Lex and Yacc

A Quick Tour

HW8—Use Lex/Yacc to Turn this: Into this:

Here's a list:

- This is item one of a list
- This is item two. Lists should be indented four spaces, with each item marked by a "*" two spaces left of four-space margin. Lists may contain nested lists, like this:
  - Hi, I'm item one of an inner list.
  - Me two.
- Item 3, inner. Item 3, outer list.

This is outside both lists; should be back to no indent.

Final suggestions:

2.

if myVar == 6.02e23**2 then f( ..

Lex / Yacc History

- Origin – early 1970’s at Bell Labs
- Many versions & many similar tools
  - Lex, flex, jflex, posix, …
  - Yacc, bison, byacc, CUP, posix, …
  - Targets C, C++, C#, Python, Ruby, ML, …
- We’ll use jflex & byacc/j, targeting java (but for simplicity, I usually just say lex/yacc)
Uses

- “Front end” of many real compilers
  - E.g., gcc
- “Little languages”:
  - Many special purpose utilities evolve some clumsy, ad hoc, syntax
  - Often easier, simpler, cleaner and more flexible to use lex/yacc or similar tools from the start

Lex: A Lexical Analyzer Generator

- Input:
  - Regular exprs defining "tokens"
  - Fragments of declarations & code
- Output:
  - A java program “yylex.java”
- Use:
  - Compile & link with your main()
  - Calls to yylex() read chars & return successive tokens.

Lex Input: "mylexer.flex"

// java stuff

//: Lex section delims
%%
%byaccj
%
{
public foo()…
%
%
}%
%
%
[a-zA-Z]+ {foo(); return(42); }
[
  \t\n]
{ /* skip whitespace */}
...

Rules/ regexps + {Actions}

Declarations & code: most copied verbatim to java pgm

Token code

No action

yacc: A Parser Generator

- Input:
  - A context-free grammar
  - Fragments of declarations & code
- Output:
  - A java program & some “header” files
- Use:
  - Compile & link it with your main()
  - Call yyparse() to parse the entire input
  - yyparse() calls yylex() to get successive tokens
Lex Regular Expressions

Letters & numbers match themselves
Ditto \n, \t, \r
Punctuation often has special meaning
But can be escaped: \* matches "*"
Union, Concatenation and Star
r|s, rs, r*; also r+, r?; parens for grouping
Character groups
[ab*c] == [^cab]. [a-z2648AEIOU],[^abc]
"^" for "not" only in char groups, not complementation

Expression lexer: "expr.l"

```c
#include "y.tab.h"

yyerror(char *msg){printf("%s,%s\n",msg,yytext);}
int yywrap(){return 1;}
```

Yacc Input: “expr.y”

```yacc
%{
import java.io.*;
%
}
%token NUM VAR
%
stmt: exp { printf("%d\n",$1); }
    ;
exp : exp '+' NUM { $$ = $1 + $3; }
    |    exp '-' NUM { $$ = $1 - $3; }
    |    NUM { $$ = $1; }
    ;
%
```

Lex/Yacc Interface: Compile Time
Lex/Yacc Interface:
Run Time

```
main()

yylex()

yyparse()

Myaction:
...  
  yylval = ...
  ...
  return(code)
```

Parser “Value” class

```
public class ParserVal
{
  public int ival;
  public double dval;
  public String sval;
  public Object obj;
  public ParserVal(int val)  
  { ival=val; }
  public ParserVal(double val)  
  { dval=val; }
  public ParserVal(String val)  
  { sval=val; }
  public ParserVal(Object val)  
  { obj=val; }
} // end class
```

```
// then do
yylval = new ParserVal(3.14);
yylval = new ParserVal(42);

// ... or something like...
yylval = new ParserVal(new myTypeOfObject());
```

```
// in yacc actions, e.g.:
$$ .ival = $1.ival + $2.ival;
$$ .dval = $1.dval - $2.dval;
```

More Yacc Declarations

```
%token BHTML BHEAD BTITLE BBODY P BR LI
%token EHTML EHEAD ETITLE EBODY
%token <sval> TEXT
%type <obj> page head title
%type <obj> words list item items

%start page
```

“Calculator” example

From http://byaccj.sourceforge.net/

```
%
  import java.lang.Math;
  import java.io.*;
  import java.util.StringTokenizer;
%
/* YACC Declarations; mainly op prec & assoc */
%token NUM
%left '-' '+'
%left '*' '/'
%left NEG  /* negation--unary minus */
%right '^'  /* exponentiation */
/* Grammar follows */
%%
...
Ambiguous grammar; prec/assoc decls are a (smart) hack to fix that.

```java
void dotest()
{...

public static void main(String args[])
{...
```
# set following 3 lines to the relevant paths on your system
JFLEX = ~ruzzo/src/jflex-1.4.3/jflex-1.4.3/bin/jflex
BYACCJ = ~ruzzo/src/byaccj/yacc.macosx
JAVAC = javac
LEXDEBUG = 0 # set to 1 for token dump

# targets:
run: Parser.class
  java Parser $(LEXDEBUG) test.ratml
Parser.class: Yylex.java Parser.java Makefile test.ratml
  $(JAVAC) Parser.java
Yylex.java: jratml.flex
  $(JFLEX) jratml.flex
Parser.java: jratml.y
  $(BYACCJ) -J jratml.y

clean:
  rm -f *~ *.class *.java

Makefile:
Not required, but convenient

General form
A: B C
  D
Means A depends on B & C and is built by running D

Parser "states"
Not exactly elements of PDA's "Q", but similar
A yacc "state" is a set of "dotted rules" – rules in G with a "dot" (or ".") somewhere in the right hand side.
In a state, "A \rightarrow \alpha_\beta" means this rule, up to and including \alpha is consistent with input seen so far; next terminal in the input must derive from the left end of some such \beta. E.g., before reading any input, "S \rightarrow _.\beta" is consistent, for every rule S \rightarrow \beta (S = start symbol)
Yacc deduces legal shift/goto actions from terminals/ nonterminals following dot; reduce actions from rules with dot at rightmost end. See examples below
Yacc In Action

Initially, push state 0 while not done {
let S be the state on top of the stack;
let i in Σ be the next input symbol;
look at the action defined in S for i:
if "accept", halt and accept;
if "error", halt and signal a syntax error;
if "shift" to state T", push i then T onto the stack;
if "reduce via rule r (A → α )", then:
pop exactly 2*i|α| symbols
(the 1st, 3rd, ... will be states, and
the 2nd, 4th, ... will be the letters of α);
let T = the state now exposed on top of the stack;
T's action for A is "goto state U" for some U;
push A, then U onto the stack.
}

Implementation note: given the tables, it's deterministic,
and fast -- just table lookups, push/pop.

Yacc Output

state 0
$accept : _expr $end
( shift 4
A shift 5
. error
expr goto 1
term goto 2
fact goto 3

state 1
$accept : expr$_send
expr : expr$_+ term
$end accept
+ shift 6
. error

state 2
expr : term$_ (2)
term : term$_* fact
. shift 7
. reduce 2

...
Goto & Lookahead

Example: input "A + A $end"

An Error Case: "A ) $end"
Q: Do you have any advice for up-and-coming programmers?
A: ... One more piece of advice – take a theoretician to lunch...

From the end of a 2008 interview with Steve Johnson, creator of yacc