Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра «Информатика»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А. И. Рубан
подпись
« _____» _____ 2016 г.

ДИПЛОМНЫЙ ПРОЕКТ

230105.65 Программное обеспечение вычислительной техники и автоматизированных систем

Визуальный редактор-генератор лексических анализаторов Visual flex

Пояснительная записка

Научный руководитель _____ доцент, к.т.н. А. С. Кузнецов
подпись, дата

Нормоконтролер _____ доцент, к.т.н. О. А. Антамошкин
подпись, дата

Выпускник _____ А. В. Кривлев
подпись, дата

Красноярск 2016

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра «Информатика»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А. И. Рубан
подпись
« _____» _____ 2016 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме дипломного проекта

Студенту Кривлеву Александру Викторовичу

Группа ЗКИ10-05 Направление (специальность) 230105.65, Программное обеспечение вычислительной техники и автоматизированных систем.

Тема выпускной квалификационной работы: «Визуальный редактор-генератор лексических анализаторов Visual flex».

Утверждена приказом по университету № 4202/с от 28.03.2016 г.

Руководитель ВКР А. С. Кузнецов, доцент кафедры «Информатика», канд. техн. наук.

Исходные данные для ВКР: спроектировать и разработать визуальный редактор-генератор лексических анализаторов Visual flex.

Перечень разделов ВКР:

- введение;
- анализ предметной области;
- программная реализация Visual flex;
- руководство пользователя;
- эргономика.

Перечень графического или иллюстративного материала с указанием основных чертежей, плакатов, слайдов: презентационные слайды PowerPoint.

Руководитель ВКР _____ А. С. Кузнецов
(подпись)

Задание принял к исполнению _____ А. В. Кривлев
(подпись)

« ___ » _____ 2016 г.

АННОТАЦИЯ

Выпускная квалификационная работа по теме «Визуальный редактор-генератор лексических анализаторов Visual flex» содержит 54 страницы текстового документа, включая 16 рисунков, 11 библиографических источников, 3 приложения.

Целью работы являлась разработка редактора-генератора лексических анализаторов. Редактор-генератор позволяет писать *flex*-спецификации с подсветкой синтаксиса и генерировать по ней код лексического анализатора, выделять конфликты при написании регулярных выражений, пошагово выполняться на тестовом входном файле, а также использовать точки останова в тестовом файле.

В дипломный проект входит введение, четыре главы и заключение.

Во введении определяется необходимость реализации и основные задачи проекта.

В первой главе проводится подробный анализ предметной области, описана доступная теория, связанная с лексическим анализом.

Во второй главе идет описание программной реализации приложения, архитектура приложения, реализация функциональных возможностей.

В третьей главе обосновывается выбор платформы и языка программирования, дается руководство пользователя, описывается генерация лексического анализатора, работа с файлами и анализатором.

Четвертая глава посвящена эргономики приложения.

В заключении представлено подведение итогов по всей проделанной работе.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В информатике при изучении компиляторных курсов, и, в частности лексического анализа используются инструменты для генерации лексических анализаторов. Большинство данных инструментов не имеют графической оболочки и кроме того для получения лексического анализатора с их помощью приходится вручную вводить соответствующий набор команд. Это усложняет процесс освоения данной темы, и увеличивает количество ошибок при создании лексического анализатора.

Поэтому возникает необходимость разработать функциональную среду, имеющую графический интерфейс и позволяющую автоматизировать генерацию лексических анализаторов с возможностью визуального контроля написания и редактирования спецификаций для анализатора.

Поскольку большинство генераторов лексических анализаторов не предоставляют удобный графический интерфейс цель данного дипломного проекта разработка Visual flex – визуального редактора-генератора лексических анализаторов на основе Flex.

Основные задачи проекта:

1) разработать графическую оболочку, в которой предоставить подсветку синтаксиса для написания *flex*-спецификаций с выделением конфликтов по мере написания регулярного выражения;

2) предоставить возможность автозавершения объявленных инструкция;

3) предоставить автоматическую генерацию исходного текста лексического анализатора с помощью Flex;

4) предоставить автоматическую генерацию лексического анализатора по исходному тексту с помощью свободного компилятора gcc;

5) предоставить возможность пошагового выполнения на тестовом входном файле;

6) предоставить возможность использования точек останова в тестовом входном файле.

1 Глава. Анализ предметной области

1.1 Лексический анализ

Лексический анализ или сканирование — это процесс аналитического разбора входной последовательности символов (например, такой как исходный код на языке программирования) с целью получения на выходе последовательности символов, называемых «токенами». Группа символов входной последовательности, идентифицируемая на выходе процесса, называется лексемой. В процессе лексического анализа производится распознавание и выделение лексем из входной последовательности символов.

Лексема – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своём составе других структурных единиц языка[1]. Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и т.п.

Лексический анализ производится с точки зрения определённого формального языка или набора языков. Язык, а точнее его грамматика, задаёт определённый набор лексем, которые могут встретиться на входе процесса. Принято организовывать процесс лексического анализа, рассматривая входную последовательность символов как поток символов. При такой организации процесс самостоятельно управляет выборкой отдельных символов из входного потока.

Распознавание лексем в контексте грамматики обычно производится путём их идентификации согласно идентификаторам токенов, определяемых грамматикой языка. При этом любая лексема, которая согласно грамматике не может быть идентифицирована как токен языка, рассматривается как ошибка.

Любой токен можно представить в виде структуры, содержащей идентификатор токена и последовательность символов лексемы, выделенной из входного потока (строку, число и т. д.). Цель такой конвертации состоит в том, чтобы подготовить входную последовательность для синтаксического анализатора, и изба-

вить его от определения лексических подробностей в контекстно-свободной грамматике.

Например, исходная программа содержит следующую инструкцию присваивания:

$$s = f * g$$

Символы в этом присваивании могут быть сгруппированы в следующие лексемы и отображены в следующие токены:

1) *s* - лексема, которая отображается в токен ‹*id, 1*›, где *id* – абстрактный символ, обозначающий идентификатор, а 1 указывает на запись в таблице идентификаторов для *a*, в которой хранится такая информация как имя и тип идентификатора.

2) Символ присваивания = - лексема, которая отображается в токен ‹=›. Этот токен не требует значения атрибута. В качестве имени токена может быть использован любой абстрактный символ, но для удобства записи в качестве имени абстрактного символа можно использовать саму лексему.

3) *f* - лексема, которая отображается в токен ‹*id, 2*›, где 2 указывает на запись в таблице идентификаторов для *b*.

4) * - лексема, отображаемая в токен ‹*›.

5) *g* – лексема, отображаемая в токен ‹*id, 3*›, где 3 указывает на запись в таблице идентификаторов для *c*.

Лексический анализ является первой фазой компиляции программы.

1.2 Лексический анализатор

Лексический анализатор — программа или часть программы, выполняющая лексический анализ[2].

На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору. На этапе лексического анализа обнаруживаются некоторые про-

стейшие ошибки, такие как недопустимые символы, неправильная запись чисел или идентификаторов и др.

Лексический анализатор входит в состав практически всех компиляторов по следующим причинам[3]:

1) структурируя поступающий на вход исходный текст программы и удаляя всю незначащую информацию, лексический анализатор упрощает работу с текстом исходной программы на этапе синтаксического разбора и сокращает объём обрабатываемой информации;
2) применяет простую, эффективную и теоретически хорошо проработанную технику анализа для выделения в тексте и разбора лексем;
3) отделяет сложный по конструкции синтаксический анализатор от работы непосредственно с текстом исходной программы.

В большинстве компиляторов лексический и синтаксический анализаторы – это взаимосвязанные части рисунок 1. Лексический разбор исходного текста в таком варианте выполняется поэтапно, так что синтаксический анализатор, выполнив разбор очередной конструкции языка, обращается к сканеру за следующей лексемой. При этом он может сообщить информацию о том, какую лексему следует ожидать. В процессе разбора может даже происходить «откат назад»[4], чтобы выполнить анализ текста на другой основе.

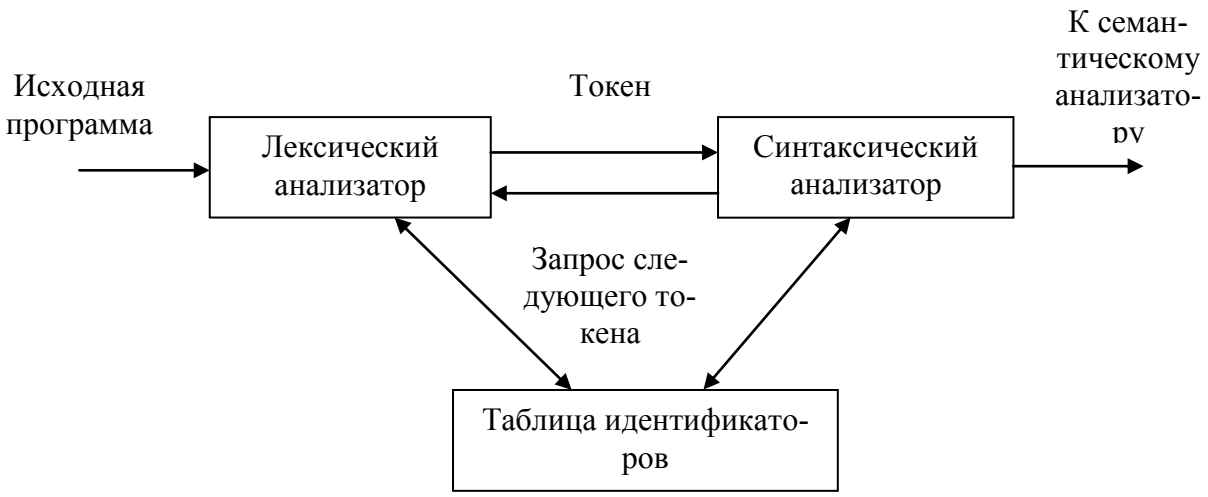

Рисунок 1 – Взаимодействие лексического анализатора с синтаксическим анализатором

1.3 Генератор лексических анализаторов Flex

Существуют различные программные средства для решения задачи построения лексических анализаторов. Наиболее известным из них является Flex [5].

Flex (Fast Lexical Analyzer) - программа для генерации лексических анализаторов.

Программный инструментарий Flex позволяет определить лексический анализатор с помощью регулярных выражений для описания шаблонов токенов. Входные обозначения для Flex обычно называют языком Flex, а сам инструмент – компилятором Flex. Компилятор Flex преобразует, входные шаблоны в конечный автомат и генерирует код (в файле с именем *lex.yy.c*), имитирующий данный автомат.

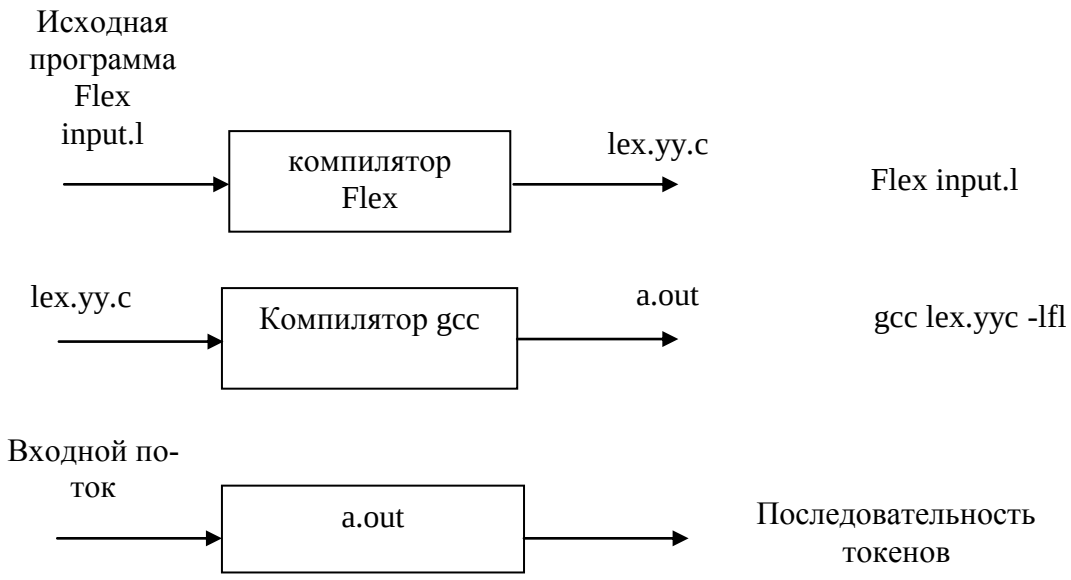

Рисунок 2 – Схема использования Flex

На рисунке 2 показаны схема использования Flex и команды, соответствующие каждому этапу генерирования лексического анализатора. Входной файл input.l написан на языке Flex и описывает генерируемый лексический анализатор. Компилятор Flex преобразует input.l в программу на языке программирования Си (файл с именем *lex.yy.c*). При компиляции *lex.yy.c* необходимо прилинковать библиотеку Flex *(-lfl*). Этот файл компилируется в файл с именем a.out. Выход компилятора Си представляет собой работающий лексический анализатор, который на основе потока входных символов выдаёт поток токенов.

Обычно полученный лексический анализатор, используется в качестве подпрограммы синтаксического анализатора.

Структура программы на языке Flex имеет следующий вид:

```
Объявления
%%
Правила трансляции
%%
Вспомогательные функции
```

Обязательным является наличие правил трансляции, а, следовательно, и символов `%%` перед ними. Правила могут и отсутствовать в файле, но `%%` должны присутствовать всё равно.

Пример самого короткого файла на языке Flex:

```
%%
```

В этом случае входной поток просто посимвольно копируется в выходной. По умолчанию, входным является стандартный входной поток (`stdin`), а выходным – стандартный выходной (`stdout`).

Раздел объявлений может включать объявления переменных, именованные константы и регулярные определения (например, `digit [0-9]` – регулярное выражение, описывающее множество цифр от 0 до 9). Кроме того, в разделе объявлений может помещаться символьный блок, содержащий определения на Си. Символьный блок всегда начинается с `%{` и заканчивается `%}`. Весь код символьного блока полностью копируется в начало генерируемого файла исходного кода лексического анализатора.

Второй раздел содержит правила трансляции вида

```
Шаблон { Действие }
```

Каждый шаблон является регулярным выражением, которое может использовать регулярные определения из раздела объявлений. Действия представляют собой фрагменты кода, обычно написанные на языке программирования Си, хотя существуют и разновидности Flex для других языков программирования.

Третий раздел содержит различные дополнительные функции на СИ, используемые в действиях. Flex копирует эту часть кода в конец генерируемого файла.

1.4 Анализ существующих решений

Среди существующих решений нет полноценных альтернатив разрабатываемому редактору-генератору лексических анализаторов.

В качестве схожего примера можно рассмотреть следующие продукты Visual BNF [6] и Visual IDE-Style LL(k) Parser Generator[7].

Данные продукты являются генераторами синтаксических анализаторов.

Visual BNF - это генератор синтаксических анализаторов общего назначения, он преобразует описание контекстно-свободной LALR/LR грамматики в детерминированный конечный автомат, который помещается в .dll файл.

Полученный файл в дальнейшем можно использовать в разрабатываемом приложении. Данный продукт получил имя Visual BNF поскольку пользователь может визуально разбирать код и видеть синтаксическое дерево разбора грамматики рисунок 3.

Visual IDE-Style LL(k) Parser Generator - использует синтаксическое дерево разбора грамматики с иконками для лексем, нетерминалов, представления символов грамматики, грамматических правил и т.д., рисунок 4.

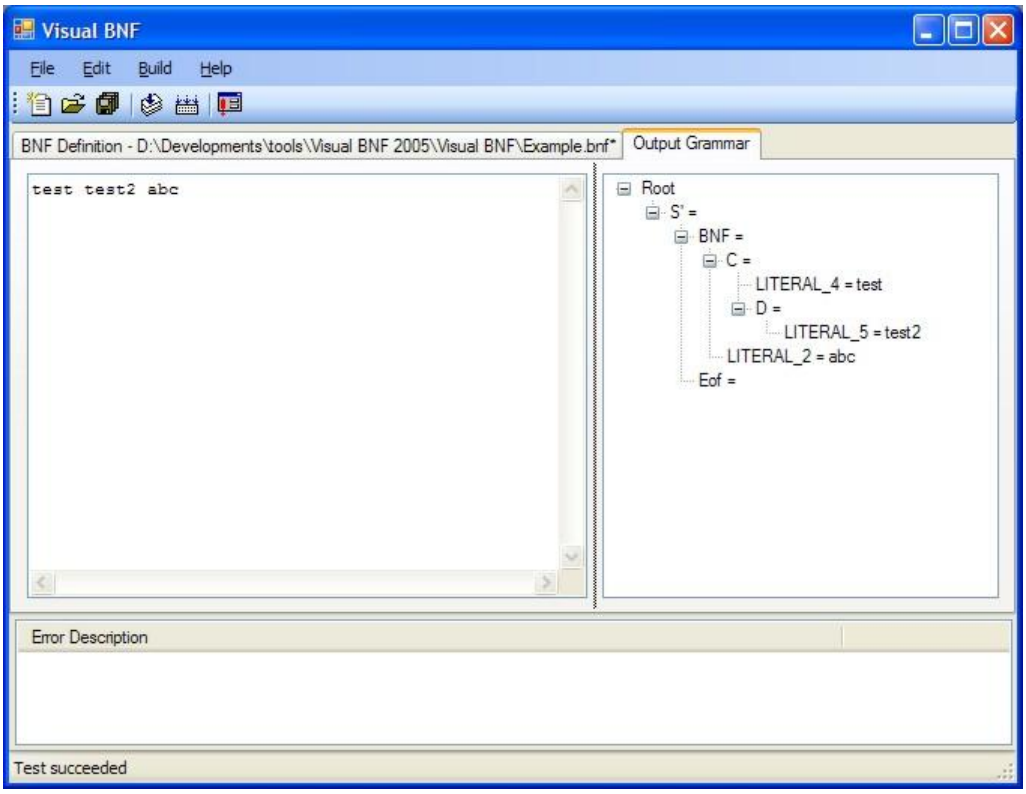

Рисунок 3 – Visual BNF

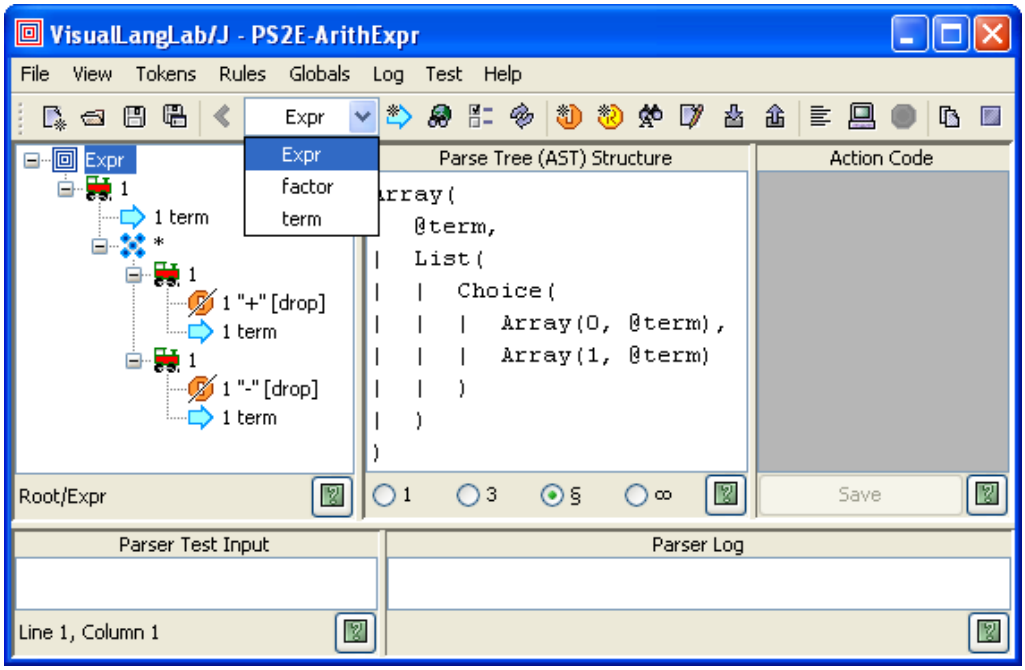

Рисунок 4 - Visual IDE-Style LL(k) Parser Generator

Данные инструменты не предоставляю текстовый редактор с подсветкой синтаксиса и выделением конфликтов грамматики. У них нет функций пошагового выполнения и точек останова при разборе входного файла.

Лексический анализ является встроенной частью синтаксического анализа данных инструментов, и генерация независимого лексического анализатора не предоставляется, что является полезным при изучении лексического анализа. К тому же данные инструменты не являются бесплатными либо предоставляются по коммерческой лицензии, что затрудняет их широкое использование.

2 Глава. Программная реализация Visual flex

2.1 Архитектура приложения

Архитектура приложения выполнена по типовой структуре рисунок 5.

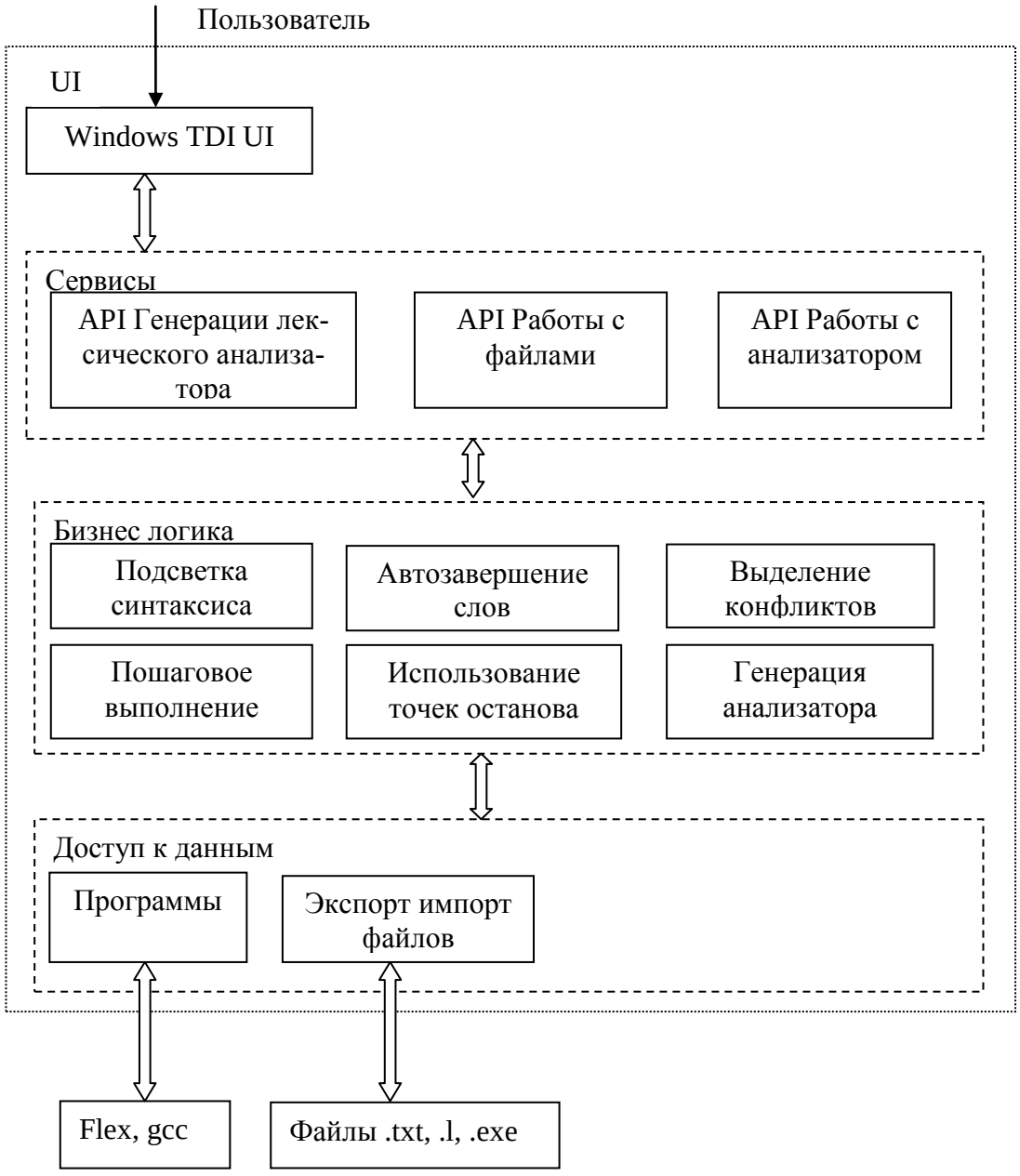

Рисунок 5 – Архитектура Visual flex

Презентационный слой реализует интерфейс, позволяющий пользователю взаимодействовать с приложением. Пользователь может:

- создавать, редактировать либо загружать файлы;
- генерировать лексический анализатор;
- работать с лексическим анализатором.

Слой бизнес логики представляет основное ядро приложения. Данный слой реализует программные интерфейсы, посредством которых осуществляется взаимодействие с презентационным слоем и внешними приложениями. В него входят основные компоненты, реализующие следующие функции:

- подсветку синтаксиса;
- автдополнение объявленного шаблона и автоматическое закрытие скобок;
- выделение конфликтов при написании регулярного выражения;
- пошаговое выполнение анализатором тестового файла;
- использование точек останова при выполнении анализатором тестового файла.

Слой доступа к данным включает компоненты позволяющие приложению использовать внешние программы и экспортировать импортировать файлы.

2.2 Реализация функциональных возможностей редактора генератора

Общий алгоритм программы представляет собой обработку сигналов от пользователя: нажатий на кнопки, выбор пунктов меню.

2.2.1 Генерация лексического анализатора

Для генерации лексического анализатора разработан статический класс `fileGenerate` расположенный в файле `fileGenerate.cs`. Данный класс предоставляет следующие функции:

- *public static void generateFlex(string flex, string nameCreateFlexFile, System.Windows.Forms.Timer timer, RichTextBox rtbInformFlexFile);*
- *public static void generateGenerator(string nameCreateFlexFile, RichTextBox rtbInformFlexFile).*

Функция *generateFlex* принимает на вход файл, написанный на языке Flex с расширением *.l* описывающий генерируемый лексический анализатор, обращается к функции *public void CreateFlexFile(string my, string namefile)* вспомогательного класса *MyProcess* которая запускает генератор лексических анализаторов Flex и подает ему на вход переданный файл *.l*. Flex преобразует входной файл в программу на языке программирования Си (файл с именем *lex.yy.c*), и сохраняет его в *C:\Visualflex*\.

Функция *generateGenerator* генерирует лексический анализатор, используя вспомогательную функцию *public void CreateGenerator(string namefile)* класса *MyProcess*. *CreateGenerator* обращается к компилятору gcc.exe и подаёт ему на вход файл *lex.yy.c* сгенерированный ранее компилятором Flex. В результате на выходе получается готовый к использованию лексический анализатор.

2.2.2 Работа с файлами

Для работы с файлами разработан статический класс *fileManagement* расположенный в файле *fileManagement.cs*. Данный класс предоставляет следующие функции:

- *public static void OpenTestFile(ref string nameRunFile, RichTextBox rtbOutText, RichTextBox rtbInformTestFile, RichTextBox rtbTestFile);*

- *public static void openFlexFile(ref string nameCreateFlexFile, FastColoredTextBox fctb, RichTextBox rtbInformFlexFile);*
- *public static void SaveTextFile(ref string nameRunFile, RichTextBox rtbTestFile, RichTextBox rtbInformTestFile);*
- *public static bool SaveFlexFile(ref string nameCreateFlexFile, FastColoredTextBox fctb, RichTextBox rtbInformFlexFile);*
- *public static void createFlexFile(ref string nameCreateFlexFile, FastColoredTextBox fctb, RichTextBox rtbInformFlexFile);*
- *public static void createTestFile(ref string nameRunFile, FastColoredTextBox fctb, RichTextBox rtbTestFile, RichTextBox rtbInformTestFile).*

Функция *OpenTestFile* открывает стандартный диалог выбора файла операционной системы Windows, считывает информацию из выбранного файла и загружает её на элемент управления предоставляющий поле для отображения текста для тестового файла, предварительно очистив его.

Функция *openFlexFile* открывает стандартный диалог выбора файла операционной системы Windows, считывает информацию из выбранного файла и загружает её на элемент управления предоставляющий поле для отображения текста с описанием грамматики для Flex, предварительно очистив его.

Функция *SaveTextFile* записывает информацию из элемента управления, предоставляющего поле для отображения текста для тестового файла в файл.

Функция *SaveFlexFile* записывает информацию из элемента управления, предоставляющего поле для отображения текста с описанием грамматики для Flex в файл.

Функция *createTestFile* отображает форму *CreateTestFile.cs*, в которой пользователю предлагается ввести имя для создаваемого тестового файла, после подтверждения создаётся тестовый файл с указанным именем.

Функция *createFlexFile* отображает форму *CreateFlexFile.cs*, в которой пользователю предлагается ввести имя для создаваемого файла с Flex-спецификациями, после подтверждения создаётся файл с указанным именем.

Все вышеописанные функции информирует об успешном или неудачном выполнении операции, выводя соответствующее сообщение в элемент правления предоставляющий поле для информации и ошибок.

2.2.3 Работа с лексическим анализатором

Для работы с лексическим анализатором разработан статический класс *workGenerate* расположенный в файле *workGenerate.cs*. Данный класс предоставляет следующие функции:

- *public static void run(string namefile, ref string output, string generator, RichTextBox rtbInformFlexFile);*
- *public static void resetFlags(ref int index, ref int index2, ref bool ifrun, ref int r, ref int w, ref string[] strings);*
- *public static void OpenAnalizator(RichTextBox rtbTestFile, RichTextBox rtbOutText, ref string nameGenerator, RichTextBox rtbInformTestFile);*
- *public static void drawPoints(Panel panel2, int counterPoint, List<StopPoint> points);*
- *public static void paintPoint(int x, int y, Panel panel2);*
- *public static void DelFile(string nameFile, RichTextBox rtbInformTestFile);*

- `public static void DelAllFiles(RichTextBox rtbInformTestFile)`.

Функция *run* запускает переданный ей лексический анализатор с тестовым файлом и записывает полученный результат в параметр *output* переданный по ссылке для дальнейшей работы с полученным результатом.

Функция `resetFlags` обнуляет все переданные ей флаги.

Функция `OpenAnalizator` открывает стандартный диалог выбора файла операционной системы Windows, для выбора лексического анализатора предварительно очистив элемент управления, предоставляющий поле для отображения текста работы анализатора. Также `OpenAnalizator` информирует об успешном или неудачном выполнении операции.

Функция `drawPoints` перерисовывает точки останова на поверхности, если они содержаться в коллекции `List<StopPoint>` .

Функция `paintPoint` рисует точку останова на поверхности `panel2`, по переданным ей координатам.

Функция `DelFile` удаляет переданный ей файл.

Функция `DelAllFiles` удаляет все временные файлы из папки *C:\Visualflex* по шаблону *temp**.

2.2.4 Подсветка синтаксиса

Для подсветки синтаксиса и работы с текстом Visual flex используется проект `FastColoredTextBox` [8] предоставляющий соответствующий набор классов и функций.

Класс `Style` предоставляет стиль для символов. В проекте используется шесть стилей символов:

- `BlueStyle` – для ключевых слов;
- `RedStyle` – для чисел;
- `GreenStyle` – для комментариев;

- *BrownStyle* – для текста;
- *MagentaStyle* – объявляемых шаблонов *flex*-спецификаций;
- *OrangeStyle* - для символов % и %% в файле *flex*-спецификаций.

Для их установки используется функция:

- *private void SyntaxHighlight(TextChangedEventArgs e).*

2.2.4.1 Комментарии

В Visual flex поддерживается многострочный стиль комментариев. Он должен начинаться символами /* и оканчиваться */. Все, что находится между этими символами, не воспринимается компилятором Flex. Как следует из его названия, многострочный комментарий может состоять из нескольких строк[9].

Пример комментария:

```
/* Это комментарий */
```

2.2.4.2 Ключевые слова

Ключевыми словами в проекте устанавливается следующий набор слов:

- *include*
- *yylineno*
- *noyywrap*
- *yytext*
- *yyin*
- *auto*
- *break*
- *case*
- *char*
- *const*
- *continue*
- *default*
- *do*
- *double*
- *else*
- *enum*
- *extern*
- *float*
- *for*
- *goto*

- *if*
- *int*
- *long*
- *register*
- *return*
- *short*
- *signet*
- *sizeof*
- *static*
- *struct*
- *switch*
- *typedef*
- *union*
- *unsigned*
- *void*
- *volatile*
- *while*

2.2.5 Автозавершение *flex*-спецификаций

Для автозавершения *flex*-спецификаций в Visual flex используются классы *AutocompleteItem* и *AutocompleteMenu* предоставляемые проектом *FastColoredTextBox*. В них определены необходимые функции и переменные для автозавершения.

AutocompleteMenu формирует открывающееся окно с доступными значениями для подстановки при написании спецификаций, а *AutocompleteItem* формирует подставляемые значения.

Функция *private void BuildAutocompleteMenu(AutocompleteMenu popupMenu)* главного класса *fVisualFlex* используется для построения списка доступных для подстановки значений. В ней объявляется коллекция *List<AutocompleteItem>*, куда добавляются значения объявленных регулярных выражений.

2.2.6 Выделение конфликтов

Для выделения конфликтов при написании *flex*-спецификаций используются две функции:

- `Check();`
- `checkReg()`.

Функция *Check* работает с коллекцией `List<myRegul> Reg2` , в которую записываются добавляемые регулярные выражения из файла с *flex*-спецификациями. Каждое выражение, в том числе вновь добавленное, сравнивается с каждым выражением из всей коллекции и если, оказывается больше одного совпадения, функция выдает сообщение о том, что выражение уже объявлено и выделяет выявленное выражение.

Функция `checkReg` считывает данные из области правил трансляции файла с *flex*-спецификациями, пропуская пустые строки и комментарии. Каждый шаблон, заключённый в фигурные скобки функция проверяет с наличием его в коллекции `List<myRegul> nameReg` и если его в ней не оказывается, выдает сообщение что данное выражение не объявлено.

2.2.7 Пошаговое выполнение

Пошаговое выполнение на тестовом файле реализовано функцией:

- `private void stepRun_Click(object sender, EventArgs e);`

Функция `stepRun_Click` является обработчиком события для кнопки «Шаг вперёд» в ней проверяется, выбран ли лексический анализатор, и если он выбран, то запускается с тестовым фалом, результат работы синтаксического анализатора помещается в строковую переменную `output`. На следующем этапе содержание строки `output` разбивается на подстроки по управляющему символу новой строки `'\n'` и помещается в массив строк

string[] outStrings. Затем при каждом следующем нажатии кнопки «Шаг вперёд» выводится последовательно строки из массива строк *string[] outStrings*, при этом в функции реализована подсветка выводимой на текущем шаге строки и соответствующего ей символу из тестового файла.

2.2.8 Реализация точек останова

В Visual flex реализован класс *StopPoint* размещённый в файле *StopPoint.cs* предоставляющий координаты и номер для точки останова. Для установки и снятия точки останова используется функция:

- *private void panel2_MouseClick(object sender, MouseEventArgs e)*

Функция *panel2_MouseClick* является событием на нажатие кнопки мыши на элементе *panel2*. Элемент *panel2* предоставляет поле, на котором отображаются точки останова. При нажатии на левую кнопку мыши над элементом *panel2* добавляется точка останова, при этом в папке *C:\Visualflex* создаётся временный файл с именем *tempX*, где *X* - номер точки останова равный строке в тестовом файле напротив которого поставлена точка. В этот файл копируется содержание тестового файла до строки, напротив которой установлена точка останова. При этом координата и номер точки останова

добавляется в коллекцию *List<StopPoint> points*. При нажатии на правую кнопку мыши над элементом *panel2* функция *panel2_MouseClick* функция проверяет, содержится ли в коллекции *List<StopPoint> points* точка с такими координатами и удаляет ее, если находит.

2.3 Перечень выводимых сообщений и ошибок

Полный перечень выводимых сообщений и ошибок указан в таблице 1.

Таблица 1 – Перечень выводимых сообщений и ошибок

Формат сообщения	Описание
Анализатор «имя анализатора» выбран.	Выводится при выборе анализатора в пункте меню.
Анализатор не выбран!	Ошибка выводимая при попытке работы с тестовым файлом без выбранного предварительнл анализатора
Анализатор сгенерирован.	Выводится при успешной генерации анализатора компилятором gcc.
Выражение «имя выражения» - не объявлено	Выводится при написании выражения в области правил транляции файла *flex*-спецификаций если выражение не было обявлено в области объявлений.
Выражение «имя выражения»- уже существует	Выводится при написании выражения в области объявлений файла flex-спецификаций если выражение уже объявлено.
Достигнута точка останова.	Выводится про достижении точки останова при исполнении тестового файла.
Здесь нельзя поставить точку!	Выводится при попытке поставить точку останова если тестовый файл не открыт или в нем отсутствует страка напротив которой ставится точка.

Продолжение таблицы 1

Формат сообщения	Описание
Не удалось сгенерировать файл 'lex.yy.c'!	Выводится при неудачной попытке сгенерировать файл компилятором Flex.
Не удалось сгенерировать файл!	Выводится при неудачной попытке сгенерировать файл Flex или gcc компиляторами.
Не удалось скопировать файл!	Выводится при неудачной попытке скопировать файл.
Не удалось удалить файл!	Выводится при неудачной попытке удалить файл.
Не удалось удалить файлы!	Выводится при неудачной попвтке удалить временные файлы
Не указано имя файла!	Выводится при попытке создания файла без указания имени.
Ошибка: Не могу прочитать файл.	Ошибка, выводимая при неудачной попытке чтения файла.
Файл «имя файла» открыт.	Выводится при успешном чтении выбранного файла.
Файл «имя файла» создан.	Выводится при успешном создании файла.
Файл «имя файла» сохранён!	Выводится при успешном сохранении файла.
Файл 'lex.yy.c' сгенерирован!	Выводится при успешной компиляции входного файла в программу на Си компилятором Flex.
Файл уже существует!	Выводится при попытке созадть файл с и менем который уже существует.

3 Глава. Руководство пользователя

3.1 Платформа

Разработанный визуальный редактор-генератор лексических анализаторов Visual flex является .Net-приложением написанный на языке C#. Взаимодействие с пользователем осуществляется с помощью оконного интерфейса TDI (Многодокументный интерфейс с вкладками — разновидность графического интерфейса пользователя, в котором каждый документ отображается на отдельной вкладке общего окна), в котором пользователь производит необходимые действия для работы с программой и получает необходимый результат.

3.1.1 Обоснование выбора платформы

Платформа .Net Framework для реализации Visual flex была выбрана по следующим причинам[10]:

- Объектно-ориентированный подход;
- Мощный инструментарий;
- Разделение кода;
- Поддержка языков высокого уровня;
- Visual Studio

Платформа .NET изначально строится на принципах объектно-ориентированного программирования. Поставляемая вместе со средой библиотека базовых классов обладает достаточным функционалом для решения задач практически любой сложности. В .NET способ разделения кода между приложениями значительно отличается от предшествующих реализаций за счет использования сборок. Сборки обладают формальными средствами для управления версиями и допускают одновременное существование рядом

нескольких различных версий сборок. Это свойство благоприятно сказывается на удобстве использования, быстроте написания и читабельности кода, что крайне важно для последующей поддержки программы. Среда разработки Visual Studio, поставляемая вместе с .NET, предоставляет необходимый инструментарий для эффективного и быстрого создания приложений с графическим интерфейсом.

3.1.2 Обоснование выбора языка программирования

- Язык программирования C# претендует на подлинную объектную ориентированность.
- Язык программирования C# призван реализовать компонентно-ориентированный подход к программированию, который способствует меньшей машинно-архитектурной зависимости результирующего программного кода, большей гибкости, переносимости и легкости повторного использования программ.
- Принципиально важным отличием от предшественников является изначальная ориентация на безопасность кода.
- Расширенная поддержка событийно-ориентированного программирования.
- Язык программирования C# является «родным» для создания приложений в среде Microsoft .NET, поскольку наиболее тесно и эффективно интегрирован с ней.

3.2 Подготовка к работе Visual flex

Поскольку для разработки Visual flex была выбрана платформа .Net Framework, а Flex и gcc являются продуктами, разработанными в системах на базе пакетов GNU, в проекте были использованы их аналоги WinFlex и

MinGW портирующие соответствующие продукты под Windows.

Для начала работы с Visual flex необходимо выполнить следующие шаги:

- Создать папку Visualflex в корне даска C:\;
- Добавить в папку Visualflex проект;
- Скомпилировать проект;
- Установить MinGW;
- Компилятору gcc.exe разрешить выполнятся от имени администратора;
- Переменной среды path операционной системы добавить путь к MinGw.

Проект с исходным кодом программы Visual flex необходимо поместить в папку Visualflex на диске C:\ поскольку все файлы, создаваемые или используемые программой, сохраняются в этой папке и обращение к сторонним ресурсам из приложения настроено через эту папку. После компиляции в папке C:\Visualflex\Visualflex\Visualflex\bin\Debug появится запускаемый файл Visualflex.exe. Поскольку Visual flex для генерации лексического анализатора использует компилятор gcc его необходимо установить на компьютер пользователя. Это можно сделать, скачав и установив MinGW [11] - компилятор, родной программный порт GNU Compiler Collection (GCC) под Windows, вместе с набором свободно распространяемых библиотек импорта и заголовочных файлов для Windows API. Разрешение выполнятся от имени администратора позволит компилятору gcc запускаться из Visual flex а добавление в переменную среды path операционной системы Windows пути к MinGw позволит gcc использовать идущие вместе с ним библиотеки.

3.3 Запуск приложения

Программа «Visual flex» представляет собой исполняемый файл «Visualflex.exe».

Для запуска программы необходимо запустить файл «Visualflex.exe».

Откроется главное окно программы рисунок 6. Главное окно разбито на две области. Первая область представляет собой элемент управления с двумя вкладками «Flex спецификации» и «Тестовый файл». Вторая область элемент управления, куда выводятся информация и ошибки выполнения при работе с программой.

Вкладка «Flex спецификации» предназначена для работы с файлом Flex-спецификаций и предоставляет следующий набор кнопок:

- Создать Flex файл;
- Открыть Flex файл;
- Сохранить Flex файл;
- Печать;
- Вырезать;
- Копировать;
- Вставка;
- Сгенерировать файл для анализатора;
- Сгенерировать анализатор.

Вкладка «Тестовый файл» предназначена для работы с тестовым файлом и предоставляет следующий набор кнопок:

- Создать тестовый файл;
- Открыть тестовый файл;
- Сохранить тестовый файл;
- Выбрать анализатор;
- Выполнить;
- Шаг вперед;

- Удалить все точки останова;
- Обновить.

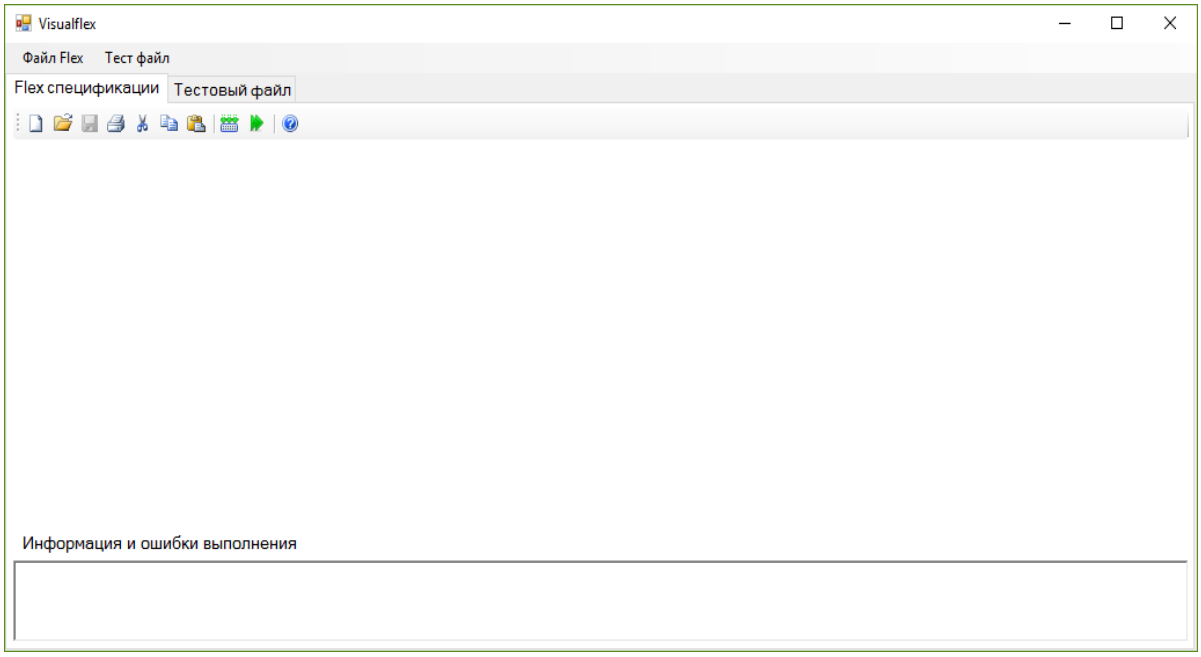

Рисунок 6 – Главное окно Visual flex

3.4 Работа с файлом Flex-спецификаций

Под работой с файлом Flex-спецификаций подразумевается его создание, открытие, редактирование и генерация по нему синтаксического анализатора.

Для того чтобы начать работать с файлом Flex-спецификаций его нужно либо создать либо открыть существующий. При создании файла нужно задать его имя, а система автоматически сохранит его в папке *C:\Visualflex* с расширением *.l*. При этом откроется подготовленный шаблон, в который нужно внести только необходимые объявления и инструкции Flex рисунок 7. Полный листинг шаблона приведен в приложении А.

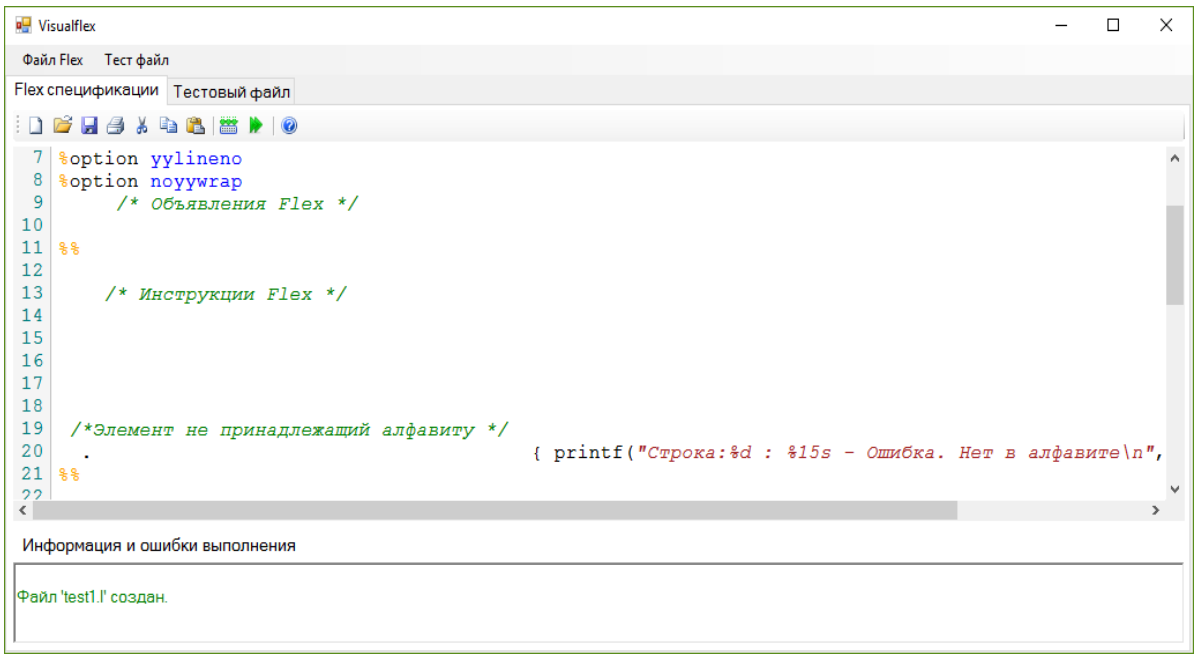

Рисунок 7 – Создание файла Flex-спецификаций

В таблице 2 перечислены специальные символы, использующиеся в регулярных выражениях Flex.

Вторым этапом работы с файлом Flex-спецификаций является генерация по нему лексического анализатора с помощью компиляторов Flex и gcc. Нажав кнопку «Сгенерировать файл для анализатора» компилятор Flex генерирует файл lex.yy.c в папку C:\Visualflex. Нажав кнопку «Сгенерировать анализатор» компилятор gcc генерирует лексический анализатор в той же папке с именем файла Flex-спецификаций.

Таблица 2 - Специальные символы, использующиеся в регулярных выражениях Flex.

Символ выражения	Значение
.	Соответствует любому символу, кроме \n
[ ]	Класс символов, соответствующий любому из символов, описанных внутри скобок. Знак '-' указывает на диапазон символов. Например, [0-9] означает то же самое, что и [0123456789], [a-z] – любая прописная буква латинского алфавита, [A-z] – все заглавные и прописные буквы латинского алфавита, а также 6 знаков пунктуации, находящихся между Z и a в таблице ASCII. Если символ '-' или ']' указан в качестве первого символа после открывающейся квадратной скобки, значит он включается в описываемый класс символов. Управляющие (escape) последовательности языка Си также могут указываться внутри квадратных скобок, например, \t.
^	Внутри квадратных скобок используется как отрицание, например, регулярное выражение [^\t\n] соответствует любой последовательности символов, не содержащей табуляций и переводов строки. Если просто используется в начале шаблона, то означает начало строки.
$	При использовании в конце регулярного выражения означает конец строки.

Продолжение таблицы 2

Символ выражения	Значение
{}	Если в фигурных скобках указаны два числа, то они интерпретируются как минимальное и максимальное количество повторений шаблона, предшествующего скобкам. Например, A{1,3} соответствует повторению буквы A от одного до трёх раз, а 0{5} – 00000. Если внутри скобок находится имя регулярного определения, то это просто обращение к данному определению по его имени.
\	Используется в escape-последовательностях языка Си и для задания метасимволов, например, * – символ ‘*’ в отличие от *
*	Повторение регулярного выражения, указанного до *, 0 или более раз. Например, [\t]* соответствует регулярному выражению для пробелов и/или табуляций, отсутствующих или повторяющихся несколько раз.
+	Повторение регулярного выражения, указанного до +, один или более раз. Например, [0-9]+ соответствует строкам 1, 111 или 123456
?	Соответствует повторению регулярного выражения, указанного до ?, 0 или 1 раз. Например, -?[0-9]+ соответствует знаковым числам с необязательным минусом перед числом.
|	Оператор «или». Например, *true*|*false* соответствует любой из двух строк.

Продолжение таблицы 2

()	Используются для группировки нескольких регулярных выражений в одно. Например, a(bc|de) соответствует входным последовательностям: abc или ade.
/	Так называемый присоединенный контекст. Например, регулярное выражение 0/1 соответствует 0 во входной строке 01, но не соответствует ничему в строках 0 или 02.
“ ”	Любое символы в кавычках рассматриваются как строка символов. Метасимволы, такие как *, теряют своё значение и интерпретируются как два символа: \ и *.

3.5 Работа с тестовым файлом

Под работой с тестовым файлом подразумевается его создание, открытие, редактирование и запуск его в лексическом анализаторе. Вкладка «Тестовый файл», предназначена для работы с тестовым файлом, для этого она разделена на две области, одна для отображения тестового файла, вторая для вывода результата после запуска тестового файла в лексическом анализаторе рисунок 8.

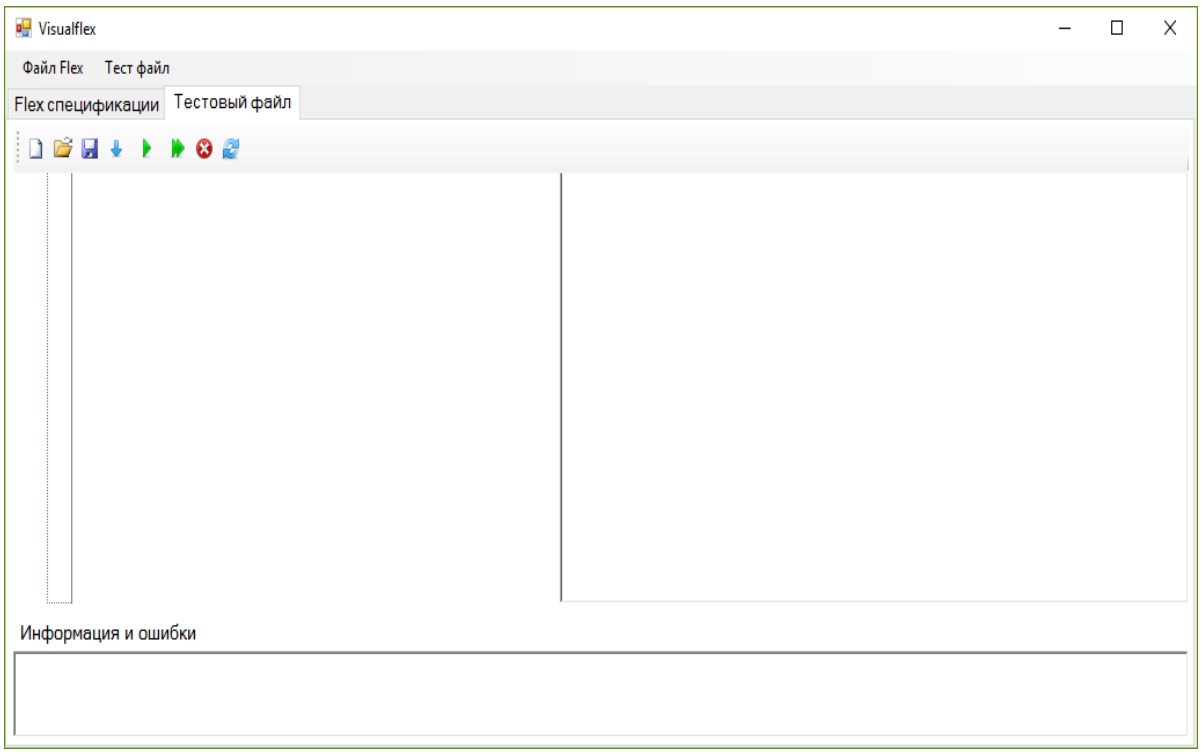

Рисунок 8 – Работа с тестовым файлом

Тестовый файл можно либо создать, либо открыть существующий. При создании файла нужно задать его имя, а система автоматически сохранит его в папке C:\Visualflex с расширением .txt.

Что бы провести лексический анализ тестового файла, необходимо выбрать лексический анализатор, для этого предназначена кнопка «Выбрать анализатор». После его выбора с тестовым файлом можно работать в трех режимах:

- Проанализировать весь тестовый файл;
- Проанализировать тестовый файл пошагово;
- Проанализировать до точки останова.

Для анализа всего файла нужно использовать кнопку «Выполнить». Для анализа тестового файла пошагово нужно использовать кнопку «Шаг вперёд». Для анализа тестового файла с точками останова, необходимо предварительно установить точки.

Точки устанавливаются в специальной области слева от текста тестового файла напротив номера строки, где необходима остановка анализатора. После установки точек воспользоваться кнопкой «Выполнить».

4 Глава. Эргономика

Визуальный редактор генератор лексических анализаторов Visual flex представляет собой .Net приложение с графическим оконным интерфейсом, запускаемое через интерфейс операционной системы Windows. Особенности поведения окон следующие:

- большинство окон может свободно перемещаться по экрану, перемещение окон производится при захвате заголовка окна, однако при необходимости можно задать для перемещения также произвольную область, либо всё окно;
- можно управлять размером окна, обычно тремя способами — через меню окна (в этом случае изменение размеров возможно с клавиатуры), путём перемещения границ окна либо специальной области в правом нижнем углу окна;
- в ОС семейства Windows большинство оконных приложений может использовать «полноэкранный режим». При нажатии на кнопку «развернуть», окно занимает всю доступную площадь экрана, кроме панели задач;
- в ОС семейства Windows большинство оконных приложений присутствует кнопка «свернуть». При нажатии на эту кнопку окно приложения скрывается, но само приложение не прекращает работу;
- для главного окна приложения закрытие окна обычно предполагает выход из приложения.

Организацию окон в программах реализуют несколькими способами:

- однооконный режим (SDI)
- многооконный режим (MDI, TDI)
- псевдо многооконный режим (PMDI)

Для данного визуального редактора генератора лексических анализаторов был выбран многодокументный оконный интерфейс с вкладками TDI по следующим причинам:

- логически отделяются окна документов от окон других программ;
- расходуется меньше памяти;
- панели управления разных окон находятся в одном и том же месте;
- возможность использовать общие интерфейсные элементы, относящиеся ко всем документам.

Редактор генератор запускается двойным щелчком по файлу «Visualflex.exe».

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была рассмотрена теория лексического анализа и существующие в ней инструментальные средства.

Исходя из полученных данных и поставленных задач, был реализован визуальный редактор-генератор лексических анализаторов Visual flex. Данный редактор-генератор является .Net приложением с многодокументным оконным интерфейсом, позволяющий писать *flex*-спецификации с подсветкой синтаксиса и выделением конфликтов по мере написания регулярного выражения, автоматически генерировать лексический анализатор по исходному тексту, пошагово выполнятся на тестовом входном файле и использовать точки останова на нём.

Так как разработанный редактор-генератор имеет образовательную значимость, было дополнено соответствующее методическое пособие для студентов, изучающих компиляторные курсы.

В перспективе в Visual flex можно добавить поддержку генерации лексических анализаторов на других языках, например C++ . Не исключается также возможность расширения и улучшения функциональных возможностей.

Примеры работы с редактором-генератором рассмотрены в приложении к данной пояснительной записке.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Серебряков В. А. Основы конструирования компиляторов : учебное пособие / В. А. Серебряков, М. П. Галочкин. – Москва : издательство «Едиториал УРСС», 1999. – 193 с.

2 Свердлов С. З. Языки программирования и методы трансляции : учебное пособие / С. З. Свердлов. – СПб.: издательство «Питер», 2007. – 638 с.

3 Молдованова О.В. Языки программирования и методы трансляции.: Учебное пособие / О. В. Молдованова. – Новосибирск/СибГУТИ, 2012. – 134с.

4 Компиляторы: принципы, технологии и инструментарий, 2-е изд. : учебное пособие / Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульфман. Москва : издательский дом «Вильямс», 2008 – 1184 с.

5 flex: The Fast Lexical Analyzer [Электронный ресурс] // Генератор лексических анализаторов. – Режим доступа: http://flex.sourceforge.net/. (дата обращения: 02.04.2016 г.)

6 VisualBNF [Электронный ресурс] // Генератор синтаксических анализаторов. – Режим доступа: http://www.intralogic.eu/VisualBNF/ (дата обращения: 09.05.2016 г.)

7 VisualLangLab - A Visual Parser-Generator IDE [Электронный ресурс] // Генератор синтаксических анализаторов. – Режим доступа: https://vll.java.net/ (дата обращения: 09.05.2016 г.)

8 Pavel Torgashov. Fast Colored TextBox for Syntax Highlighting [Электронный ресурс] // Подсветка синтаксиса . – Режим доступа: http://www.codeproject.com/Articles/161871/Fast-Colored-TextBox-for-syntax-highlighting (дата обращения: 20.05.2016 г.)

9 Шилдт Г. C# 4.0: полное руководство.: учебное пособие / Г. Шилдт. – Москва : издательский дом «Вильямс», 2012 – 1056 с.

10 Рихтер Д. CLR via C#. Программирование на платформе Microsoft .Net Framework 4.5 на языке C#. : Учебное пособие / Д. Рихтер. – СПб.: Питер, 2016. – 896с.

11 MinGW-w64 - for 32 and 64 bit Windows [Электронный ресурс] // Компилятор . - Режим доступа: https://sourceforge.net/projects/mingw-w64/ (дата обращения: 07.03.2016 г.)

ПРИЛОЖЕНИЕ А

Листинг шаблона при создании файла flex-спецификаций

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
%}

%option yylineno
%option noyywrap
     /* Объявления Flex */

%%

    /* Инструкции Flex */


 /*Элемент не принадлежащий алфавиту */
  .                                          {
printf("Строка:%d : %15s - Ошибка. Нет в алфавите\n",
yylineno, yytext);}
%%
int main(int argc, char* argv[])
{
  FILE* infile;
  /* Проверка на правильность запуска */
  if(argc < 2)
  {
    printf("Мало параметров в командной строке.\n");
    return 1;
  }
  else if(argc > 2)
  {
    printf("Много  параметров в командной строке.\n");
    return 1;
  }
```

```
    infile = fopen(argv[1], "r");
    if (NULL == infile)
    {
       printf("Не могу открыть файл.\n");
    }
    yyin = infile;
    yylex();

    fclose(infile);

    return 0;
}
```

ПРИЛОЖЕНИЕ Б

Листинг тестового файла test1.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
%}

%option yylineno
%option noyywrap

      /* Объявления Flex */

digit         0|[1-9][1-9]*
intconst      [+\-]?{digit}+
cicl          for|while|do|to
stringch      [^"\n]
ws            [ \t\n]
punctuator    ";"|","|"."
identifier    [a-zA-Z][a-zA-Z_]*
assignment    ":="
comparison    "<"|">"|"="|"=="|"<>"|"!="|">="|"<="
brackets      "("|")"|"{"|"}"
operator      "+"|"-"|"*"|"/"
error         .

%%

       /* Инструкции Flex */

      /* Пропускаем все пробельные символы. */
{ws}            {}
     /* Целочисленная константа в*/
{intconst}*                          {
printf("Строка:%d : %15s%s\n", yylineno, yytext, " -
Целочисленная константа.");}

       /*Символ точки с запятой */
```

```
{punctuator}                            {
printf("Строка:%d : %15s - Знак пунктуации.\n",
yylineno, yytext);}

        /*Знак операций*/
{operator}                              {
printf("Строка:%d : %15s - Знак операций.\n", yylineno,
yytext);}

        /*Операторы цикла.    */
{cicl}                                  {
printf("Строка:%d : %15s - Оператор цикла.\n",
yylineno, yytext);}

       /* Идентификатор.   */
{identifier}                            {
printf("Строка:%d : %15s - Идентификатор.\n", yylineno,
yytext);}

        /* Знак присваивания.*/

{assignment}                            {
printf("Строка:%d : %15s - Знак присваивания.\n",
yylineno, yytext);}

       /* Знаки операций сравнения */
{comparison}                            {
printf("Строка:%d : %15s - Знак операции сравнения.\n",
yylineno, yytext);}

        /* Круглые скобки*/
{brackets}                              {
printf("Строка:%d : %15s - Знак скобки.\n", yylineno,
yytext);}

        /*Элемент не принадлежащий алфавиту */
{error}                                 {
printf("Строка:%d : %15s - Ошибка. Нет в алфавите\n",
yylineno, yytext);}
%%
```

```
int main(int argc, char* argv[])
{
  FILE* infile;
  /* Проверка на правильность запуска */
  if(argc < 2)
  {
    printf("Мало параметров в командной строке.\n");
    return 1;
  }
  else if(argc > 2)
  {
    printf("Много  параметров в командной строке.\n");
    return 1;
  }

  infile = fopen(argv[1], "r");
  if (NULL == infile)
  {
     printf("Не могу открыть файл.\n");
  }
  yyin = infile;
  yylex();

  fclose(infile);

  return 0;
}
```

ПРИЛОЖЕНИЕ В

Примеры работы редактора-генератора

В1. Подсветка синтаксиса

Visual flex предоставляет подсветку для ключевых слов, коментариев, текста, цифр и добавляемых Flex-спецификации рисунок В.1

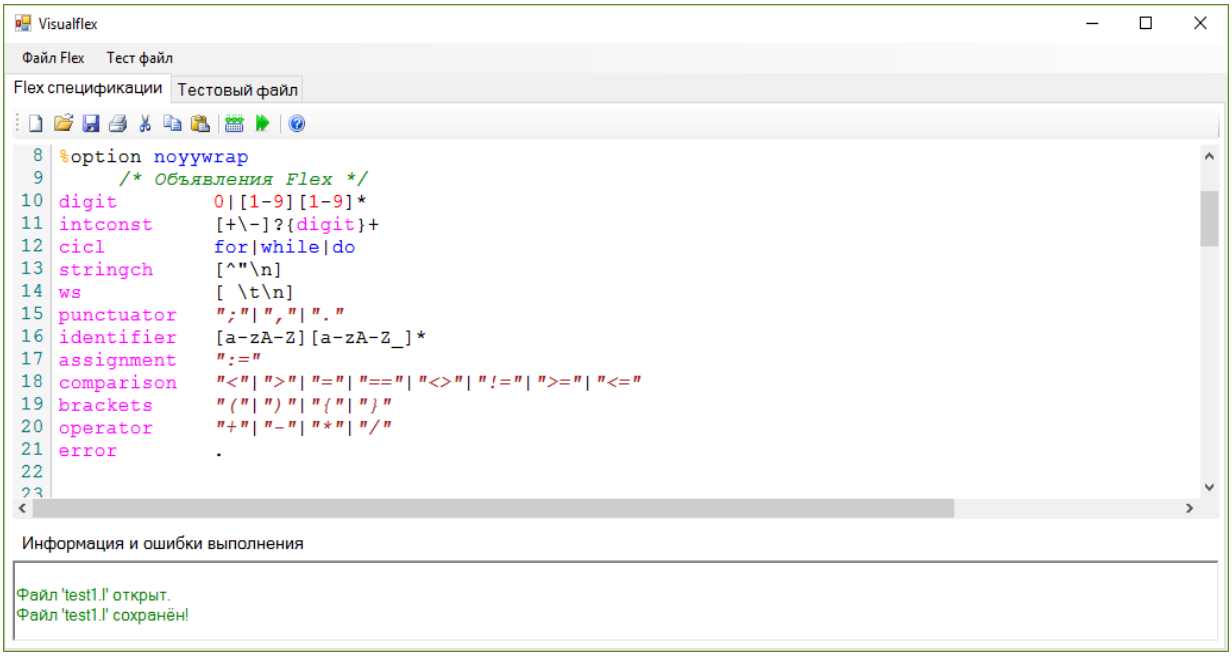

Рисунок В.1 – Подсветка синтаксиса

В2. Выделение конфликта в регулярном выражении

При написании регулярного выражения при повторном введении выражения программа выделит конфликт и выведет сообщение об ожибке рисунок В.2

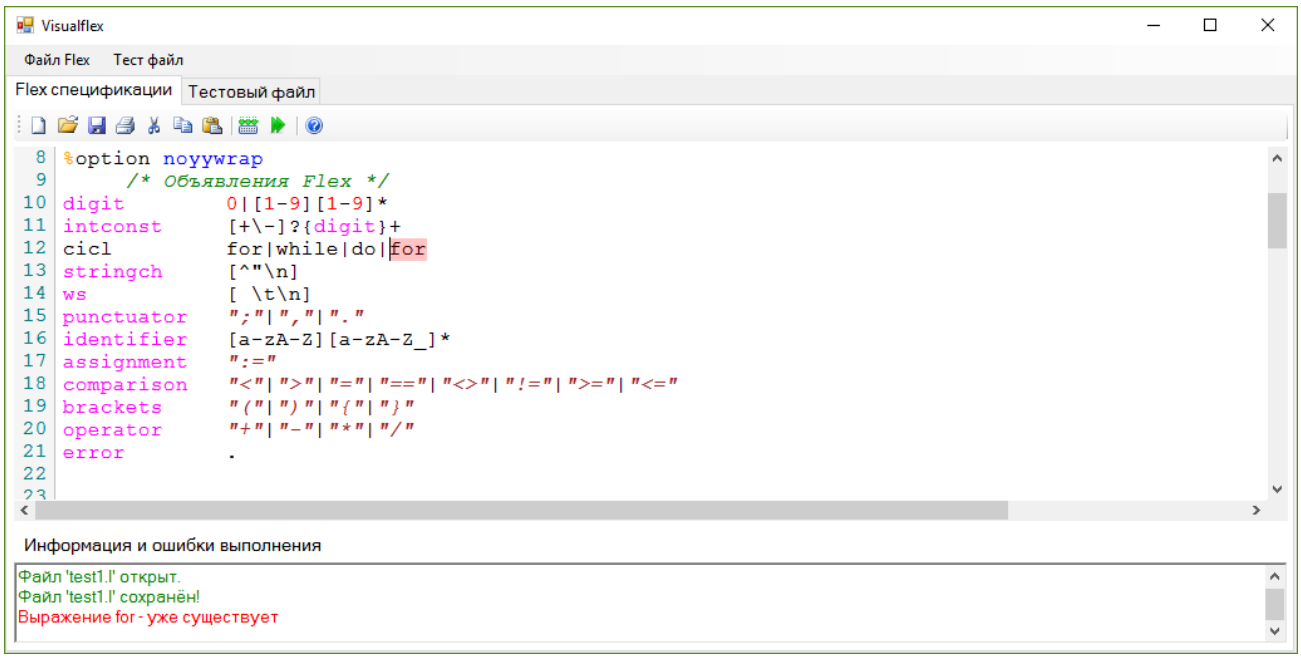

Рисунок В.2 – Выделение повторно вводимого выражения

При добавлении инструкции в правилах трансляции, которая не объявлена в блоке объявлений Flex, программа выделит добавляемую инструкцию и выведит сообщение об ошибке рисунок В.3.

Рисунок В.3 – Выделение необъявленной инструкции

В3. Автоподстановка объявленных выражений

При добавлении инструкции в правилах трансляции, которая была объявлена в блоке объявлений Flex, программа предложит варианты объявленных инструкции из которых пользователь может выбрать подходящий вариант рисунок В4.

Рисунок В.4 – Автоподстановка объявленных выражений

В4. Генерация лексического анализатора

Генерация лексического анализатора происходит в два этапа, сначала генерируется файл *lex.yy.c*, затем по этому файлу сам лексический анализатор рисунок В.5.

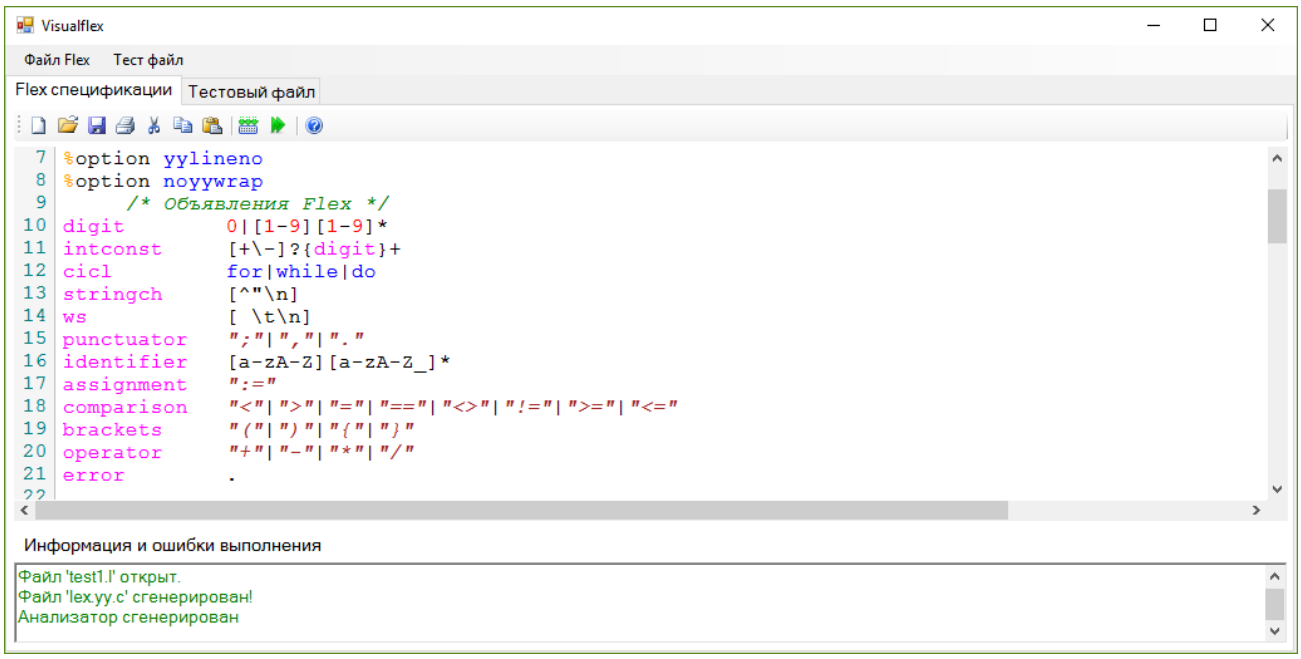

Рисунок В.5 – Генерация лексического анализатора

В5. Работа с лексическим анализатором

Для работы с лексическим анализатором необходимо создать либо открыть тестовый файл и выбрать анализатор.

При необходимости проанализировать весь тестовый файл необходимо нажать кнопку «Выполнить» программа выдаст результат рисунок В.6.

При необходимости проанализировать тестовый файл до определённого места можно использовать точки останова рисунок В.7. Количество точек и их последовательность не ограничивается.

При необходимости проанализировать тестовый файл пошагово нужно воспользоваться кнопкой «Шаг вперёд» рисунок В.8.

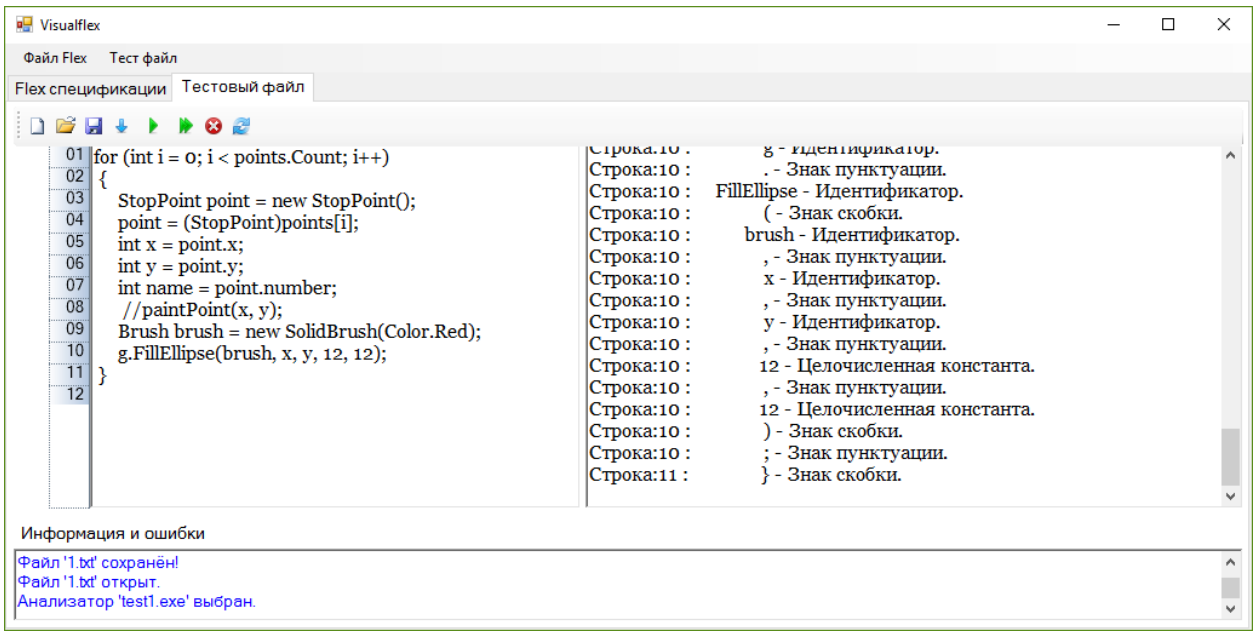

Рисунок В.6 – Результат анализа тестового файла

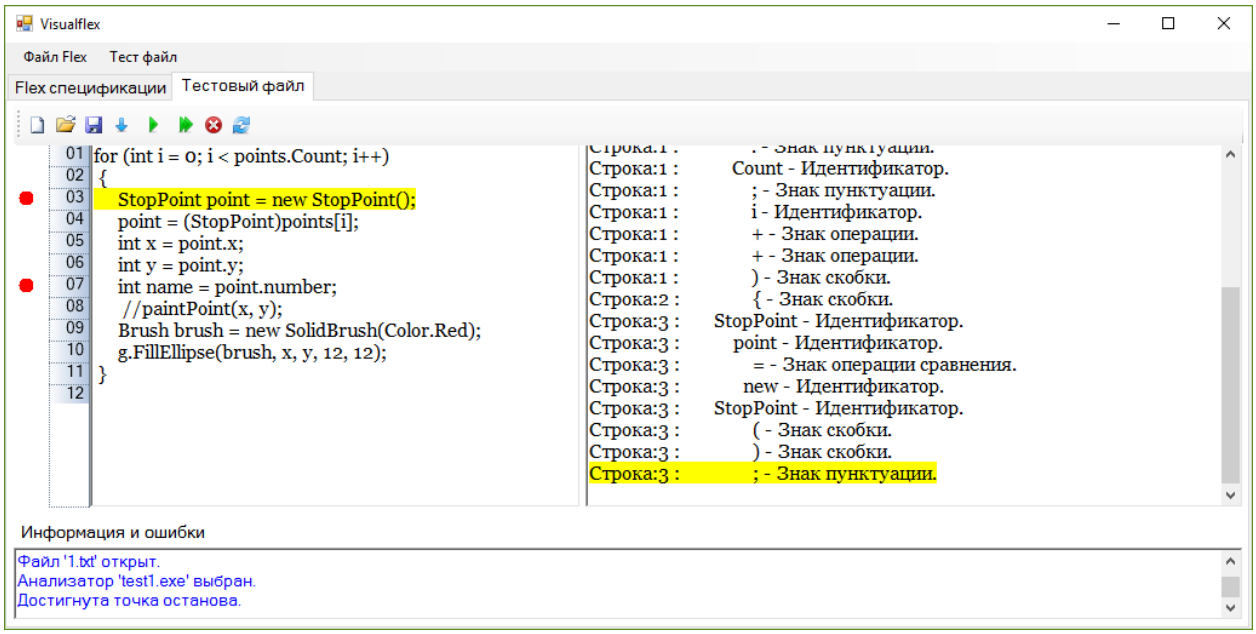

Рисунок В.7 – Результат анализа тестового файла с использованием точек останова

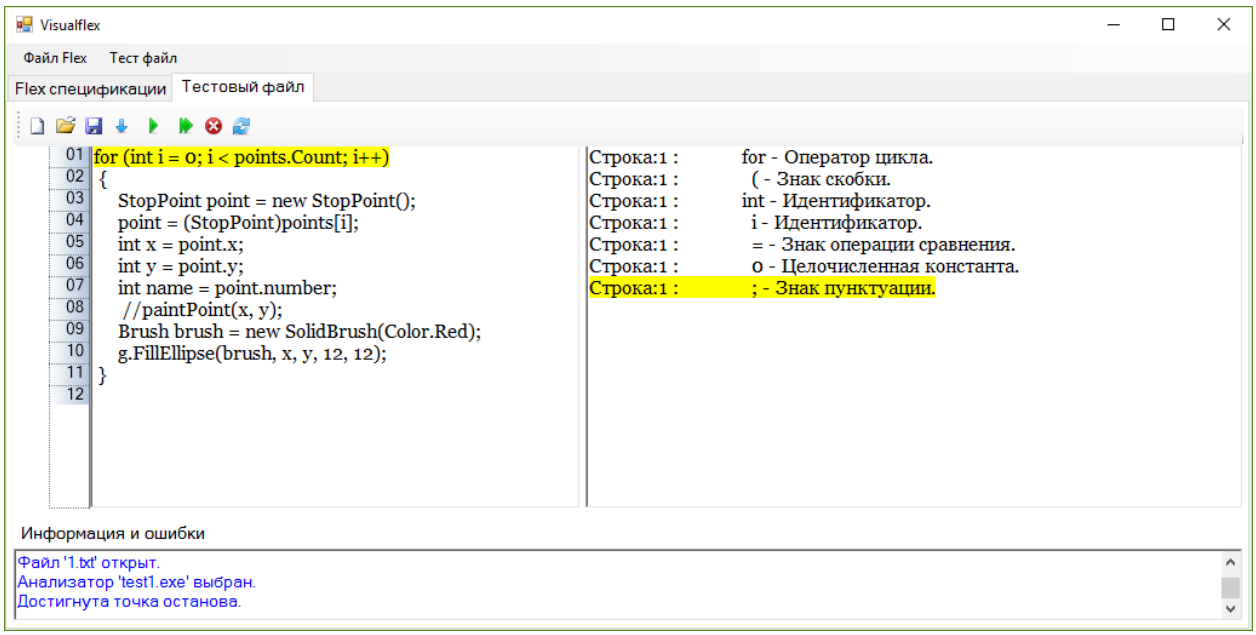

Рисунок В.8 – Результат пошагового анализа тестового