ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Доцент факультета компьютерных наук, заместитель декана по учебно-методической работе, канд. социол. наук

_____ И. Ю. Самоненко
«_____» _____ 2020 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2020 г.

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.13-01 ПЗ 01-1				

Приложение для визуализации метода рекурсивного спуска

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.13-01 ПЗ 01-1-ЛУ

Исполнитель:
студент группы БПИ 199
_____ К. Н. Борисов
«_____» _____ 2020 г.

Москва 2020

УТВЕРЖДЕН
RU.17701729.04.13-01 ПЗ 01-1-ЛУ

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.13-01 ПЗ 01-1				

Приложение для визуализации метода рекурсивного спуска

Пояснительная записка

RU.17701729.04.13-01 ПЗ 01-1

Листов 22

Москва 2020

Содержание

1 Введение

1.1 Наименование программы

«Приложение для визуализации метода рекурсивного спуска»

1.2 Документы, на основании которых ведется разработка

Приказ № 2.3-02/2004-04 от 20.04.2020 «Об изменении тем, руководителей курсовых работ студентов образовательной программы «Программная инженерия» факультета компьютерных наук».

2 Назначение и область применения

2.1 Назначение программы

2.1.1 Функциональное назначение

Разрабатываемое приложение «Приложение для визуализации метода рекурсивного спуска» предназначено для визуализации синтаксического анализа методом рекурсивного спуска.

2.1.2 Эксплуатационное назначение

Программа наглядно демонстрирует работу метода рекурсивного спуска и помогает лучше понять процесс его работы, может использоваться как для изучения непосредственно методов синтаксического анализа, так и для симуляции условий в задачах связанных с работой таких методов.

2.2 Область применения

Программа используется в сфере образования для демонстрации работы методов синтаксического анализа, в том числе и метода рекурсивного спуска.

3 Технические характеристики

3.1 Постановка задачи на разработку программы

Программа разработана в рамках курсовой работы на образовательной программе «Программная инженерия» ФКН ВШЭ, тема которой – «Приложение для визуализации метода рекурсивного спуска», а цель – написать программу способную визуализировать метод рекурсивного спуска.

3.2 Описание алгоритмов и функционирования программы

Программа представляет собой Windows приложение, написанное с использованием технологии WPF, которое визуализирует синтаксический анализ заданной пользователем строки методом рекурсивного спуска.

3.2.1 Описание общей схемы работы приложения

1) При запуске программы, из папки parsers подгружается файл simple.rtf, содержащий описание формальной грамматики рассматриваемого синтаксического анализатора.
2) Запускается файл simple.exe, содержащий синтаксический анализатор. Ему на вход в stdin подаётся строка, введённая пользователём.
3) Строки, которые simple.exe вывёл в stdout, по одной добавляются в экземпляр класса ParserHistory (см. приложение 1). Он использует полученные строки для построения синтаксического дерева и сохраняет каждую промежуточную версию своего дерева (см. приложение 2).
4) После завершения работы simple.exe пользователь получает полный контроль над тем, какой кадр будет выведен на экран. Он может вручную выбирать показываемый кадр, а может включить автоматическую прокрутку. Он также может поменять входную строку, тогда этот алгоритм будет выполнен заново, начиная с шага 2.

3.2.2 Описание алгоритма построения синтаксического дерева

Это тот алгоритм, который упоминается во втором шаге алгоритма, описанного в разделе 3.2.1. Сначала создаются пустой стек узлов и пустой список узлов, а потом для каждой полученной строки выполняются следующие шаги:

1) Если эта строка сообщает нам о начале правила, то мы создаём новый узел и добавляем его в стек и в список. Предыдущая верхушка стека становится родителем нового узла.
2) Если эта строка сообщает нам об удачном конце правила, то мы удаляем верхний элемент из стека.
3) Если эта строка сообщает нам о неудачном конце правила, то мы удаляем верхний элемент из стека. Мы также удаляем его и всех его потомков из списка.
4) Текущие состояние списка сохраняется в историю версий дерева. Туда также сохраняется код с правильной подсветкой (см. разд. 3.2.4).

3.2.3 Описание алгоритма «гравитации» синтаксического дерева

– Создаётся массив чисел, длинна которого совпадает с длинной строки, и заполняется нулями.

– Список узлов сортируется по их уровню либо по возрастанию, либо по убыванию, в зависимости от направления гравитации.

– Для каждого узла в отсортированном списке:

1) Находится сегмент массива, соответствующий этому узлу.
2) Визуальному уровню этого узла присваивается максимальное значение из найденного сегмента.
3) Все значения в найденном сегменте увеличиваются на один.

3.2.4 Описание алгоритма подсветки кода

– RTF код разбивается на строки.

– Для каждого узла в стеке (см. разд. 3.2.2):

1) Находится строка, начинающаяся с имени родителя этого узла.
2) В этой строке находится n-ая подстрока вида `"\\b{}"`, где n это Index нашего узла (см. табл. 18).
3) Найденная подстрока заменяется на `"\\b "`. В RTF `"\\b{}"` и `"\\b "` имеют одинаковое значение – выделение жирным [9].
4) Все остальные подстроки вида `"\\b{}"` удаляются.

– Во всем документе все `"{}"` заменяются на `"0"`. Это выключает выделение жирным везде, где оно не было включено этим алгоритмом.

3.3 Обоснование выбора алгоритма решения задачи

Система с внешними синтаксическими анализаторами, описанная в разд. 3.2.1, была выбрана для того, чтобы мою программу можно было использовать для визуализации любого синтаксического анализатора, не меняя её код. Также внешние синтаксические анализаторы вместе с их формальной грамматикой в формате RTF можно автоматически генерировать из формальной грамматики на БНФ, записанной в обычном текстовом формате. Я для этого использовал PEG.js, так как у него самая понятная документация и он может генерировать синтаксические анализаторы, которые выводят свой прогресс [10].

Для хранения форматированного текста (формальной грамматики и объяснений алгоритма) был выбран формат RTF, потому что его поддерживают richTextBox-ы в WinForms и в WPF [8]. Его также может открывать Microsoft Word, и, поскольку он хранится как обычный текст, его изменения видны в таких системах контроля версий как Git.

Итеративный алгоритм, описанный в разд. 3.2.3, был выбран, потому что, в отличие от рекурсивного, для него не надо хранить полную структуру дерева (нужен только список узлов) и он понятнее рекурсивного.

3.4 Описание и обоснование выбора метода организации входных и выходных данных

3.4.1 Описание метода организации входных и выходных данных

Входные данные представляют собой строку, синтаксический анализ которой визуализироваться. Строка задаётся посредством поля ввода, расположенного непосредственно над синтаксическим деревом.

Выходные данные, согласно требованиям из технического задания, представляют собой рисунок дерева в формате PNG. Пользователь может сохранить этот файл в удобное ему место, используя диалоговое окно, которое появляется при нажатии на кнопку «Сохранить».

3.4.2 Обоснование выбора метода организации входных и выходных данных

Поле ввода строки, синтаксический анализ которой будет визуализироваться, расположено непосредственно над синтаксическим деревом, потому что каждый символ этой строки является листом синтаксического дерева.

Выходные данные организованы в формате PNG в силу требований технического задания. Также использование любого другого формата растровых изображений было бы не логично, так как JPEG сжимает изображение с потерями и не поддерживает прозрачность, а более сложные форматы могут не поддерживается. В целях экономии места на панели управления процессом визуализации, для экспорта и для сохранения используется одна и та же кнопка.

3.5 Описание и обоснование выбора состава технических и программных средств

3.5.1 Состав технических и программных средств

Состав технических средств, необходимых для работы системы:

1) Процессор архитектуры x86 или x64 с частотой не менее 1 ГГц;
2) Не менее 2 ГБ ОЗУ;
3) Не менее 5 МБ свободного места на жестком диске;
4) Графическое устройство DirectX 9 с драйвером WDDM 1.0 или более поздней версии.
5) Windows 7 или более пздняя версия операционной системы (32-разрядные или 64-разрядные);
6) Установленный .NET Framework версии 4.5 и выше;

4 Технико-экономические показатели

4.1 Предполагаемая потребность

Синтаксический анализ имеет много различных областей применения. Его часто надо использовать при написание программного обеспечения. Но большинство алгоритмов синтаксического анализа настолько сложные, что даже те программисты, которые их понимают, не пишут их от руки, а используют так называемые «компиляторы компиляторов» [11], о существование которых почти никто не знает. Поэтому, когда людям надо сделать синтаксический аналий какой-то строки, они вынужденны использовать неоптимальные методы.

Например, многие, при попытке написать десериализатор CSV, сначала разбивают файл на строки по разделителю `"\n"`, а потом разбивают каждую строку по разделителю `","`. Такой код написать очень просто, так как в большинстве языков программирования есть метод String.Split, но это будет неправильный десериализатор CSV, потому что в CSV каждое поле может быть заключено в двойные кавычки [13]. Чтобы написать десериализатор CSV, который корректно обрабатывает поля в к двойных кавычках, надо будет весь этот код переписывать заново и он станет раз в 10 сложнее.

Большинство проблем, для решения которых нужен синтаксический анализ, можно плохо решить используя String.Split и регулярные выражения. Поэтому многие программисты так и не узнают о существовании синтаксических анализаторов и об методах их автоматической генерации.

Моя программа позволяет понять как работает метод рекурсивного спуска, простейший из методов синтаксического анализа, и как формальная грамматика записывается в БНФ. Это поможет им понять как пользоваться генераторами синтаксических анализаторов, так как они использует БНФ [14].

5 Список использованной литературы

[1] ГОСТ 19.201-78. ЕСПД. Техническое задание. Требования к содержанию и оформлению. — М.: ИПК Издательство стандартов, 2001.

[2] ГОСТ 19.301-79. ЕСПД. Программа и методика испытаний. Требования к содержанию и оформлению. — М.: ИПК Издательство стандартов, 2001.

[3] ГОСТ 19.401-78. ЕСПД. Текст программы. Требования к содержанию и оформлению. — М.: ИПК Издательство стандартов, 2001.

[4] ГОСТ 19.404-79. ЕСПД. Пояснительная записка. Требования к содержанию и оформлению. — М.: ИПК Издательство стандартов, 2001.

[5] ГОСТ 19.505-79. ЕСПД. Руководство оператора. Требования к содержанию и оформлению. — М.: ИПК Издательство стандартов, 2001.

[6] ГОСТ 15150-69 Машины, приборы и другие технические изделия. Исполнения для различных климатических районов. Категории, условия эксплуатации, хранения и транспортирования в части воздействия климатических факторов внешней среды. — М.: Изд-во стандартов, 1997.

[7] LMS [Электронный ресурс] //URL: `https://lms.hse.ru` (Дата обращения: 27.11.2019, режим доступа: свободный)

[8] Документация Microsoft WPF [Электронный ресурс] //URL: `https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/` (Дата обращения: 19.05.2020, режим доступа: свободный)

[9] Rich Text Format (RTF) Version 1.5 Specification [Электронный ресурс] //URL: `http://www.biblioscape.com/rtf15_spec.htm` (Дата обращения: 19.05.2020, режим доступа: свободный)

[10] PEG.js Documentation [Электронный ресурс] //URL: `https://pegjs.org/documentation` (Дата обращения: 19.05.2020, режим доступа: свободный)

[11] Статья «Recursive descent parser» Wikipedia.org //URL: `https://en.wikipedia.org/wiki/Recursive_descent_parser` (Дата обращения: 19.05.2020, режим доступа: свободный)

[12] Статья «Backus-Naur form» Wikipedia.org //URL: `https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form` (Дата обращения: 19.05.2020, режим доступа: свободный)

[13] Статья «Comma-separated values» Wikipedia.org //URL: `https://en.wikipedia.org/wiki/Comma-separated_values` (Дата обращения: 23.05.2020, режим доступа: свободный)

[14] Статья «Компилятор компиляторов» Wikipedia.org //URL: `https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80_%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80%D0%BE%D0%B2` (Дата обращения: 23.05.2020, режим доступа: свободный)

ПРИЛОЖЕНИЕ 1

Описание и функциональное назначение классов

Таблица 1 — Классы проекта

Класс	**Назначение**
MainWindow	Логика интерфейса главного окна
TreeCanvas	Отрисовка синтаксического дерева и палитры его цветов
HistoryEntry	Синтаксическое дерево в какой-то момент времени
HistoryToken	ParserTreeToken но без служебных полей
Parser	Запуск синтаксического анализатора
ParserHistory	Строит синтаксическое дерево и хранит каждую его версию
ParserTreeToken	Узел синтаксического дерева
RtfBuilder	Подсветка того кода, который сейчас выполняются

ПРИЛОЖЕНИЕ 2

Описание и функциональное назначение методов, полей и свойств

Таблица 2 — Описание методов класса MainWindow

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
InitializeEvents	private	void		Инициализирует события элементов интерфейса
SelectTutorialPage	private	void	int	Подгружает страницу объяснений с данным индексом
SetRtf	private	void	string	Меняет содержимое richTextBox-а на то, что написано в строке
DisplayHistoryEntry	private	void		Обновляет текущий кадр
CanvasLegend	private	void	bool	Обновляет цветовую палитру
RunParser	private	void	string	Запускает синтаксический анализатор
SetSpeed	private	void	double	Изменяет скорость автоматической прокрутки
Load	private	void	string	Загружает сохранение
Save	private	void	string	Сохраняет текущее состояние в файл
MainSliderChange	private	void	object, EventArgs	Обработчик события, которое вызывается при изменении значения основного бегунка
SpeedSliderChange	private	void	object, EventArgs	Обработчик события, которое вызывается при изменении значения бегунка скорости
SpeedBoxChange	private	void	object, EventArgs	Обработчик события, которое вызывается при изменении значения поля ввода скорости
InputBoxChange	private	void	object, EventArgs	Обработчик события, которое вызывается при изменении значения поля ввода строки
MainWindowDrop	private	void	object, DragEventArgs	Обработчик события, которое вызывается после перетаскивания файла в это окно
HyperlinkClick	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии на гиперссылку
NextFrame	private	void		Показывает следующий кадр
PrevFrame	private	void		Показывает предыдущий кадр
FirstFrame	private	void		Показывает первый кадр

Продолжение таблицы 2

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
LastFrame	private	void		Показывает последний кадр
TogglePause	private	void		Переключает паузу
Reverse	private	void		Меняет направление автоматической прокрутки
NextFrameEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для показа следующего кадра
PrevFrameEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для показа предыдущего кадра
FirstFrameEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для показа первого кадра
LastFrameEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для показа последнего кадра
TogglePauseEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для переключения паузы
ReverseEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для изменения направления автоматической прокрутки
SaveEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для сохранения текущего состояния
LoadEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для загрузки сохранения

Продолжение таблицы 2

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
NextTutorialEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для показа следующей страницы объяснений
PrevTutorialEvent	private	void	object, EventArgs	Обработчик события, которое вызывается при нажатии кнопки или клавиши быстрого вызова для показа предыдущей страницы объяснений
ExportToPng	private static	void	string, FrameworkElement	Делает снимок данного элемента и сохраняет его в файл с данным именем

Таблица 3 — Описание полей класса MainWindow

Поля			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
historyIndex	private	int	Индекс текущего кадра
isPaused	private	bool	Включена ли пауза
isReversed	private	bool	Идёт ли прокрутка в обратною сторону
colors	private	List<Brush>	Список цветов
tutorialIndex	private	int	Индекс текущей страницы объяснений
mainTimer	private	DispatcherTimer	Таймер автоматической прокрутки
speedBoxTimer	private	DispatcherTimer	Таймер для проверки корректности числа, введенного в поле ввода скорости
parser	private	Parser	Синтаксический анализатор
theHistory	private	ParserHistory	История
autosavePath	private	const string	Путь к файлу автосохранения

Таблица 4 — Описание свойств класса MainWindow

Свойства			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
speed	private	double	Скорость автоматической прокрутки
treeTrim	private	bool	Надо ли подровнять дерево
treeOrientation	private	bool	Ориентация дерева
treeGravity	private	bool	Гравитация дерева
treeHelp	private	bool	Режим новичка
inputString	private	string	Строка, синтаксический анализ которой будет визуализироваться

Таблица 5 — Описание методов класса TreeCanvas

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
WriteString	public	void	string	Рисует строку
InitLegend	public	void	List<Brush>, IEnumerable< string>, bool	Рисует цветовую палитру
DrawRect	private	void	HistoryToken, int	Рисует один узел дерева
DisplayHistoryEntry	public	void	HistoryEntry, bool	Рисует дерево
DrawConventionalTree	private	void	Dictionary< HistoryToken, HistoryToken >	Рисует ветки дерева
GetNodeCenterX	private	double	HistoryToken	Считает x координату центра узла дерева
GetNodeCenterY	private	double	HistoryToken	Считает y координату центра узла дерева
GetColor	private static	Brush		Запрашивает у пользователя собственный цвет, открывая диалоговое окно

Таблица 6 — Описание полей класса TreeCanvas

Поля			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
font	private	readonly FontFamily	Шрифт, который везде используется
colorDict	private	Dictionary<string, Brush>	Ставит в соответствие названия правил и их цвета
inputBox	private	TextBox	Поле ввода, где пользователь вводит строку, синтаксический анализ которой будет визуализироваться.
lastHistoryEntry	private	HistoryEntry	Тот кадр, который сейчас нарисован
lastHelp	private	bool	Нарисовано ли сейчас дерево в режиме новичка
CharWidth	private	const int	Ширина каждого символа в строке
TextStart	private	const int	Начальная позиция строки
FontSize	private	const int	Размер шрифта
LegendFontSize	private	const int	Размер шрифта подписей к цветам палитры
TextblockTop	private	const int	Позиция поля ввода
TreeTop	private	const int	Позиция верхушки дерева

Продолжение таблицы 6

Поля			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
TreeVSpace	private	const int	Вертикальное расстояние между узлами дерева
RectHeight	private	const int	Высота каждого узла дерева

Таблица 7 — Описание методов класса HistoryEntry

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
SetSettings	public	void	bool, bool, bool	Меняет настройки
GetEdges	public	Dictionary< HistoryToken, HistoryToken >		Находит список всех веток дерева, но возвращает словарь, где ключ это узел, а значение это его родитель
InvertDisplayLevels	private	void		Переворачивает дерево
CalculateDisplayLevels	private	void	bool	Рассчитывает гравитацию (см разд. 3.2.3)
ToString	public override	string		Переопределяет метод класса Object
GetEnumerator	public	IEnumerator< HistoryToken >		Реализует интерфейс IEnumerable< HistoryToken>
IEnumerable.GetEnumerator		IEnumerator		Реализует интерфейс IEnumerable

Таблица 8 — Описание полей класса HistoryEntry

Поля			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
treeRanges	private	HistoryToken[]	Узлы дерева
isBroken	internal	bool	Сломано ли это дерево и надо ли его отображать
isTrimmed	private	bool	Надо ли подравнивать это дерево
edges	private	Dictionary<HistoryToken, HistoryToken>	Ветки дерева

Таблица 9 — Описание свойств класса HistoryEntry

Свойства			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
RtfGrammar	public	string	Описание формальной грамматики в формате RTF
CursorPos	public	int	Индекс последнего анализированного символа

Таблица 10 — Описание методов класса HistoryToken

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
ToString	public override	string		Переопределяет метод класса Object

Таблица 11 — Описание свойств класса HistoryToken

Свойства			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
Name	public	string	Название правила, по которому был постоен этот узел
StartPos	public	int	Индекс первого символа, входящего в этот узел
EndPos	public	int	Индекс последнего символа, входящего в этот узел
RecLevel	internal	int	Глубина этого узла в дереве, его уровень
Trimmable	public	bool	Можно ли этот узел убирать при обрезке дерева
DisplayLevel	public	int	Визуальный уровень узла

Таблица 12 — Описание методов класса Parser

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
Run	public	ParserHistory	string	Запускает синтаксический анализатор

Таблица 13 — Описание полей класса Parser

Поля			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
name	private	string	Название синтаксического анализатора

Таблица 14 — Описание методов класса ParserHistory

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
CopyState	private	HistoryToken[]		Делает копию поля state
SaveState	private	void		Сохраняет текущее состояние в поле history
Add	public	void	string	Добавляет узел в дерево. Принимает на вход строки от синтаксического анализатора. Здесь происходит основное построение дерева
GetEnumerator	public	IEnumerator< HistoryToken >		Реализует интерфейс IEnumerable< HistoryToken>
IEnumerable.GetEnumerator		IEnumerator		Реализует интерфейс IEnumerable

Таблица 15 — Описание полей класса ParserHistory

Поля			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
stack	private	Stack<ParserTreeToken>	Внутренний стек, используемый для построения дерева
state	private	List<ParserTreeToken>	Список узлов, которые сейчас есть в дереве
prevPos	private	int	Предыдущая позиция конца узла
history	private	List<HistoryEntry>	Список всех старых деревьев

Таблица 16 — Описание свойств класса ParserHistory

Свойства			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
OriginalRtf	public	string	Оригинальная грамматика, из которой делаются все остальные
InputString	public	string	Входная строка
RuleNames	public	IEnumerable<string>	Имена всех правил
this	public	HistoryEntry	Индексатор

Таблица 17 — Описание методов класса ParserTreeToken

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
ToString	public override	string		Переопределяет метод класса Object
Clone	public	HistoryToken		Делает копию этого узла, выбрасывая служебные поля

Таблица 18 — Описание свойств класса ParserTreeToken

Свойства			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
Parent	public	ParserTreeToken	Родитель
Name	public	string	Название правила, по которому был постоен этот узел
Index	public	int	Каким по счету ребёнком является этот узел
StartPos	public	int	Индекс первого символа, входящего в этот узел
RecLevel	public	int	Глубина этого узла в дереве
EndPos	public	int	Индекс последнего символа, входящего в этот узел
ChildCount	public	int	Количество детей
Dict	public	Dictionary<string, int>	Количество детей каждого типа

Таблица 19 — Описание методов класса RtfBuilder

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
HighlightIdentifier	private	void	ParserTreeToken	Выделяет жирным правило, соответствующее данному узлу дерева
End	private	string		Завершить построение грамматики
Build	public static	string	string, IEnumerable< ParserTreeToken >	Собирает грамматику путём создания экземпляра класса RtfBuilder и вызова его методов HighlightIdentifier и End

Продолжение таблицы 19

Методы				
Имя	**Мод. Доступа**	**Тип**	**Аргументы**	**Назначение**
GetNames	public static	IEnumerable<string>	string	Возвращает имена правил, которые присутствуют в грамматике

Таблица 20 — Описание полей класса RtfBuilder

Поля			
Имя	**Мод. Доступа**	**Тип**	**Назначение**
lines	private	string[]	Массив строк RTF кода

ПРИЛОЖЕНИЕ 3

Диаграмма классов

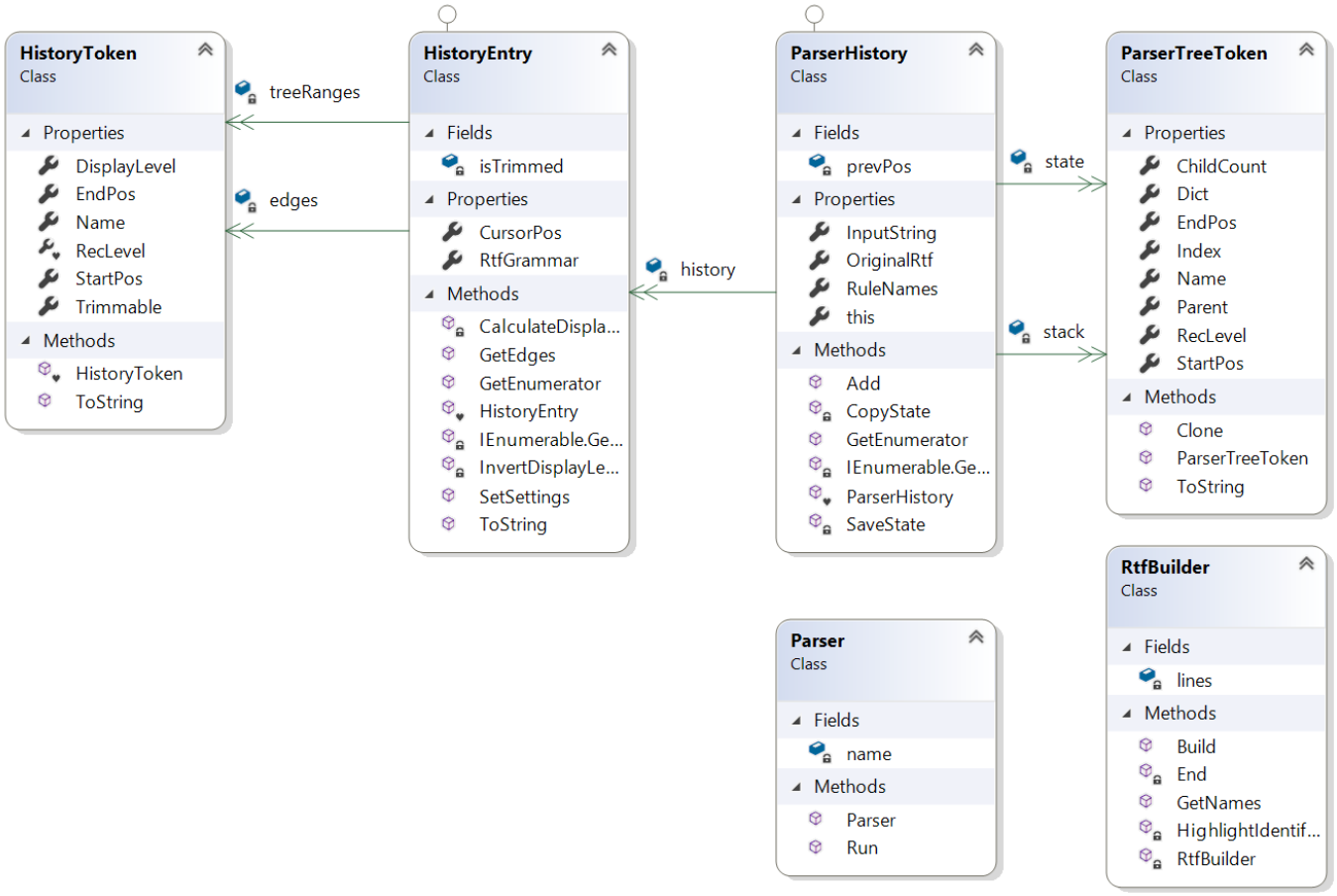

Рисунок 1 — Диаграмма классов

ПРИЛОЖЕНИЕ 4

Терминология

Термин	**Определение**
Граф	Множество вершин V и набор неупорядоченных пар вершин E.
Дерево	Граф, в котором между любыми двумя вершинами имеется ровно один путь.
Формальная грамматика	Способ описания формального языка. Определяется набором символов и правил, состоящих из левой и правой части. Эти символы могут быть терминальными и нетерминальными. Терминальные имеют конкретное, неизменяемое значение (обычно это символы ASCII), а нетерминальные могут стоять в левой части правил.
Синтаксическое дерево	Дерево, в котором листья сопоставлены терминальным символам формальной грамматики, а все остальные вершины – нетерминальным.
Синтаксический анализ	Процесс составления синтаксического дерева.
Синтаксический анализатор	Программа, выполняющая синтаксический анализ строки символов.
Метод рекурсивного спуска	Один из самых простых алгоритмов синтаксического анализа, реализуемый путём рекурсивного вызова функций.
Форма Бэкуса – Наура (БНФ)	Язык, на котором можно записывать формальную грамматику. Обычно используется для описания языков программирования.

6 Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопроводит. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					