ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ»

Лексический анализ транслируемой программы

Методические указания

Волгоград
2009

УДК 004.4'412

Р е ц е н з е н т

д-р техн. наук, профессор *А. В. Заболеева-Зотова*

Издается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Лексический анализ транслируемой программы : метод. указания / сост. О. А. Сычев ; ВолгГТУ. – Волгоград, 2009. – 16 с.

Изложены основы синтаксиса регулярных выражений и их использования при проведении лексического анализа. Приведены материалы по генератору лексических анализаторов flex, а также рекомендации по выполнению наиболее сложных моментов лабораторных работ. Предлагается список контрольных вопросов.

Предназначены для студентов специальности 230105.65 «Программное обеспечение вычислительной техники и автоматизированных систем» по дисциплине «Теория языков программирования и методы трансляции».

У ч е б н о е и з д а н и е

Олег Александрович **Сычев**

ЛЕКСИЧЕСКИЙ АНАЛИЗ ТРАНСЛИРУЕМОЙ ПРОГРАММЫ

Методические указания

Темплан выпуска электронных изданий 2009 г., поз. № 29.

На магнитоносителе. Уч.-изд. л. 0,70.
Подписано на «Выпуск в свет» 15.06.2009. Заказ № 10.

Волгоградский государственный технический университет.
400131, г. Волгоград, пр. им. В. И. Ленина, 28.

1 Лабораторная работа №1. Регулярные выражения

1.1 Цель работы

Целью работы является ознакомление студента с назначением и синтаксисом регулярных выражений, приобретение навыков их составления и анализа с использованием современных программных средств.

1.2 Задание

Необходимо составить регулярное выражение для совпадения с любой строкой из задания, данного преподавателем.

1.3 Порядок выполнения

1. Изучить теоретический материал
2. Получить задание на составление регулярного выражения у преподавателя
3. Составить выражение, используя для его проверки RegexBuilder
4. Показать составленное выражение преподавателю
5. Отчитать лабораторную работу

1.4 Синтаксис регулярных выражений

Регулярные выражения предназначены для описания шаблонов, поиск совпадений с которыми будет производится в исходном тексте.

Обычные символы в регулярных выражениях означают буквальное совпадение с написанным текстом. Исключением являются служебные символы, например `[ ] \ ^ $ . | ? * + ( ) { } < >` Если в регулярном выражении требуется вставить совпадение с одним из служебных символов, его следует экранировать, поместив перед ним символ обратной дробной черты(\), например `\+ \. \\` (для обозначения совпадения с символом обратной дробной черты). Экранирование неслужебных символов, не являющихся буквами, игнорируется, поэтому если вы сомневаетесь, является ли символ служебным, следует для надежности экранировать его. При экранировании символов `a, b, f, n, t, v` или `r,` они будут соответствовать служебным последовательностям в строковых константах языка C. Например, `\n` — перенос строки, `\t` — символ табуляции.

Символьный класс используется в случае, если в каком-либо месте шаблона может присутствовать любой из определенного множества символов. Символьный класс описывается в квадратных скобках перечислением всех входящих символов. Например, символьному классу `[123]` соответствуют строки «1», «2» и «3». В символьном классе можно указать

диапазон символов, используя символ -. Например классу `[0-5]`, соответствует любой символ от «0» до «5»; `[a-z]` — любая строчная латинская буква. Можно использовать в одном классе и диапазоны, и конкретные значения. Например, класс `[0-3aA-Z]` — даст совпадение с любой цифрой от 0 до 3, буквой a, либо любой заглавной латинской буквой. Примеры некорректных диапазонов: `[a-Z]` (надо указывать символы одного регистра, в таких случаях необходимо писать `[a-zA-Z])`, `[0-a]` (не может быть диапазона между буквой и цифрой), `[%-&]` (диапазоны нецифровых и небуквенных символов недопустимы). Если требуется указать совпадение с любыми символами кроме определенных, то сразу за открывающей квадратной скобкой ставится знак отрицания `^`. Например, выражению `[^1-3]` соответствует любая строка из одного символа, кроме «1», «2» и «3». Внутри квадратных скобок обычные служебные символы теряют свои значения и не требуют экранирования; служебными считается лишь символы `]` (конец класса), `^` (отрицание, только если стоит сразу за открывающей скобкой), − (диапазон символов) и `\` (символ экранирования).

Вне символьного класса может использовать служебный символ **.** (точка). Он соответствует любому символу, кроме символа перевода строки `(\n)`. В отличие от точки, символьный класс с отрицанием будет включать в себя перевод строки.

Для указания того, что символы при совпадении должны следовать друг за другом, используется конкатенация. Символы (или символьные классы) в шаблоне указываются непосредственно друг за другом. Например, выражению `12` соответствует строка «12», выражению `1[0-9]` соответствует любое двузначное число, начинающееся с единицы.

Для задания вариантов строк используется перечисление, варианты записываются через вертикальную черту `|`. Например выражению `for|while|if`, соответствуют строки «for», «if» и «while». В случае одиночных символов эффективнее использовать символьный класс. Если варианты составляют только часть совпадения, то их следует заключить в круглые скобки, при этом один из вариантов может быть пустым, например `(do |)while` будет соответствовать строкам «do while» или «while».

Если необходимый нам символ может встречаться в строке определенное количество раз, то используются служебные символы-квантификаторы. Квантификатор + означает, что предшествующий ему символ (символьный класс, выражение в скобках) может встречаться любое количество раз, но как минимум один; квантификатор `*` - ни разу или любое количество раз, `?` - ни разу или один раз. Например, выражение `ab?` означает «a» или «ab», `a+b` означает «ab», «aab» и т.д. с любым количеством букв «a»; `a*b` — «b», «ab», «aab» и т.д.

Если необходимо более точно указать количество совпадений, то применяется квантификатор в виде фигурных скобок: `a{3}` означает ровно три буквы «a», `a{1,3}` — любое количество от одного до трех, `a{,3}` — любое количество от нуля до трех, `a{3,}` - не менее трех букв «a».

При применении квантификаторов к символьному классу или выражению в скобках с применением вертикальной черты следует помнить, что варианты при каждой подстановке могут быть различными, т.е. `[ab]{2}` даст совпадение не только с «aa» и «bb», но и с «ab» и «ba».

Квантификаторы по умолчанию являются «жадными», то есть при наличии нескольких возможных совпадений будет выбрано наибольшее. Рассмотрим регулярное выражение `\(.*\)`. Оно должно находить любые строки, заключенные в скобки. Если строка для поиска будет такой «(12345)2341», то совпадение будет «(12345)». Но если строка будет такой «0(12345)124(32521)7», то мы получим не два совпадения, а одно «(12345)124(32521)», поскольку квантификатор `*` будет искать совпадение до последней закрывающей скобки, даже встретив первую. Для решения этой проблемы существует 2 способа:

– при поиске символов внутри скобок указать любые символы кроме закрывающей скобки: `\([^)\n]*\)`

– использовать «нежадные» квантификаторы, которые ищут минимальное по длине совпадение, указав ? сразу после квантификатора `(*)`: `\(.*?\)`

В этом случае будут найдены обе строки, что и требовалось.

Если символ `^` указать первым в регулярном выражении, то он будет означать, что совпадение должно начинаться с начала строки. Например, выражение `^a+`, в строке «aaabbb» выделит «aaa», а в строке «bbbaaa» не найдет ни одного вхождения. Аналогично используется символ `$` для обозначения конца строки. Например, выражение `b+$`, в строке «aaabbb» выделит «bbb», а в строке «bbbaaa» не найдет ни одного вхождения. В других позициях эти символы не имеют служебного значения.

1.5 Приоритеты операций в регулярных выражениях

Для того, чтобы правильно составить выражение необходимо понимать порядок выполнения операций в нем: например, означает ли `ab{3}` три повторения `ab` или же `a`, за которым следует три буквы `b`?

Для того, чтобы легко находить ответ на подобные вопросы следует рассмотреть регулярное выражение как выражение — т.е. последовательность операндов и операций, порядок выполнения которых определяется приоритетом операций (и, разумеется, использованием скобок). Операндами в случае регулярных выражений являются символы и символьные классы (в квадратных скобках). Выделяют четыре базовых типа операций:

1) конкатенация — простое следование одного операнда за другим, т.е. `ab` означает совпадение сначала с `a` и следующей за ней `b`;

2) круглые скобки — определяют группировку выражений и позволяют менять приоритет;

3) квантификаторы — позволяют задавать количество повторений образца шаблона: это символы `?` `+` и `*`, а также числа или интервалы в фигурных скобках;

4) операцию альтернативы — вертикальную черту.

Приоритеты этих операций следующие (от высшего к низшему):

1) круглые скобки,
2) квантификатор,
3) конкатенация,
4) альтернатива.

Таким образом, конкатенация осуществляется после применения квантификатора, но до альтернативы. То есть в приведенном выше примере `ab{3}` сначала будет применен квантификатор (к одной букве `b`), а лишь затем — конкатенация букв `a` и повторенной три раза `b`. Наоборот, альтернатива применяется позже конкатенации, поэтому `ab|cd` означает `ab` или `cd`, а не `abd` или `acd`.

Для управления этим процессом используются круглые скобки: если нам нужно получить `ab` три раза, то выражение будет выглядеть `(ab){3}`. Соответственно выражение `a(b|c)d` совпадает со строками `abd` и `acd`.

1.6 Контрольные вопросы

1. Использование символа ^ в синтаксисе регулярных выражений
2. Использование символа $ в синтаксисе регулярных выражений
3. Символьный класс и способы его задания.
4. Операции-квантификаторы в регулярных выражениях.
5. Сравните приоритет операций конкатенации, альтернативы (|) и квантификатора в регулярных выражениях.
6. Составьте регулярное выражение, под которое подходят все правильно заданные адреса электронной почты.
7. Составьте регулярное выражение, под которое подходят все правильно заданные адреса web-страниц
8. Составьте регулярное выражение, под которое подходят все правильно заданные пути к файлам в ОС Unix
9. Составьте регулярное выражение, под которое подходят все правильно заданные пути к файлам в ОС Windows
10. Укажите все варианты, совпадающие с регулярным выражением ab|cd{2,3}

11. Укажите все варианты, совпадающие с регулярным выражением (ab|cd){2,3}

12. Укажите все варианты, совпадающие с регулярным выражением [ab]|[cd]{2,3}

13. Укажите все варианты, совпадающие с регулярным выражением ([ab]|[cd]){2,3}

2 Лабораторная работа №2. Разработка лексического анализатора

2.1 Цель работы

Целью работы является изучение современных средств автоматизации разработки лексических анализаторов и разработка собственного лексического анализатора с помощью генератора flex.

2.2 Задание

Необходимо разработать лексический анализатор для выбранного языка программирования с помощью генератора лексических анализаторов flex.

2.3 Генератор лексических анализаторов flex

2.3.1 Назначение и использование flex

Программа flex (fast lexical analyzer generator) предназначена для генерации лексических анализаторов (сканеров). Входом этой программы является текстовый файл, состоящий из трех секций, разделенных строками из двух символов процента (%%):

- секция объявлений — предназначена для начальной настройки сканера;
- секция правил — задает работу сканера
- секция пользовательского кода — задает дополнительные функции (например `main`), описываемые автором анализатора а не генерируемые flex'ом.

Результатом работы flex является код программы — сканера на языке Си (по умолчанию файл называется lex.yy.c). Этот код в дальнейшем может быть скомпилирован для получения исполняемого файла анализатора. Файл содержит функцию `int yylex(void)`, выполняющую лексический анализ.

2.3.2 Основы работы анализатора

Секция правил содержит набор правил типа: `шаблон действие` (действие от шаблона отделяется пробелом). **Шаблон** представляет собой регулярное выражение (для использования специальных возможностей flex синтаксис регулярных выражений несколько расширен). **Действие** — это код на языке Си, который будет выполняться при нахождении в анализируемом тексте (текст читается из глобальной переменной `FILE *yyin`) совпадения с шаблоном. При этом коду доступны две переменных: `yytext` содержит текст, совпавший с шаблоном, а `yyleng` — длину этого текста. Например следующее правило будет печатать сообщения обо всех найденных целых числах:

```
[0-9]+ { printf("Found a number %d\n",
atoi(yytext));}
```

Если код действия занимает более одной строки, то его следует заключить в фигурные скобки. Действия могут содержать произвольный код, включая оператор `return`, выполнение которого приведет к возврату из функции `yylex`. Переменная `yytext` является указателем, указывающим внутрь разбираемого текста, поэтому обращаться с ее содержимым следует аккуратно (в частности, ничего нельзя добавлять к концу этой строки, поскольку такие действия могут испортить анализируемый текст). Если планируется дальнейшее использование этого текста, то следует скопировать его в отдельное место с помощью функции `strcpy`.

Специальный шаблон `<<EOF>>` позволяет задать действие, выполняемое в конце файла.

Если для каких-то символов входного текста сканера совпадения не было найдено, то они просто копируются в выходной поток — по умолчанию на экран. Нередко с данного места в тексте возможно несколько совпадений. Даже в случае приведенного выше правила с целым числом и входного текста 19 возможны 2 совпадения: 1 и 19. Для понимания правил, которыми руководствуется flex при выборе совпадения и шаблона, рассмотрим фрагмент лексического анализатора языка Си, содержащий два шаблона:

`[A-Za-z_][A-Za-z_0-9]+` задает идентификатор (имя функции или переменной);

`while` — задает ключевое слово `while`, оператор цикла с предусловием.

При наличии нескольких возможных совпадений flex руководствуется следующими правилами:

1) выбирается самое длинное из всех возможных совпадений — это гарантирует, что если в тексте встретится переменная `whileAbc` то она будет распознана как идентификатор, совпадение ее начала с ключевым словом не введет анализатор в заблуждение;

2) если самое длинное совпадение подходит под несколько шаблонов (в данном случае это будет само ключевое слово `while`) то выбирается самый первый из них — поэтому для правильной работы анализатора **всегда необходимо** сначала указывать все частные случаи-исключения (в данном примере `while` и другие ключевые слова), и лишь затем — общий случай (в данном примере идентификатор). При соблюдении этого порядка ключевое слово `while` будет распознано правильно, в противном случае анализатор посчитает его именем переменной.

В коде действия возможно использование некоторых специальных функций. Наиболее часто используются `BEGIN` (рассмотрена ниже в главе про стартовые условия) и `unput`. Функция `unput` позволяет поместить символ, переданный ей, во входной поток. Будьте осторожны - ее вызов портит содержимое `yytext`, поэтому его надо обработать до вызова `unput`.

Главная функция простейшего лексического анализатора обычно записывается в секции пользовательского кода и выполняет следующие действия:

1) открыть файл с анализируемым текстом с помощью функции `fopen`, сохранив возвращенный функцией результат в объявленную flex'ом (не требующую дополнительного объявления) переменную `yyin`;

2) вызвать функцию `yylex()` для выполнения лексического анализа.

2.3.3 Секция объявлений

В секции объявлений размещаются объявления и настройки, управляющие работой анализатора. Сюда входят опции анализатора, объявления именованных шаблонов, объявления стартовых условий, а также часть кода на языке Си.

Текст, расположенный между `%{` и `%}` в секции объявлений копируется в результирующий файл на языке Си до функции `yylex`. Эта секция обычно используется для подключения заголовочных файлов и (при необходимости) объявления собственных глобальных переменных (особенно внешних — `extern` — предназначенных для связи с другими частями программы). Например:

```
%{
    #include <stdio.h>
    #include "myheader.h"
    extern int _result;
%}
```

Секция правил также может содержать код между %{ и %} - такой код будет вставлен в начало функции yylex, и может использоваться для объявления локальных переменных и их инициализации.

Также секция объявлений обычно содержит опции анализатора, управляющие ее работой — они начинаются с директивы %option. Для того, чтобы сгенерированный flex'ом анализатор мог быть скомпилирован под ОС Windows необходимо указать опции

```
%option noyywrap
%option never-interactive
```

Первая из них говорит анализатору, что вы не предоставляете функции yywrap (необходимой в случае, если сканер анализирует несколько файлов, например при реализации директивы #include перенаправлением сканера на другой файл), вторая отключает оптимизацию, применяемую при анализе текстов, вводимых с клавиатуры — этот код содержит функции, специфичные для ОС семейства Unix.

Также в секции объявлений могут объявляться именованные шаблоны, состоящие из имени и записанного через пробел шаблона, например

```
DIGIT [0-9]
```

Такие шаблоны могут затем применяться в шаблонах секции правил записанными в фигурные скобки, например шаблон для восьмеричных констант языка Си с применением объявленного выше подшаблона будет выглядеть так:

```
0{DIGIT}*
```

2.3.4 Стартовые условия

В некоторых случаях правила лексического разбора могут изменяться в зависимости от ситуации. Текст комментария, например, необходимо проигнорировать. Внутри строковой константы языка Си, записанной в двойных кавычках, правила меняются еще сильнее: большинство обычных правил перестают действовать (ключевые слова, например, распознавать не надо), зато вступают в игру управляющие последовательности: \n, \\ и даже \".

Далеко не все такие ситуации могут быть обработаны одним шаблоном — так в приведенном выше примере со строкой просто кавычки означают конец строки, последовательность \" означает символ кавычки, но не конец строки, однако если ситуация принимает вид \\" то это все-же символ обратной дробной черты и конец строки. Для обработки таких ситуаций были введены стартовые условия (start condition).

Стартовые условия позволяют управлять работой шаблонов. Они записываются перед шаблонами в угловых скобках, например <STRING>\\n. В угловых скобках можно перечислить несколько стартовых условий через запятую (<STRING,COMMENT>), угловые скобки с

символом звездочки (<*>) означают правило, применимое ко всем стартовым условиям. Стартовые условия бывают двух видов:

1) включающие (inclusive) — если активно включающее стартовое условие, то работают шаблоны, перед которыми оно указано (включая звездочку), и шаблоны перед которыми не указано ни одного стартового условия;

2) исключающие (exclusive) — если активно исключающее стартовое условие, то работают только шаблоны, перед которыми оно указано (включая звездочку) — именно их следует использовать, например, для обработки строковых констант, поскольку обычные правила языка Си перестают в них действовать.

Стартовые условия должны быть объявлены в секции объявлений: объявления включающих условий начинаются с `%s`, за которым идет список условий; объявления исключающих — с `%x`.

В начале анализатор находится в специальном стартовом условии `INITIAL`, которое является включающим (в противном случае анализатор, написанный без использования стартовых условий, не работал бы). Для перехода в другое стартовое условие в коде действия следует указать инструкцию `BEGIN`, указав в скобках имя условия, например `BEGIN(STRING)`. `BEGIN(INITIAL)` может использоваться для возврата в начальное состояние.

Подробнее работа со стартовыми условиями разобрана в разделе «обработка строковых констант и многострочных комментариев».

2.3.5 C++ сканеры

В обычном режиме flex генерирует сканер на чистом языке C. Для генерирования C++ сканера, с использованием классов, следует указать `%option c++` в секции объявлений. Следует учитывать, что C++ сканеры до сих пор считаются экспериментальной возможностью flex'a и не поддерживают всех его возможностей.

В этом случае flex генерирует два класса `FlexLexer`, который содержит абстрактный интерфейс сканера и `yyFlexLexer`, являющийся его потомком, содержащий собственно сгенерированный сканер. Для работы сканера необходимо создать объект типа `yyFlexLexer`. Такой объект содержит все внутреннее состояние сканера (которое в C-сканерах хранилось в глобальных переменных), что открывает возможность иметь в одной программе несколько параллельно работающих сканеров. Наиболее важные функции, используемые для этого объекта:

`yyFlexLexer(istream *arg_yyin=0,ostream *arg_yyout=0)` — конструктор, позволяет указать входной и (при необходимости) выходной потоки сканера. Выходной поток перенаправляется редко, так как служит, в основном, для вывода символов с которыми сов-

падений не было; входной поток как правило перенаправляется в анализируемый файл;

`virtual int yylex()` - функция сканирования, аналогична `yylex`;

`const char * YYText()` - функция, возвращающая текст совпадения — аналогична переменной `yytext` в сканерах на языке C;

`int YYLeng()` - функция, возвращающая длину совпадения — аналогична переменной `yyleng` в сканерах на языке C.

2.4 Обработка строковых констант и многострочных комментариев

Наиболее сложной частью лексического анализатора является обработка сложных лексем, которые:

- не могут быть обработаны одним шаблоном (регулярным выражением) и
- правила обычного лексического разбора в них неприменимы.

Для обработки таких лексем обычно используют исключающие стартовые условия. Примерами таких лексем в языке Си могут служить строковые литералы (константы) и многострочные комментарии. Строковые литералы представляют проблему потому, что внутри них может встречаться кавычка, предваренная символом обратной дробной черты. Причем если количество таких символов является нечетным (один, три и т.д.), то кавычка должна распознаваться как символ в строке, если же количество обратных дробных черт четно (ноль, два, ...) то кавычка означает конец литерала (поскольку двойной символ обратной дробной черты должен обрабатываться отдельно как служебная последовательность). Многострочные комментарии могут доставлять проблемы потому, что они должны заканчиваться последовательностью */, при этом одиночные звездочки и дроби должны считаться частью комментария.

Типовой способ работы с такими ситуациями следующий:

1) создайте стартовое условие для обрабатываемой ситуации;
2) напишите правило, обеспечивающее вход в это стартовое условие;
3) напишите правила, обеспечивающие обработку обычных символов и специальных служебных последовательностей внутри стартового условия;
4) напишите правило, обрабатывающее завершение стартового условия и возвращение к обычному разбору.

Случай комментариев более простой. Они открываются последовательностью `/*` и заканчиваются `*/`. Лексический анализатор должен про-

игнорировать содержимое комментария. В этом случае внутри стартового состояния нам понадобятся правила для следующих действий:

1) войти в состояние комментария, если встретилась /*
2) пропустить все, кроме символов *
3) пропустить любое количество *, если за ними не следует /
4) завершить состояние комментария, если нашли одну (или несколько) *, за которыми следует /

В расположенном ниже примере правила приведены в указанном выше порядке.

```
%x comment
%%
"/*" BEGIN(comment); /* правило 1 */
<comment>[^*]* /* правило 2 */
<comment>"*"+[^*/]* /* правило 3 */
<comment>"*"+"/" BEGIN(INITIAL); /* правило 4 */
```

Обратите внимание, что правила 2 и 3 стараются взять как можно больше символов за одно срабатывание — это улучшает производительность сканера. Правило 2 обрабатывает также и символы дроби. Это не мешает работе анализатора, поскольку если перед дробью стоит * то правило 2 остановится перед ней (оно не обрабатывает *), после чего правило 3 даст совпадение из одного символа (*), а правило 4 — из двух (*/). По правилам работы flex будет выбрано наибольшее по длине совпадение, то есть правило 4. Сколько бы звездочек не стояло перед дробью, правило 4 даст совпадение на 1 символ длиннее, чем правило 3.

При обработке строковых литералов следует учитывать возможное наличие внутри них управляющих последовательностей. В языке Си они начинаются с символа обратной дробной черты (\), и служат для:

1) вставки символов, отсутствующих на клавиатуре (с указанием восьмеричного кода символа, для наиболее употребительных служебных символов присутствуют сокращения, например \n, \t);
2) вставки символа двойной кавычки (так, чтобы он оказался частью строки, а не закрывал строку);
3) вставки собственно символа обратной дробной черты в строку.

Обработка этих последовательностей и замена их необходимыми символами внутри строки являются задачами лексического анализатора. Например два символа \t должны уже после лексического анализа быть заменены в строке одним символом табуляции и т. д.

В этом случае требуются правила для:

1) начала строки;
2) обработки обычных символов;
3) обработки служебных последовательностей с кодом символа;
4) обработки служебных последовательностей с сокращениями;

5) обработки последовательностей \\ и \";

6) обработки кавычки, закрывающей строку.

Следующий упрощенный пример демонстрирует частичную обработку строки в двойных кавычках, сохраняя ее в специальную строковую переменную `literal`. При этом учитываются три управляющих последовательности: `\n`, `\"` и `\\` (не забывайте, что символы кавычек и обратной дробной черты в шаблонах являются служебными и требуют экранирования):

```
%x STRING
%%
\" strcpy(literal,""); BEGIN(STRING);
/*начало строки, 1 */
<STRING>[^\\\n\"]+ strcat(literal,yytext);
/*обычные символы, обратите внимание что перево-
ды строки не обрабатываются тоже — они не могут
встречаться внутри строковых констант, 2 */
<STRING>\\n strcat(literal,"\n");
/*обработка последовательности \n , 3 */
<STRING>\\\\ strcat(literal,"\\");
/*обработка последовательности \\ , 4 */
<STRING>\\\" strcat(literal,"\"");
/*обработка последовательности \" , 5 */
<STRING>\" {
   printf("String literal: %s",literal);
   BEGIN(INITIAL);
}
/*кавычка, закрывающая строку, 6 */
```

В этом случае, при наличии перед кавычкой нескольких символов обратной дробной черты сначала будут убраны пары символов (правило 4), затем если перед кавычкой осталась еще одна обратная дробная черта, то сработает правило 5, в противном случае — правило 6, завершающее строку.

2.5 Методика выполнения работы

1. Изучите руководство по выбранному языку программирования. Выпишите из руководства следующие данные:

1.1. перечень ключевых слов языка;

1.2. правила описания имен классов, функций и переменных;

1.3. виды констант, встречающихся в языке, и правила их описания; особо отметьте служебные символы и комбинации символов, используемые при задании строковых и символьных констант;

1.4. операции и другие синтаксические знаки (а также их сочетания), использующиеся в языке;

1.5. способы задания комментариев.

2. На основании этих данных составьте таблицу лексем выбранного языка программирования с подробных их описанием. Составьте тестовый пример — фрагмент программы на выбранном языке программирования, содержащий все типы лексем (и служебных последовательностей в них), присутствующие в таблице.

3. Для каждой лексемы составьте регулярное выражение, ее описывающее.

4. Для сложных лексем (многострочные комментарии, строковые и символьные константы со служебными символами) создайте исключающие стартовые состояния и опишите регулярные выражения, обеспечивающие переход в эти состояния и обработку текста в них (включая возвращение в начальное состояние).

5. Создайте flex-файл, впишите в него объявление стартовых состояний и подключение необходимых библиотек. Задайте опции `never-interactive` и `noyywrap`.

6. В основной секции файла впишите составленные регулярные выражения и код на языке C, обеспечивающий печать типа и значения найденных лексем.

7. В заключительной секции файла напишите функцию main, которая открывает файл с исходным текстом программы (предпочтительно получение его имени как аргумента командной строки), присваивает его в переменную `yyin` и вызывает функцию `yylex`.

8. Запустите flex указав ему в качестве аргумента созданный flex-файл.

9. Создайте проект консольного приложения в используемой среде разработки, добавьте в него созданный flex-ом файл `lex.yy.c` и пустой файл `unistd.h` (если разработка ведется в среде Windows). Скомпилируйте получившийся код.

10. Запустите результирующую программу на разработанном ранее тестовом примере и проверьте ее работоспособность. При необходимости отладьте программу.

2.6 Контрольные вопросы

1. Включающие стартовые условия в генераторе сканеров flex
2. Исключающие стартовые условия в генераторе сканеров flex
3. Особенности синтаксиса регулярных выражений в генераторе сканеров flex
4. Работа с несколькими входными файлами в генераторе сканеров flex

5. Особенности использования переменной yytext в генераторе сканеров flex

6. Устройство сканера, генерируемого flex

7. Составьте набор правил для обработки однострочных комментариев, начинающихся символами ?-

8. Составьте набор правил для обработки многострочных комментариев, заключенных в знаки %{...}%

9. Составьте набор правил, для обработки строковых констант в двойных кавычках, содержащих служебные последовательности \\, \n, \r и \"