Lex and Yacc

<<Lex_and_Yacc.ext>>

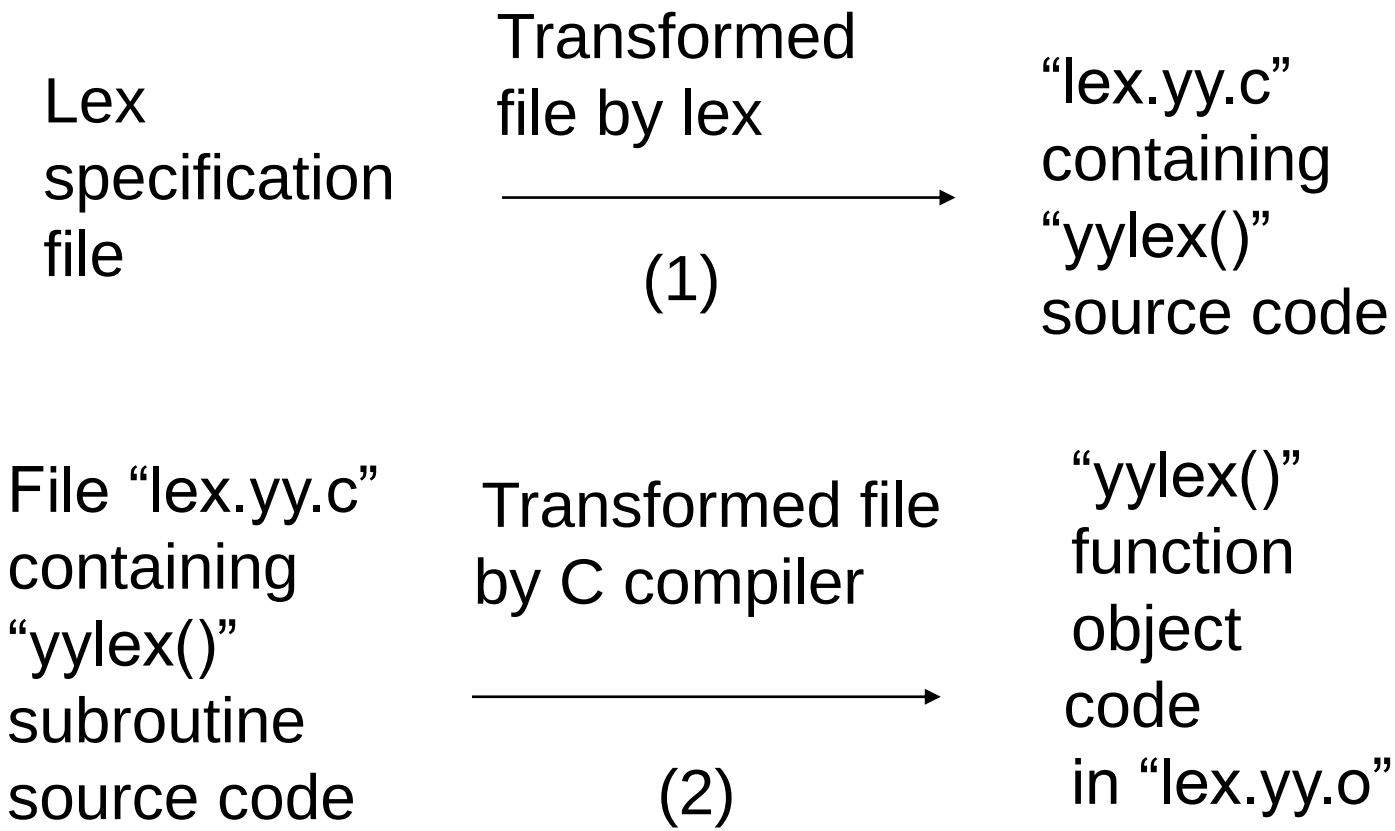
Lex and Yacc

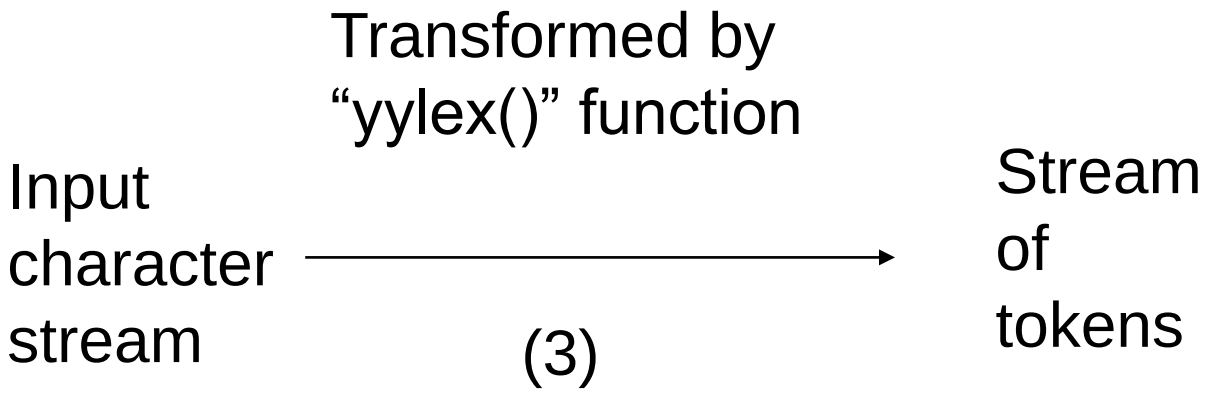
- Lexical analyzer: a program that perform lexical analysis.
- Parser: a program that performing parsing.
- Yacc: Yet Another Compiler Compiler.
- The input to the lexical analysis phase is **a stream of characters**.
- When the lexical program recognizes an object, it outputs an indication of the type of object that it has just encountered. The indication is usually called a **token**.

- A lexical analyzer follows rules to recognize certain sequences or groups of characters, whereas a parser follows rules that may be self-referential to recognize more complicated constructs.

- The Lex program is used to generate a lexical analysis routine. Lex reads as input a specification of the lexical analyzer, and it produces either a C or a RATDOR subroutine as its output.

Lex


Lex specification file
Transformed file by lex
(1)
“lex.yy.c” containing “yylex()” source code
File “lex.yy.c” containing “yylex()” subroutine source code
Transformed file by C compiler
(2)
“yylex()” function object code in “lex.yy.o”


Transformed by
“yylex()” function
Input
character
stream
(3)
Stream
of
tokens

* The lexical analyzer takes a stream of characters as input, and produces a stream of tokens as output.

rules: regular expression. EX. **`[0-9]+`**

The action code will be performed every time the rule leads to a pattern match.

```
[0-9]+ {printf("a number");}
[0-9]+ {return(NUMBER);}

[0-9]+ {
        yylval = atoi(yytext);
        return(NUMBER);
}
```

EX. Numbers (digit strings).
The keywords set, bit, on, and off.
Either a new line or a semicolon representing a command terminator.

```
$ cat y.tab.h

# define SET 257
# define BIT 258
# define ONCMD 259
# define OFFCMD 260
# define NUMBER 261
# define ENDCMD 262
# define UNKNOWN 263
```

The Format of a LEX File

```
%{
   C program declaration
%}
%%

lex rules
%%
user subroutines
```

```
$ cat lexdemo.l

%{
/*
* a lex specification to recognize
* numbers, 4 words, and delimiters
*/
# include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+     {    /* rule 1 */
       yylval = atoi(yytext);
       return(NUMBER);
       }
;       return(ENDCMD);    /* rule 2 */
\n      return(ENDCMD);    /* rule 3 */
set     return(SET);          /* rule 4 */
bit     return(BIT);          /* rule 5 */
on     return(ONCMD);       /* rule 6 */
off     return(OFFCMD);      /* rule 7 */
[ \t]+                  ;     /* rule 8 */
 .      return(UNKNOWN);    /* rule 9 */
%%
```

- Lex works through the rules from the **top down**, so rule 9 will apply only when all of the other rules fail.

- The rule that specifies the **longest match** is applied useless several rules specify a match of the same length, in which case the **topmost** rule is applied.

```
$ cat lextst.c
#include "y.tab.h"
int yylval;
extern char yytext[];
/*
*          DEMONSTRATION
* call yylex() to acquire tokens
*/
main()
{
int token;
while( token = yylex()) {
      switch(token)
      {
```

```
case NUMBER:
     printf("Number: %d\n",yylval);
     break;
case SET:
     printf("set\n");
     break;
case BIT:
     printf("Bit\n");
     break;
case ONCMD:
     printf("On\n");
     break;
case OFFCMD:
     printf("Off\n");
     break;
```

```
        case UNKNOWN:
                printf("Unknown: %s\n",yytext);
                break;
        case ENDCMD:
                printf("End marker\n");
                break;
        default:
                printf("Unknown token: %d\n",token);
                break;
        }
}
}
```

```
$ cc lextst.c lex.yy.c -o lextst
$ echo "set bit 5 on;set20" | lextst
Set
Bit
Number: 5
On
End marker
Set
Number: 20
End marker
$ echo "set bit 3 On" | lextst
Set
Bit
Number: 3
Unknown: O
Unknown: n
End marker
$ _
```

Yacc

- The Yacc utility program is used to create a **parser** subroutine.
- Yacc accepts a syntax specification and then produces wither C or RATFOR source code for the parser routine.

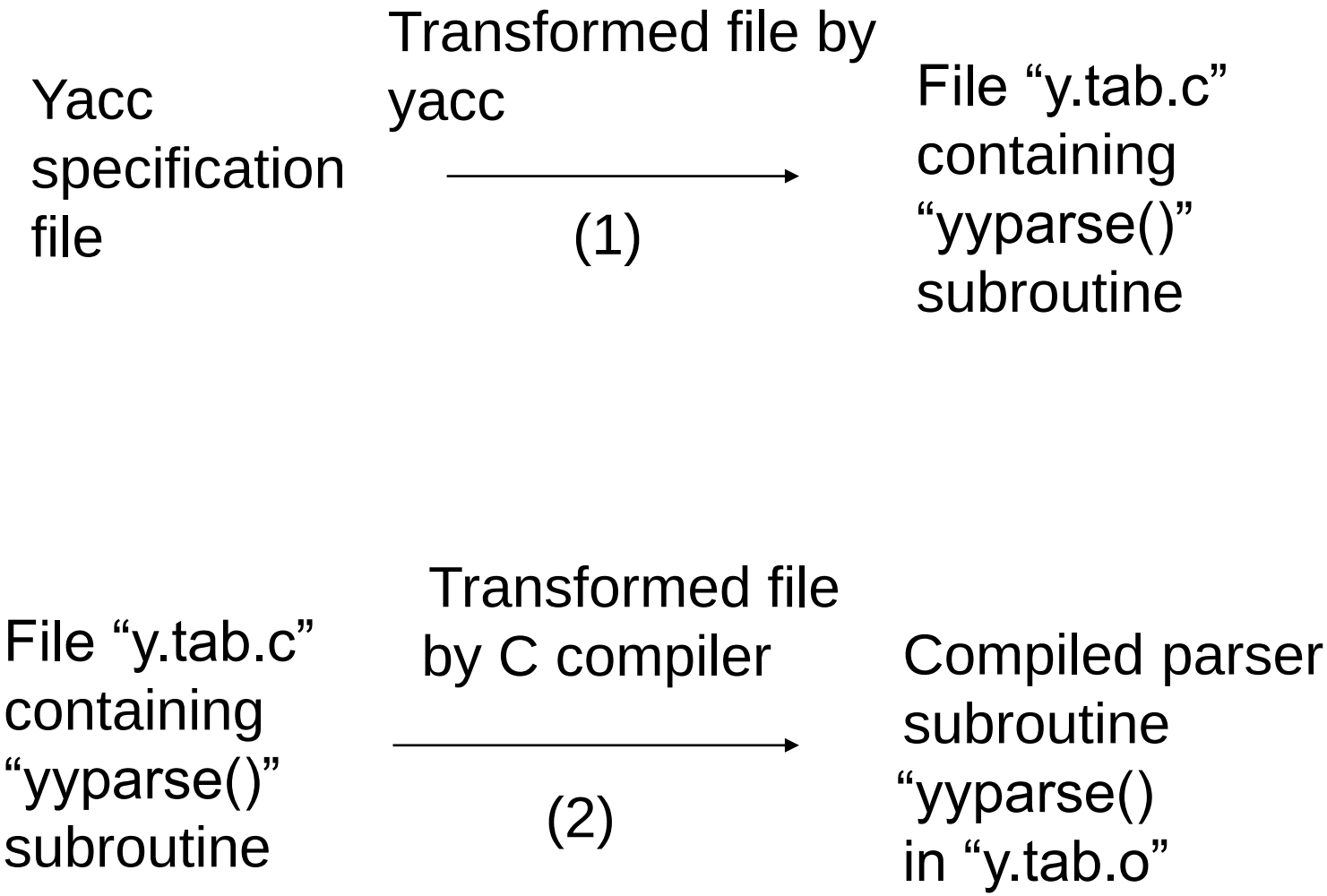
* Format for the rules (lower/upper cases)

1)

```
cmd: SET BIT numb ONCMD ENDCMD
          |
      SET BIT numb OFFCMD ENDCMD
          ;
```

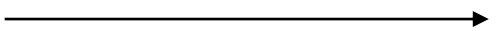
2)

```
cmd: SET BIT numb onoff ENDCMD
          ;
onoff:   ONCMD
               |
         OFFCMD
         ;
numb: NUMBER
          ;
```


Yacc specification file
Transformed file by yacc
(1)
File “y.tab.c” containing “yyparse()” subroutine
File “y.tab.c” containing “yyparse()” subroutine
Transformed file by C compiler
(2)
Compiled parser subroutine “yyparse() in “y.tab.o”

Stream of Tokens delivered by "yylex()" function

Transformed by "yyparse()" subroutine



(3)

Sequence of appropriate actions

- A low-level rule (e.g., numb) can return a value to a high-level rule (e.g., cmd) by passing a value to the pseudo-variable **$$**.

```
num: NUMBER
 {$$ = yylval;}
;
```

- A high-level rule can pick up a value by examining the pseudo-variable $1 for the first member of the definition, $2 for the second, and so on.

The Format of a YACC File

```
%{
  C program declaration
%}
%TOKEN A,B
%%

yacc rules
%%
main()
{
}
```

```
$ cat yaccdemo.y
%{
/*
 * Yacc Specification File
 * - The First Part -
 *    Declarations
 */
int testvar = 0
int yylval;
#define off 0
#define on 1
%}
%TOKEN     SET,BIT,ONCMD,OFFCMD
%TOKEN     NUMBER,ENDCMD,UNKNOWN
%%
```

```
/*
 * - The Second Part -
 *      Rules
 */
section:            /* Rule 1 */
      |
      cmds
      ;
cmds: cmd           /* Rule 2 */
      |
      cmds cmd
      ;
```

```
cmd: ENDCMD          /* Rule 3 - Alternative 1*/
                    { /* the null cmd */ }
     |
     SET BIT numb onoff ENDCMD /* Alternative 2 */
     {
        if (($3 <= 15) && ($3 >=0)) {
           if ($4 == off)
              testvar = testvar & ~(1 << $3);
           else
              testvar = testvar | (1 << $3);
        }
        else
            printf("Illegal bit number: %d\n",$3);
        printf("testvar - %o\n",testvar);
     }
```

```
        |
        SET numb ENDCMD   /* Rule 3 Alternative 3 */
        {
              testvar = $2;
              printf("Testvar - %0\n",testvar);
        }
        ;
numb: NUMBER          /* Rule 4 */
        { $$ = yylval;}
        ;
onoff: ONCMD  /* Rule5 - Alternative 1 */
        {$$ = on;}
        |
         OFFCMD           /* Rule 5- Alternative 2 */
         {$$ = off;}
        ;
%%
```

```
/*
 *
 *   - The Third Part -
 *   Support Subroutines
 */
main()
{
      yyparse();
}
```

* Processing

```
$ yacc  -d  yaccdemo.y
$ cc  -o  yaccdemo  y.tab.c lex.yy.c -lfl
```

EX. command:

```
set trial 3 amplitude 10 (X)
set bit 5 on trial 3 csr (X)
set bit 3 on
set 10
set bit 4 off; set bit 0 on
```