|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт информационных технологий | |
| Кафедра вычислительной техники | |
|  |
|  |
|  |
| **КУРСОВАЯ РАБОТА** | |

по дисциплине «Структура и алгоритмы обработки данных»

**Тема курсовой работы:** Разработка программы «Частотный словарь»

**Студент группы** ИКБО-03-18 Петров Анатолий Валерьевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 (учебная группа, фамилия, имя, отчество студента) (подпись студента)

**Руководитель курсовой работы** зав.каф. ВТ, доцент, к.т.н. Платонова О.В \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(должность, звание, учёная степень) (подпись руководителя)

**Консультант** старший преподаватель Асадова Ю.С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 (должность, звание, учёная степень) (подпись консультанта)

Работа представлена к защите «\_\_\_\_» \_\_\_\_\_\_\_\_\_20\_\_г.

Допущен к защите «\_\_\_\_» \_\_\_\_\_\_\_\_\_20\_\_г.

Москва 2019

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт информационных технологий | |
| Кафедра вычислительной техники | |
|  | **Утверждаю** |
|  | Заведующий кафедрой ВТ  Платонова О. В.  (Подпись)  *ФИО* |
|  | «05» сентября 2019 г. |
| **ЗАДАНИЕ** | |
| **на выполнение курсовой работы** по дисциплине | |
| «Структуры и алгоритмы обработки данных» | |

Студент Петров Анатолий Валерьевич Группа ИКБО-03-18

**Тема** Разработка программы «Частотный словарь»

**Исходные данные:** Язык программирования С++, MS Visual Studio, техническое задание

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:** Частотный словарь – это алфавитный список всех слов заданного текста с указанием количества вхождений слова в текст. Операции словаря: создание пустого словаря; добавление нового элемента в словарь; увеличение значения счетчика слова в элементе словаря с заданным словом; вывод элементов словаря в алфавитном порядке; поиск элемента словаря по слову. В демонстрационной программе предусмотреть считывание текста для построения словаря из текстового файла и запись словаря в алфавитном порядке в другой текстовый файл перед завершением работы программы, считывание частотного словаря из текстового файла для просмотра и поиска в нем элементов. В текстовом файле не предусмотрено переноса слов.

|  |
| --- |
| **Срок представления к защите курсовой работы:** **до** «19» декабря 2019 г. |
|  |
| **Задание на выполнение курсовой работы выдал \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Платонова О. В.)**  *Подпись руководителя Ф.И.О. руководителя* |

**«5» сентября 2019г.**

**Задание на курсовую работу получил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Петров А.В.)**

*Подпись обучающегося* *Ф.И.О. исполнителя*

**«5» сентября 2019г.**

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 3](#_Toc27037091)

[1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 5](#_Toc27037092)

[1.1 Введение 5](#_Toc27037093)

[1.1.1 Наименование программы 5](#_Toc27037094)

[1.1.2 Краткая характеристика области применения 5](#_Toc27037095)

[1.2 Основание для разработки 5](#_Toc27037096)

[1.3 Назначение разработки 5](#_Toc27037097)

[1.4 Требования, предъявляемые к программе 6](#_Toc27037098)

[1.4.1 Требования к функциональным характеристикам программ 6](#_Toc27037099)

[1.4.2 Требования к техническим средствам, используемым при работе программы 6](#_Toc27037100)

[1.4.3 Требования к языкам программы и среде разработки программы 7](#_Toc27037101)

[1.4.4 Требования к информационным структурам на входе и выходе программы 7](#_Toc27037102)

[1.4.5 Специальные требования 7](#_Toc27037103)

[1.5 Требования к программной документации 7](#_Toc27037104)

[1.6 Этапы разработки 8](#_Toc27037105)

[2 ОБЗОР СПОСОБОВ ОРГАНИЗАЦИИ ДАННЫХ И ОБОСНОВАНИЕ ВЫБОРА СТРУКТУРЫ ДАННЫХ ДЛЯ ЭФФЕКТИВНОГО ВЫПОЛНЕНИЯ ОПЕРАЦИЙ 9](#_Toc27037106)

[2.1 Стек 9](#_Toc27037107)

[2.2 Очередь 9](#_Toc27037108)

[2.3 Словарь 10](#_Toc27037109)

[2.4 Массив 10](#_Toc27037110)

[2.5 Динамический массив 11](#_Toc27037111)

[2.6 Связанный список 11](#_Toc27037112)

[2.7 Вектор 11](#_Toc27037113)

[2.8 Выводы и окончательный выбор структуры 12](#_Toc27037114)

[3 ОПИСАНИЕ ПРОГРАММЫ 13](#_Toc27037115)

[3.1 Общие сведения 13](#_Toc27037116)

[3.1.1 Наименование программы 13](#_Toc27037117)

[3.1.2 Программное обеспечение, необходимое для функционирования программы 13](#_Toc27037118)

[3.1.3 Язык программирования, на котором написана программа 14](#_Toc27037119)

[3.2 Функциональное назначение программы (классы решаемых задач и функциональные ограничения на применения) 14](#_Toc27037120)

[3.3 Описание логической структуры программы 15](#_Toc27037121)

[3.3.1 Алгоритмы, используемые в программе 15](#_Toc27037122)

[3.3.2 Структура программы с описанием функций составных частей и связей между ними 20](#_Toc27037123)

[3.4 Технические средства, которые используются при работе программы 22](#_Toc27037124)

[3.5 Вызов программы 23](#_Toc27037125)

[3.5.1 Главное окно приложения 23](#_Toc27037126)

[3.6 Входные данные (организация и предварительная подготовка входных данных) 27](#_Toc27037127)

[3.7 Выходные данные 27](#_Toc27037128)

[ЗАКЛЮЧЕНИЕ 28](#_Toc27037129)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 29](#_Toc27037130)

[ПРИЛОЖЕНИЯ 30](#_Toc27037131)

[Приложение А 31](#_Toc27037131)

[Приложение Б 31](#_Toc27037131)

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

## 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.

### Наименование программы

Название данного приложения будет напрямую перекликаться с темой курсовой работы: “Операции частотного словаря”. Данное название целиком и полностью отражает функционал будущего приложения. Английский вариант названия: “OperationsOfFrequencyDictionary”. Краткое наименование: “OOFD”.

### Краткая характеристика области применения

Данная программа пригодится в первую очередь в сфере гуманитарных дисциплин, главный потенциал в области изучения иностранного языка.

## Основание для разработки

Основанием для разработки является курсовая работа по дисциплине «Структуры и алгоритмы обработки данных», предусмотренная учебным планом, по программе бакалавриата 09.03.04 «Программная инженерия», с учётом специфики профиля «Разработка программных приложений и проектирование информационных систем»

## Назначение разработки

Создание приложения, способного дать пользователю возможность работать с частотным словарём для запоминания слов, используя все особенности данного способа хранения данных.

## Требования, предъявляемые к программе

### Требования к функциональным характеристикам программ

В курсовой работе должна быть разработана программа с модульной структурой: каждая подзадача должна быть оформлена в виде функции С++. Программа должна содержать удобный пользовательский интерфейс (текстовый или графический).

Частотный словарь – это алфавитный список всех слов заданного текста с указанием количества вхождений слова в текст.

Операции словаря:

* создание пустого словаря;
* добавление нового элемента в словарь;
* увеличение значения счетчика слова в элементе словаря с заданным словом;
* вывод элементов словаря в алфавитном порядке;
* поиск элемента словаря по слову.

В демонстрационной программе предусмотреть считывание текста для построения словаря из текстового файла и запись словаря в алфавитном порядке в другой текстовый файл перед завершением работы программы, считывание частотного словаря из текстового файла для просмотра и поиска в нем элементов. В текстовом файле не предусмотрено переноса слов.

### Требования к техническим средствам, используемым при работе программы

Персональный компьютер пользователя должен быть оснащён графическим адаптером, также должна быть установлена ОС Windows (желательно, последней версии).

### Требования к языкам программы и среде разработки программы

Для разработки используется язык программирования «C++», в качестве среды разработки выступает «MS Visual Studio», а для разработки графического интерфейса пользователя задействован механизм «Windows Forms».

### Требования к информационным структурам на входе и выходе программы

Входные данные могут представлять из себя, как данные вводимые пользователем с клавиатуры, так и данные, которые считываются из файла при последующем запуске программы. Программа должна предоставлять возможности к считыванию данных с клавиатуры в поле ввода.

Выходные данные будут представлять из себя текст в таблице, в которой будет отображена вся требуемая пользователю информацию в комфортном для чтения формате. После завершения работы выходные данные можно просмотреть в файле.

### Специальные требования

В демонстрационной программе предусмотреть считывание словаря в начале работы программы из текстового файла и запись словаря в файл перед завершением ее работы.

## Требования к программной документации

1. Пояснительная записка в соответствии со стандартами РТУ МИРЭА (СМКО МИРЭА 7.5.1/04.И.05-16);
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 выше перечисленное должно быть отражено в пояснительной записке, которая прилагается проекту.

## Этапы разработки

1. Обзор способов организации данных и обоснование выбора структуры данных для эффективного выполнения операций: 05.09.19 – 19.09.19.
2. Разработка программы: 20.09.19 – 10.11.19.
3. Разработка программной документации: 11.11.19 – 30.11.19.
4. Оформление пояснительной записки: 1.12.19 – 19.12.19.
5. Защита курсовой работы: 23.12.19 – 31.12.19.

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

# ОБЗОР СПОСОБОВ ОРГАНИЗАЦИИ ДАННЫХ И ОБОСНОВАНИЕ ВЫБОРА СТРУКТУРЫ ДАННЫХ ДЛЯ ЭФФЕКТИВНОГО ВЫПОЛНЕНИЯ ОПЕРАЦИЙ

Обзор различных структур таких как стек, очередь, массив, динамический массив, словарь, связанный список, вектор для выявления наиболее эффективных для написание программы.

## Стек

Стек — это структура LIFO. Это расшифровывается как Last In First Out («последним вошёл, первым вышел»). При добавлении и удалении из стека последний добавленный элемент будет первым удаляемым.

Для стека нужно всего три операции: Push, Pop и Top.

Push добавляет объект в стек. Pop удаляет объект из стека. Top даёт самый последний объект в стеке. Эти контейнеры в большинстве языков являются частью стандартных библиотек. В C++ они называются stack.

## Очередь

Очередь — это структура FIFO (First In First Out, «первым зашёл, первым вышел»).

При добавлении и удалении из очереди первый добавляемый элемент будет первым извлекаемым. Очереди нужно только несколько операций: Push\_Back, Pop\_Front, Front и Back. Push\_Back добавляет элемент к концу очереди. Pop\_Front удаляет элемент из начала очереди. Front и Back позволяют получить доступ к двум концам очереди.

Программистам часто нужно добавлять или удалять элементы из обоих концов очереди. Такая структура называется двухсторонней очередью (double ended queue, deque). В этом случае добавляется ещё пара операций: Push\_Front и Pop\_Back. Эти контейнеры тоже включены в большинство основных языков. В C++ это queue и deque.

Внутри очередь и двухсторонняя очередь могут быть устроены довольно сложно. Поскольку объекты могут поступать и извлекаться с любого конца, внутренний контейнер должен уметь наращивать и укорачивать очередь с начала и с конца. Во многих реализациях используются множественные страницы памяти. Когда любой из концов разрастается за пределы текущей страницы, добавляется дополнительная страница. Если страница больше не нужна, то она удаляется.

## Словарь

Это ассоциативный контейнер, который работает по принципу — [ключ — значение]. Он схож по своему применению с вектором и массивом, но есть некоторые различия:

* Ключом может быть все что угодна. От обычной переменной до класса.
* При добавлении нового элемента контейнер будет отсортирован по возрастанию.

Добавление, удаление, обращение к элементам происходит за log n.

## Массив

Массив — это структура данных, представленная в виде группы ячеек одного типа, объединенных под одним единым именем. Массивы используются для обработки большого количества однотипных данных. Имя массива является указателем. Отдельная ячейка данных массива называется элементом массива. Элементами массива могут быть данные любого типа. Массивы могут иметь как одно, так и более одного измерений. В зависимости от количества измерений массивы делятся на одномерные массивы, двумерные массивы, трёхмерные массивы и так далее до n-мерного массива.

## Динамический массив

Динамический массив — это массив, который может менять свой размер. Основные языки программирования в своих стандартных библиотеках поддерживают динамические массивы. В C++ это vector. В своей сути динамический массив — это простой массив, однако имеющий ещё два дополнительных блока данных. В них хранятся действительный размер простого массива и объём данных, который может на самом деле храниться в простом массиве.

## Связанный список

Массив — это непрерывный блок памяти, и каждый элемент его расположен после другого. Связанный список — это цепочка объектов. Связанные списки тоже присутствуют в стандартных библиотеках основных языков программирования. В C++ они называются list.

Самое большое преимущество связанного списка заключается в добавлении и удалении объектов из списка. Внесение изменений в середину списка выполняется очень быстро. Помните, что динамический массив теоретически мог вызывать смещение каждого элемента, а связанный список сохраняет каждый другой объект на своём месте.

## Вектор

Стандартный шаблон обобщённого программирования языка C++ std::vector<T> — реализация динамического массива переменного размера. Вектор в C++ — это замена стандартному динамическому массиву. А массив – структура данных, поэтому рассмотрение вектора условию задания никак не противоречит.

Вспомним о том, что для создания динамического массива (вручную) нам нужно пользоваться конструктором new и вдобавок указателями. Но в случае с векторами всего этого делать не нужно. Вообще, по стандарту пользоваться динамическим массивом через конструктор new — не есть правильно. Так как в компьютере могут происходить различные утечки памяти.

Вектор – крайне удобный и понятный динамический массив. Также вектор легко сортировать с помощью отдельно написанного компаратора, а благодаря точкам входа и выхода можно сортировать даже отдельную часть вектора.

## Выводы и окончательный выбор структуры

Сравним указанные варианты по удобству реализации и выполнению предписанных функций. Так как, согласно заданию, нужно создать «Частотный словарь», был выбран «**Вектор**», с помощью которого будет реализована структура данных. Можно было бы выбрать и «Словарь», потому что многие действия у них схожи, но все-таки словарь намного дольше обращается к элементам (log n) и предпочтительнее использовать вектор.

В данной программе понадобится изменять элементы на промежутке всего вектора, сортировать слова в алфавитном порядке. Для этих действий хорошо подходит вектор.

# ОПИСАНИЕ ПРОГРАММЫ

## Общие сведения

В процессе выполнения курсовой работы и исполнения назначенных условий было создано приложение для операционной системы Windows для работы со множествами, наборами элементов одного типа. Множества способны хранить в себе эти элементы, а программа предоставляет инструменты для их отображения и проведения операций, обозначенных в техническом задании.

### Наименование программы

Краткое наименование: “OOFD”. Название программы формировалось исходя из его предназначения и основного функционала. “OOFD” (OperationsOfFrequencyDictionary) – такое название было выбрано для данного приложения, переводится оно как операции частотного словаря , что целиком и полностью отражает суть данного программного обеспечения.

### Программное обеспечение, необходимое для функционирования программы

Для корректного функционирования данного программного продукта необходимо, чтобы на персональном компьютере или ноутбуке была установлена ОС от компании Microsoft, а именно Windows, кране желательно, чтобы была установлена последняя версия, именно на ней гарантируется наиболее корректная работа всех функций программы, а также наилучший пользовательский опыт. Также требуется наличие графического адаптера, чтобы устройство могло справляться с обработкой отображения окна приложения. Другие требования к устройству пользователя не предусмотрены.

### Язык программирования, на котором написана программа

Язык программирования для создания курсовой работы избирался исходя из основных требований к структуре для работы со множествами, а также из программных средств и методов, которые позволят наиболее успешно реализовать все запланированные функции. Таким образом, был выбран язык программирования C++, который предоставляет удобные средства для работы с памятью, в целом, и с указателями, в частности. Также выбранная среда разработки и фреймворк предназначены для этого языка программирования.

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

Данный программный продукт был создан с целью помочь пользователям в изучении новых слов, посредством удобного хранения и отображения однотипных данных в таблице, а также для проведения операций над данными словами.

Основной функционал приложения перечислен в следующем списке: создание пустого словаря, добавление нового элемента в словарь, увеличение значения счетчика слова в элементе словаря с заданным словом, вывод элементов словаря в алфавитном порядке, поиск элемента словаря по слову.

Разработанная курсовая работа имеет ряд ограничений на использование и определённые особенности, которые стоит учитывать во время её эксплуатации. Программное решение не является свободно переносимым с одного устройства на другое.

Программа позволяет работать лишь с английскими и русскими буквами. Других типов данных для хранения и обработки данное программное обеспечение не предусматривает, в том числе не предусмотрено создание пользовательских типов данных.

Операция добавления нового слова в словарь выполняется при условии записи букв. Если попробовать записать числа или математические выражения, то появится предупреждение об ошибке.

## Описание логической структуры программы

Логическая структура программы подразумевает использование двух парадигм программирования.

Первая, объектно-ориентированный подход для удобного и наглядного представления иерархии классов и данных, хранящихся в этих классах [5]. Данный подход непременно должен быть использован, когда речь заходит о графическом интерфейсе или о представлении различных типов данных, как единой структуры. Ведь ООП было создано для улучшения типизации и стандартизации данных в программном коде.

Вторая, процедурный стиль программирования [6] для различного рода дополнительных функций или операций, где производятся лишь видоизменения и преобразования над уже созданными объектами классов и переменными. ПП позволяет сохранить простоту читаемости кода, когда необходимо создать функции, внесение которых в конкретный класс неуместно, а создание нового усложняет читаемость.

### Алгоритмы, используемые в программе

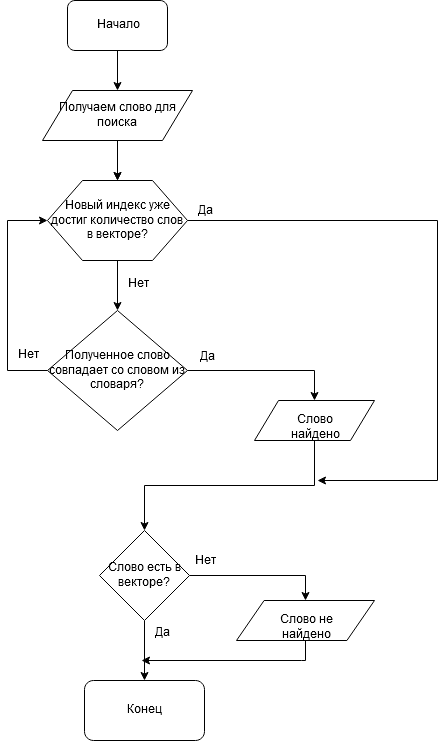
Основными алгоритмами для работы данного приложения являются алгоритмы работы поиска слова и добавления слова в словарь, в котором производятся хранение и обработка введённых пользователем данных. Также важную роль играют различные алгоритмы работы графической части программы, проверки входных данных, отображение слов в частотном словаре, а также интеграция операций вектора в работу самого программного продукта.

#### Алгоритмы работы поиска слова и добавления слова в вектор

При начале работы программы происходит считывание данных из файла, а именно файла с текстом, каждое считанное слово попадает в вектор и получает своё конкретное место внутри самого вектора и таблицы частотного словаря. Для расположения внутри вектора нового слово, оно сравнивается со всеми значениями в векторе. Если это слово уже встречалось, надо увеличить количество. Сам процесс заполнения вектора происходит с помощью цикла. Для разбора процесса заполнения стоит разобрать поиск слова и добавления слова в вектор.

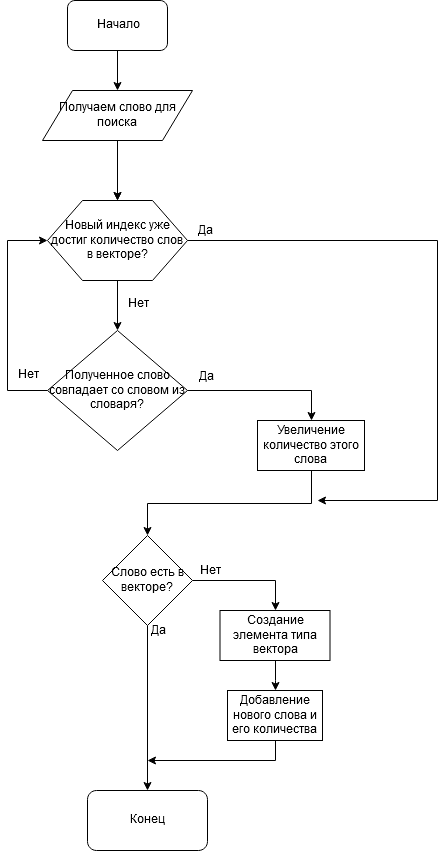
В графическом интерфейсе есть строка для ввода слова. Над ней есть две кнопки для работы с этим словом.

Для поиска слова в векторе нам понадобится логическая переменная, чтобы установить: встречается ли это слово в словаре или нет. С помощью цикла и условий получаем результат. Алгоритм поиска элемента в векторе представлен на рисунке 3.1.



**Рисунок 3.1 – Алгоритм поиска слова в векторе**

Чтобы добавить слово в вектор, надо для начала проверить это слово с помощью предыдущего поиска. Затем, при наличии этого слова надо увеличить его на указанное количество в векторе. Если слово новое, то увеличиваем вектор с помощью добавления нового слова в конец вектора. Это реализовано через метод push\_back(). На рисунке 3.2 представлен вариант обработки добавления нового слова в вектор.

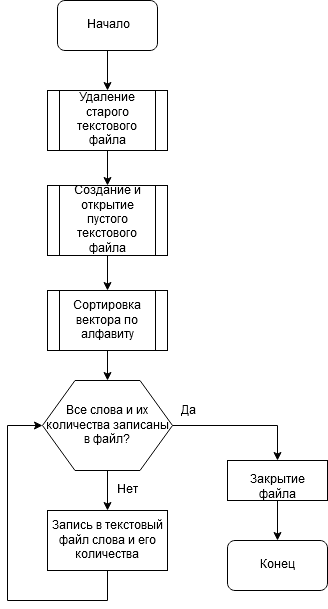


**Рисунок 3.2 – Алгоритм обработки добавления нового слова в вектор**

#### Алгоритмы работы с текстовыми файлами

Также был создан ряд алгоритмов для работы с текстовыми файлами, хранящие текст и сохраненный частотный словарь. Данные алгоритмы касаются чтения данных и добавление оных в вектор для последующей их обработки в выбранной форме обхода, а также для записи данных в файл [8]. Созданные алгоритмы не являются методами вектора, но напрямую связаны с организацией работы структур с текстовым файлом. Функций записи данных в файл было создано несколько, так как каждая из них обрабатывает разные случаи работы программы, от загрузки частотного словаря до добавления или удаления словаря из файла.

Программа не меняет текстовый файл походу выполнения различных операций. Она запоминает все слова в векторе, сортирует его по алфавиту и уже отсортированный вектор записывает через разделительный знак “/” в текстовый файл. Алгоритм кнопки сохранения словаря в текстовый файл представлен на рисунке 3.3.

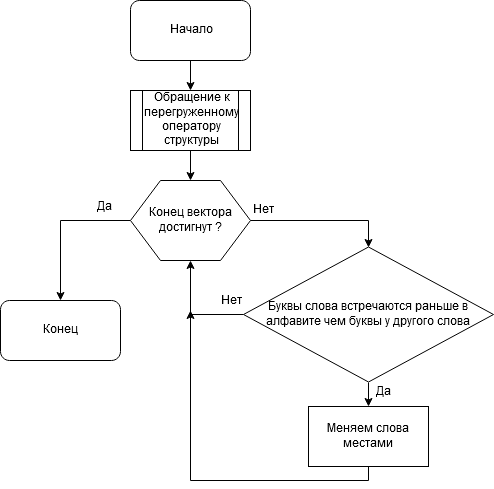


**Рисунок 3.3 – Алгоритм кнопки сохранения словаря в файл**

#### Алгоритм сортировки слов по алфавиту

Было создано множество алгоритмов для произведения операций с графической оболочкой программы, то есть, самим окном приложения. Большая часть из них представляет из себя работу различного вида кнопок, полей ввода и так далее. Особенным образом выделяются функции, которые выполняют основные операции над словарём, так как их реализация является ключевой для приложения подобного толка [10].

Важным алгоритмом в данной программе является алгоритм сортировки слов по алфавиту, причём можно сортировать и английский, и русский алфавит. Это сделано с помощью перегрузки оператора в структуре info для сравнения двух слов и последующей сортировки. Для этого надо указать в качестве параметров сортировки sort, алгоритм которой был построен на базе быстрой сортировки и для использования которого требуется подключенная библиотека <algorithm>, начало и конец вектора words, а также саму структуру info с перегруженным оператором скобки.



**Рисунок 3.4 – Алгоритм сортировки слов по алфавиту**

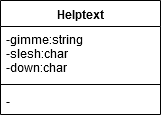
### Структура программы с описанием функций составных частей и связей между ними

Для наглядной и удобной работы программистов с однотипными структурами данных задействуется концепция ООП, которая позволяет произвести категоризацию данных для последующей более структурированной их обработки. Данное приложение также использует данную парадигму, в программе был создан собственный дополнительный класс, который нужен для упрощения работы с файлами. Основой этой программы является вектор, который не имеет собственного класса, к сожалению, однако у него не так уж много дополнительных методов и скорее наличие этого класса лишь усложнило бы программу для понимания, поэтому я использовал структуру и записал её прямо в MyForm.h. Был задействован и класс графических библиотек Visual Studio для создания графического интерфейса.

#### Дополнительный класс для работы с файлом и структура для вектора

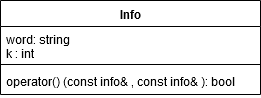
Этот класс необязателен, но он очень помогает. К примеру, строковая переменная gimme нужна для записи слова до пробела. Ты не забываешь, что она существует и работаешь с ней. А использование переменных slesh и down избавляют от постоянного написания слэша и прочего в конце предложений или разделений слов при записи в файл. Методы у класса отсутствуют.

Диаграмму, созданную методами нотации UML [11], дополнительного класса можно увидеть на рисунке 3.5.



**Рисунок 3.5 – Класс Helptext**

Для структуры Info не было создано никаких методов за исключением перегрузки оператора () для сравнения слов. Также используются переменные word и k для хранения слова и его количества.



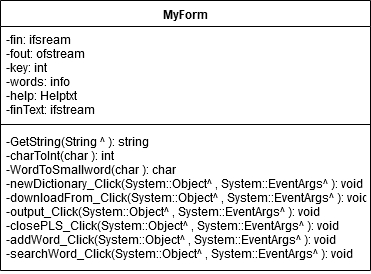
**Рисунок 3.6 – Структура Info**

#### Класс главного окна программы

Одной из самых важных составляющих программы является графическая оболочка, в целом, и класс главного окна программы, в частности, так как в нём происходят все важные проверки и операции.

Большинство методов класса MyForm – это различные события, которые привязаны к определённым компонентам на форме, что вызывает реакцию на нажатие кнопок пользователем, но также были созданы стандартные методы, которые являются проверками на входные данные, введённые пользователем, а также методы для нужного преобразования типов цифр и букв из заглавных в строчные.

На рисунке 3.7 отражены все поля, основные методы и события класса MyForm.



**Рисунок 3.7 – Класс MainWindow и перечисление ElementStatus**

В качестве основных обычных методов класса MyForm были выделены следующие методы:

1. Метод GetString создан для преобразования типов из String^ в string;
2. Метод charToInt представляет из себя перевод чисел типов из char в int;
3. Метод WordToSmallword переводит заглавные буквы в строчные.

Для класса главного окна приложения были выделены следующие основные события кнопок, отвечающие за отклик программы на действия пользователя:

1. Слот newDictionary\_Click удаляет старый словарь и создает новый;
2. Слот downloadFrom\_Click загружает частотный словарь, если он существует;
3. Слот output\_Click нужен для вывода всех слов в словаре в алфавитном порядке;
4. Слот closePLS\_Click отвечает за сохранения словаря и закрытия приложения;
5. Слот addWord\_Click добавляет слово в словарь;
6. Слот searchWord\_Click производит поиск слова по частотному словарю.

## Технические средства, которые используются при работе программы

Для запуска программы необходимо открыть исполняемый файл, который предоставляется пользователю в формате “.exe”. Как было отмечено ранее, для корректной работы продукта необходимо наличие графического адаптера. Чем больше видеопамяти будет предоставлено для приложения, тем более быстрая работа приложения будет осуществлена.

В качестве средства хранения и обработки данных была использована структура динамического массива в виде вектора, созданная самостоятельно с помощью соответствующего шаблона.

Программа использует 2 текстового файла формата “.txt”. Первый файл содержит текст, с которого считываются слова и заносятся в словарь при загрузке приложения. Если он будет отсутствовать, программа сообщит об этом.

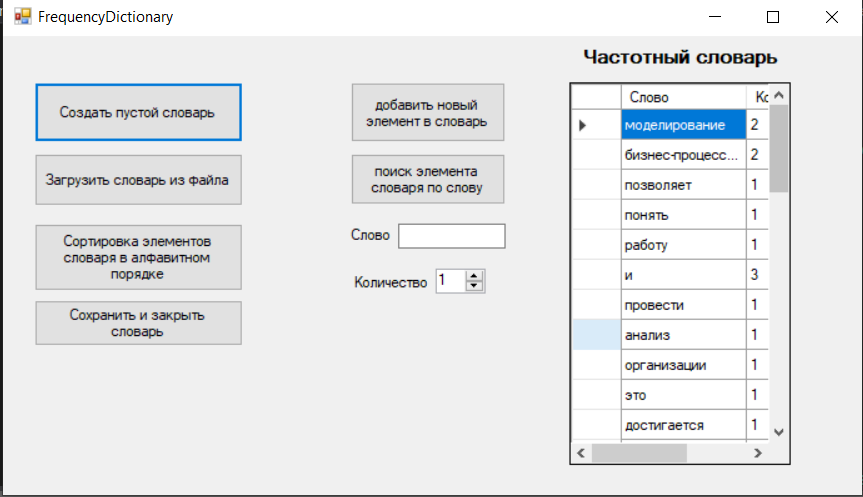
Второй файл формата “.txt” выступает в качестве статического способа хранения данных после завершения работы программы. Данный файл считывается программой при нажатии на кнопку “Загрузить словарь”. При этом он стирает все слова ранее присутствующее в словаре. Если файл не был создан на момент запуска программы или был удалён во время её работы, то кнопка окажется недоступной.

## Вызов программы

При запуске программного продукта появляется главное окно приложения, на котором расположен частотный словарь. Именно в него будут добавляться новые слова, также там будут отображаться все слова из текстового файла.

### Главное окно приложения

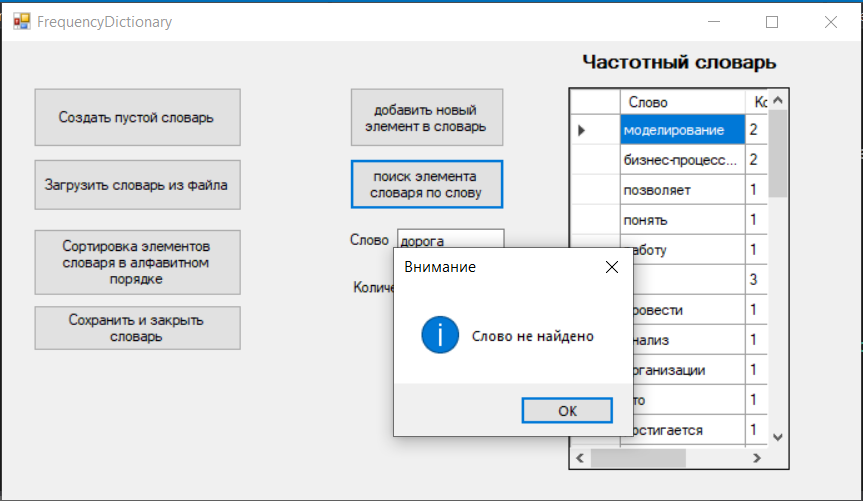
Окно приложения включает в себя не только саму таблицу частотного словаря, но и кнопки, взаимодействующие с ним, а также кнопки, работающие с введенным словом, и строку ввода слова, c привязанным к ней элементом numericUpDown. Помимо этого, были созданы предупреждающие сообщения об ошибках или уведомлениях действий пользователя.



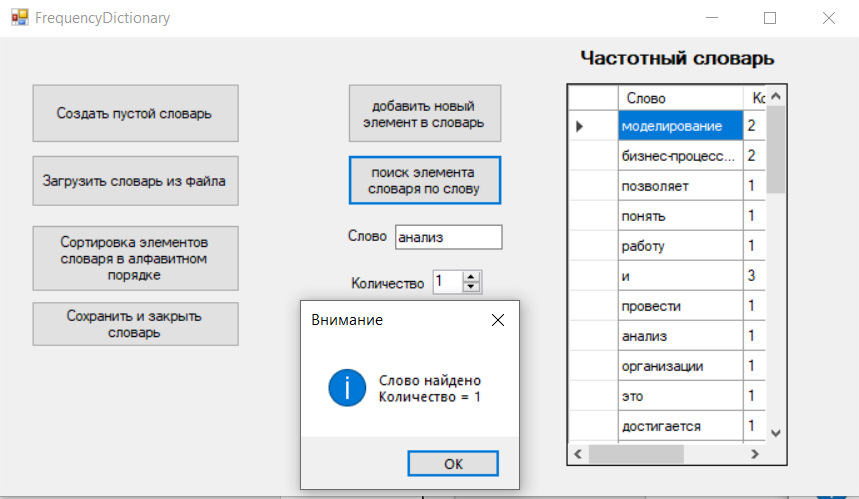
**Рисунок 3.8 – Окно приложения “Частотный словарь”**

Кнопки, взаимодействующие со словарем: создать пустой словарь, загрузить словарь из файла, сортировка элементов словаря в алфавитном порядке, сохранить и закрыть словарь. Для работы с введенным словом понадобились кнопки, которые добавляют слово в словарь и ищут слово по словарю.

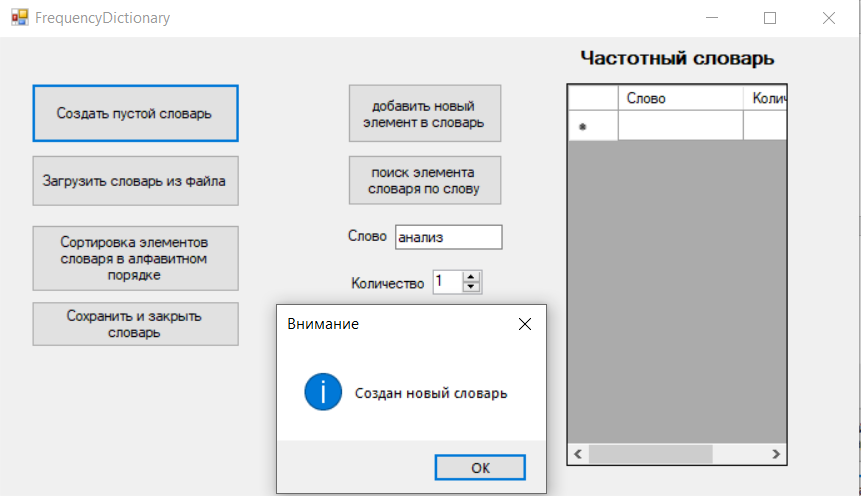
Перечисленные кнопки и загрузка приложения могут вызывать предупреждения о работе со словарём. А также кнопка “Загрузить словарь” не доступна при создание пустого словаря или удаление старого. На рисунках 3.9, 3.10, 3.11, 3.12, 3.13 представлены эти сообщения. Недоступность кнопки “Загрузить словарь” показано на рисунке 3.14.



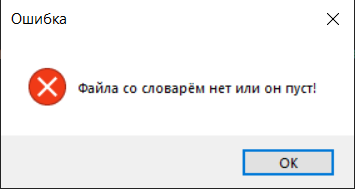
**Рисунок 3.9 – Сообщение “Слово не найдено”**



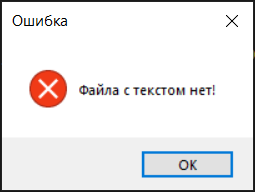
**Рисунок 3.10 – Сообщение “Слово найдено”**



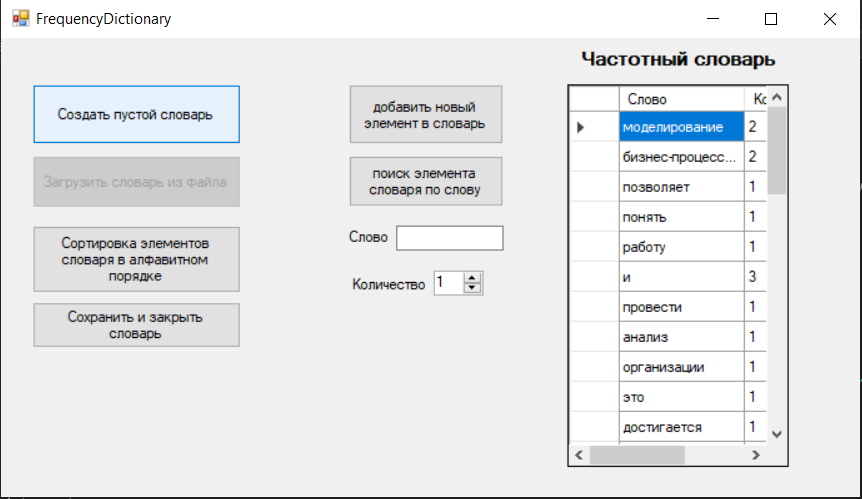
**Рисунок 3.11 – Сообщение “Создан новый словарь”**



**Рисунок 3.12 – Ошибка “Файла со словарём нет или он пуст!”**



**Рисунок 3.13 – Ошибка “Файла с текстом нет!”**



**Рисунок 3.14 – Недоступность кнопки “Загрузить словарь”**

## Входные данные (организация и предварительная подготовка входных данных)

В качестве входных данных программа принимает текстовый файл под названием “TextForDictionary.txt”, хранящий текст, по которому требуется создать частотный словарь. В начале работы программы происходит его считывание, и все слова записываются в ячейку таблицы, а рядом со словом пишется его количество повторений в тексте. Также во время использования приложения есть возможность загрузить словарь.

## Выходные данные

В качестве выходных данных выступают все данные, которые пользователь вводил в программу, то есть, добавленные им слова. Весь этот поток данных заносится в текстовый файл, название которого обозначено в предыдущем подразделе. Запись данных в файл происходит перед завершением программы.

# ЗАКЛЮЧЕНИЕ

На протяжении всего процесса проектирования и создания программного продукта были изучены основные структуры данных, которые используются для хранения и обработки различных типов данных, также были разобраны основные алгоритмы для работы с этими структурами данных и их грамотное использование.

В рамках реализации графической оболочки приложения были приобретены основные навыки для совмещения работы графических компонентов и различного вида алгоритмов и структур, для их органичного взаимодействия и удобства использования программы.

Основными задачами во время выполнения курсовой работы были: создание программы, ориентированной для работы с частотным словарем и словами и проведения основных операций над ними, которая будет использовать вектор, который реализует динамический массив в качестве основной структуры для обработки данных, используя ключевые алгоритмы, которые были созданы для вектора, также обучение основным навыкам в анализе и выборе наиболее подходящей структуры в зависимости от специализации программного продукта.

Все из перечисленных задач были успешно выполнены и качественно интегрированы в созданное приложение. В дополнении к этому был создан ряд дополнительных функций, повышающий удобство приложения.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Пышкин, Е. В. Структуры данных и алгоритмы: реализация на C/C++ / Е. В. Пышкин.— СПб.: ФТК СПБГПУ, 2009.— 200 с.
2. Ахо, А. Структуры данных и алгоритмы / Ахо Альфдред, Хопкрофт Джон, Ульман Джеффри. — М.: Издательский дом “Вильяме”, 2003.— 384 с.
3. Макконелл, Дж. Основы современных алгоритмов / Джон Макконелл.— 2-е изд., перераб.— М.: Техносфера, 2004.— 368 с.
4. Готтшлинг, П. Современный C++. Для программистов, инженеров и учёных / Питер Готтшлинг.— М.: ООО “И. Д. Вильямс”, 2017.— 512 c.
5. **Лафоре,**Р. Объектно-ориентированное программирование в C++ / Роберт Лафоре.— 4-е изд., перераб. и доп.— М.: Питер, 2004.— 924 с.
6. Седжвик, Р. Алгоритмы на C++ / Роберт Седжвик.— М.: Издательский дом “Вильямс”, 2011.— 1056 с.
7. Алгоритмы: построение и анализ / Томас Кормен Х. [и др.].— 2-е изд.— М.: Издательский дом “Вильямс”, 2005.— 1296 с.
8. Уоррен-Младший, Г. Алгоритмические трюки для программистов / Генри Уоррен-Младший.— 2-е изд.— М.: ООО “И. Д. Вильямс”, 2014.— 512 с.
9. **Лафоре,**Р. Структуры данных и алгоритмы в Java / Роберт Лафоре.— 2-е изд., перераб. и доп.— СПб.: Питер, 2013.— 704 с.
10. Белова, Л. Ю. Элементы теории множеств и математической логики: учебное пособие / Л. Ю. Белова, Ю. А. Белов; Яросл. гос. ун-т им. П.Г. Демидова.— Ярославль: ЯрГУ, 2012.— 204 с.
11. Фаулер**,**М. Язык UML. Основы / Мартин Фаулер.— 3-е изд.— СПб.: Символ, 2005.— 192 с.
12. Шлее, М. Qt 5.3. Профессиональное программирование на C++ / Макс Шлее.— СПб.: БХВ-Петербург, 2015.— 928 с.

# ПРИЛОЖЕНИЯ

Приложение А. Дополнительный класс программы

Приложение Б. Основной класс программы

### Приложение А

Дополнительный класс программы

*Листинг А.1 – Helptext.h:*

#pragma once

#include <string>

#include <fstream>

class Helptext {

public:

std::string gimme;

char slesh = '/';

char down = '\n';

};

*Листинг А.2 – Helptext.cpp:*

#include "Helptext.h"

### Приложение Б

Основной класс программы

*Листинг Б.1 -MyForm.h:*

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <algorithm>

#include "Helptext.h"

namespace OperationsOfFrequencyDictionary {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace std;

/// <summary>

/// Сводка для MyForm

/// </summary>

ifstream fin; //чтение файла

ofstream fout; // запись файла

int key = 0; // количество слов

struct info // информация о слове

{

string word;

int k;

bool operator()(const info& lhs, const info& rhs) const {// сравнение для сортировки

return lhs.word < rhs.word;

}

};

vector <info> words; // словарь

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

fin.open("Dictionary.txt");

fin.get();

if (!fin.is\_open() || fin.eof())

{

MessageBox::Show("Файла со словарём нет или он пуст!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

downloadFrom->Enabled = false;

}

fin.close();

Helptext \*help = new Helptext;

ifstream finText;

finText.open("TextForDictionary.txt");

if (!finText.is\_open())

{

MessageBox::Show("Файла с текстом нет!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

else

{

char ch, chfull[50];

int x = 0;

for (int l1 = 0; l1 < 50; l1++)

{

chfull[l1] = 0; //объявляю строки

}

bool povtor;

info newwords;

while (!finText.eof())

{

povtor = false;

x = 0;

for (int l = 0; l < 50; l++)

{

chfull[l] = 0;

}

help->gimme = "";

while (finText.get(ch))

{

if (int(ch) > 47 && int(ch) < 59)//48-58 это ASCII-коды цифр

{

while (ch != ' ')

finText.get(ch);

}

if (ch != ' ' && ch != ',' && ch != '.' && ch != '!' && ch != '?' && ch != help->down)

{

if ((int(ch) < - 32) || ((int(ch) > 64) && (int(ch) < 91)))

{

ch = WordToSmallword(ch);

}

if (ch == 'Ё')

ch = 'ё';

chfull[x] = ch;

x++;

}

else break;

}

help->gimme = chfull;

if (help->gimme != "")

{

for (int i = 0; i < key; i++)

{

if (words[i].word == help->gimme)

{

words[i].k++;

povtor = true;

break;

}

}

if (!povtor)

{

newwords.word = help->gimme;

newwords.k = 1;

key++;

dataGridView1->Rows->Add();

words.push\_back(newwords);

}

}

}

}

for (int i = 0; i < key; i++)

{

dataGridView1->Rows[i]->Cells[0]->Value = gcnew String(words[i].word.c\_str());

dataGridView1->Rows[i]->Cells[1]->Value = words[i].k;

}

finText.close();

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Button^ newDictionary;

private: System::Windows::Forms::Button^ addWord;

private: System::Windows::Forms::Button^ output;

protected:

private: System::Windows::Forms::Button^ searchWord;

private: System::Windows::Forms::Button^ downloadFrom;

private: System::Windows::Forms::Button^ closePLS;

private: System::Windows::Forms::TextBox^ textBox1;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::DataGridView^ dataGridView1;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::NumericUpDown^ numericUpDown1;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column1;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column2;

protected:

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->newDictionary = (gcnew System::Windows::Forms::Button());

this->addWord = (gcnew System::Windows::Forms::Button());

this->output = (gcnew System::Windows::Forms::Button());

this->searchWord = (gcnew System::Windows::Forms::Button());

this->downloadFrom = (gcnew System::Windows::Forms::Button());

this->closePLS = (gcnew System::Windows::Forms::Button());

this->textBox1 = (gcnew System::Windows::Forms::TextBox());

this->label1 = (gcnew System::Windows::Forms::Label());

this->dataGridView1 = (gcnew System::Windows::Forms::DataGridView());

this->label2 = (gcnew System::Windows::Forms::Label());

this->numericUpDown1 = (gcnew System::Windows::Forms::NumericUpDown());

this->label3 = (gcnew System::Windows::Forms::Label());

this->Column1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column2 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->BeginInit();

this->SuspendLayout();

//

// newDictionary

//

this->newDictionary->Location = System::Drawing::Point(33, 45);

this->newDictionary->Name = L"newDictionary";

this->newDictionary->Size = System::Drawing::Size(223, 59);

this->newDictionary->TabIndex = 0;

this->newDictionary->Text = L"Создать пустой словарь";

this->newDictionary->UseVisualStyleBackColor = true;

this->newDictionary->Click += gcnew System::EventHandler(this, &MyForm::newDictionary\_Click);

//

// addWord

//

this->addWord->Location = System::Drawing::Point(371, 45);

this->addWord->Name = L"addWord";

this->addWord->Size = System::Drawing::Size(165, 59);

this->addWord->TabIndex = 1;

this->addWord->Text = L"добавить новый элемент в словарь";

this->addWord->UseVisualStyleBackColor = true;

this->addWord->Click += gcnew System::EventHandler(this, &MyForm::addWord\_Click);

//

// output

//

this->output->Location = System::Drawing::Point(33, 184);

this->output->Name = L"output";

this->output->Size = System::Drawing::Size(223, 67);

this->output->TabIndex = 2;

this->output->Text = L"Сортировка элементов словаря в алфавитном порядке";

this->output->UseVisualStyleBackColor = true;

this->output->Click += gcnew System::EventHandler(this, &MyForm::output\_Click);

//

// searchWord

//

this->searchWord->Location = System::Drawing::Point(371, 116);

this->searchWord->Name = L"searchWord";

this->searchWord->Size = System::Drawing::Size(165, 50);

this->searchWord->TabIndex = 3;

this->searchWord->Text = L"поиск элемента словаря по слову";

this->searchWord->UseVisualStyleBackColor = true;

this->searchWord->Click += gcnew System::EventHandler(this, &MyForm::searchWord\_Click);

//

// downloadFrom

//

this->downloadFrom->Location = System::Drawing::Point(33, 116);

this->downloadFrom->Name = L"downloadFrom";

this->downloadFrom->Size = System::Drawing::Size(223, 52);

this->downloadFrom->TabIndex = 4;

this->downloadFrom->Text = L"Загрузить словарь из файла";

this->downloadFrom->UseVisualStyleBackColor = true;

this->downloadFrom->Click += gcnew System::EventHandler(this, &MyForm::downloadFrom\_Click);

//

// closePLS

//

this->closePLS->Location = System::Drawing::Point(33, 260);

this->closePLS->Name = L"closePLS";

this->closePLS->Size = System::Drawing::Size(223, 46);

this->closePLS->TabIndex = 6;

this->closePLS->Text = L"Сохранить и закрыть словарь";

this->closePLS->UseVisualStyleBackColor = true;

this->closePLS->Click += gcnew System::EventHandler(this, &MyForm::closePLS\_Click);

//

// textBox1

//

this->textBox1->Location = System::Drawing::Point(422, 184);

this->textBox1->Name = L"textBox1";

this->textBox1->Size = System::Drawing::Size(114, 22);

this->textBox1->TabIndex = 7;

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(368, 187);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(48, 17);

this->label1->TabIndex = 8;

this->label1->Text = L"Слово";

//

// dataGridView1

//

this->dataGridView1->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView1->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(2) {

this->Column1,

this->Column2

});

this->dataGridView1->Location = System::Drawing::Point(604, 45);

this->dataGridView1->Name = L"dataGridView1";

this->dataGridView1->RowTemplate->Height = 24;

this->dataGridView1->Size = System::Drawing::Size(236, 377);

this->dataGridView1->TabIndex = 9;

//

// label2

//

this->label2->AutoSize = true;

this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10.8F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label2->Location = System::Drawing::Point(615, 9);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(203, 24);

this->label2->TabIndex = 10;

this->label2->Text = L"Частотный словарь";

//

// numericUpDown1

//

this->numericUpDown1->Location = System::Drawing::Point(462, 229);

this->numericUpDown1->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

this->numericUpDown1->Name = L"numericUpDown1";

this->numericUpDown1->Size = System::Drawing::Size(54, 22);

this->numericUpDown1->TabIndex = 11;

this->numericUpDown1->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(370, 234);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(86, 17);

this->label3->TabIndex = 12;

this->label3->Text = L"Количество";

//

// Column1

//

this->Column1->HeaderText = L"Слово";

this->Column1->Name = L"Column1";

this->Column1->ReadOnly = true;

//

// Column2

//

this->Column2->HeaderText = L"Количество";

this->Column2->Name = L"Column2";

this->Column2->ReadOnly = true;

this->Column2->Width = 90;

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(916, 452);

this->Controls->Add(this->label3);

this->Controls->Add(this->numericUpDown1);

this->Controls->Add(this->label2);

this->Controls->Add(this->dataGridView1);

this->Controls->Add(this->label1);

this->Controls->Add(this->textBox1);

this->Controls->Add(this->closePLS);

this->Controls->Add(this->downloadFrom);

this->Controls->Add(this->searchWord);

this->Controls->Add(this->output);

this->Controls->Add(this->addWord);

this->Controls->Add(this->newDictionary);

this->Name = L"MyForm";

this->Text = L"FrequencyDictionary";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: std::string GetString(String ^ s)

{

std::string os;

using namespace Runtime::InteropServices;

const char\* chars =

(const char\*)(Marshal::StringToHGlobalAnsi(s)).ToPointer();

os = chars;

Marshal::FreeHGlobal(IntPtr((void\*)chars));

return os;

}

private: System::Void newDictionary\_Click(System::Object^ sender, System::EventArgs^ e) {

remove("Dictionary.txt");

dataGridView1->Rows->Clear();

key = 0;

fout.open("Dictionary.txt");

words.clear();

MessageBox::Show("Создан новый словарь", "Внимание", MessageBoxButtons::OK, MessageBoxIcon::Information);

fout.close();

downloadFrom->Enabled = false;

}

int charToInt(char c) {

return c - '0';

}

char WordToSmallword(char c)

{

return c + 32; }

private: System::Void downloadFrom\_Click(System::Object^ sender, System::EventArgs^ e) {

fin.open("Dictionary.txt");

words.clear();

dataGridView1->Rows->Clear();

key = 0;

char ch, chfull[50], chnumber;

Helptext \*help = new Helptext;

int l = 0, i = 0;

while (!fin.eof())

{

for (int l1 = 0; l1 < 50; l1++)

{

chfull[l1] = 0;

}

dataGridView1->Rows->Add();

while (fin.get(ch))

{

if (ch == '\n')

{

chfull[l] = 0;

break;

}

if (ch != '/')

{

chfull[l] = ch;

l++;

}

else

{

fin.get(ch);

chnumber = ch;

}

}

info newwords;

help->gimme = chfull;

key++;

dataGridView1->Rows[key - 1]->Cells[0]->Value = gcnew System::String(help->gimme.c\_str());

newwords.word = help->gimme;

dataGridView1->Rows[key - 1]->Cells[1]->Value = charToInt(chnumber);

newwords.k = charToInt(chnumber);

words.push\_back(newwords);

help->gimme = "";

l = 0;

}

fin.close();

}

private: System::Void output\_Click(System::Object^ sender, System::EventArgs^ e) {

sort(words.begin(), words.end(), info());

for (int i = 0; i < key; i++)

{

dataGridView1->Rows[i]->Cells[0]->Value = gcnew System::String(words[i].word.c\_str());

dataGridView1->Rows[i]->Cells[1]->Value = words[i].k;

}

}

private: System::Void closePLS\_Click(System::Object^ sender, System::EventArgs^ e) {

//загрузить в файл словаря

remove("Dictionary.txt");

fout.open("Dictionary.txt", ofstream::app);

sort(words.begin(), words.end(), info());

for (int i = 0; i < key-1; i++)

fout << words[i].word << "/" << words[i].k << "\n";

if (key != 0)

fout << words[key-1].word << "/" << words[key - 1].k;

fout.close();

exit(0);

}

private: System::Void addWord\_Click(System::Object^ sender, System::EventArgs^ e) {

bool proverka = true;

string s = GetString(textBox1->Text);

for (int i = 0; i < key; i++)

if (s == words[i].word)

{

proverka = false;

words[i].k += int(numericUpDown1->Value);

dataGridView1->Rows[i]->Cells[1]->Value = words[i].k;

break;

}

if (proverka == true)

{

info newwords;

newwords.word = s;

newwords.k = int(numericUpDown1->Value);

words.push\_back(newwords);

key++;

dataGridView1->Rows->Add();

dataGridView1->Rows[key - 1]->Cells[0]->Value = textBox1->Text;

dataGridView1->Rows[key - 1]->Cells[1]->Value = newwords.k;

}

numericUpDown1->Value = 1;

}

private: System::Void searchWord\_Click(System::Object^ sender, System::EventArgs^ e) {

string s = GetString(textBox1->Text);

bool proverka = false;

for (int i = 0; i < key; i++)

if (s == words[i].word)

{

proverka = true;

MessageBox::Show("Слово найдено\n" + "Количество = " + words[i].k, "Внимание", MessageBoxButtons::OK, MessageBoxIcon::Information);

break;

}

if (!proverka)

MessageBox::Show("Слово не найдено", "Внимание", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

};

}

*Листинг Б.2 - MyForm.h:*

#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void main(array<String^>^ args) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

OperationsOfFrequencyDictionary::MyForm form;

Application::Run(%form);

}