

ツールとしての yacc と lex

大浦 真@ LILO

ohura@netfort.gr.jp

<http://www.netfort.gr.jp/~ohura/>

目次

- yacc って何?
- lex って何?
- yacc と lex
- yacc のファイルの構造
- lex のファイルの構造
- lex の利用例
- yacc と lex の利用例
- おわりに

yacc って何?

- UNIX で古くから使われているツール。
- yacc = **Y**et **A**nother **C**ompiler **C**ompiler
- 文法定義から構文解析プログラム (パーサ) を生成するプログラム
パーサとは 入力された文について、正しい文かどうか、
どういう構造をした文か、などを分析するプログラム。
- yacc は単なるプログラムではなく、プログラムを生成するプログラムである。
- 現在は、yacc と上位互換な GNU Bison が良く使われる。

lex って何?

- こちらも UNIX で古くから使われているツール。
- lex = **LEX**ical analyser generator
- 字句解析プログラム (スキャナ) を生成するプログラム
スキャナとは 入力された文から、文法を構成する意味のある最小単位を識別する。
- lex も単なるプログラムではなく、プログラムを生成するプログラムである。
- 現在は、lex と上位互換である Flex が良く使われる。

yacc と lex

- yacc と lex は一緒に使われることが多い。
- 簡単な電卓から、gcc のような compiler まで、何らかの「文法」を持ったプログラムを作るのに使われる。
- ちなみに、fetchmail は設定ファイルを解析する部分に yacc と lex を使っている。

yacc のファイルの構造

- 基本的な構造

宣言部

%%

ルール部

%%

プログラム部

Figure 1: yacc のファイル構造

yacc のファイルの構造

- 拡張子には、通常 .y が使われる。
- 中身は、宣言部、ルール部、プログラム部の 3 つに分かれている。

宣言部 C 言語の宣言 (#include など) や、yacc の宣言、yacc に与えるオプションを記述する。省略可。

ルール部 構文規則を記述する。

プログラム部 C 言語のプログラムを記述する。省略可。

yacc のファイルの構造

● 構文規則の書き方

```
symbol : boby1 { action1 }  
       | boby2 { action2 }  
       ;
```

- ‘symbol’ は ‘body1’、もしくは、‘body2’ からできている。
- action の部分には、C 言語の式を書く。
- 「‘symbol’ が ‘body1’ からできている」という規則が成り立つ場合には、‘action1’ が実行される。

lex のファイルの構造

- 基本的な構造

定義部

%%

ルール部

%%

サブルーチン部

Figure 2: lex のファイル構造

lex のファイルの構造

- 拡張子には、通常 .l が使われる。
- 中身は、定義部、ルール部、サブルーチン部の 3 つに分かれている。

定義部 C 言語の宣言、lex に与えるオプションを記述する。省略可。

ルール部 スキャナのルールを記述する。

サブルーチン部 C 言語のプログラムを記述する。省略可。

lex のファイルの構造

- ルールの書き方

```
pattern  action
```

- pattern には、識別したいものを記述。正規表現が使える。
- action には、pattern が見付かった時に実行される C 言語の式を記述する。

lex の利用例

- 最小の lex プログラム

```
minimum.l  
%%
```

- 何のルールも記述されていないので、単に標準入力からの入力をそのまま標準出力に出力するだけ。

lex の利用例

- コンパイルの仕方
 - この lex ファイルを minimum.l とする。
 - lex を実行

```
% flex minimum.l
```

- lex.yy.c というファイルができるので、それをコンパイル。ただし、この時 libfl とリンクさせる必要がある。

```
% gcc lex.yy.c -o minimum -lfl
```

lex の利用例

- 例その 2。入力から指定した文字列を削除する。

```
delete.l  
  
%%  
user ;
```

- 入力の中に 'user' という pattern にマッチする文字列があったらそれに対応する action が実行される。ただし、この例では、action は何もしない。
- マッチしない文字列はそのまま出力される。

lex の利用例

- 例その3。指定した文字列を置換する。

```
replace.l  
  
%{  
#include <stdio.h>  
%}  
%%  
user[0-9]          { printf("USER"); }
```

- pattern には、正規表現が使えるので、この場合は、
'user0'、'user1'、、、という文字列にマッチする。
- さらに、action に printf("USER"); が記述されているので、
マッチした文字列のかわりに 'USER' が出力される。

lex の利用例

- 「これだけなら sed だけで十分じゃない。」
- 例その4。少し複雑な置換の例。スタート状態を使う。

state1.l

```
%{
#include <stdio.h>
%}
%s comment
%%
<INITIAL>"%"      { printf("%%");
                    BEGIN comment; }
<INITIAL>user     { printf("USER"); }
<comment>\n      { printf("\n");
                    BEGIN INITIAL; }
```

lex の利用例

- その時の「状態」によってマッチする pattern を切替える。
- 例えば、コメント内部 (この場合、‘%’ から改行まで。) 以外の文字列のみを置換したい場合
- pattern の前に <state> という形で状態を記述する。
- 最初は、‘INITIAL’ という状態で始まる。
- ‘%’ にマッチすると ‘comment’ という状態が始まる。
- ‘comment’ という状態にある時、‘\’ (改行文字) にマッチすると、‘INITIAL’ という状態が始まる。

lex の利用例

- 例その 5(state2.1 別紙)。
- C 言語のコメント (‘/*’ から始まり ‘*/’ で終わる。) の行数とコメント以外の行数を数える。
- スキャナ自身は、C 言語になると yylex() という関数になるので、yylex() を main 関数の中で呼び出す。
- ‘INITIAL’ 状態の時、‘\n’ にマッチすると、変数 `line_number` の値が一つ増える。
- ‘comment’ 状態の時、‘\n’ にマッチすると、変数 `comment_line_number` の値が一つ増える。

yacc と lex の利用例

- yacc はしばしば lex と一緒に使われる。
- yacc が生成するパーサは、yylex() という関数を呼び出すようになっている。
- 例その6(english.y english.l 別紙)。
- ごくごく簡単な英語を解析するプログラム。

yacc と lex の利用例

- コンパイルの仕方
 - yacc を実行

```
% bison -d english.y
```

- english.tab.c と english.tab.h ができる。この english.tab.h を lex のファイルで include する。
- lex と gcc を実行

```
% flex english.l  
% gcc -Wall english.tab.c lex.yy.c  
-o english -lfl
```

yacc と lex の利用例

- 例その 7(別紙)。簡単な電卓。
- 例その 8(手前味噌)。b2c(BASIC to C translator)。

おわりに

- yacc や lex は、本格的に使うと少し専門的になってしまふ。
- しかし、特に lex は、簡単な使い方を覚えるだけで、文字列を操作する強力な「ツール」として使うことができる。
- 設定ファイルなどを解析するためにも使える。
- GNUjdoc
`http://openlab.ring.gr.jp/gnujdoc/` に日本語の info マニュアルがある。日本語の書籍もちらほら。
- ぜひ、「ツール」としての yacc と lex を使ってみましょう。