Lex & Yacc

A compiler or an interpreter performs its task in 3 stages:

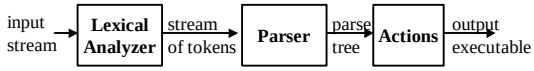
1) **Lexical Analysis:** scans the input stream and converts sequences of characters into tokens (token is a classification of groups of characters)

   Lex is a tool for writing lexical analyzers.

2) **Syntactic Analysis (Parsing):** A parser reads tokens and assembles them into language constructs using the grammar rules of the language.

   Yacc is a tool for constructing parsers.

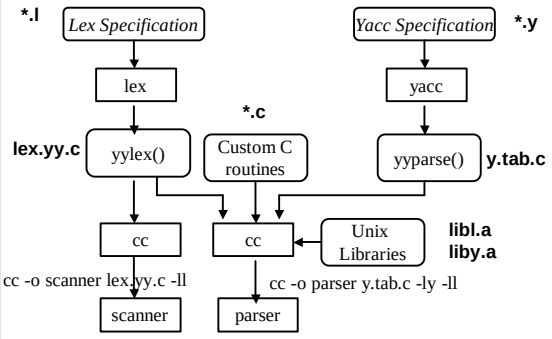
3) **Actions:** Acting upon input is done by code supplied by the compiler writer.



Example: Lexical analysis

Lexeme	Token
Sum	identifier
i	identifier
:=	assignment_op
=	equal_sign
57	integer_constant
*	multiplication_op
,	comma
(	left_paran

Using Lex and Yacc Tools



Lex: reads a spec file containing regular expressions and generates a C routine that performs lexical analysis.
Matches sequences that identify tokens.

Lex declares an external variable called **`yytext`** which contains the matched string.

Yacc: reads a spec file that codifies the grammar of a language and generates a parsing routine.

Examples

```
%%
[+-]?[0-9]*(\.)?[0-9]+   printf("FLOAT");
```

```
digit [0-9]
%%
[+-]?{digit}*(\.)?{digit}+    printf("FLOAT");
```

```
digit [0-9]
sign  [+-]
%%
float val;
{sign}?{digit}*(\.)?{digit}+
                  {sscanf(yytext, "%f", &val);
                   printf(">%f<", val);
```



```
%%
[A-Z]+[ \t\n]  printf("%s", yytext);
.    ;    /* no action specified */
```

```
alphabetic   [A-Za-z]
digit        [0-9]
alphanumeric ({alphabetic}|{digit})
%%
{alphabetic}{alphanumeric}*  printf("variable");
\,                           printf("Comma");
\{                           printf("Left brace");
\:\=                         printf("Assignment");
```

Compare the following two specs:

```
%%
for        printf("FOR");
[a-z]+     printf("IDENTIFIER");
```

```
%%
[a-z]+     printf("IDENTIFIER");
for        printf("FOR");
```


Yacc

Yacc specification describes a CFG, that can be used to generate a parser.

Elements of a CFG:

- Terminals: tokens and literal characters,
- Nonterminals (variables): syntactical elements,
- Production rules,
- Start rule.

Example:

symbol: definition
{action}
;

A ® Bc is written as **a : b 'c'**

Declarations in a yacc spec file define tokens and their characteristics.

%token: declare name of tokens
%left: define left-associative operators
%right: define right-associative operators
%nonassoc: define operators that may not associate with themselves.
%type: declare the type of variables
%union: declare multiple data types for semantic values.
%start: declare the start symbol (first var in rules)
%prec: assign precedence to a rule
%{ C declarations %} directly copied to the resulting C program


```
/* print-int.l */
%%
[0-9]+  {sscanf(yytext, "%d", &yylval);
          return(INTEGER);
        }
\n      return NEWLINE;
.       return(yytext[0]);
```

```
/* print-int.y */
token INTEGER NEWLINE
%%
```