S実験'95　S3 (LexとYacc) ＃1

久野　靖 *

1995.4.24, 1995.5.22, 1995.5.31

1　Lexとパターン認識

1.1　Lexとは

さて、皆様これまでの課題でC言語をマスターされた(?!)と思うので、今度はなるべくCで書かないで済むツールについて勉強して頂く。その第1段はlexというツールである。このツールは、標準入力に現れる特定のパターンを認識するC関数(名前はyylexと決まっている)を自動生成してくれる。

例えば、入力に「名前」(というのは、英字で始まり、英数字が並んだものですよね)と「非負整数」(というのは、数字が並んだものですよね)が現れ、それの区別をつける必要があるとしよう。これを行なうためのlexソースは次の通り。

```
digit   [0-9]
alpha   [a-zA-Z]
white   [\n\t ]
%%
{digit}+                        { return NUM; }
{alpha}({alpha}|{digit})*       { return IDENT; }
{white}                         { ; }
```

これはどう読むかというと。まずlexのソースファイルは次のような形をしている。

*筑波大学大学院経営システム科学専攻

```
定義
...
%%
パターン　{ 動作... }
パターン　{ 動作... }
...
```

従って、最初の 3 行は「定義」である。これはそれぞれ、「digit とは 0〜9 までの字のことである」「alpha とは小文字の a〜z、および大文字の A〜Z までの字のことである」「white とは改行、タブ、および空白文字のことである」と読む。

次に「パターン」であるが、最初のは「digit が 1 個以上繰り返したもの」と読む。この時は yylex は NUM という値 (これはあとで define しよう) を返す、という動作を行なう。もちろん、動作の部分は C 言語で書くわけだ。2 番目のパターンは「alpha がまずあり、続いて (alpha または digit) が 0 回以上繰り返したもの」で、この時はもちろん IDENT という値を返す。最後に white があった時は「何もしない」から、無視されてさらに先の入力を読みに行くことになる。

さて、これを動かすためには次のようにやる。

```
% ls
t1.c    t1.lex
% lex t1.lex
% ls
lex.yy.c        t1.c            t1.lex
```

つまり、lex は C 言語ソースを lex.yy.c という決まった名前のファイルに作成する。ここではこれを t1.c という C ソースファイルに取り込んで使うようにしている。その中身を見て頂こう。

```
#define yywrap() 1        ← これはおまじないでいると覚えて。
#define NUM     1         ← NUM は実は 1。
#define IDENT   2         ← IDENT は実は 2。

#include "lex.yy.c"       ← ここに生成されたのを取り込む。

main() {                  ← 主プログラム。
  int code;
  while(code = yylex())   ← 0 が帰ったらファイルのおわり。
    switch(code) {
case NUM:   printf("num: %s\n", yytext); break;
case IDENT: printf("id:  %s\n", yytext); break;
```

```
    }
}
```

見ての通り、yylex が NUM または IDENT を返すので、そのどちらであるか、を表示する。なお、yylex が「動作」を実行する時には「パターン」に一致した入力文字列は yytext という配列に保存されているので、これを打ち出せばどんな文字列が来たかも分かる。実行させてみよう。

```
% cc t1.c
% a.out
123
num: 123
abc
id:  abc
abc123
id:  abc123
123abc
num: 123
id:  abc
^D
%
```

よろしいでしょう?

練習 1: この例題を打ち込んで動かしてみよ。

1.2 自前で書くと...

上では lex が yylex を自動生成してくれたが、これを自前で書くことも当然できる。例えば次の通り。

```
#include <stdio.h>

char yytext[100];
int nextc = 0;

yylex() {
  int i = 0;
  if(nextc == 0) nextc = getchar();
  while(nextc != EOF) {
    if(nextc == ' ' || nextc == '\t' || nextc == '\n') {
      nextc = getchar(); }
    else if(nextc >= '0' && nextc <= '9') {
```

```
      do {
        yytext[i++] = nextc; nextc = getchar();
      } while(nextc >= '0' && nextc <= '9');
      yytext[i++] = 0; return NUM; }
    else if(nextc >= 'a' && nextc <= 'z' || nextc >= 'A' && nextc <= 'Z') {
      do {
        yytext[i++] = nextc; nextc = getchar();
      } while(nextc >= 'a' && nextc <= 'z' || nextc >= 'A' && nextc <= 'Z' ||
              nextc >= '0' && nextc <= '9');
      yytext[i++] = 0; return IDENT; }
    else {
      fprintf(stderr, "invalid char: %x\n", nextc); nextc = getchar(); }
  }
  return 0;
}
```

これがt1yylex.cに入っていたとして、さっきのt1.cのincludeを"t1yylex.c"に直せばまったく同じに動作する。

練習 2: この、自前で書いた版も動かしてみよ。

練習 3: 何か、名前や数が沢山あるファイル (たとえば/usr/dict/words とか) を食べさせてみて、両者の能率を比べてみよ。能率を比べるのならmainのprintfは取ってしまった方がよい。実行時間の測り方は:

```
% time a.out
1.2u 0.2s ....
 ↑ この最初の数字がこのプログラムがCPUを使用した秒数です。
```

1.3 「パターン」について

先に出てきたものを含めて、Lexで使うことができるパターンについてひと通り説明いたしませう。以下でα、βなどは任意の (部分) パターンを意味する。

```
[文字... 文字]    :  [] 内の文字のどれか。
    (「X-Y」と書いてコード順でXからYまで全部、を表わせる)
[^文字... 文字]   :  [] 内の文字以外の文字のどれか。
.                 :  任意の1文字。
文字              :  その文字。ただし [や . など特殊文字は別扱い。
\文字 or "文字"   :  その文字。[や . など特殊文字を指定するのに使う。
{定義名}          :  定義の参照。
```

$\alpha\ \beta$	:	α に続いて β 。
α | β	:	α または β
^α	:	行の先頭にある α 。
α $	:	行の末尾にある α 。
α *	:	α の 0 回以上の繰り返し。
α +	:	α の 1 回以上の繰り返し。
α ?	:	0 個または 1 個の α 。
α / β	:	α 、ただしすぐ次に β がある。
(α)	:	α 。(くくるのに使う。)

ところで、「定義」というのは単にパターン部が長くて読みにくくなるのを防ぐためにある機能で、定義参照が現れたらその部分が機械的に定義の右辺に置き変わるだけである。だからさっきの

```
digit   [0-9]
alpha   [a-zA-Z]
white   [\n\t ]
%%
{digit}+                        { return NUM; }
{alpha}({alpha}|{digit})*       { return IDENT; }
{white}                         { ; }
```

は次のように置き換えても構わない。

```
%%
[0-9]+                          { return NUM; }
[a-zA-Z][a-zA-Z0-9]*            { return IDENT; }
[\n\t ]                         { ; }
```

さて、パターンについてまとめたところでもう少し複雑な例として、符合つき整数や実数も扱えるように改良してみよう。

```
sign    [+-]
dot     "."
digit   [0-9]
alpha   [a-zA-Z]
white   [\n\t ]
%%
{sign}?{digit}+                 { return NUM; }
{sign}?{digit}+{dot}{digit}*    { return RNUM; }
{alpha}({alpha}|{digit})*       { return IDENT; }
{white}                         { ; }
```

読めますよね?

練習 4: これに対応するように main のファイルも直して動かせ。

練習 5: lex を使わず C 言語で書いた版も作ってみよ。

練習 6: 指数形式 (`12.5e-6` とか `3e15` とか) も使えるように改良してみよ。

練習 7: その lex を使わず C 言語で書いた版も作ってみよ。(これはすごく大変だから、よっぽどやりたい人のみでよい。)

1.4 演算子とキーワード

演算子、というのは要するに'+' とか'*' などのことである。これをいちいち

```
"+"             { return PLUS; }
```

のように扱ってもよいが、実は ASCII の (つまり日本語でない) 文字コードというのは 0〜255 の数値だから、これを直接返すようにすればいちいち名前を考えるより楽である。

```
"+"             { return '+'; }
```

ただし、それをやる場合には NUM などの define を文字コードとぶつからないように 256 から順に振るようにすること。

にもかかわらず、演算子の中で 2 文字以上を組み合わせたもの (== とか+=など) は 1 文字のコードでは表せないのでやっぱり名前を使うようにしないといけないのに注意。

```
"=="             { return EQUAL; }
```

あと、プログラム言語には名前と同じ形をしているけど特別な意味を持つ語 (予約語) が指定されている場合が多い。Pascal だと if とか while とかがそうである。そういうのは

```
begin            { return KBEGIN; }
```

のようにして扱える。(b とか e とかいうのは b という字や e という字を表す「パターン」でもあることに注意。) 従って「begin」という入力はこのパターンにマッチする。ところが、これまで名前に使っていた

```
{alpha}({alpha}|{digit})*       { return IDENT; }
```

も「begin」という入力にマッチするのでちょっと困る。しかし、lex では「同じパターンにマッチする行が複数あるときは、先に書いてあるものが優先する」という規則があるため、KBEGIN の方を先に書いておけば OK である。じゃあ、beginning という名前は begin と ing に分けられてしまうのか、というとこれも「マッチする長さが長い方が優先する」という規則が上の規則に優先する、ということになっているので大丈夫である。

練習 8: C 言語の演算子や記号 (かっこ等)、および予約語も扱えるような lex 記述を作成してみよ。(自分で覚えている範囲で良い。)

練習 9: その lex を使わず C 言語で書いた版も作ってみよ。(これもすごく大変なので、よっぽどやりたい人のみでよい。)

1.5 Lex の限界

これは次回につながる話題なので、できるだけ考えてみてください。ただし、決してできないものが含まれていますので、できないと思ったらそう書いてください。

練習 10: 「(」と「)」で囲まれた数、というのをただの NUM とは別に認識するようにしてみよ。

練習 11: 今度は、2 重かっこと 3 重かっこで囲まれた数、というのをただの NUM とは別に認識するようにしてみよ。

練習 12: 一般に n 個のかっこで囲まれた数、というのをただの NUM とは別に認識するようにしてみよ。

A 本日の練習問題兼出席

「練習 n」と記されているものを順にやってください。ただし、やらなくてもいい、という指示が書かれているものはやりたい人だけで結構です。

できたら、その実行例のうち比較的高度な (と自分で思った) 部分を 2〜3 ページぶん選んで整理して打ち出してください。そして**最後の**紙の裏に次の内容:

- 学籍番号、氏名、所属、本日の日付。
- 以下のアンケートに対する答え (簡単でいいですよ)。

  Q1. これまで、Lex のようなツールを使ったことがありますか。ある人は、それと比べてどうですか。

  Q2. 本日やったことのうち面白かった/興味深かった部分はどこですか。また、難しいと思った部分はどこですか。

  Q3. 本日の感想、今後の要望などお書きください。

を記入したものを出席用に提出してください。出席として認定されるためには、本日 5 時までに事務室のレポートボックスに提出のこと。