МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)

Институт № 7 «Интеллектуальные и робототехнические системы»

Кафедра 703 «Системное проектирование авиакомплексов»

**Лабораторная работа**

**по курсу «Технологии разработки программного обеспечения»**

Тема: «**Коллективная разработка программного комплекса**

**анализа текстов русского языка**

**с целью поиска различных членов предложения**»

Работу выполнили

студенты группы М70-406С-20

Ибрагимов Т.Р.

Кокурин К.А.

Путинцев И.С.

Сухомейло Н. Я.

студенты группы М7О-406с-20

Поляков Е.А.

Трофимов Н.Д.

Работу принял

ст. преподаватель кафедры 703

Барчев Н.Б.

**Москва**

**2024**

**Оглавление**

[1. Задание 3](#__RefHeading___Toc351_1486839431)

[2. Псевдокод алгоритмов решения поставленных задач 4](#__RefHeading___Toc353_1486839431)

[Общий вид 4](#__RefHeading___Toc12162_1486839431)

[Алгоритмы поиска союзов, предлогов, частиц 4](#__RefHeading___Toc12164_1486839431)

[Алгоритм поиска сказуемых 4](#__RefHeading___Toc12166_1486839431)

[Алгоритм поиска определений 5](#__RefHeading___Toc12168_1486839431)

[3. Сведения о программной реализации 5](#__RefHeading___Toc355_1486839431)

[Описание входных и выходных данных 5](#__RefHeading___Toc357_1486839431)

[Краткое описание спроектированных программных единиц 6](#__RefHeading___Toc359_1486839431)

[Файл «MyForm.cpp» 6](#__RefHeading___Toc361_1486839431)

[Файл «MyForm1.cpp» 8](#__RefHeading___Toc12170_1486839431)

[Файл «Frame.cpp» 9](#__RefHeading___Toc12172_1486839431)

[Файл «definition.cpp» 10](#__RefHeading___Toc12174_1486839431)

[Файл «predicate.cpp» 10](#__RefHeading___Toc12176_1486839431)

# 1. Задание

Разработать программный комплекс, предназначенный для проведения анализа произвольных текстов русского языка с целью поиска в них различных членов предложения (подлежащих, сказуемых, определений, дополнений, обстоятельств), а также слов, не являющихся членами предложения, и последующей обработки полученной информации.

Начальные требования к разработке:

* реализация (по выбору пользователя) возможности анализа как текста, поступающего из текстового файла, так и вводимого с клавиятуры;
* учет отсутствия ограничений на размер входного файла;
* подсчет искомых слов в поступающем тексте и формирование упорядоченных списков найденных слов с информацией о числе появлений каждого слова в соответствующей роли, а также идентификацией и визуализацией предложений, в которых данное слово появляется в этой роли;
* реализация (по выбору пользователя) возможности выполнения как полного
* анализа поступающего текста, так и анализа с целью поиска членов предложения конкретного типа, либо слов, не являющихся членами предложення;
* реализация (по выбору пользователя) возможности вывода результатов анализа на экран, либо в текстовый файл (файлы); вывод результатов анализа на экран должен предоставлять возможности удобного просмотра выводимой информации, а также возможность визуализации в
* процессе просмотра предложений, в которых те или иные слова появляются в соответствующей роли;
* реализация пользовательского интерфейса, адекватного сложности решаемой задачи;
* при разработке использовать средства одной из реализаций языка программирования С++.

Дополнительно выполнить по результатам работы анализ полученных результатов с целью оценки степени их близости к реальности. Учесть ошибки первого и второго рода.

# 2. Псевдокод алгоритмов решения поставленных задач

## Общий вид

Выбрать файл с текстом или ввести его в окне ввода

Разделить исходный текст на предложения построчно и пронумеровать их

Сформировать матрицу слов исходного текста, где в каждая строка — предложение, а каждый элемент в строке — слово или знак препинания

Осуществить фильтрацию матрицы слов исходного текста, проанализировав ее элементы с помощью функций поиска подлежащих, сказуемых, определений, обстоятельств, дополнений, союзов, частиц, знаков препинаний

Сформировать матрицу уникальных слов исходного текста, исключив повтор слов, учет служебных частей речи и знаков препинаний, подсчитывая при этом число повторений и номера предложений, связанных с рассматриваемым словом

Сформировать название выходного файла

Записать матрицу уникальных слов исходного текста в файл

## Алгоритмы поиска союзов, предлогов, частиц

Итерация по предложениям

итерация по словам предложения

если слово совпадает с тем, что входит в словарь союзов, предлогов или частиц

то отметить его как служебную часть речи

## Алгоритм поиска сказуемых

Итерация по предложениям

итерация по словам предложения

если у слова не опредён тип, то поместить его в буферные потоки для составных именного и глагольного сказуемых

проанализировать слово из буферных потоков на соответствие признаков вспомогательных глаголов

если соответствует вспомогательному глаголу для составного именного сказуемого, то выставить тип сказуемое для рассматриваемого слова и последующего

если соответствует вспомогательному глаголу для составного глагольного сказуемого, то выставить тип сказуемое для рассматриваемого слова и если последующее слово соответствует признакам глагола также выставить ему тип сказуемое, иначе провести анализ последующего за последующим слова и если оно соответствует признакам глагола также выставить ему тип сказуемое

## Алгоритм поиска определений

Итерация по предложениям

итерация по словам предложения

если окончание слова совпадает с элементом массива окончаний определений, то определить слово по типу определение

# 3. Сведения о программной реализации

При написании программы использовался язык программирования С++. Стандарт языка ISO C++ 17 Standard. Среда разработки – Microsoft Visual Studio 2019, версия 16.11.18. Минимальные системные требования к программе: ОС Windows 10, процессор Intel Core i1-10105F с частотой 3.7 ГГц, оперативная память 8 Гб, 200 Мб свободного места на носителе информации. Необходимо наличие пакетов .NET Framework 4,4.5, 4.5.1, 4.5.2, 4.6, 4.6.1 targeting Pack, SDK для .NET Framework 4.8, Поддержка C++/CLI для средств сборки версии 141 (14.16), MSVC версии 141 — библиотеки C++ для VS 2017 для ARM64 с устранением рисков Spectre (версия 14.16).

## Описание входных и выходных данных

В качестве входных данных выступает любой текст на русском языке, служебные файлы для анализа слов на признаки. В качестве выходных данных выступает пронумерованный входной текст в виде текстового файла и размеченный текстовый файл с найденными сказуемыми в формате: название сказуемого, частота встречаемости, номера предложений, где они встретились, статистика по словам в виде таблицы внутри программы.

## Краткое описание спроектированных программных единиц

### Файл «MyForm.cpp»

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm::SubmitButton\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | Вызов функций разбиения текста построчно, анализа на члены предложения, вывода исходного текста на экран |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | String^ Project4::MyForm::get\_current\_directory() |
| Краткое описание | Получение в качестве строки String^ местоположения выбранного пользователем файла |
| Входные параметры | Отсутствуют |
| Выходные параметры | Строка формата String^ |

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm::создатьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | Очистка матрицы исходных слов, очистка окна, запрещение использования кнопок |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm::открытьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | Выбор файла с исходным текстом |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm::ShowInTextByColour\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | Отображение членов предложения цветом |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | string wordtype\_int\_to\_string(const int& need\_i) |
| Краткое описание | Переработка выдачи типа словами под вектор уникальных слов |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Целое число, соответствующее типу члена предложения из перечисления |

|  |  |
| --- | --- |
| Название функции | void write\_to\_file(ofstream& writing\_filestream) |
| Краткое описание | Запись в выходной файл результатов анализа текста |
| Входные параметры | Файловый поток на запись |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm::сохранитьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | Формирование названия выходного файла и запись в файл матрицы уникальных слов |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

## Файл «MyForm1.cpp»

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm1::ShowSubject\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | отображение всех подлежащих |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm1::ShowPredicate\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | отображение всех сказуемых |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm1::ShowAttribute\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | отображение всех определений |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm1::ShowObject\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | отображение всех обстоятельств |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | System::Void Project4::MyForm1::ShowAdverbialModifier\_Click(System::Object^ sender, System::EventArgs^ e) |
| Краткое описание | отображение всех дополнений |
| Входные параметры | Форма вызова, событие вызова |
| Выходные параметры | Отсутствуют |

## Файл «Frame.cpp»

|  |  |
| --- | --- |
| Название функции | void split\_text(string& line, const string& file\_text) |
| Краткое описание | преобразование исходного текста в пронумерованный файл |
| Входные параметры | Входной текст в виде строки и название файла для записи пронумерованного текста |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | void find\_union(vector<vector<Word>>& whole\_text) |
| Краткое описание | поиск союзов в тексте |
| Входные параметры | Матрица исходных слов |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | void find\_preposition(vector<vector<Word>>& whole\_text) |
| Краткое описание | поиск предлогов в тексте |
| Входные параметры | Матрица исходных слов |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | void find\_particle(vector<vector<Word>>& whole\_text) |
| Краткое описание | поиск частиц в тексте |
| Входные параметры | Матрица исходных слов |
| Выходные параметры | Отсутствуют |

|  |  |
| --- | --- |
| Название функции | void filter(vector<vector<Word>>& whole\_text) |
| Краткое описание | фильтрация слов на члены предложения и части речи |
| Входные параметры | Матрица исходных слов |
| Выходные параметры | Отсутствуют |

## Файл «definition.cpp»

|  |  |
| --- | --- |
| Название функции | void find\_definitions() |
| Краткое описание | поиск определений в тексте |
| Входные параметры | Матрица исходных слов |
| Выходные параметры | Отсутствуют |

## Файл «predicate.cpp»

|  |  |
| --- | --- |
| Название функции | bool check\_for\_exclusion(const string& inp, const string& reason) |
| Краткое описание | проверка слова на исключения сказуемых |
| Входные параметры | Матрица исходных слов |
| Выходные параметры | True если слово соответствует сказуемому, false в ином случае |

|  |  |
| --- | --- |
| Название функции | bool chk\_end(const string& inp, const string& rf) |
| Краткое описание | проверка слова на окончания глаголов |
| Входные параметры | Слово, признак проверки |
| Выходные параметры | True если слово соответствует сказуемому, false в ином случае |

|  |  |
| --- | --- |
| Название функции | bool check\_for\_verb\_signs(const string& inp) |
| Краткое описание | проверка на признаки сказуемого по всем его особенностям |
| Входные параметры | слово |
| Выходные параметры | True если слово соответствует сказуемому, false в ином случае |

|  |  |
| --- | --- |
| Название функции | string ComplexVerb(const string& inp, stringstream& buff, int& offset) |
| Краткое описание | анализ на составное глагольное сказуемое |
| Входные параметры | Слово, буферный строковый поток для составных глагольных сказуемых, смещение по матрице исходных слов |
| Выходные параметры | Составное глагольное сказуемое или ничего |

|  |  |
| --- | --- |
| Название функции | string CompexlNominal(const string& str, stringstream& buff) |
| Краткое описание | анализ на составное именное сказуемое |
| Входные параметры | Слово, буферный строковый поток для составных именных сказуемых, смещение по матрице исходных слов |
| Выходные параметры | Составное именное сказуемое или ничего |

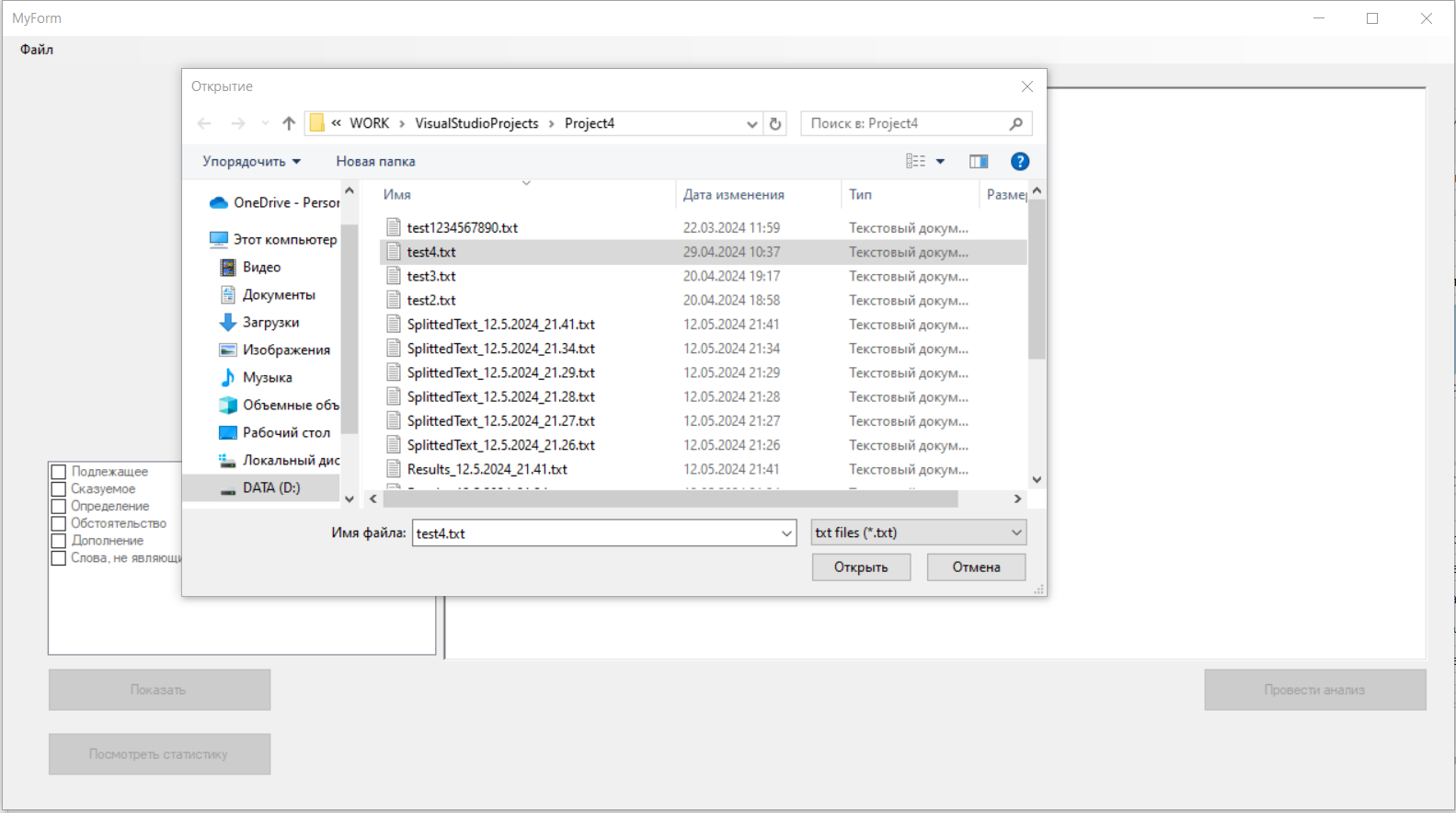
|  |  |
| --- | --- |
| Название функции | void find\_predicate() |
| Краткое описание | основная функция поиска сказуемых |
| Входные параметры | Отсутствуют |
| Выходные параметры | Отсутствуют |

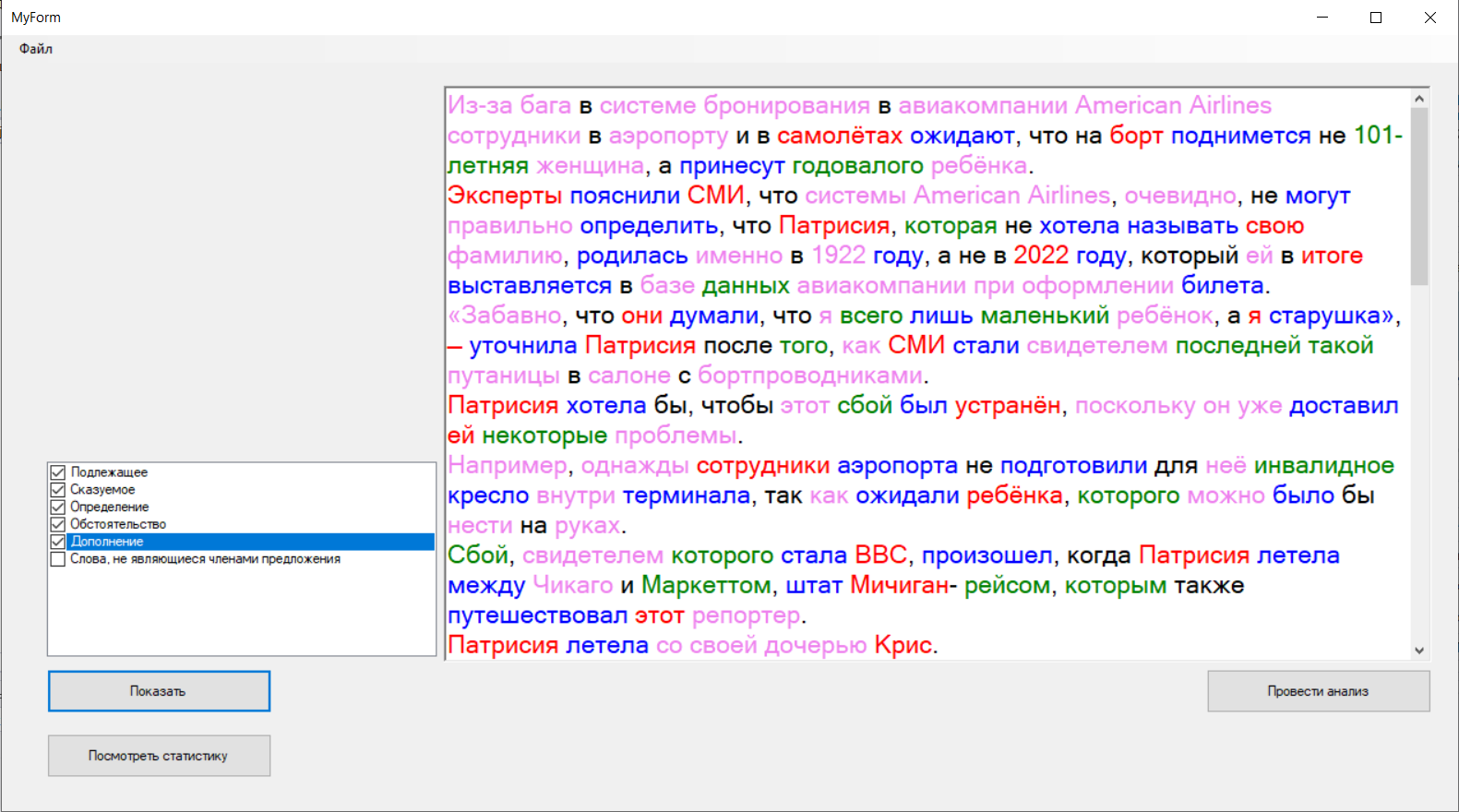
# 4. Инструкция пользователя

1. Нажать кнопку «Файл»
2. Нажать кнопку «Открыть»
3. В открывшемся диалоговом окне проводника выбрать файл с текстом и нажать кнопку «Открыть»
4. Нажать кнопку «Провести анализ». Убедиться в доступности выбора отображения членов предложения и кнопки «Посмотреть статистику»
5. Повторить п.1 настоящей инструкции
6. Нажать кнопку «Сохранить». Проверить наличие полученного файла в директории с программой.

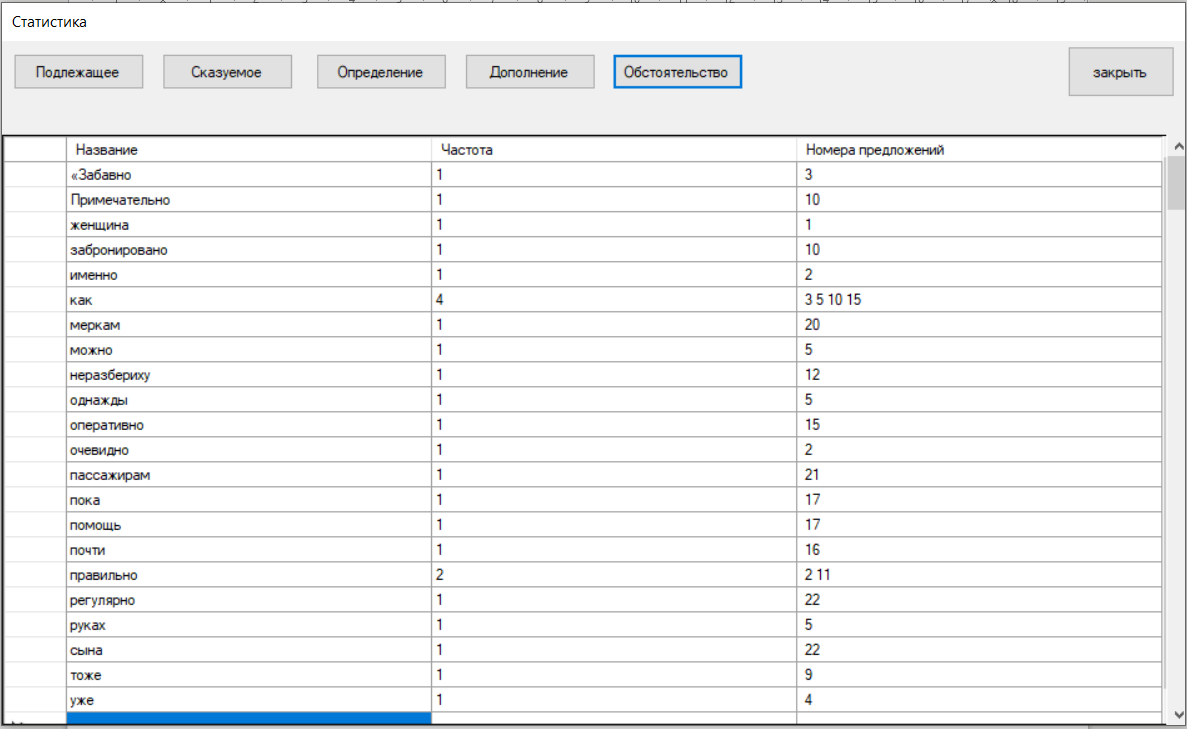
# 5. Тестирование

Выбор исходного файла:



Отображение цветом членов предложения:

Отображение матрицы уникальных слов в табличном виде. Выбор обстоятельств для отображения:



# Приложение. Листинг программной разработки

## Файл «MyForm.h»

#pragma once

#pragma comment(lib, "winmm.lib")

#pragma warning(disable : 2020) // для доступности стандарта языка C++ 17

#include<sstream>

#include<exception>

#include<Windows.h>

#include<msclr\marshal\_cppstd.h> // библиотека для преобразования строк типа String^ в std::string и обратно

#include"Frame.hpp"

#include"MyForm1.h"

namespace Project4

{

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 System::IO;

using namespace std;

using namespace msclr::interop;

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

{

public:

MyForm(void) // конструктор окна формы

{

InitializeComponent();

LoadLibs(); // Это запускаем в начале

}

~MyForm() // деструктор окна формы

{

if (components)

{

UnloadLibs(); // Это запускаем в конце

delete components;

}

}

private: System::Windows::Forms::MenuStrip^ menuStrip1;

private: System::Windows::Forms::ToolStripMenuItem^ файлToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ создатьToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ открытьToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ сохранитьToolStripMenuItem;

private: System::Windows::Forms::RichTextBox^ TextDisplay;

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

private: System::Windows::Forms::CheckedListBox^ SelectSentenceMembers;

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

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

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->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());

this->файлToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->создатьToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->открытьToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->сохранитьToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->TextDisplay = (gcnew System::Windows::Forms::RichTextBox());

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

this->SelectSentenceMembers = (gcnew System::Windows::Forms::CheckedListBox());

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

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

this->menuStrip1->SuspendLayout();

this->SuspendLayout();

//

// menuStrip1

//

this->menuStrip1->ImageScalingSize = System::Drawing::Size(20, 20);

this->menuStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(1) { this->файлToolStripMenuItem });

this->menuStrip1->Location = System::Drawing::Point(0, 0);

this->menuStrip1->Name = L"menuStrip1";

this->menuStrip1->Size = System::Drawing::Size(1262, 28);

this->menuStrip1->TabIndex = 0;

this->menuStrip1->Text = L"menuStrip1";

//

// файлToolStripMenuItem

//

this->файлToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(3) {

this->создатьToolStripMenuItem,

this->открытьToolStripMenuItem, this->сохранитьToolStripMenuItem

});

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

this->файлToolStripMenuItem->Size = System::Drawing::Size(59, 24);

this->файлToolStripMenuItem->Text = L"Файл";

//

// создатьToolStripMenuItem

//

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

this->создатьToolStripMenuItem->Size = System::Drawing::Size(224, 26);

this->создатьToolStripMenuItem->Text = L"Создать";

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

//

// открытьToolStripMenuItem

//

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

this->открытьToolStripMenuItem->Size = System::Drawing::Size(224, 26);

this->открытьToolStripMenuItem->Text = L"Открыть";

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

//

// сохранитьToolStripMenuItem

//

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

this->сохранитьToolStripMenuItem->Size = System::Drawing::Size(224, 26);

this->сохранитьToolStripMenuItem->Text = L"Сохранить";

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

//

// TextDisplay

//

this->TextDisplay->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->TextDisplay->Location = System::Drawing::Point(383, 44);

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

this->TextDisplay->Size = System::Drawing::Size(856, 500);

this->TextDisplay->TabIndex = 1;

this->TextDisplay->Text = L"";

this->TextDisplay->TextChanged += gcnew System::EventHandler(this, &MyForm::TextDisplay\_TextChanged);

//

// SubmitButton

//

this->SubmitButton->Enabled = false;

this->SubmitButton->Location = System::Drawing::Point(1044, 550);

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

this->SubmitButton->Size = System::Drawing::Size(195, 38);

this->SubmitButton->TabIndex = 2;

this->SubmitButton->Text = L"Провести анализ";

this->SubmitButton->UseVisualStyleBackColor = true;

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

//

// SelectSentenceMembers

//

this->SelectSentenceMembers->CheckOnClick = true;

this->SelectSentenceMembers->Enabled = false;

this->SelectSentenceMembers->FormattingEnabled = true;

this->SelectSentenceMembers->Items->AddRange(gcnew cli::array< System::Object^ >(6) {

L"Подлежащее", L"Сказуемое", L"Определение",

L"Обстоятельство", L"Дополнение", L"Слова, не являющиеся членами предложения"

});

this->SelectSentenceMembers->Location = System::Drawing::Point(39, 370);

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

this->SelectSentenceMembers->Size = System::Drawing::Size(338, 174);

this->SelectSentenceMembers->TabIndex = 4;

this->SelectSentenceMembers->SelectedIndexChanged += gcnew System::EventHandler(this, &MyForm::SelectSentenceMembers\_SelectedIndexChanged);

//

// ShowInTextByColour

//

this->ShowInTextByColour->Enabled = false;

this->ShowInTextByColour->Location = System::Drawing::Point(39, 550);

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

this->ShowInTextByColour->Size = System::Drawing::Size(195, 38);

this->ShowInTextByColour->TabIndex = 5;

this->ShowInTextByColour->Text = L"Показать";

this->ShowInTextByColour->UseVisualStyleBackColor = true;

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

//

// TableButton

//

this->TableButton->Enabled = false;

this->TableButton->Location = System::Drawing::Point(39, 606);

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

this->TableButton->Size = System::Drawing::Size(195, 38);

this->TableButton->TabIndex = 6;

this->TableButton->Text = L"Посмотреть статистику";

this->TableButton->UseVisualStyleBackColor = true;

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

//

// MyForm

//

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::None;

this->ClientSize = System::Drawing::Size(1262, 673);

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

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

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

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

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

this->Controls->Add(this->menuStrip1);

this->MainMenuStrip = this->menuStrip1;

this->Name = L"MyForm";

this->ShowIcon = false;

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"MyForm";

this->menuStrip1->ResumeLayout(false);

this->menuStrip1->PerformLayout();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private:

String^ get\_current\_directory(); // получение текущей папки с программой

System::Void SubmitButton\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void открытьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void ShowInTextByColour\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void создатьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void сохранитьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void TableButton\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void SelectSentenceMembers\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e);

System::Void TextDisplay\_TextChanged(System::Object^ sender, System::EventArgs^ e);

};

}

## Файл «MyForm.cpp»

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "MyForm.h"

#include"divide.hpp"

using namespace System;

using namespace System::Windows::Forms; // пространство имён, предоставляющее доступ к функциям работы с окнами

using namespace std;

using namespace msclr::interop; // пространство имён для работы преобразователя строк marshal

[STAThreadAttribute] // обязательное требование для обмена сообщениями с сервером сообщений Windows с

int main(cli::array<String^> ^ arguments)

{

Application::EnableVisualStyles(); // включает визуальные стили для приложения

Application::SetCompatibleTextRenderingDefault(false); // задаёт по умолчанию значения для свойств

Project4::MyForm Page;

Application::Run(% Page); // запуск приложения

Application::Exit(); // завершение работы приложения

return 0;

}

System::Void Project4::MyForm::SubmitButton\_Click(System::Object^ sender, System::EventArgs^ e) // обработка нажатия кнопки для начала анализа

{

string whole\_text;

if (this->TextDisplay->Text != "") // после ввода текста вручную в окне ввода-показа

{

String^ MegaText = "";

MegaText = this->TextDisplay->Text; // получение всего текста из окна ввода-показа

whole\_text = marshal\_as<string>(MegaText); // преобразования текста-строки из System^ в std::string

// удаление символов переноса строки

while (whole\_text.find('\n') != string::npos)

{

whole\_text.replace(whole\_text.find('\n'), 1, 1, ' ');

}

}

main\_function(whole\_text); // вызов основной функции

// разрешение выбора кнопок

this->SelectSentenceMembers->Enabled = true;

this->TableButton->Enabled = true;

}

String^ Project4::MyForm::get\_current\_directory() // получение в качестве строки String^ местоположения выбранного пользователем файла

{

char buffer[MAX\_PATH];

GetCurrentDirectoryA(256, buffer);

string x = string(buffer) + "\\";

return marshal\_as<String^>(x);// преобразование типов строк

}

System::Void Project4::MyForm::открытьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (sentences.size() != false) // если до этого анализировался текст

{

sentences.clear(); // очистка вектора структур слов

}

if (SelectSentenceMembers->CheckedItems->Count) // если до этого был выбор вывода членов предложения

{

SelectSentenceMembers->ClearSelected(); // очистка выбора

}

if (TextDisplay->Text != "") // если окно ввода-вывода было использовано

{

TextDisplay->Clear(); // очистка окна

}

if (unique\_words.size() != false) // если вектор уникальных слов непустой

{

unique\_words.clear(); // очистить его

}

// запрещение использования кнопок

this->SelectSentenceMembers->Enabled = false;

this->TableButton->Enabled = false;

this->SubmitButton->Enabled = false;

// очистка выбора отображения

for (int i = 0; i < this->SelectSentenceMembers->Items->Count; i++)

{

this->SelectSentenceMembers->SetSelected(i, false);

this->SelectSentenceMembers->SetItemChecked(i, false);

}

this->ShowInTextByColour->Enabled = false;

// выбрать файл и загрузить структуру

Stream^ stream;

OpenFileDialog^ Select\_File = gcnew OpenFileDialog(); // открыть диалоговое окно для выбора исходного файла

Select\_File->InitialDirectory = get\_current\_directory(); // установить первоначальное место открытия окна для

Select\_File->Filter = "txt files (\*.txt)|\*.txt"; // выбор только текстовых файлов

this->DialogResult = Select\_File->ShowDialog();

if (this->DialogResult == System::Windows::Forms::DialogResult::OK)

{

// обработки кнопки ок при выборе файла в проводнике

if ((stream = Select\_File->OpenFile())) // если файл открывается

{

// добавить ветвление по типу ричтекстбокстстримтайп в зависимости от rtf и txt

this->TextDisplay->LoadFile(Select\_File->FileName, System::Windows::Forms::RichTextBoxStreamType::PlainText);

stream->Close();

}

} else

if (this->DialogResult == System::Windows::Forms::DialogResult::Cancel)

{

// обработка кнопки выход при выборе файла в проводнике

}

delete Select\_File;

this->SubmitButton->Enabled = true;

}

System::Void Project4::MyForm::ShowInTextByColour\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (SelectSentenceMembers->CheckedItems->Count == false) // вывод диагностического сообщения при попытке отображения цветом членов предложения без конкретного выбора

{

MessageBox::Show("Выберите элемент из списка!", "", MessageBoxButtons::OK, MessageBoxIcon::Warning);

}

else

{

TextDisplay->Clear();

int lines\_counter = 0; // счётчик строк окна

this->TextDisplay->Clear();

int TextDisplayPosition = 0; // счётчик символов в окне ввода-вывода

for (auto& sentence : sentences) // итерация по предложениям

{

int count = 0;

bool flag = false; // показатель установки пробела перед словом

for (auto& word : sentence) // итерация по предложению

{

String^ line = marshal\_as<String^>(word.data); // преобразование в String^ строки std::string структуры

switch (word.type)

{

case SUBJECT: // подлежащее

{

if (this->SelectSentenceMembers->GetItemChecked(0)) // если выбрана опция показать подлежащие

{

TextDisplay->SelectionColor = Color::Red;

}

else

{

TextDisplay->SelectionColor = Color::Black;

}

break;

}

case PREDICATE:

{

if (this->SelectSentenceMembers->GetItemChecked(1)) // если выбрана опция показать сказуемое

{

TextDisplay->SelectionColor = Color::Blue;

}

else

{

TextDisplay->SelectionColor = Color::Black;

}

break;

}

case DEFINITION:

{

if (this->SelectSentenceMembers->GetItemChecked(2))

{

TextDisplay->SelectionColor = Color::Green;

}

else

{

TextDisplay->SelectionColor = Color::Black;

}

break;

}

case CIRCUMSTANCE:

{

if (this->SelectSentenceMembers->GetItemChecked(3))

{

TextDisplay->SelectionColor = Color::DarkOrange;

}

else

{

TextDisplay->SelectionColor = Color::Black;

}

}

case ADDITION:

{

if (this->SelectSentenceMembers->GetItemChecked(4)) // если выбрана опция показать дополнение

{

TextDisplay->SelectionColor = Color::Violet;

}

else

{

TextDisplay->SelectionColor = Color::Black;

}

break;

}

default:

TextDisplay->SelectionColor = Color::Black;

break;

}

if ((word.type != PUNCTUATION) && (flag != false)) // если слово не является первым в предложении и не является знаком пунктуации

{

TextDisplay->AppendText(" ");

TextDisplayPosition++;

}

else

{

flag = true; // разрешить постановку пробела

}

TextDisplay->Select(TextDisplayPosition, 0); // выбор одного символа для формата цвета по образцу

TextDisplay->AppendText(line); // запись слова в строку окна ввода-вывода

TextDisplayPosition += line->Length; // увеличение счётчика символов в окне ввода-вывода

}

TextDisplay->AppendText("\n"); // переход на следующую строку окна

TextDisplayPosition++;

lines\_counter++; // увеличение счётчика строк окна

}

}

}

System::Void Project4::MyForm::создатьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) // обработка нажатия кнопки работы с новым текстом

{

if (sentences.size() != false) // если до этого анализировался текст

{

sentences.clear(); // очистка вектора структур слов

}

if (SelectSentenceMembers->CheckedItems->Count) // если до этого был выбор вывода членов предложения

{

SelectSentenceMembers->ClearSelected(); // очистка выбора

}

if (TextDisplay->Text != "") // если окно ввода-вывода было использовано

{

TextDisplay->Clear(); // очистка окна

}

if (unique\_words.size() != false) // если вектор уникальных слов непустой

{

unique\_words.clear(); // очистить его

}

// запрещение использования кнопок

this->SelectSentenceMembers->Enabled = false;

this->TableButton->Enabled = false;

this->SubmitButton->Enabled = false;

// очистка выбора отображения

for (int i = 0; i < this->SelectSentenceMembers->Items->Count; i++)

{

this->SelectSentenceMembers->SetSelected(i, false);

this->SelectSentenceMembers->SetItemChecked(i, false);

}

this->ShowInTextByColour->Enabled = false;

}

string wordtype\_int\_to\_string(const int& need\_i)//переработка выдачи типа словами под вектор уникальных слов

{

switch (need\_i)

{

case 0:return "Подлежащее";

case 1:return "Сказуемое";

case 2:return "Определение";

case 3:return "Дополнение";

case 4:return "Обстоятельство";

default:return "Не определено";

}

}

void write\_to\_file(ofstream& writing\_filestream) // функция записи в выходной файл результатов анализа текста

{

for (const auto& word : unique\_words) // итерация по массиву уникальных слов

{

if (word.type != PUNCTUATION)

{

writing\_filestream << "data: " << word.data << setw(30 - word.data.length()) << " type: " << wordtype\_int\_to\_string(word.type) << setw(32 - wordtype\_int\_to\_string(word.type).length()) << " frequency: " << word.frequency << setw(30) << " sentences\_included: " << word.sentences\_included << endl;

}

}

}

System::Void Project4::MyForm::сохранитьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (unique\_words.size()) // если массив уникальных слов сформирован

{

string filename = "Results\_";

filename = form\_filename(filename);

ofstream writing\_filestream(filename);

write\_to\_file(writing\_filestream); // записать в файл поля элементов массива уникальных элементов

writing\_filestream.close();

}

else

{

MessageBox::Show("Сохранять нечего!", "", MessageBoxButtons::OK, MessageBoxIcon::Warning);

}

}

System::Void Project4::MyForm::TableButton\_Click(System::Object^ sender, System::EventArgs^ e) // отображение в табличном виде матрицы уникальных слов

{

MyForm1^ next\_page = gcnew MyForm1(this);

this->Hide();

next\_page->Show();

}

System::Void Project4::MyForm::SelectSentenceMembers\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

this->ShowInTextByColour->Enabled = true;

}

System::Void Project4::MyForm::TextDisplay\_TextChanged(System::Object^ sender, System::EventArgs^ e)

{

this->SubmitButton->Enabled = true;

}

## Файл «MyForm1.h»

#pragma once

#include"TextReader.hpp"

#include<msclr\marshal\_cppstd.h> // библиотека для преобразования строк типа String^ в std::string и обратно

namespace Project4

{

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 msclr::interop;

/// <summary>

/// Сводка для MyForm1

/// </summary>

public ref class MyForm1 : public System::Windows::Forms::Form

{

public:

MyForm1(void)

{

InitializeComponent();

}

MyForm1(System::Windows::Forms::Form^ prev)

{

prev\_page = prev;

InitializeComponent();

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm1()

{

if (components)

{

delete components;

}

}

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

protected:

protected:

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

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column3;

System::Windows::Forms::Form^ prev\_page;

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

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

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

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

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

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

private: System::Windows::Forms::VScrollBar^ vScrollBar1;

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

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

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

this->Column3 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

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

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

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

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

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

this->vScrollBar1 = (gcnew System::Windows::Forms::VScrollBar());

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

this->SuspendLayout();

//

// Screen

//

this->Screen->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;

this->Screen->AutoSizeRowsMode = System::Windows::Forms::DataGridViewAutoSizeRowsMode::AllCells;

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

this->Screen->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->Column1, this->Column2,

this->Column3

});

this->Screen->Dock = System::Windows::Forms::DockStyle::Bottom;

this->Screen->Location = System::Drawing::Point(0, 92);

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

this->Screen->ReadOnly = true;

this->Screen->RowHeadersWidth = 51;

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

this->Screen->Size = System::Drawing::Size(1262, 581);

this->Screen->TabIndex = 0;

//

// Column1

//

this->Column1->HeaderText = L"Название";

this->Column1->MinimumWidth = 6;

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

this->Column1->ReadOnly = true;

//

// Column2

//

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

this->Column2->MinimumWidth = 6;

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

this->Column2->ReadOnly = true;

//

// Column3

//

this->Column3->HeaderText = L"Номера предложений";

this->Column3->MinimumWidth = 6;

this->Column3->Name = L"Column3";

this->Column3->ReadOnly = true;

//

// CloseButton

//

this->CloseButton->Location = System::Drawing::Point(1136, 5);

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

this->CloseButton->Size = System::Drawing::Size(114, 50);

this->CloseButton->TabIndex = 1;

this->CloseButton->Text = L"закрыть";

this->CloseButton->UseVisualStyleBackColor = true;

this->CloseButton->Click += gcnew System::EventHandler(this, &MyForm1::CloseButton\_Click);

//

// ShowSubject

//

this->ShowSubject->Location = System::Drawing::Point(12, 12);

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

this->ShowSubject->Size = System::Drawing::Size(140, 36);

this->ShowSubject->TabIndex = 2;

this->ShowSubject->Text = L"Подлежащее";

this->ShowSubject->UseVisualStyleBackColor = true;

this->ShowSubject->Click += gcnew System::EventHandler(this, &MyForm1::ShowSubject\_Click);

//

// ShowPredicate

//

this->ShowPredicate->Location = System::Drawing::Point(171, 12);

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

this->ShowPredicate->Size = System::Drawing::Size(140, 36);

this->ShowPredicate->TabIndex = 3;

this->ShowPredicate->Text = L"Сказуемое";

this->ShowPredicate->UseVisualStyleBackColor = true;

this->ShowPredicate->Click += gcnew System::EventHandler(this, &MyForm1::ShowPredicate\_Click);

//

// ShowAttribute

//

this->ShowAttribute->Location = System::Drawing::Point(335, 12);

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

this->ShowAttribute->Size = System::Drawing::Size(140, 36);

this->ShowAttribute->TabIndex = 4;

this->ShowAttribute->Text = L"Определение";

this->ShowAttribute->UseVisualStyleBackColor = true;

this->ShowAttribute->Click += gcnew System::EventHandler(this, &MyForm1::ShowAttribute\_Click);

//

// ShowObject

//

this->ShowObject->Location = System::Drawing::Point(493, 12);

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

this->ShowObject->Size = System::Drawing::Size(140, 36);

this->ShowObject->TabIndex = 5;

this->ShowObject->Text = L"Дополнение";

this->ShowObject->UseVisualStyleBackColor = true;

this->ShowObject->Click += gcnew System::EventHandler(this, &MyForm1::ShowObject\_Click);

//

// ShowAdverbialModifier

//

this->ShowAdverbialModifier->Location = System::Drawing::Point(650, 12);

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

this->ShowAdverbialModifier->Size = System::Drawing::Size(140, 36);

this->ShowAdverbialModifier->TabIndex = 6;

this->ShowAdverbialModifier->Text = L"Обстоятельство";

this->ShowAdverbialModifier->UseVisualStyleBackColor = true;

this->ShowAdverbialModifier->Click += gcnew System::EventHandler(this, &MyForm1::ShowAdverbialModifier\_Click);

//

// vScrollBar1

//

this->vScrollBar1->Location = System::Drawing::Point(1241, 92);

this->vScrollBar1->Name = L"vScrollBar1";

this->vScrollBar1->Size = System::Drawing::Size(21, 581);

this->vScrollBar1->TabIndex = 7;

//

// MyForm1

//

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

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

this->ClientSize = System::Drawing::Size(1262, 673);

this->ControlBox = false;

this->Controls->Add(this->vScrollBar1);

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

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

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

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

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

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

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

this->Name = L"MyForm1";

this->ShowIcon = false;

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"Статистика";

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

this->ResumeLayout(false);

}

#pragma endregion

private:

System::Void CloseButton\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void ShowSubject\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void ShowPredicate\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void ShowAttribute\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void ShowObject\_Click(System::Object^ sender, System::EventArgs^ e);

System::Void ShowAdverbialModifier\_Click(System::Object^ sender, System::EventArgs^ e);

};

}

## Файл «MyForm1.cpp»

#include "MyForm1.h"

System::Void Project4::MyForm1::CloseButton\_Click(System::Object^ sender, System::EventArgs^ e) // обработка кнопки закрытия страницы

{

prev\_page->Show();

this->Close();

}

System::Void Project4::MyForm1::ShowSubject\_Click(System::Object^ sender, System::EventArgs^ e) // отображение всех подлежащих

{

// очищение таблицы показа

while (this->Screen->Rows->Count > 1)

{

for (int i = 0; i < this->Screen->Rows->Count - 1; i++)

{

this->Screen->Rows->Remove(this->Screen->Rows[i]); // очистка строки таблицы

}

}

int counter = 0; // переменная-счетчик

for (const auto& word : unique\_words) // итерация по массиву уникальных слов

{

if (word.type == SUBJECT) // если у слова тип - подлежащее

{

this->Screen->Rows->Add(); // добавить строку в таблицу

this->Screen->Rows[counter]->Cells[0]->Value = marshal\_as<String^>(word.data); // преобразовать в формат String^ поля объекта std::string

this->Screen->Rows[counter]->Cells[1]->Value = word.frequency; // вывести частоту встречаемости в тексте этого слова

this->Screen->Rows[counter++]->Cells[2]->Value = marshal\_as<String^>(word.sentences\_included); // вывести номера предложений, где встретилось это слово

}

}

}

System::Void Project4::MyForm1::ShowPredicate\_Click(System::Object^ sender, System::EventArgs^ e) // отображение всех сказуемых

{

// очищение таблицы показа

while (this->Screen->Rows->Count > 1)

{

for (int i = 0; i < this->Screen->Rows->Count - 1; i++)

{

this->Screen->Rows->Remove(this->Screen->Rows[i]);// очистка строки таблицы

}

}

int counter = 0; // переменная-счетчик

for (const auto& word : unique\_words) // итерация по массиву уникальных слов

{

if (word.type == PREDICATE) // если у слова тип - сказуемое

{

this->Screen->Rows->Add(); // добавить строку в таблицу

this->Screen->Rows[counter]->Cells[0]->Value = marshal\_as<String^>(word.data); // преобразовать в формат String^ поля объекта std::string

this->Screen->Rows[counter]->Cells[1]->Value = word.frequency; // вывести частоту встречаемости в тексте этого слова

this->Screen->Rows[counter++]->Cells[2]->Value = marshal\_as<String^>(word.sentences\_included); // вывести номера предложений, где встретилось это слово

}

}

}

System::Void Project4::MyForm1::ShowAttribute\_Click(System::Object^ sender, System::EventArgs^ e) // отображение всех определений

{

// очищение таблицы показа

while (this->Screen->Rows->Count > 1)

{

for (int i = 0; i < this->Screen->Rows->Count - 1; i++)

{

this->Screen->Rows->Remove(this->Screen->Rows[i]);

}

}

int counter = 0;

for (const auto& word : unique\_words) // итерация по массиву уникальных слов

{

if (word.type == DEFINITION) // если у слова тип - определение

{

this->Screen->Rows->Add(); // добавить строку в таблицу

this->Screen->Rows[counter]->Cells[0]->Value = marshal\_as<String^>(word.data); // преобразовать в формат String^ поля объекта std::string

this->Screen->Rows[counter]->Cells[1]->Value = word.frequency; // вывести частоту встречаемости в тексте этого слова

this->Screen->Rows[counter++]->Cells[2]->Value = marshal\_as<String^>(word.sentences\_included); // вывести номера предложений, где встретилось это слово

}

}

}

System::Void Project4::MyForm1::ShowObject\_Click(System::Object^ sender, System::EventArgs^ e) // отображение всех обстоятельств

{

// очищение таблицы показа

while (this->Screen->Rows->Count > 1)

{

for (int i = 0; i < (this->Screen->Rows->Count - 1); i++)

{

this->Screen->Rows->Remove(this->Screen->Rows[i]);

}

}

int counter = 0;

for (const auto& word : unique\_words) // итерация по массиву уникальных слов

{

if (word.type == ADDITION) // если у слова тип - дополнение

{

this->Screen->Rows->Add(); // добавить строку в таблицу

this->Screen->Rows[counter]->Cells[0]->Value = marshal\_as<String^>(word.data); // преобразовать в формат String^ поля объекта std::string

this->Screen->Rows[counter]->Cells[1]->Value = word.frequency; // вывести частоту встречаемости в тексте этого слова

this->Screen->Rows[counter++]->Cells[2]->Value = marshal\_as<String^>(word.sentences\_included); // вывести номера предложений, где встретилось это слово

}

}

}

System::Void Project4::MyForm1::ShowAdverbialModifier\_Click(System::Object^ sender, System::EventArgs^ e) // отображение всех дополнений

{

// очищение таблицы показа

while (this->Screen->Rows->Count > 1)

{

for (int i = 0; i < this->Screen->Rows->Count - 1; i++)

{

this->Screen->Rows->Remove(this->Screen->Rows[i]);

}

}

int counter = 0;

for (const auto& word : unique\_words) // итерация по массиву уникальных слов

{

if (word.type == CIRCUMSTANCE) // если у слова тип - обстоятельство

{

this->Screen->Rows->Add(); // добавить строку в таблицу

this->Screen->Rows[counter]->Cells[0]->Value = marshal\_as<String^>(word.data); // преобразовать в формат String^ поля объекта std::string

this->Screen->Rows[counter]->Cells[1]->Value = word.frequency; // вывести частоту встречаемости в тексте этого слова

this->Screen->Rows[counter++]->Cells[2]->Value = marshal\_as<String^>(word.sentences\_included); // вывести номера предложений, где встретилось это слово

}

}

}

## Файл «Frame.hpp»

#pragma once

#include"TextReader.hpp"

#include"Predicate.hpp"

#include"addition.hpp"

#include"circumstance.hpp"

using namespace std;

void split\_text(string&, const string&); // функция преобразования исходного текста в пронумерованный файл

void main\_function(string&); // основная функция

## Файл «Frame.cpp»

#include"Frame.hpp"

#include"Predicate.hpp"

#include"addition.hpp"

#include"trpo\_podl.hpp"

#include"definition.hpp"

#include"divide.hpp"

void split\_text(string& line, const string& file\_text) // функция преобразования исходного текста в пронумерованный файл

{

ofstream splitted(file\_text);

size\_t sentence\_counter = 1; // счётчик числа предложений

string part\_of\_sentence; // строка, отвечающая за хранение первой части предложения при переносе

string sentence; // строка, отвечающая за хранение предложения из строки копии входного файла

for (int i = 0; i < line.size(); i++) // итерация по строке из файла

{

sentence += line[i]; // посимвольное копирование предложения из строки

if (line[i] == '.' || line[i] == '?' || line[i] == '!') // если встретился символ окончания предложения

{

if ((i != line.size() - 1) && line[i + 1] != ' ') // если нет пробела после этого знака и все предложение не одно во всей строке

{

continue;

}

while (sentence[0] == ' ') // если в самом начале предложения есть лишние пробелы

{

sentence.erase(0, 1); // удаление пробелов в самом начале предложения

}

splitted << sentence\_counter++ << ") " << sentence << endl; // записать в файл предложение

sentence.clear();

}

}

splitted.close(); // закрыть файловые потоки

}

void find\_union(vector<vector<Word>>& whole\_text) // функция поиска союзов в тексте

{

const vector<string> unions = { "и", "да", "тоже", "также", "а", "но", "однако", "зато", "же", "или", "либо", "то", "не", "только", "если", "столько", "сколько", "как", "так", "хотя", "что", "чтобы", "будто", "когда", "пока", "едва", "если", "раз", "ибо", "дабы", "для", "хотя", "хоть", "пускай", "как", "словно","кто", "каков", "который", "куда", "откуда", "где", "почему", "зачем" };

for (auto& sentence : whole\_text) // итерация по предложениям

{

for (auto& word : sentence) // итерация по предложению

{

string temp = word.data;

for (auto& it : unions)

{

if (temp == it)

{

word.type = UNION;

}

}

}

}

}

void find\_preposition(vector<vector<Word>>& whole\_text) // функция поиска предлогов в тексте

{

const vector<string> prepositions = { "в","это", "с","к", "до", "по", "через", "после", "из-за", "за", "над", "под", "перед", "у", "через", "возле", "мимо", "около", "от", "ради", "благодаря", "ввиду", "вследствие", "для", "на", "вопреки", "несмотря", "о", "об", "обо", "про", "насчёт","вроде", "наподобие", "без" };

for (auto& sentence : whole\_text) // итерация по предложениям

{

for (auto& word : sentence) // итерация по предложению

{

string temp = word.data;

if (temp[0] >= -33 && temp[0] <= -64)

{

temp[0] += 32;

}

for (auto& it : prepositions)

{

if (temp == it)

{

word.type = PREPOSITION;

}

}

}

}

}

void find\_particle(vector<vector<Word>>& whole\_text) // функция поиска частиц в тексте

{

const vector<string> particles = { "бы", "пусть", "пускай", "да", "давай", "давайте", "не", "ни", "вовсе", "далеко", "отнюдь", "уже"};

for (auto& sentence : whole\_text) // итерация по предложениям

{

for (auto& word : sentence) // итерация по предложению

{

string temp = word.data;

if (temp[0] >= -33 && temp[0] <= -64)

{

temp[0] += 32;

}

for (auto& it : particles)

{

if (temp == it)

{

word.type = PARTICLE;

}

}

}

}

}

void filter(vector<vector<Word>>& whole\_text) // функция фильтрации слов на члены предложения и части речи

{

find\_union(sentences); // поиск союзов

find\_preposition(sentences); // поиск предлогов

find\_particle(sentences); // поиск частиц

find\_definitions(); // поиск определений

find\_predicate(); // поиск сказуемых

find\_subjects(); // поиск подлежащих

find\_circumstances(); // поиск обстоятельств

find\_addition(); // поиск дополнений

}

void main\_function(string& whole\_text)

{

string file\_text = "SplittedText\_";

file\_text = form\_filename(file\_text);

split\_text(whole\_text, file\_text); // разделение изначального текста на формат: [номер предложения; правая круглая скобка; пробел; само предложение]

text\_handler(file\_text); // формирование вектора векторов структур слов и выставка знаков препинания

filter(sentences); // фильтрация слов на члены предложения

uniquify\_words(); // функция, осуществляющая подсчет уникальных слов и выставку номеров предложений, где они есть

}

## Файл «TextReader.hpp»

#pragma once

#include<iostream>

#include<fstream>

#include<sstream>

#include<vector>

#include<string>

#include<cctype>

#include<windows.h>

#include<algorithm>

#include<iomanip>

using namespace std;

//Структура слова: само слово + тип

enum Word\_type

{

// основные части речи

SUBJECT, // подлежащее

PREDICATE, // сказуемое

ADDITION, // дополнение

DEFINITION, // определение

PUNCTUATION, // знаки пунктуации

CIRCUMSTANCE, // обстоятельство

UNKNOWN,

// служебные части речи

UNION, // союз

PREPOSITION, // предлог

PARTICLE // частица

};

struct Word // структура слово

{

string data; // название

Word\_type type; // тип слова

int frequency; // частота встречаемости слова в тексте

string sentences\_included; // номера предложений, в которых встречается слово

vector<Word\*> derived; // указатель на дочерний элемент

};

//Предложение - вектор слов

inline vector<vector<Word>> sentences;

inline vector<Word> unique\_words;

void text\_handler(const string&);

string form\_filename(string&);

## Файл «TextReader.cpp»

#include"TextReader.hpp"

//Расшифровка перечисляемого типа для удобства

string type\_decoding(Word\_type type)

{

switch (type)

{

case Word\_type::SUBJECT:

return "Подлежащее";

case Word\_type::PREDICATE:

return "Сказуемое";

case Word\_type::ADDITION:

return "Дополнение";

case Word\_type::DEFINITION:

return "Определение";

case Word\_type::PUNCTUATION:

return "Знак препинания";

case Word\_type::CIRCUMSTANCE:

return "Обстоятельство";

case Word\_type::UNION:

return "Союз";

case Word\_type::PREPOSITION:

return "Предлог";

case Word\_type::PARTICLE:

return "Частица";

}

return "Unknown";

}

string form\_filename(string& filename) // формирование названия файла по системному времени

{

string h;

stringstream temp;

time\_t t = std::time(nullptr);

tm\* now = std::localtime(&t);

temp << now->tm\_mday << '.' << (now->tm\_mon + 1) << '.'

<< (now->tm\_year + 1900) << "\_" << (now->tm\_hour) << "." << (now->tm\_min);

temp >> h;

filename += h;

filename += ".txt";

return filename;

}

void text\_handler(const string& file\_text)

{

ifstream file(file\_text);

string line;

while (getline(file, line))

{

// Поиск позиции escape последовательности

size\_t pos = line.find("\n");

// Если escape последовательность найдена, обрезаем строку до этой позиции

if (pos != string::npos) {

line = line.substr(0, pos);

}

// Если строка начинается с числа + ")" (закрывающая скобка),

// то пропускаем эту нумерацию.

if (!line.empty()) {

size\_t index\_pos = line.find\_first\_of(") ");

if (index\_pos != string::npos && index\_pos != 0 && isdigit(line[index\_pos - 1])) {

line = line.substr(index\_pos + 1);

}

}

// Разделение строки на слова

string current\_word;

vector<Word> words;

for (size\_t i = 0; i < line.size(); ++i)

{

unsigned char check = line[i];

if (isspace(check) || ispunct(check))

{

// Если текущий символ является пробелом или знаком пунктуации,

// сохраняем собранное слово в вектор слов

if (!current\_word.empty())

{

words.push\_back({ current\_word, UNKNOWN });

current\_word.clear();

}

// Если текущий символ является знаком пунктуации, добавляем его как отдельное слово

if (ispunct(check)) {

string punctuations(1, check);

words.push\_back({ punctuations, PUNCTUATION });

}

}

else if (check == '-') {

// Если текущий символ - дефис, добавляем его к текущему слову

current\_word += check;

}

else {

// Добавляем текущий символ к текущему слову

current\_word += check;

// Проверяем, если следующий символ - дефис и не является последним в строке

if (i + 1 < line.size() && line[i + 1] == '-') {

// Добавляем следующий символ (дефис) к текущему слову

current\_word += line[++i];

}

}

}

sentences.push\_back(words);

}

file.close();

}

## Файл «addition.hpp»

#pragma once

#include <sstream>

#include <algorithm>

#include"TextReader.hpp"

using namespace std;

void find\_addition();

## Файл «addition.сpp»

#include "addition.hpp"

//Массивы и счетчики

string\* exceptions; // Массив исключений

string\* mas\_of\_endings\_verb; //массив окончаний глаголов

string\* mas\_of\_prepositions; //масив предлогов

string\* mas\_of\_obst\_preposition; //Массив предлогов обстоятельства

string\* mas\_of\_suffiks\_of\_oborot;//Массив суффиксов

string\* mas\_of\_unions; //Массив союзов

string\* mas\_of\_podl\_endings; //Массив окончаний подлежащего

string\* mas\_of\_adjectives; //Массив определений

string\* obst\_word; //Массив обстоятельства

string\* mest\_word; //Массив местоимений

string\* particles; //Массив частиц

string\* interjections; //Массив междометий

int endings\_count = 0; //Счетчик массива окончаний глаголов

int exception\_count = 0; //Счетчик массива исключений

int prepositions\_count = 0; //Счетчик массива предлогов

int obst\_preposition = 0; //Счетчик массива предлогов обстоятельства

int obst\_count = 0; //Счетчик массива обстоятельств

int mest\_count = 0; //счетчик массива местоимений

int count\_of\_suffiks\_of\_oborot = 0; //Счетчик массива суффиксов

int count\_of\_unions\_soch = 0; //Счетчик массива союзов

int count\_of\_podl\_endings = 0; // Счетчик массива подлежащих

int endings\_adjectives = 0; //Счетчик массива определений

int count\_particles = 0; //счетчик массива частиц

int count\_interjections = 0; //Счетчик массива междометий

//Прототипы функций проверок(анализа)

bool is\_verb(const string& word);

bool is\_preposition(const string& word);

bool is\_unions(const string& word);

bool is\_obst(const string& word);

bool is\_mest(const string& word);

bool is\_exception(const string& word);

bool is\_adjective(const string& word);

bool is\_interjection(const string& word);

bool is\_particles(const string& word);

//Прототипы функций анализа

void find\_oborots(string\* words\_in\_sentence, int word\_count);

//Прототипы функций заполнения

int countLines(const string& general\_file, const string& startMarker, const string& endMarker, string\*& temp\_array);

string\* create\_array(int count);

void destroy\_arr();

//Рабочие файлы

const string general\_file = "all\_information.txt";

const string file\_text = "splitted\_text.txt";

int countLines(const string& general\_file, const string& startMarker, const string& endMarker, string\*& temp\_array) {

ifstream file(general\_file);

string line;

bool shouldRead = false;

int wordCount = 0;

while (getline(file, line)) {

if (line == startMarker) {

shouldRead = true;

continue;

} else if (line == endMarker) {

shouldRead = false;

break;

}

if (shouldRead) {

istringstream iss(line);

string word;

while (iss >> word) {

wordCount++;

}

}

}

//Сброс указателя в файле в начало для повторного чтения

file.clear();

file.seekg(0, ios::beg);

int index = 0;

temp\_array = new string[wordCount];

while (getline(file, line)) {

if (line == startMarker) {

shouldRead = true;

continue;

} else if (line == endMarker) {

shouldRead = false;

break;

}

if (shouldRead) {

istringstream iss(line);

string word;

while (iss >> word) {

temp\_array[index] = word; // Запись слова в массив

index++;

}

}

}

file.close();

return wordCount;

}

string\* create\_array(int count){

string \*a;

a = new string[count];

return a;

}

//Заполнений массивов данными из файлов

//Если создавать массив, то здесь

//Объявлять сверху указатель и счетчик и дальше по аналогии, как в функции

void filling\_array()

{

//Следует избавиться от дублированного кода\*

//\*Требуется рефакторинг

string \*temp\_array = nullptr;

//1

endings\_count = countLines(general\_file, "глаголы:", "конец1", temp\_array);

mas\_of\_endings\_verb = create\_array(endings\_count);

for( int i = 0; i<endings\_count;i++)

{

mas\_of\_endings\_verb[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

//2

prepositions\_count = countLines(general\_file, "предлоги:", "конец2", temp\_array);

mas\_of\_prepositions = create\_array(prepositions\_count);

for( int i = 0; i<prepositions\_count; i++)

{

mas\_of\_prepositions[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

//3 и так далее

count\_of\_suffiks\_of\_oborot = countLines(general\_file, "суффиксы:", "конец3", temp\_array);

mas\_of\_suffiks\_of\_oborot = create\_array(count\_of\_suffiks\_of\_oborot);

for( int i = 0; i<count\_of\_suffiks\_of\_oborot; i++)

{

mas\_of\_suffiks\_of\_oborot[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

count\_of\_unions\_soch = countLines(general\_file, "союзы:", "конец4", temp\_array);

mas\_of\_unions = create\_array(count\_of\_unions\_soch);

for( int i = 0; i<count\_of\_unions\_soch; i++)

{

mas\_of\_unions[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

obst\_preposition = countLines(general\_file, "обстоятельственные-предлоги:", "конец5", temp\_array);

mas\_of\_obst\_preposition = create\_array(obst\_preposition);

for( int i = 0; i<obst\_preposition;i++)

{

mas\_of\_obst\_preposition[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

count\_of\_podl\_endings = countLines(general\_file, "подлежащие:", "конец6", temp\_array);

mas\_of\_podl\_endings = create\_array(count\_of\_podl\_endings);

for( int i = 0; i<count\_of\_podl\_endings; i++)

{

mas\_of\_podl\_endings[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

endings\_adjectives = countLines(general\_file, "определение:", "конец7", temp\_array);

mas\_of\_adjectives = create\_array(endings\_adjectives);

for( int i = 0; i<endings\_adjectives; i++)

{

mas\_of\_adjectives[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

mest\_count = countLines(general\_file, "местоимения:", "конец8", temp\_array);

mest\_word = create\_array(mest\_count);

for( int i = 0; i<mest\_count; i++)

{

mest\_word[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

exception\_count = countLines(general\_file, "исключения:", "конец9", temp\_array);

exceptions = create\_array(exception\_count);

for( int i = 0; i<exception\_count; i++)

{

exceptions[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

obst\_count = countLines(general\_file, "наречия:", "конец10", temp\_array);

obst\_word = create\_array(obst\_count);

for( int i = 0; i<obst\_count; i++)

{

obst\_word[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

count\_particles = countLines(general\_file, "частицы:", "конец11", temp\_array);

particles = create\_array(count\_particles);

for( int i = 0; i<count\_particles; i++)

{

particles[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

count\_interjections = countLines(general\_file, "междометия:", "конец12", temp\_array);

interjections = create\_array(count\_interjections);

for( int i = 0; i<count\_interjections; i++)

{

interjections[i] = temp\_array[i];

}

delete[]temp\_array;

temp\_array = nullptr;

}

void find\_oborots(string\* words\_in\_sentence,int word\_count){

//Анализ частых случаев причатных/деепричатных оборотов

bool oborot\_found=false;

bool oborot\_flag=false;

int oborot\_start=0;

int oborot\_finish=0;

for(int k=0;k<count\_of\_unions\_soch;k++){

for(int i=1;i<word\_count;i++){

if((words\_in\_sentence[i-1]=="," && oborot\_found==false)||( is\_particles(words\_in\_sentence[i-1]) && oborot\_found==false)){

for(int j=0;j<count\_of\_suffiks\_of\_oborot;j++){

if(words\_in\_sentence[i].rfind(mas\_of\_suffiks\_of\_oborot[j])!=-1){

oborot\_found = true;

oborot\_start = i-1;

}

}

}

}

for(int i=1; i < word\_count; i++){

if(words\_in\_sentence[i-1]=="," && oborot\_found==true && oborot\_start <= i-1 && oborot\_start != i-1 ){

for(int j=0;j<count\_of\_suffiks\_of\_oborot;j++){

if(words\_in\_sentence[i].rfind(mas\_of\_suffiks\_of\_oborot[j])==-1){

oborot\_found = false;

oborot\_finish = i;

}

if(words\_in\_sentence[i].rfind(mas\_of\_suffiks\_of\_oborot[j])!=-1){

oborot\_found = true;

}

}

}

else if(words\_in\_sentence[i]=="." && oborot\_found==true){

oborot\_found = false;

oborot\_finish = i;

}

}

for(int i = oborot\_start;i < oborot\_finish;i++){

words\_in\_sentence[i].erase();

}

}

}

bool is\_preposition(const string& word) {

for (int i = 0; i < prepositions\_count; i++) {

if (word == mas\_of\_prepositions[i]) {

return true;

}

}

return false;

}

bool is\_obst\_preposition(const string& word) {

for (int i = 0; i <obst\_preposition ; i++) {

if (word == mas\_of\_obst\_preposition[i]) {

return true;

}

}

return false;

}

bool is\_obst(const string& word) {

for (int i = 0; i < obst\_count; i++) {

if (word == obst\_word[i]) {

return true;

}

}

return false;

}

bool is\_mest(const string& word) {

for (int i = 0; i < mest\_count; i++) {

if (word == mest\_word[i]) {

return true;

}

}

return false;

}

bool is\_unions(const string& word) {

for (int i = 0; i < count\_of\_unions\_soch; i++) {

if (word == mas\_of\_unions[i]) {

return true;

}

}

return false;

}

bool is\_exception(const string& word)

{

for (int i = 0; i < exception\_count; i++)

{

if (word == exceptions[i]) {

return true;

}

}

return false;

}

bool is\_verb(const string& word )

{

for (int i = 0; i < endings\_count; i++) {

if (word.size() >= mas\_of\_endings\_verb[i].size() &&

word.compare(word.size() - mas\_of\_endings\_verb[i].size(), mas\_of\_endings\_verb[i].size(), mas\_of\_endings\_verb[i]) == 0)

{

if(!is\_exception(word))

{

return true;

}

}

}

return false;

}

bool is\_adjective(const string& word)

{

for (int i = 0; i < endings\_adjectives; i++) {

if (word.size() >= mas\_of\_adjectives[i].size() &&

word.compare(word.size() - mas\_of\_adjectives[i].size(), mas\_of\_adjectives[i].size(), mas\_of\_adjectives[i]) == 0)

{

if(!is\_exception(word))

{

return true;

}

}

}

return false;

}

bool is\_interjection(const string& word){

for (int i = 0; i < count\_interjections; i++)

{

if (word == interjections[i]) {

return true;

}

}

return false;

}

bool is\_particles(const string& word){

for (int i = 0; i < count\_particles; i++)

{

if (word == particles[i]) {

return true;

}

}

return false;

}

void find\_addition()

{

filling\_array();

int senteces\_count = sentences.size();

int word\_count = 0;

vector<string> remaining\_words;

for (int i = 0; i<senteces\_count;++i )

{

word\_count = sentences[i].size();

if(word\_count == 0)

{

break;

}

string\* words\_in\_sentence = new string[word\_count];

for(int k = 0; k <word\_count; k++)

{

words\_in\_sentence[k] = sentences[i][k].data;

}

find\_oborots( words\_in\_sentence, word\_count);

for (int i = 0; i < word\_count; i++)

{

if(i!=0 && is\_obst\_preposition(words\_in\_sentence[i-1]) == true){

for(int j=0;j<count\_of\_podl\_endings;j++){

if(words\_in\_sentence[i].rfind(mas\_of\_podl\_endings[j])!=-1 ||words\_in\_sentence[i]==mas\_of\_podl\_endings[j] ){

words\_in\_sentence[i].erase();

}

}

}

}

for (int i = 0; i < word\_count; i++)

{

string lower\_word = words\_in\_sentence[i]; //Перевод в нижний регистр для анализа

for (int k = 0; k < lower\_word.length(); k++)

{

lower\_word[0] = tolower(lower\_word[0]);

}

if (is\_verb(lower\_word))

{

words\_in\_sentence[i].erase();

}

else if(is\_unions(lower\_word)==true){

words\_in\_sentence[i].erase();

}

else if(is\_preposition(lower\_word))

{

words\_in\_sentence[i].erase();

}

else if(is\_adjective(lower\_word))

{

words\_in\_sentence[i].erase();

}

else if (is\_obst(lower\_word))

{

words\_in\_sentence[i].erase();

}

else if (is\_mest(lower\_word))

{

words\_in\_sentence[i].erase();

}

else if (is\_interjection(lower\_word))

{

words\_in\_sentence[i].erase();

}

else if (is\_particles(lower\_word))

{

words\_in\_sentence[i].erase();

}

else if(words\_in\_sentence[i]=="—"|| words\_in\_sentence[i]=="-" || words\_in\_sentence[i]=="–"|| words\_in\_sentence[i].find('«')!=string::npos || words\_in\_sentence[i].find('»')!=string::npos ||

words\_in\_sentence[i].find('"')!=string::npos )

{

words\_in\_sentence[i].erase();

}

}

//Заполнение вектора теми словами. которые остались после анализа текста(т.е предполагаемые дополнения)

bool detect\_vector\_str = false;

for(int i = 0;i< word\_count;i++)

{

detect\_vector\_str = false;

if(!words\_in\_sentence[i].empty()&& words\_in\_sentence[i]!="—" && words\_in\_sentence[i]!="." && words\_in\_sentence[i]!="," &&

words\_in\_sentence[i]!=";"&& words\_in\_sentence[i]!="!" && words\_in\_sentence[i]!="?" && words\_in\_sentence[i]!=":" )

{

for(int j = 0; j<remaining\_words.size();j++){

if(remaining\_words[j]!=words\_in\_sentence[i])

{

detect\_vector\_str=false;

}

else

{

detect\_vector\_str = true;

break;

}

}

if(detect\_vector\_str == false)

{

remaining\_words.push\_back(words\_in\_sentence[i]);

}

}

}

delete[] words\_in\_sentence;

}

for (size\_t k = 0; k < remaining\_words.size(); k++)

{

for (size\_t i = 0; i < sentences.size(); i++)

{

for (size\_t j = 0; j < sentences[i].size(); j++)

{

if (remaining\_words[k] == sentences[i][j].data && sentences[i][j].type == UNKNOWN)

{

sentences[i][j].type = ADDITION;

}

}

}

}

destroy\_arr();

}

void destroy\_arr()

{

delete[] exceptions;//

delete[] mas\_of\_endings\_verb; //

delete[] mas\_of\_prepositions;//

delete[] mas\_of\_obst\_preposition;

delete[] mas\_of\_suffiks\_of\_oborot;//

delete[] mas\_of\_unions;//

delete[] mas\_of\_podl\_endings;//

delete[] mas\_of\_adjectives;//

delete[] obst\_word;//

delete[] mest\_word;//

delete[] particles;//

delete[] interjections;//

}

## Файл «**circumstance**.hpp»

#ifndef CIRC\_H

#define CIRC\_H

#include"TextReader.hpp"

bool LoadLibs(); // Это запускаем в начале

int UnloadLibs(); // Это запускаем в конце

void find\_circumstances();

#endif

## Файл «**circumstance**.cpp»

#include"circumstance.hpp"

#include"TextReader.hpp"

#include <string>

#include <fstream>

#include <vector>

char\*\* Predlog;

std::string\*\* Okonchanie, \* NotAdvNO, \* NotAdvPronoun, \* Top50Adv;

int PredNum, \* OkNum, NotAdvNONum, NotAdvPronounNum, Top50AdvNum;

bool LoadLibs()

{

std::ifstream fin;

fin.open("res/Предлоги.txt");

fin >> PredNum;

Predlog = new char\* [PredNum];

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

{

Predlog[i] = new char[8];

fin >> Predlog[i];

}

fin.close();

OkNum = new int[PredNum];

Okonchanie = new std::string \* [PredNum];

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

{

fin.open("res/" + std::string(Predlog[i]) + ".txt");

fin >> OkNum[i];

Okonchanie[i] = new std::string[OkNum[i]];

for (int j = 0; j < OkNum[i]; j++)

{

fin >> Okonchanie[i][j];

}

fin.close();

}

fin.open("res/НО.txt");

fin >> NotAdvNONum;

NotAdvNO = new std::string[NotAdvNONum];

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

{

fin >> NotAdvNO[i];

}

fin.close();

fin.open("res/Местоимения.txt");

fin >> NotAdvPronounNum;

NotAdvPronoun = new std::string[NotAdvPronounNum];

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

{

fin >> NotAdvPronoun[i];

}

fin.close();

fin.close();

fin.open("res/Топ.txt");

fin >> Top50AdvNum;

Top50Adv = new std::string[Top50AdvNum];

for (int i = 0; i < Top50AdvNum; i++)

{

fin >> Top50Adv[i];

}

fin.close();

return 0;

}

int UnloadLibs()

{

delete[] NotAdvNO;

delete[] NotAdvPronoun;

delete[] Top50Adv;

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

{

delete[] Predlog[i];

}

delete[] Predlog;

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

{

delete[] Okonchanie[i];

}

delete[] OkNum;

delete[] Okonchanie;

return 0;

}

bool IsPredlog(std::string word) {

if ((word[0] >= 192) && (word[0] <= 223)) word[0] = word[0] + 32;

for (int i = 0; i < PredNum; ++i) {

if (word == Predlog[i]) return true;

}

return false;

}

std::string Vowels = "АаЕеЁёИиОоУуЫыЭэЮюЯя";

bool CheckNotVowel(char ch) { //Проверка не является ли данная буква гласной

for (int i = 0; i < Vowels.length(); i++)

{

if (ch == Vowels[i]) return false;

}

return true;

}

void FindAdverbLite(std::vector<std::vector<Word>>& sentences)

{

for (size\_t i = 0; i < sentences.size(); ++i) {

for (size\_t j = 0; j < sentences[i].size(); ++j) {

if (sentences[i][j].data.length() > 2) {

std::string substr = sentences[i][j].data.substr(sentences[i][j].data.length() - 2, 2);

if (substr == "но") {

bool flag = 1;

for (int c = 0; c < NotAdvNONum; c++)

{

if (sentences[i][j].data == NotAdvNO[c]) flag = 0;

}

if (flag)

{

sentences[i][j].type = CIRCUMSTANCE;

}

}

}

if (sentences[i][j].data.length() > 3) {

std::string substr = sentences[i][j].data.substr(sentences[i][j].data.length() - 3, 3);

if (substr == "гда") { //поиск слов, кончающихся на "гда"

if ((sentences[i][j].data != "Когда") && (sentences[i][j].data != "когда"))

sentences[i][j].type = CIRCUMSTANCE;

continue;

}

}

if (sentences[i][j].data.length() > 4) {

std::string substr = sentences[i][j].data.substr(sentences[i][j].data.length() - 4, 4);

if (!CheckNotVowel(substr[0])) {

if (CheckNotVowel(substr[1])) {

if ((substr[2] == ('н'))) {

if (substr[3] == ('е')) {

sentences[i][j].type = CIRCUMSTANCE;

continue;

}

}

}

}

}

if (sentences[i][j].data.length() > 5) {

std::string substr = sentences[i][j].data.substr(sentences[i][j].data.length() - 5, 5);

if (substr == "чески") {

sentences[i][j].type = CIRCUMSTANCE;

continue;

}

if (substr == "-таки") {

sentences[i][j].type = CIRCUMSTANCE;

continue;

}

}

if (sentences[i][j].data.length() > 3) {

std::string substr = sentences[i][j].data.substr(0, 3);

if (substr == "по-") {

sentences[i][j].type = CIRCUMSTANCE;

continue;

}

}

for (int k = 0; k < Top50AdvNum; k++)

{

if (sentences[i][j].data == Top50Adv[k])

{

sentences[i][j].type = CIRCUMSTANCE;

break;

}

}

}

}

}

void FindByEndings(std::vector<Word>& sentence, const int k, size\_t from, int sentenceNum) {

for (size\_t i = from; i < sentence.size(); ++i) {

bool flag = true;

for (int j = 0; j < NotAdvPronounNum; j++) //вынести в 253

{

if (sentence[i].data == NotAdvPronoun[j]) flag = 0;

}

for (int j = 0; j < Top50AdvNum; j++) //вынести в 253

{

if (sentence[i].data == Top50Adv[j]) flag = 0;

}

if (sentence[i].type != UNKNOWN) flag = 0;

if (flag)

{

for (size\_t c = 0; c < OkNum[k]; ++c) {

if (sentence[i].data.length() > Okonchanie[k][c].length()) {

std::string substr = sentence[i].data.substr(sentence[i].data.length() - Okonchanie[k][c].length(), Okonchanie[k][c].length());

if (substr == Okonchanie[k][c]) {

if (CheckNotVowel(sentence[i].data[sentence[i].data.length() - Okonchanie[k][c].length() - 1])) {

sentence[i].type = CIRCUMSTANCE;

}

else if (sentence[i].data == "ше" + Okonchanie[k][c]) {

sentence[i].type = CIRCUMSTANCE;

}

return;

}

}

}

}

}

}

void FindPredlogs(std::vector<std::vector<Word>>& sentences)

{

for (size\_t i = 0; i < sentences.size(); ++i)

{

for (size\_t j = 0; j < sentences[i].size(); ++j)

{

if (sentences[i][j].type == PREPOSITION)

if (IsPredlog(sentences[i][j].data))

{

// номер предложения, номер слова, что за предлог

for (int k = 0; k < PredNum; k++)

{

if (sentences[i][j].data == Predlog[k])

{

FindByEndings(sentences[i], k, j, i);

}

}

}

}

}

}

void find\_circumstances()

{

FindAdverbLite(sentences); // Это запускаем между

FindPredlogs(sentences); // Это запускаем между, тоже

}

## Файл «**definition**.hpp»

#pragma once

#include"TextReader.hpp"

void find\_definitions();

## Файл «**definition**.cpp»

#include"definition.hpp"

const vector<string> reg = { //Массив окончаний, свойственных определениям

"ий", "ый", "им",

"ым","ого","его",

"ему","ому","нем","шем","ьем",

"ом","ней","жей","шей","чей","ой",

"ое","ее","ая",

"яя","ую","юю",

"ыми","ими",

"ые","ие",

"ых","их"

};

void find\_definitions() // функция поиска определений

{

for (auto& Sentence : sentences) // итерация по предложениям

{

for (auto& word : Sentence) // итерация по словам в конкретном предложении

{

if (word.type == UNKNOWN) // если тип слова не был определен

{

for (const auto& it : reg) // итерация по окончаниям прилагательных

{

int position = word.data.rfind(it); // поиск окончания в слове

if ((position != string::npos) && (it.size() != word.data.size())) // если найдено окончание в слове

{

int length\_of\_ending = it.size(); // длина окончания из файла

if (word.data.size() - position == length\_of\_ending) // если в конце слова окончание

{

word.type = DEFINITION; // установка типа слова как член предложения определение

}

}

}

}

}

}

}

## Файл «**divide**.hpp»

#pragma once

#include"TextReader.hpp"

void uniquify\_words();

## Файл «**divide**.**c**pp»

#include"divide.hpp"

vector <Word> New\_word;

void word\_with\_same\_name\_and\_type(int need\_i, int need\_j)

{

int count = 0;

vector<int> members = { 0 };

string list\_of\_members;

for (int i = 0; i < sentences.size(); i++)

{

for (int j = 0; j < sentences[i].size(); j++)

{

if (i <= need\_i && sentences[i][j].data == sentences[need\_i][need\_j].data && sentences[i][j].type == sentences[need\_i][need\_j].type)

{

count++;

members.push\_back(i);

list\_of\_members += " " + std::to\_string(i + 1);

}

}

}

for (int k = 0; k < members.size(); k++)

{

for (int i = 0; i < sentences.size(); i++)

{

for (int j = 0; j < sentences[i].size(); j++)

{

if (i == k && sentences[i][j].data == sentences[need\_i][need\_j].data && sentences[i][j].type == sentences[need\_i][need\_j].type) {

sentences[i][j].frequency = count;

for (int g = 0; g < members.size(); g++)

{

sentences[i][j].sentences\_included = list\_of\_members;

}

}

}

}

}

sentences[need\_i][need\_j].frequency = count;

for (int g = 0; g < members.size(); g++)

{

sentences[need\_i][need\_j].sentences\_included = list\_of\_members;

}

}

void analysis()

{

for (int i = 0; i < sentences.size(); i++)

{

for (int j = 0; j < sentences[i].size(); j++)

{

if (sentences[i][j].type != UNKNOWN)

{

word\_with\_same\_name\_and\_type(i, j);

}

}

}

for (int i = 0; i < sentences.size(); i++)

{

for (int j = 0; j < sentences[i].size(); j++)

{

bool not\_uniq\_detect = false;

for (int k = 0; k < unique\_words.size(); k++)

{

if (sentences[i][j].data == unique\_words[k].data && sentences[i][j].type == unique\_words[k].type)

{

not\_uniq\_detect = true;

}

}

if (not\_uniq\_detect == false)

{

unique\_words.push\_back(sentences[i][j]);

}

}

}

}

bool comparator(Word& a, Word& b)

{

return a.data < b.data;

}

void uniquify\_words()

{

analysis();

sort(unique\_words.begin(), unique\_words.end(), comparator);

}

## Файл «**Predicate**.**h**pp»

#pragma once

#include"TextReader.hpp"

void find\_predicate();

## Файл «**Predicate**.**c**pp»

#include"Predicate.hpp"

#include<unordered\_map>

unordered\_map<string, vector<string>> files // словарь признаков глаголов

{

pair<string, vector<string>>{ "RF\_FC", { "ешь", "ет","уем","аем", "юем", "яем", "ают","ете","ут","ать","ять","еть" }},

pair<string, vector<string>>{ "RF\_SC", {"ишь","ит","дим","рим","зим","тим","оим","пим", "ите","ат","ят","ить"}},

pair<string, vector<string>>{ "RF\_Refl", {"ся","ось", "ись","ась"}},

pair<string, vector<string>>{ "RF\_Pt", {"л","ал","ял","ил","ла","ло","али","или","ели","яли",

"ули","сли","шли"}},

pair<string, vector<string>>{ "RF\_ImpM", {"ите"}},

pair<string, vector<string>>{"RF\_participle", {"та","ана", "мо", "то","ано","ыты", "ена"}},

pair<string, vector<string>>{ "RF\_First\_Person", {"му","шу","щу","ду","ву","ру","ну","зу","су","чу","аю","яю","ую","ею","рю","лю"}}

};

unordered\_map<string, vector<string>> exclusions // словарь исключений

{

pair<string, vector<string>>{ "EM", { "ем", "едем", "пишем", "тонем", "топнем","завладеем", "двинем","печем"}},

pair<string, vector<string>>{"U", {"укажу", "снаряжу", "нагружу", "подрежу", "разрежу"}}

};

bool check\_for\_exclusion(const string& inp, const string& reason) // проверка слова на исключительные признаки сказуемого

{

for (int i = 0; i < exclusions[reason].size(); i++) // итерация по словам-исключениям с заданным окончанием

{

if (exclusions[reason][i] == inp) // если исходное слово найдено в словаре исключений

{

return true;

}

}

return false;

}

bool chk\_end(const string& inp, const string& rf) // функция проверки слова на окончания глаголов

{

for(int i = 0; i < files[rf].size(); i++) // итерация по словарю окончаний сказуемых

{

int position = inp.rfind(files[rf][i]); // поиск окончания в слове

if (position != string::npos) // если окончание найдено в исходном слове

{

int length\_of\_ending = files[rf][i].size(); // длина окончания из файла

if (inp.size() - position == length\_of\_ending) // если в конце слова окончание

{

return true;

}

}

}

return false;

}

bool check\_for\_verb\_signs(const string& inp) // проверка на признаки сказуемого по всем его особенностям

{

if (chk\_end(inp, "RF\_FC") == true) // проверить на окончание первого спряжения

{

return true;

}

if (chk\_end(inp, "RF\_SC") == true) // проверить на окончание второго спряжения

{

return true;

}

if (chk\_end(inp, "RF\_Refl") == true) // проверить на окончание возвратного глагола

{

return true;

}

if (chk\_end(inp, "RF\_Pt") == true) // проверить на окончание глагола прошедшего времени

{

return true;

}

if (chk\_end(inp, "RF\_ImpM") == true) // проверить на окончание глаголов повелительного наклонения

{

return true;

}

if (chk\_end(inp, "RF\_participle") == true) // проверить на окончание глаголов повелительного наклонения

{

return true;

}

if (chk\_end(inp, "RF\_First\_Person") == true) // проверить на окончание глаголов первого лица

{

return true;

}

// проверка на вхождение в число исключений

if (check\_for\_exclusion(inp, "EM") == true) // окончание -ем

{

return true;

}

if (check\_for\_exclusion(inp, "U") == true) // окончание -ем

{

return true;

}

return false;

}

string ComplexVerb(const string& inp, stringstream& buff, int& offset) // функция анализа на составное глагольное сказуемое

{

vector<string> binding\_verbs = { "нача","ста","пусти",

"продолжа","остал","переста","броси","прекрати","намеревал",

"обязал","готов","долж","рад","был","хотел","мечта","буде","пытал" }; // массив начальных частей основных вспомогательных глаголов

string result;

string temp;

for(const auto& verb: binding\_verbs) // итерация по массиву начальных частей вспомогательных глаголов

{

int position = inp.find(verb); // поиск в исходном слове начала вспомогательного глагола

if (position != string::npos) // если слово найдено

{

if (position == 0) // если найдено в самом начале слова

{

result += inp; // записать вспомогательный глагол как сказуемое

buff >> temp; // извлечь следующее за ним слово

if (check\_for\_verb\_signs(temp) == true) // если следующее слово также глагол

{

offset += 1; // добавить к вспомогательному глаголу основной и

return (result + " " + temp); // записать его в результат вместе с вспомогательным глаголом в результирующую строку

}

else // если наречие

{

buff >> temp; // извлечь второе слово за вспомогательным глаголом

if (check\_for\_verb\_signs(temp) == true) // если слово - глагол

{

offset += 2; // добавить к глаголу еще один и

return (result + " " + temp); // записать его в результирующую строку

}

else

{

return result; // если следующее слово не является частью составного сказуемого, то определить вспомогательный глагол как самостоятельный простой

}

}

}

}

}

return result;

}

string CompexlNominal(const string& str, stringstream& buff) // функция анализа на составные именные сказуемые

{

vector<string> binding\_verbs = { "был", "будет", "является",

"стал", "остался", "бывал", "оказался", "считался", "казался", "является",

"слыл", "называли", "были", "бывают", "бываем", "явился"};

for(auto &verb :binding\_verbs) // поиск сказуемого среди списка вспомогательных для составных именных

{

int position = verb.find(str); // если слово совпадает с вспомогательным глаголом составного именного сказуемого

if (position != string::npos && position == 0) // если нашли слово и они по-настоящему совпадают

{

string next\_word; // строка хранения второй части составного сказуемого

buff >> next\_word; // извлечь из строки следующее за ним слово

return str + " " + next\_word; // вернуть вспомогательный глагол и следующее слово за ним

}

}

return ""; // если глагол не является вспомогательным, вернуть пустую строку

}

void find\_predicate() // основная функция поиска сказуемых

{

for (auto& Sentence : sentences) // итерация по предложениям в тексте

{

stringstream buffer\_verb; // буферный строковый поток для составных глагольных сказуемых

stringstream buffer\_for\_nominal; // буферный строковый поток для составных именных сказуемых

Word temporary; // временный объект структуры Слово

for (int i = 0; i != Sentence.size(); i++) // итерация по словам в предложении

{

if (Sentence[i].type == UNKNOWN) // если у слова тип не определен

{

buffer\_verb >> Sentence[i].data; // занести в строковый поток для составных глагольных сказуемых слово из предложения

buffer\_for\_nominal >> Sentence[i].data;// занести в строковый поток для составных именных сказуемых слово из предложения

if (Sentence[i].data.size() < 3) // если слово меньше, чем приблизительная минимальная длина глагола

{

continue; // не рассматривать его при анализе

}

else

{

// составное именное сказуемое

string temp = CompexlNominal(Sentence[i].data, buffer\_for\_nominal); // функция проверки на составное именное сказуемое

if (temp.size() != false)

{

Sentence[i].type = PREDICATE;

}

else

{

int offset = 0;

// составное глагольное сказуемое

string temp = ComplexVerb(Sentence[i].data, buffer\_verb, offset); // функция проверки на составное глагольное сказуемое

if (temp.size() != false)

{

Sentence[i].type = PREDICATE;

}

else if (check\_for\_verb\_signs(Sentence[i].data) == true)

{

Sentence[i].type = PREDICATE;

}

}

}

}

}

}

}

## Файл «**trpo\_podl**.**h**pp»

#pragma once

#include"TextReader.hpp"

#include <iostream>

#include <string>

#include <fstream>

#include <conio.h>

#include <iomanip>

#include <Windows.h>

#include <vector>

void find\_subjects();

## Файл «**trpo\_podl**.**c**pp»

#include"trpo\_podl.hpp"

using namespace std;

vector <string> isc\_podl;//найденные в текущем предложении подлежащие

bool mestoimenia(vector <string>& piece\_of\_sentence);

bool imena(string first\_word);

bool side\_of\_podl(vector <string>& piece\_of\_sentence, int poz\_skaz);//poz\_skaz позиция рассматриваемого сказцемого выраженного глаголом

void analysis(vector <string>& piece\_of\_sentence);

bool Capitalized\_word(vector <string>& piece\_of\_sentence);

void check\_for\_predicate(const int& i)

{

bool part\_of\_sentence\_mentioned = false;

for (int j = 0; j < sentences[i].size(); j++)

{

if (part\_of\_sentence\_mentioned == false)

{

if (sentences[i][j].type == PREDICATE)

{

if (j != 0)

{

if (sentences[i][j - 1].type == UNKNOWN)

{

sentences[i][j - 1].type = SUBJECT;

part\_of\_sentence\_mentioned = true;

}

else if (j != sentences[i].size() - 1)

{

if (sentences[i][j + 1].type == UNKNOWN)

{

sentences[i][j + 1].type = SUBJECT;

part\_of\_sentence\_mentioned = true;

}

}

}

}

}

if (sentences[i][j].type == PUNCTUATION)

{

part\_of\_sentence\_mentioned = false;

}

}

}

void find\_subjects()

{

for (int i = 0; i < sentences.size(); i++)//цикл по всем предложениям

{

check\_for\_predicate(i);

//цикл по опредленному предложению для нахождения знаков препинания и и количества ячеек до него

int counter = 0;

int g = 0;//запоминает старт части предложения

string new\_word;

vector <string> piece\_of\_sentence;

bool No\_commas = false;//флаг если нет запятой

while (counter < sentences[i].size())//разбиение предложения на подпредложения

{

string h = sentences[i][counter].data;

piece\_of\_sentence.push\_back(h);//кладем слово во временный вектор piece\_of\_sentence

if (sentences[i][counter + 1].data == "," || sentences[i][counter + 1].data == "." || sentences[i][counter + 1].data == "!" || sentences[i][counter + 1].data == "?")//+ 1 так как у нас знаки препинания тоже часть вектора и надо их скипать

{

No\_commas = true;

analysis(piece\_of\_sentence);//вызываем функцию анализатора

int d = 100;

for (int r = counter; r >= g; --r )//обратный ход по части предложения для установки соответсвия

{

if (isc\_podl.size() && isc\_podl[0] == sentences[i][r].data && sentences[i][r].type == UNKNOWN)//если найденное подлежащее равно тому что есть в предложении И оно еще не было назначено другим членом предложения

{

sentences[i][r].type = SUBJECT;

}

}

counter = counter + 2;//скипаем занк препинания в тексте

g = counter;

isc\_podl.clear();

piece\_of\_sentence.clear();

}

else

{

counter++;

}

}

}

}

void analysis(vector <string>& piece\_of\_sentence)

{

bool flag\_spec = false;

char break\_point = 0;//номер аналзируемого признака

while (!flag\_spec)

{

++break\_point;

switch (break\_point)

{

case 1: flag\_spec = mestoimenia(piece\_of\_sentence);

break;

case 2: flag\_spec = Capitalized\_word(piece\_of\_sentence);//imena(piece\_of\_sentence)

break;

case 3: flag\_spec = true;//что бы выйти

break;

}

}

}

bool mestoimenia(vector <string>& piece\_of\_sentence)

{

int h = 0;

vector <string> mestoim{ "я", "ты", "мы", "вы", "он", "она", "оно", "они", "Я", "Ты", "Мы", "Вы", "Он", "Она", "Оно", "Они" };

while (mestoim.size() < h)

{

for (int i = 0; i < piece\_of\_sentence.size(); ++i)

{

if (i == piece\_of\_sentence.size() - 1)

{

string last\_word = piece\_of\_sentence[i];

last\_word.erase(last\_word.end() - 1);

if (last\_word == mestoim[h])

{

isc\_podl.push\_back(mestoim[h]);

return true;

}

}

if (piece\_of\_sentence[i] == mestoim[h])

{

isc\_podl.push\_back(mestoim[h]);

return true;

}

}

++h;

}

return false;

}

bool imena(string first\_word)

{

vector <string> imena{ "Саша", "Александр", "Маша", "Мария", "Таня", "Татьяна", "Герасим", "Иван", "Ваня", "Анастасия", "Настя", "Алексей", "Оля", "Ольга", "Дима", "Дмитрий", "Анна", "Аня", "Дмитрий", "Дима", "Сергей", "Катя", "Алексей", "Леша", "Захар", "Михей" };

int k = 0;

while (imena.size() < k)

{

if (first\_word[first\_word.length() - 1] == '.' || first\_word[first\_word.length() - 1] == '!' || first\_word[first\_word.length() - 1] == '?')

{

first\_word.erase(first\_word.end() - 1);

}

if (first\_word == imena[k])

{

return true;

}

++k;

}

return false;

}

bool Capitalized\_word(vector <string>& piece\_of\_sentence)

{

int i = 0;//счетчик элементов вектора

bool capital\_first = false;

while (i < piece\_of\_sentence.size())

{

if (i == 0)

{

string first\_word = piece\_of\_sentence[i];

capital\_first = imena(first\_word);

if (capital\_first == true)

{

isc\_podl.push\_back(first\_word);

return true;

}

}

else

{

string anyword = piece\_of\_sentence[i];

if (anyword[0] > 'А' && anyword[0] < 'Я')

{

isc\_podl.push\_back(anyword);

return true;

}

}

++i;

}

return capital\_first;

}

bool side\_of\_podl(vector <string>& piece\_of\_sentence, int poz\_skaz)

{

if (piece\_of\_sentence.size() <= 1)//если в векторе одно слово

{

return false;

}

if (poz\_skaz == 0)

{

isc\_podl.push\_back(piece\_of\_sentence[poz\_skaz + 1]);//если это начало предложения, то тогда второе слово подлежащее

return true;

}

if (poz\_skaz == piece\_of\_sentence.size() - 1)

{

isc\_podl.push\_back(piece\_of\_sentence[poz\_skaz - 1]);//если это конец предложения, то тогда предыдущее слово подлежащее

return true;

}

else

{

vector <string> predlog{ "Без", "Между", "Под", "В", "На", "По", "Вокруг", "О", "Про", "До", "Об", "С", "Для", "Около", "Из-за", "За", "От", "Из-под", "К", "Перед", "Не", "И", "Что", "Его", "Только", "без", "между", "под", "в", "на", "по", "вокруг", "о", "про", "до", "об", "с", "для", "около", "из-за", "за", "от", "из-под", "к", "перед", "не", "и", "что", "его", "только" };

int i = 0;

while (predlog.size() < i)

{

int j = 0;

while (j < piece\_of\_sentence.size())

{

if (predlog[i] == piece\_of\_sentence[j])//добавить типо расстояние до нашего сказуемного равное или меньшее 0

{

if (j < poz\_skaz)

{

isc\_podl.push\_back(piece\_of\_sentence[poz\_skaz + 1]);

return true;

}

if (j > poz\_skaz)

{

isc\_podl.push\_back(piece\_of\_sentence[poz\_skaz - 1]);

return true;

}

}

++j;

}

}

++i;

}

return false;

}