lex - Eine Einführung

Axel Kohnert

9th May 2005

Abstract

`lex` ist ein Unixprogramm, welches die Erstellung eines C-programms für die lexikalische Analyse unterstützt. Dazu kann man Aktionen definieren, die beim Erkennen von regulären Ausdrücken in der Eingabe ausgeführt werden. Diese Aktionen werden in C-Syntax definiert.

1 Syntax eines lex-programms

1.1 Aufruf von lex

Ein lex-Programm, steht üblicherweise in einem file mit der Endung `.l`, dieser file, z.B. bsp.l, wird mit dem Kommando
`lex bsp.l`
in den file `lex.yy.c` kompiliert, und diesen kann man mit dem Kommando
`cc lex.yy.c -ll`
in den ausführbaren file `a.out` übersetzen. Mit dem Aufruf
`make bsp`
wird automatische aus dem file `bsp.l` ein ausführbarer file `bsp` erzeugt. Dies gilt nicht auf allen UNIX systemen, manchmal muß man dafür dem make etwas nachhelfen, indem man die Umwandlungsregel explizit angibt (siehe Anhang)

1.2 Syntax

Das kleinste zulässige lex-Programm, besteht aus der Zeile
%%
es kopiert die Standardeingabe auf die Standardausgabe, dies ist das default-verhalten des lexikalischen Analyse programms, falls kein regulärer Ausdruck gematcht wurde. Die allgemeine Syntax ist folgende:
{Definitionen}
%%
{Regeln}
%%
{Unterprogramme}
Wie schon obiges Beispiel gezeigt hat, sind alle Teile bis auf das erste ‘ %%‘ optional.

1.3 Regeln

Regeln, die im zweiten Teil des lex-programms definiert werden, bestehen aus zwei Teilen, der erste Teil ist ein regulärer Ausdruck, der zweite Teil eine Aktion (in C-Syntax), die bei Erkennen des regulären Ausdrucks ausgeführt werden soll. Dazu ein Beispiel

```
%%
integer printf("token:INT\n");
```

dieses Programm kopiert Standardeingabe auf Standardausgabe, erstetzt dabei jedoch die Zeichenkette `integer` durch den angegebenen Text.

1.4 reguläre Ausdrücke

Zur Definition von regulären Ausdrücken im Regelteil hat man mehrere Sonderzeichen zur Verfügung, diese sind

```
 "   \  [  ]  ^  -  ?  .  *  |  (  )  $  /  {  }  %  ,  >
```

will man diese Zeichen ohne ihre Sonderbedeutung verwenden, so müssen sie mit '\' maskiert werden. Dazu ein Beispiel

```
%{
int i=0;
%}
%%
\?   printf("bereits %d Fragezeichen\n",++i);
```

in diesem Beispiel sehen wir auch schon die Einführung von Variablen. Dazu später mehr. Der Teil zwischen von lex erstellten C Programms eingefügt. Nun ein Aufstellung der regulären Ausdrücke mit Sonderzeichen

1. **[xya]** steht für das Zeichen `x` oder `y` oder `a`.
2. **[a-z]** steht für alle Zeichen zwischen `a` und `z`. Es geht auch **[a-zA-Z]** um z.B. alle Buchstaben zu matchen.
3. **[^a-z]** steht für alle Zeichen außer `a` bis `z`.
4. **.** steht für alle Zeichen außer \n.
5. Man kann zeichen auch durch die oktale Darstellung ansprechen, so steht \40 für ein blamk.
6. **ab?c** matcht auf `ac` oder `abc` (0 oder 1-malige Wiederholung)
7. **ab*** 0 mal oder öfter
8. **ab+** 1 mal oder öfter
9. **ab{1,5}** 1 mal bis maximal 5 mal

10. **(ab|cd+)?** man kann klammern, | steht für oder und gilt für die gesamten Ausdrücke d.h. der Ausdruck matcht auf `ab, cd, cdd, cddd ...` Nicht jedoch auf `abd`.

11. **ab/cd** matcht auf `ab`, aber nur wenn es von `cd` gefolgt wird.

12. **ab$** matcht auf `ab` am Zeilenende.

13. ^**ab** matcht auf `ab` am Zeilenanfang.

2 Aktionen

In diesem Abschnitt schauen wir uns noch weitere Möglichkeiten für die Definition von Aktionen an.

2.1 globale C-Variablen

Aktionen werden ausgeführt, wenn ein gelesener String matcht, auf diesen String kann man mit der globalen Variable
`char * yytext;`
zugreifen. Die Länge des Strings steht in der Varibale
`int yyleng;`
hierzu ein Beispiel, das Programm soll Namen erkennen, ihre Anzahl und die Gesamtzahl der verwendeten Buchstaben speichern. Alle integer-zahlen der Eingabe sollen außerdem addiert werden.

```
%{
int buchstaben=0, namen=0, summe=0;
%}
%%
[a-zA-Z]+{namen++; buchstaben += yyleng;}
[1-9][0-9]*summe+= atoi(yytext);
```

2.2 Wie matcht lex?

Dies geschieht nach zwei Regeln

1. Der längste match wird genommen.

2. Falls zwei gleichlang sind, so wird der zuerst definierte genommen.

Hierzu ein Beispiel:

```
integer         printf("token:INT");
[a-z]+         printf("identifier:%s",yytext);
```

wird nun das wort 'integerwert' gelesen, so wird dies als identifier genommen, wird hingegen 'integer' gelesen, so word dies als token erkannt. Dies ist auch das gewünschte Verhalten.

2.3 Definitionen

Im Beispiel sahen man bereits, daß zwischen '%{' und '%}' C-source eingegeben werden kann. Ferner können Macro-definitionen für den lex-code vorgenommen werden, dies geschieht durch Zeilen der Form:

```
Name            Bedeutung
```

die Definition `name` wird dann in regulären Ausdrücken des Regelteil durch `'{name}'` angesprochen. Obiges Beispiel hätte man auch so schreiben können:

```
D   integer
I   [a-z]+
%%
{D}  printf("token:INT");
{I}  printf("identifier:%s",yytext);
```

2.4 weitere Variablen und Routinen

Neben den beiden Variablen `yytext, yyleng` gibt es noch weitere Variablen und Routinen.

- `int yylineno`, die aktuelle Zeilennummer.
- `ECHO`, dies Macro ist äquivalent zu `printf("%s",yytext)`.
- `int yymore()`, eine Funktion, die den nächsten match sucht und das Ergebnis an `yytext` anhängt und auch die zugehörigen Aktionen durchführt.
- `int yyless(int n)`, eine Funktion die Zeichen in die Eingabe zurückschreibt, sodaß `yytext` nur noch n Zeichen lang ist.
- `int yywrap()`, diese Funktion wird ganz am Ende der lexikalischen Analyse durchgeführt. Der Benutzer kann sie auch umschreiben, was im Unterprogrammteil geschieht.
- `char input(), int output(char c), int unput(char c)`, dies sind die I/O routinen von lex, zum Lesen des nächsten Zeichens, zur Ausgabe eines Zeichens, zum Zurückgeben eines Zeichens in die Eingabe.

Ein Beispiel zur Verwendung der Routinen:

```
%%
=-[a-zA-Z] {
printf("1");
yyless(yyleng-1);
}
[a-zA-Z] printf("2");
- printf("3");
= printf("4");
%%
```

bei der Eingabe von '=-a' erfolgt die Ausgabe '1' und das a wird in die Eingabe zurückgeschrieben, nun wird der nächste Match gesucht, und beginnend beim 'a' wird das 'a' gematcht und eine '2' ausgegeben.

3 Rest

3.1 Bewertung von lex

Vorteile:

- Nur ein Durchlauf durch den Text
- Zeit zum Erkennen eines Tokens proportional zur Länge des Tokens
- Leicht zu ändern

Nachteile:

- Fehlermeldungen nicht dokumentiert
- Namenskonflikte bei der Verknüpfung von mehreren mit lex erstellten Programmen
- nicht alle Konstanten von lex können geändert werden
- Manchmal große Tabellen

3.2 Literatur

1. **Heckhoff H.** LEX, unix/mail (1985/1) p. 62-68
2. **Lesk M., Schmidt E.** LEX - A Lexical-Analyser Generator, Computer Science Technical Report 39 (1975)
3. **Schreiner Axel** Professor Schreiners UNIX Sprechstunde, Hanser Verlag München 1987