ГЕНЕРАТОР ЛЕКСИЧЕСКИХ АНАЛИЗАТОРОВ LEX

Генератор lex строит лексический анализатор, задача которого – распознать из входного потока символов очередную лексему. Список лексем, который должен распознавать конкретный лексический анализатор, задается регулярными выражениями в секции правил входного файла генератора lex. По содержимому входного файла, генератор lex строит детерминированный конечный автомат в виде программы на языке C.

Генератор лексических анализаторов может применяться для построения различных преобразователей текстовой информации (конверторов), создания пакетных редакторов, реализации распознавателя директив в диалоговой программе и т.д.

Однако, наиболее важное применение лексического анализатора – это использование его в компиляторе или интерпретаторе специализированного языка. Здесь лексический анализатор выполняет функцию ввода и первичной обработки данных. Он распознает из входного потока лексемы и передает их синтаксическому анализатору (в качестве терминальных символов грамматического разбора).

Лексический анализатор может не только выделять лексемы, но и выполнять над ними некоторые преобразования. Например, если лексема – число, то его необходимо перевести во внутреннюю (двоичную) форму записи как число с плавающей или фиксированной точкой. А если лексема – идентификатор, то его необходимо разместить в таблице, чтобы в дальнейшем обращаться к нему не по имени, а по адресу в таблице и т.д. Все такие преобразования записываются на языке программирования C.

Регулярные выражения

Регулярное выражение является шаблоном, определяющим некоторое множество последовательностей символов.

Регулярное выражение применяется как шаблон к последовательности символов текстового файла или ко входному потоку символов. Фрагмент текста считается соответствующим регулярному выражению, если он входит в множество последовательностей символов, определяемых этим выражением.

Синтаксически регулярное выражение – это слово, т.е. последовательность символов, не содержащая разделителей (пробелов, символов табуляции и т. д.). При построении регулярных выражений используются метасимволы (символы, имеющие внутри выражений специальный смысл).

Регулярное выражение строится из первичных элементов, которые могут соединяться знаками операций. Первичный элемент обозначает одиночный символ, описывая множество символов, которому он должен принадлежать. Операции над первичными элементами определяют последовательности символов и позволяют строить более сложные регулярные выражения.

Первичные элементы

C – где C – произвольный символ, не являющийся метасимволом; определяет множество, состоящее из одного символа C.

[…] – определяет множество, состоящее из символов, перечисленных в квадратных скобках; внутри квадратных скобок допустима конструкция из трех соседних символов X-Y, где X и Y – произвольные символы; такая конструкция определяет множество, состоящее из символов, коды которых попадают в интервал между кодами символов X и Y (символы X и Y также включаются в это множество).

[^…] – определяет множество, состоящее из символов, не перечисленных в квадратных скобках (отрицание множества, определенного предыдущим первичным).

. – определяет множество, состоящее из всех символов, кроме символа перевода строки.

Унарные операции

Пусть Rexp – произвольное регулярное выражение.

Rexp? – определяет необязательное вхождение фрагмента, соответствующего регулярному выражению Rexp, т.е. повторение фрагмента 0 или 1 раз.

Rexp* – определяет n-кратное (n>=0) последовательное повторение фрагмента, соответствующего регулярному выражению Rexp.

Rexp+ – определяет n-кратное (n>0) последовательное повторение фрагмента, соответствующего регулярному выражению Rexp.

Rexp{n1,n2} – определяет n-кратное (n1<=n<=n2) последовательное повторение фрагмента, соответствующего регулярному выражению Rexp.

^Rexp – определяет фрагмент, соответствующий регулярному выражению Rexp и стоящий в начале строки.

Rexp$ – определяет фрагмент, соответствующий регулярному выражению Rexp и стоящий в конце строки.

(Rexp) – эквивалентно Rexp.

Бинарные операции

Пусть Rexp1 и Rexp2 – произвольное регулярное выражение.

Rexp1|Rexp2 – определяет фрагмент, соответствующий либо выражению Rexp1, либо выражению Rexp2.

Rexp1/Rexp2 – определяет фрагмент, соответствующий выражению Rexp1, если за ним следует фрагмент, соответствующий выражению Rexp2.

Rexp1Rexp2 – определяет последовательность фрагментов, из которых первый соответствует выражению Rexp1, а второй – выражению Rexp2.

Унарные операции старше бинарных. Бинарные операции имеют одинаковый приоритет и выполняются слева направо. Порядок выполнения операций может быть изменен с помощью круглых скобок.

Метасимволы

К **метасимволам** относятся следующие символы:

“ \ | / ^ $? . * + - ( ) [ ] { }

Любой метасимвол можно использовать как обычный символ, сняв с него специальный смысл посредством экранирования. Для экранирования используется символ \. Обратная наклонная черта снимает специальный смысл у непосредственно следующего за ней символа. Кроме того, в последовательности символов, заключенной в кавычки, все символы являются обычными и не несут специального смысла.

Для того чтобы использовать в регулярном выражении пробелы и символы табуляции, их нужно экранировать или заключать в кавычки. Не обязательно экранировать метасимволы и промежутки внутри квадратных скобок, поскольку экранирование там подразумевается. Символ минус имеет специальный смысл только внутри квадратных скобок.

В регулярных выражениях можно использовать неграфические символы. Произвольный неграфический символ записывается в виде \xxx, где xxx – восьмеричное представление кода символа.

Некоторые символы имеют следующее обозначение:

\n – символ перевода строки;

\t – горизонтальная табуляция;

\b – шаг назад;

\\ – обратная наклонная черта.

Примеры регулярных выражений

[a-z0-9_]

– определяет одиночный символ, который может быть либо строчной буквой, либо цифрой, либо символом подчеркивания;

abc\+

– определяет множество, состоящее из одной последовательности символов – **abc+**;

abc+

– определяет бесконечное множество последовательностей, начинающихся с **ab** и далее содержащих любое количество символов **c**;

[+-]?[0-9]+

– определяет множество целых чисел, возможно начинающихся со знака;

[+-]?0|([1-9][0-9]*)

– определяет множество целых чисел, возможно начинающихся со знака и не содержащих ведущих нулей;

[a-zA-Z] [a-zA-Z0-9]*

– определяет множество идентификаторов, т.е. последовательностей символов, состоящих из букв и цифр и начинающихся с буквы;

[a-c]+b/xy

– определяет пустое множество, т.к. символ **b** всегда будет распознаваться регулярным выражением в квадратных скобках;

^[01]{1,7}/

– определяет множество последовательностей из не более чем семи нулей и единиц, стоящих в начале строки, если за ней следует обратная наклонная черта.

Структура входного файла

Входной файл генератора lex имеет следующий формат:

```
секция определений
%%
секция правил
%%
секция подпрограмм
```

Секции определений и подпрограмм могут отсутствовать. Два подряд идущих символа %% являются разделителями секций и должны располагаться с первой позиции в отдельной строке. Минимальный входной файл для lex:

```
%%
```

Нет описаний, нет правил и нет подпрограмм. Будет сгенерирован анализатор, копирующий входную последовательность символов на выход без изменений.

Секция определений

Секция определений может содержать:

- список состояний лексического анализатора,
- определения имен регулярных выражений,
- фрагменты текста программы на языке С.

Список состояний лексического анализатора задается следующей директивой:

```
%start name1 name2 ...
```

где name1 name2 ... – имена состояний лексического анализатора, разделенные пробелами.

Директива %start должна начинаться с первой позиции строки. Имена состояний используются в секции правил и все они обязательно должны быть описаны в директиве %start.

Определения имен регулярных выражений имеют следующий формат:

Name Rexp

где Name – имя;

Rexp – регулярное выражение.

Имя Name должно начинаться с первой позиции строки. Имена регулярных выражений можно использовать в секции правил. В любом регулярном выражении секции правил имя регулярного выражения из секции определений будет заменяться на определяемое им регулярное выражение. Определение имен регулярных выражений обычно используют для сокращения записи регулярных выражений секции правил.

Фрагменты текста программы на языке С задаются двумя способами:

```
%{
    строки
    фрагмента
    программы
%}
```

Директивы %{ и %} записываются в отдельных строках, начиная с первой позиции.

Другой способ заключается в записи текста фрагмента программы, начиная не с первой позиции строки.

Все такие фрагменты без изменений размещаются в начале программы, построенной генератором lex, и будут внешними для любой функции сгенерированной программы.

Обычно фрагменты текста программы содержат определения глобальных переменных, массивов, структур, внешних переменных и т.д., а также операторы препроцессора.

Секция правил

В секции правил описываются лексемы, которые должен распознавать лексический анализатор, и действия, выполняемые при распознавании каждой лексемы.

Общий формат правила:

```
<States>Rexp      Action
```

где States – имена состояний лексического анализатора;

Rexp – регулярное выражение;

Action – действие.

Каждое правило должно располагаться в отдельной строке и начинаться с первой позиции этой строки. Конструкция <States> является необязательной, т. е. правило может начинаться с выражения Rexp. Имена состояний заключаются в угловые скобки и разделяются запятыми. Действие Action должно отделяться от выражения Rexp по крайней мере одним пробелом.

Выражение Rexp правила определяет лексему, которую должен распознавать лексический анализатор. Если заданы состояния States, то распознавание лексемы по данному правилу производится только тогда, когда анализатор находится в одном их указанных состояний.

Действие Action задается в виде фрагмента программы на языке С и выполняется всякий раз, когда применяется данное правило, т.е. когда

входная последовательность символов принадлежит множеству, определяемому регулярном выражением Rexp данного правила.

Действие может состоять из одного оператора языка С или содержать блок операторов С, заключенный в фигурные скобки.

Кроме операторов языка С, в действии можно использовать встроенные переменные, функции и макрооперации анализатора lex.

Встроенные переменные

yytext[]

– одномерный массив (последовательность символов), содержащий фрагмент входного текста, удовлетворяющего регулярному выражению и распознанного данным правилом;

yyleng

– целая переменная, значение которой равно количеству символов, помещенных в массив yytext.

Встроенные переменные позволяют определить конкретную последовательность символов, распознанных данным правилом. При применении правила анализатор lex автоматически заполняет значениями встроенные переменные. Эти значения можно использовать в действии примененного правила.

Пример правила:

```
[a-z]+   printf(“%s”,yytext);
```

Регулярное выражение правила определяет бесконечное множество последовательностей символов, состоящих из букв латинского алфавита. Данное правило применяется, когда из входного потока символов поступает конкретная последовательность символов, удовлетворяющих его регулярному выражению. Оператор языка С *printf* выводит в выходной поток эту последовательность символов.

Встроенные функции

yymore()

– В обычной ситуации содержимое **yytext** обновляется всякий раз, когда производится применение некоторого правила. Иногда возникает необходимость добавить к текущему содержимому **yytext** цепочку символов, распознанных следующим правилом. yymore() вызывает переход анализатора к применению следующего правила. Входная последовательность символов, распознанная следующим правилом, будет добавлена в массив **yytext**, а значение переменной **yyleng** будет равно суммарному количеству символов, распознанными этими правилами.

yyless(n)

– Оставляет в массиве **yytext** первые n символов, а остальные возвращает во входной поток. Переменная **yyleng** принимает значение n. Лексический анализатор будет читать возвращенные символы для распознавания следующей лексемы. Использование yyless(n) позволяет посмотреть правый контекст.

input()

– Выбирает из входного потока очередной символ и возвращает его в качестве своего значения. Возвращает ноль при обнаружении конца входного потока.

output(c)

– Записывает символ c в выходной поток.

unput(c)

– Помещает символ c во входной поток.

yywrap()

– Автоматически вызывается при обнаружении конца входного потока. Если возвращает значение 1, то лексический анализатор завершает свою работу, если 0 – входной поток продолжается текстом нового файла. По умолчанию

yywrap возвращает 1. Если имеется необходимость продолжить ввод данных из другого источника, пользователь должен написать свою версию функции yywrap(), которая организует новый входной поток и возвратит 0.

Пример: Входной файл.

```
%%
\"[^"]*  { if( yytext[yyleng - 1] == '\\')
              yymore();
           else
           { /*     здесь должна быть часть
                  программы, обрабатывающая
                  закрывающую кавычку.
             */
            }
         }
```

Входной файл генератора lex содержит одно правило. Анализатор распознает строки символов, заключенные в двойные кавычки, причем символ двойная кавычка внутри этой строки может изображаться с предшествующей косой чертой.

Анализатор должен распознавать кавычку, ограничивающую строку, и кавычку, являющуюся частью строки, когда она изображена как \".

Допустим, на вход поступает строка "абв\"эюя". Сначала будет распознана цепочка "абв\ и, так как последним символом в этой цепочке будет символ "\", выполнится вызов **yymore**(). В результате повторного применения правила к цепочке "абв\ будет добавлено "эюя, и в **yytext** мы получим: "абв\"эюя, что и требовалось.

Встроенные макрооперации

ECHO

– Эквивалентно printf(“%s”,yytext); . Печать в выходной поток содержимого массива **yytext.**

BEGIN st

– Перевод анализатора в состояние с именем st.

BEGIN 0

– Перевод анализатора в начальное состояние.

REJECT

– Переход к следующему альтернативному правилу. Последовательность символов, распознанная данным правилом, возвращается во входной поток, затем производится применение альтернативного правила.

Альтернативные правила

Регулярное выражение, входящее в правило, определяет множество последовательностей символов.

Два правила считаются альтернативными, если определяемые ими два множества последовательностей символов имеют непустое пересечение, либо существуют такие две последовательности из этих множеств, начальные части которых совпадают.

Примеры: Альтернативные правила.

1. Регулярное выражение

SWITCH

определяет единственную последовательность символов SWITCH, а регулярное выражение

[A-Z]+

определяет бесконечное множество последовательностей символов, в том числе и SWITCH.

2. Регулярное выражение

INT

определяют последовательность INT, которая является подпоследовательностью последовательности INTEGER, определяемой регулярным выражением

INTEGER

3. Регулярное выражение

AC+

определяет множество, являющееся пересечением множеств, определяемых выражениями

A[BC]+

A[CD]+

В лексическом анализаторе каждый входной символ учитывается один раз. Поэтому в ситуации, когда возможно применение нескольких правил, действует следующая стратегия:

1. Выбирается правило, определяющее самую длинную последовательность входных символов;
2. Если таких правил оказывается несколько, выбирается то из них, которое текстуально стоит раньше других.

Если требуется несколько раз обработать один и тот же фрагмент входной цепочки символов, то можно воспользоваться функцией yyless или макрооператором REJECT.

Примеры.

1. Предположим, что мы хотим подсчитать все вхождения цепочек **she** и **he** во входном тексте. Для этого мы могли бы написать следующий входной файл для lex:

```
%{
int s=0,h=0;
%}
%%
```

```
she     { s++;
          yyless(1);
        }
he      h++;
```

Так как **she** включает в себя **he**, анализатор (без использования функции yyless(1)) не распознает те вхождения **he**, которые включены в **she**, так как, прочитав один раз **she**, эти символы он не вернет во входной поток.

2. Рассмотрим следующий входной файл для lex:

```
%%
A[BC]+    { /* операторы
             обработки
           */
           REJECT;
          }
A[CD]+    { /* операторы
             обработки
           */
          }
```

Входная последовательность символов ACCB сначала будет распознана первым правилом, а затем первые три символа этой последовательности ACC будут распознаны вторым правилом.

Активные правила

Секция правил может содержать активные и неактивные правила. В распознавании входной последовательности символов принимают участие только активные правила. Все правила, в которых не указаны состояния (<States>) всегда являются активными. Правило, в котором заданы состояния, активно только тогда, когда анализатор находится в одном из перечисленных состояний.

Все имена состояний лексического анализатора, используемые в правилах, обязательно должны быть описаны директивой %start в секции определений.

Для перевода анализатора в некоторое состояние используется макрооперация BEGIN. Анализатор всегда находится только в одном состоянии. В начальном состоянии активны только правила, в которых отсутствуют состояния.

Действия в правилах Lex-программы выполняются, если правило активно, и если автомат распознает цепочку символов из входного потока как соответствующую регулярному выражению данного правила.

Любая последовательность входных символов, не соответствующая ни одному правилу, копируется в выходной поток без изменений. Можно сказать, что действие – это то, что делается вместо копирования входного потока символов на выход. Часто бывает необходимо не копировать на выход некоторую цепочку символов, которая удовлетворяет некоторому регулярному выражению. Для этой цели используется пустой оператор C, например:

```
[ \t\n]+      ;
```

Это правило игнорирует (запрещает) вывод пробелов, табуляций и символа перевода строки. Запрет выражается в том, что на указанные символы во входном потоке осуществляется действие ";" – пустой оператор языка C, и эти символы не копируются в выводной поток символов.

Пример.

```
%start  COMMENT
COMM_BEGIN         "/*"
COM_END       "*/"
%%
{COM_BEGIN}   { ECHO;
                BEGIN COMMENT;
              }
<COMMENT>.          ECHO;
<COMMENT>\n   ECHO;
<COMMENT>{COM_END}{ ECHO;
                    BEGIN 0;
                  }
```

```
.               ;
\n              ;
```

lex построит лексический анализатор, который выделяет комментарии в программе на языке С и записывает их в стандартный файл вывода. В случае распознавания начала комментария (/*) анализатор переходит в состояние COMMENT. При распознавании конца комментария (*/) анализатор переводится в начальное состояние.

Два последних правила позволяют игнорировать символы, не входящие в состав комментариев.

Секция подпрограмм

В секции подпрограмм размещаются функции, написанные на языке С, которые необходимы в конкретном лексическом анализаторе. Эти функции могут вызываться в действиях правил и, как обычно, передавать и возвращать значения аргументов.

Здесь же можно переопределить стандартные и встроенные функции лексического анализатора, дав им свою интерпретацию. Пользовательские версии этих функций должны быть согласованы между собой по выполняемым действиям и возвращаемым значениям.

Содержимое этой секции без изменений копируется в выходной файл, построенный генератором lex.

Примеры входных файлов генератора lex

Пример 1.

```
%%
[jJ][aA][nN][uU][aA][rR][yY]              printf("Январь");
[fF][eE][bB][rR][uU][aA][rR][yY]          printf("Февраль");
[mM][aA][rR][cC][hH]                      printf("Март");
[aA][pP][rR][iI][lL]                      printf("Апрель");
[mM][aA][yY]                             printf("Май");
[jJ][uU][nN][eE]                          printf("Июнь");
[jJ][uU][lL][yY]                          printf("Июль");
```

```
[aA][uU][gG][uU][sS][tT]                printf("Август");
[sS][eE][pP][tT][eE][mM][bB][eE][rR]    printf("Сентябрь");
[oO][cC][tT][oO][bB][eE][rR]            printf("Октябрь");
[nN][oO][vV][eE][mM][bB][eE][rR]        printf("Ноябрь");
[dD][eE][cC][eE][mM][bB][eE][rR]        printf("Декабрь");
```

Генератор построит конечный автомат, который распознает английские наименования месяцев и выводит русские значения найденных английских слов. Все другие последовательности входных символов без изменений копируются в выходной поток.

Пример 2.

```
%start AA BB CC
%{
    /*
    *   Строится лексический анализатор,
    *   который распознает наличие
    *   включений файлов в Си-программе,
    *   условных компиляций,
    *   макроопределений,
    *   меток и головной функции main.
    *   Анализатор ничего не выводит, пока
    *   осуществляется чтение входного
    *   потока, а по его завершении
    *   выводит статистику.
    */
%}
БУКВА             [A-ZA-Яa-za-я_]
ЦИФРА           [0-9]
ИДЕНТИФИКАТОР   {БУКВА}({БУКВА}|{ЦИФРА})*

int a1,a2,a3,b1,b2,c;
a1 = a2 = a3 = b1 = b2 = c = 0;
%%
^#                      BEGIN AA;
^[ \t]*main             BEGIN BB;
^[ \t]*{ИДЕНТИФИКАТОР} BEGIN CC;
[ \t]+              ;
\n                  BEGIN 0;
<AA>define          { a1++; }
```

```
<AA>include       { a2++; }
<AA>ifdef         { a3++; }
<BB>”(“.")"       { b1++; }
<BB>"()"          { b2++; }
<CC>":"           { c++;  }
.                 ;
%%
yywrap(){
     if( b1 == 0 && b2 == 0 )
          printf("В программе отсутствует функция main.\n");
     if( b1 >= 1 && b2 >= 1 ){
          printf("Многократное определение функции main.\n");
     } else {
          if(b1 == 1 )
               printf("Функция main с аргументами.\n");
          if( b2 == 1 )
               printf("Функция main без аргументов.\n");
     }
     printf("Включений файлов: %d.\n",a2);
     printf("Условных компиляций: %d.\n",a3);
     printf("Определений: %d.\n",a1);
     printf("Меток: %d.\n",c);
     return(1);
}
```

Оператор **return**(1) в функции **yywrap** указывает, что лексический анализатор должен завершить работу.

Пример 3.

```
%{
#include   "y.tab.h"
extern int yylval;
%}
%%
^\n           ;
[ \t]*        ;
[A-Za-z]      { yylval = yytext[yyleng-1] - 'a';
                return(LETTER);
              }
[0-9] +       { int i;
                yylval = yytext[0] - '0';
                for(i=1; i< yyleng; i++)
```

```
            yylval = yylval *10 + yytext[i] - '0';
        return(DIGIT);
    }
```

Лексический анализатор распознает однобуквенные идентификаторы и целые положительные числа и возвращает номера типов этих лексем, определенных в файле **y.tab.h**. Во внешнюю переменную **yylval** помещаются следующие значения: для идентификатора – порядковый номер в английском алфавите; для числа – значение в двоичном представлении.

Использование генератора lex

Вызов выполнения генератора lex имеет вид:

lex Lfile

где **Lfile** – имя входного файла, построенного в соответствии с требованиями структуры входного файла lex.

В результате выполнения этой команды lex создаст лексический анализатор в виде текста программы на языке С и поместит его в файл со стандартным именем **lex.yy.c**

Файл **lex.yy.c** содержит две основных функции и несколько вспомогательных.

Основными являются две следующие функции:

yylex()

– содержит разделы действий всех правил, которые определены пользователем;

yylook()

– реализует детерминированный конечный автомат, который осуществляет разбор входного потока символов в соответствии с регулярными выражениями правил входного файла генератора lex.

Для получения выполняемой программы лексического анализатора (загрузочного модуля) необходимо выполнить компиляцию программы

lex.yy.c и скомпоновать ее с программами из стандартной библиотеки генератора lex. Это выполняется следующей командой:

cc lex.yy.c -ll

В результате выполнения этой команды будет создан выполняемый файл (загрузочный модуль) со стандартным именем **a.out.**

Головная функция **main** из стандартной библиотеки генератора lex имеет вид:

```
main(){
     yylex();
     exit(0);
}
```

Разработчик лексического анализатора имеет возможность подключить собственную функцию **main**() вместо библиотечной. Для этого достаточно поместить текст собственной функции **main**() в секцию подпрограмм входного файла lex.

Наличие сгенерированного текста программы в файле **lex.yy.c** дает программисту дополнительные возможности для внесения корректив в работу лексического анализатора.