCount()+1);
                QTableWidgetItem* book = new QTableWidgetItem(it.key());
                QTableWidgetItem* pages = new QTableWidgetItem(it.value());
                book->setFlags(Qt::ItemIsEnabled);
                pages->setFlags(Qt::ItemIsEnabled);
                ui->list->setItem(ui->list->rowCount()-1, 0, book);
                ui->list->setItem(ui->list->rowCount()-1, 1, pages);
                }
            addWindow->ui->massage->setText("Термин успешно добавлен.");
            addWindow->ui->massage->show();
    }
    file.close();
}


//изменение существующего термина
void Open::changeData(QString data){

    QString term = data.section(";", 0,0);
    QString pages = data.section (";", 1,1);

    for(QMap<QString,QString>:: iterator it = map.begin(); it != map.end();
it++){
        if (it.key() == term){
            it.value() = pages;
        }
    }

    for (int i = 0; i < ui->list->rowCount(); i++){
        if(ui->list->item(i,0)->text() == term){
            ui->list->item(i,1)->setText(pages);
```

```
            changeWindow->ui->success->show();
        }
    }

    writeToFile(filePath, &map, PROGRAMME);

}


//удаление термина
void Open:: removeData(QString term){
    bool deleted = false; //удалено - нет
    int row = 0;

    for(QMap<QString,QString>:: iterator it = map.begin(); it != map.end();){

        if (it.key() == term){
            deleted = true;

            it = map.erase(it);

            if (ui->list->item(row, 0)->text() == term){
                ui->list->removeRow(row);

            removeWindow->ui->message->setText("Термин удален.");
            removeWindow->ui->message->show();

            writeToFile(filePath, &map, PROGRAMME);
            }
        }
        else {
            it++;
        }
        row++;
    }

    if (deleted == false){
        removeWindow->ui->message->setText("Такого термина нет в указателе.");
        removeWindow->ui->message->show();
    }
}


//сохранение в файл
void Open:: saveInFile(){
   QString saveFileName = "";

   saveFileName = QFileDialog:: getSaveFileName(nullptr, "Сохранить предметный
указатель") + ".txt";

   if (saveFileName != ".txt"){
        writeToFile(saveFileName, &map, USER);
        QMessageBox::information(this, "Информация", "Файл успешно
сохранен.\nПуть:"+saveFileName);
   }
}
```

Приложение В

Класс Create

Листинг В.1 – Заголовочный файл Create.h

```
#ifndef CREATE_H
#define CREATE_H

#include <QWidget>

namespace Ui {
class Create;
}

class Create : public QWidget
{
    Q_OBJECT

public:
    explicit Create(QWidget *parent = nullptr);
    ~Create();
    Ui::Create *ui;

signals:
   void sendState(int state);

private slots:
   void on_createButton_released();
   void on_cancel_released();
};

#endif // CREATE_H
```

Листинг В.2 – Файл реализации Create.cpp

```
#include "ui_create.h"
#include <QPalette>
#include <QColor>
#include <QFile>
#include <QDir>
#include <QFileDialog>

#include <functions.h>

Create::Create(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Create)
{
    ui->setupUi(this);
    QPalette window;
    window.setColor(QPalette::Window, Qt::white);

    setAutoFillBackground(true);
    setPalette(window);

    ui->message->hide();

}

Create::~Create()
{
    delete ui;
```

```
}

void Create::on_createButton_released()
{
    if ((ui->name->text() == "")&(ui->author->text() == "")){
        ui->message->setText("Заполните поля!");
        ui->message->show();
    }
    else{ //формируем имя файла
         QString filename;
         if (ui->year->text() != ""){
            filename = ui->name->text() +
                       " (" +
                        ui->author->text() +
                        ", " +
                        ui->year->text() +
                        ").csv"   ;
         }
         else{
            filename = ui->name->text() +
                      " (" +
                       ui->author->text() +
                       ").csv"   ;
            }
         QFileDialog choice;
         QFile newPointer;

         newPointer.setFileName(choice.getExistingDirectory(nullptr,
"Сохранить как...") + "/" + filename);

         newPointer.open(QIODevice:: WriteOnly);
         newPointer.close();

         emit sendState(OPEN);
    }
}

void Create::on_cancel_released()
{
    emit sendState(START);
}
```

Приложение Г

Класс Text

Листинг Г.1 – Заголовочный файл Text.h

```
#ifndef TEXT_H
#define TEXT_H

#include <QWidget>
#include <QFile>
#include "light.h"

namespace Ui {
class Text;
}

class Text : public QWidget
{
    Q_OBJECT

public:
    explicit Text(QString filePath, QWidget *parent = nullptr);
    ~Text();

private slots:
    void on_searchButton_released();
    void changeMessageText(int);
    void sendStartF();

private:
    QFile file;
    QString filePath;
    Ui::Text *ui;
    Light* light;

signals:
   void sendStart(int);
};

#endif // TEXT_H
```

Листинг Г.2 – Файл реализации Text.cpp

```
#include "text.h"
#include "ui_text.h"
#include "light.h"
#include "functions.h"

Text::Text (QString filePath, QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Text)
{
    ui->setupUi(this);
    ui->message->hide();

    QPalette window;
    window.setColor(QPalette::Window, Qt::white);
    setAutoFillBackground(true);
    setPalette(window);

    this->filePath = filePath;
```

```
    file.setFileName(filePath);
    file.open(QIODevice::ReadOnly);

    this->filePath = filePath;

    ui->text->setText(file.readAll());
    ui->text->setReadOnly(true);
}

Text::~Text()
{
    delete ui;
}

void Text::on_searchButton_released()
{
    light = new Light (ui->text->document());
    connect(light,SIGNAL(setMessageText(int)), this,
SLOT(changeMessageText(int)));
    light->searchStr(ui->lineEdit->text());
}

void Text::changeMessageText(int t){
    if (t==0){
        ui->message->setText("Совпадений не найдено.");
        ui->message->show();
        ui->message->setStyleSheet("color: rgb(255, 44, 44)");
    }
    else{
        ui->message->setText("Найдено " + QString::number(t) + "
совпадение(й).");
        ui->message->show();
        ui->message->setStyleSheet("color: rgb(90, 185, 174)");
    }
}

void Text:: sendStartF(){
    emit sendStart(START);
}
```

Приложение Д

Класс Add

Листинг Д.1 – Заголовочный файл Add.h

```
#ifndef ADD_H
#define ADD_H

#include <QWidget>

namespace Ui {
class Add;
}

class Add : public QWidget
{
    Q_OBJECT

public:
    explicit Add(QWidget *parent = nullptr);
    ~Add();

    Ui::Add *ui;

signals:
   void sendData(QString,QString);

private slots:
   void on_addButton_released(); //ОТПРАВКА СИГНАЛА
};

#endif // ADD_H
```

Листинг Д.2 – Файл реализации Add.cpp

```
#include "add.h"
#include "ui_add.h"
#include <QStyle>
Add::Add(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Add)
{
    ui->setupUi(this);
    ui->massage->hide();
}
Add::~Add()
{
    delete ui;
}
void Add::on_addButton_released()
{
    if ((ui->termEdit->text() != "")&&(ui->pageEdit->text() != "")){
        emit sendData(ui->termEdit->text(),ui->pageEdit->text());
    }
    else {
        ui->massage->setText("Заполните поля!");
        ui->massage->show();
    }
}
```

Приложение Е

Класс Search

Листинг Е.1 – Заголовочный файл Search.h

```
#ifndef SEARCH_H
#define SEARCH_H

#include <QWidget>

namespace Ui {
class Search;
}

class Search : public QWidget
{
    Q_OBJECT

public:
    explicit Search(QWidget *parent = nullptr);
    ~Search();
    Ui::Search *ui;

private slots:
    void findTheKey(QMap<QString,QString>*);
};

#endif // SEARCH_H
```

Листинг Е.2 – Файл реализации Search.cpp

```
#include "open.h"
#include "open.h"
#include "search.h"
#include "ui_search.h"

#include <QMap>
#include <QString>
#include <QTableWidget>
#include <QTableWidgetItem>

Search::Search(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Search)
{
    ui->setupUi(this);
    ui->result->removeRow(0);

    ui->result->hide();
    ui->notFound->hide();
}

Search::~Search()
{
    delete ui;
}

void Search:: findTheKey(QMap<QString,QString>* map){
    ui->result->hide();
    ui->notFound->hide();
    QString term = ui->input->text(); //получили термин для поиска
```

```
    if (term==""){
        ui->notFound->setText("Заполните поле!");
        ui->notFound->show();
    }
    else{
    bool flag = false;

    ui->result->setColumnWidth(0,297);
    ui->result->setColumnWidth(1, 409);

    QMap<QString,QString>:: iterator it;
    for (it = map->begin(); it!= map->end(); it++){

        if (it.key() == term) { //если найден нужный ключ

            flag = true;
            ui->result->setRowCount(1);

            QTableWidgetItem* key = new QTableWidgetItem(term);
            ui->result->setItem(0, 0, key);
            key->setFlags(Qt::ItemIsEnabled);

            QTableWidgetItem* val = new QTableWidgetItem (it.value());
            ui->result->setItem(0, 1, val);
            val->setFlags (Qt::ItemIsEnabled);
        }
    }

    if (flag==false){
        ui->notFound->setText("Заданный термин не найден.");
        ui->notFound->show();
    }
    else{
        ui->result->show();
    }
  }
}
```

Приложение Ж

Класс Remove

Листинг Ж.1 – Заголовочный файл Remove.h

```
#ifndef REMOVE_H
#define REMOVE_H
#include <QWidget>

namespace Ui {
class Remove;
}

class Remove : public QWidget
{
    Q_OBJECT

public:
    explicit Remove(QWidget *parent = nullptr);
    ~Remove();
    Ui::Remove *ui;

signals:
    void sendData(QString);

private slots:
    void on_removeButton_released();
};
#endif // REMOVE_H
```

Листинг Ж.2 – Файл реализации Remove.cpp

```
#include "remove.h"
#include "ui_remove.h"
#include <QString>
Remove::Remove(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Remove)
{
    ui->setupUi(this);
    ui->message->hide();
}
Remove::~Remove()
{
    delete ui;
}
void Remove::on_removeButton_released()
{
    QString str = ui->termEdit->text();
    if (str != ""){
        emit sendData(str);
        ui->termEdit->clear();
    }
    else{
        ui->message->setText("Заполните поле!");
        ui->message->show();
    }
}
```

Приложение И

Класс Change

Листинг И.1 – Заголовочный файл Change.h

```
#ifndef CHANGE_H
#define CHANGE_H

#include <QWidget>
#include <QPair>

namespace Ui {
class Change;
}

class Change : public QWidget
{
    Q_OBJECT

public:
    explicit Change(QWidget *parent = nullptr);
    ~Change();

    Ui::Change *ui;

signals:
    void sendChangedData(QString);

private slots:
   void on_saveButton_released();
   void checkInput(QMap<QString, QString>*);
};
#endif // CHANGE_H
```

Листинг И.2 – Файл реализации Change.cpp

```
#include "change.h"
#include "ui_change.h"
#include <QPair>
Change::Change(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Change)
{
    ui->setupUi(this);
    ui->message->hide();
    ui->saveButton->hide();
    ui->Lpages->hide();
    ui->pagesEdit->hide();
    ui->success->hide();
}

Change::~Change()
{
    delete ui;
}

void Change:: checkInput(QMap<QString, QString>* map)
{
    ui->message->hide();
    ui->saveButton->hide();
    ui->Lpages->hide();
    ui->pagesEdit->hide();
```

```
    ui->success->hide();

    if (ui->termEdit->text() == ""){
        ui->message->setText("Заполните поле!");
        ui->message->show();
    }
    else{
    bool shit = true;

    for(QMap<QString, QString>:: iterator it = map->begin(); it != map->end();
it++ ){
        if (it.key() == ui->termEdit->text()) {
            shit = false;
            break;
        }
    }

    if (shit == true){
        ui->message->setText("Термин не найден. Для добавления нового термина
используйте функцию Добавить.");
        ui->message->show();
    }

    else{
        ui->Lpages->show();
        ui->pagesEdit->show();
        ui->saveButton->show();
    }
  }
}

void Change::on_saveButton_released()
{
    if(ui->pagesEdit->text() == ""){
        ui->message->setText("Заполните поле \"Страницы\"!");
        ui->message->show();
    }
    else {
      ui->message->hide();

      QString term = ui->termEdit->text();
      QString pages = ui->pagesEdit->text();
      QString str = (term + ";" + pages+";");

      emit sendChangedData(str);

      ui->termEdit->clear();

      ui->pagesEdit->clear();
      ui->pagesEdit->hide();

      ui->Lpages->hide();
      ui->saveButton->hide();
    }
}
```

Приложение К

Класс Start

Листинг К.1 – Заголовочный файл Start.h

```
#ifndef START_H
#define START_H

#include <QWidget>

namespace Ui {
class Start;
}

class Start : public QWidget
{
    Q_OBJECT

public:
    explicit Start(QWidget *parent = nullptr);
    ~Start();

    Ui::Start *ui;

signals:
    void sendState (int state);

private slots:
    void on_OpenButton_released();
    void on_CreateButton_released();
    void on_TextButton_released();
};

#endif // START_H
```

Листинг К.2 – Файл реализации Start.cpp

```
#include "start.h"
#include "ui_start.h"
#include "mainwindow.h"

#include "functions.h"

#include <QPushButton>
#include <QSize>
#include <QIcon>
#include <QPalette>
#include <QColor>

Start::Start(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Start)
{
    ui->setupUi(this);
    QSize size(302, 110);

    QPalette window;
    window.setColor(QPalette::Window, Qt::white);

    setAutoFillBackground(true);
    setPalette(window);
```

```
    ui->CreateButton->setIcon(QIcon(":/images/icons/create.png"));
    ui->CreateButton->setIconSize(size);
    ui->CreateButton->setStyleSheet("text-align: left;");

    ui->OpenButton->setIcon(QIcon(":/images/icons/open.png"));
    ui->OpenButton->setIconSize(size);
    ui->OpenButton->setStyleSheet("text-align: left;");

    ui->TextButton->setIcon(QIcon(":/images/icons/text.png"));
    ui->TextButton->setIconSize(size);
    ui->TextButton->setStyleSheet("text-align: left;");
}

Start::~Start()
{
    delete ui;
}

void Start::on_OpenButton_released()
{
    emit sendState(OPEN);
}

void Start::on_CreateButton_released()
{
    emit sendState(CREATE);
}

void Start::on_TextButton_released()
{
    emit sendState(TEXT);
}
```

Приложение Л

Класс Light

Листинг Л.1 – Заголовочный файл Light.h

```
#ifndef LIGHT_H
#define LIGHT_H

#include <QSyntaxHighlighter>
#include <QRegularExpression>

class Light : public QSyntaxHighlighter
{
    Q_OBJECT
    using BaseClass = QSyntaxHighlighter;

public:
    explicit Light(QTextDocument* parent = nullptr);
    void searchStr(const QString& str);
protected:
    void highlightBlock(const QString &text) override;
private:
    QRegularExpression m_pattern;
    QTextCharFormat m_format;
    int t;
signals:
    void setMessageText(int);
};

#endif // SEARCHHIGHLIGHT_H
```

Листинг Л.2 – Файл реализации Light.cpp

```
#include "light.h"
#include <QTextCharFormat>

Light::Light(QTextDocument* parent) : BaseClass(parent)
{
    m_format.setBackground(QColor(121,232,219));
    t = 0;
}
void Light::highlightBlock(const QString& text)
{
    QRegularExpressionMatchIterator matchIterator =
m_pattern.globalMatch(text);
    while (matchIterator.hasNext())
    {
        QRegularExpressionMatch match = matchIterator.next();
        setFormat(match.capturedStart(), match.capturedLength(), m_format);
        t++;
    }
    emit setMessageText(t);
}
void Light::searchStr(const QString &str)
{
    m_pattern = QRegularExpression(str);
    rehighlight();
}
```

Приложение М

Главная функция, вспомогательные программные элементы

Листинг М.1 – Главная функция

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.move(450,10);
    w.show();
    return a.exec();
}
```

Листинг М.2 – Заголовочный файл functions.h

```
#ifndef WRITETOFILE_H
#define WRITETOFILE_H
#include <QString>
#include <QFile>

enum purpose{
    PROGRAMME,
    USER,
};
enum states{
    OPEN,
    CREATE,
    START,
    TEXT
};
void writeToFile (QString filePath, QMap<QString,QString>* map, int purpose);
#endif // WRITETOFILE_H
```

Листинг М.3 – Файл реализации functions.cpp

```
#include "functions.h"
#include <QFile>
#include <QString>
#include <QMap>

void writeToFile (QString filePath, QMap<QString,QString>* map, int purpose){
    QFile file (filePath);
    file.open(QIODevice:: WriteOnly | QIODevice::Truncate);
    QString newLine;
    for(QMap<QString, QString>:: iterator it = map->begin(); it != map->end();
it++){
        if (purpose == PROGRAMME){
            newLine = it.key() + ";"+ it.value() +";\n";
        }
        else{
            newLine = it.key() + " - "+ it.value() +";\n";
        }
        file.write(newLine.toUtf8());
    }
    file.close();
}
```