

Syntax Analysis

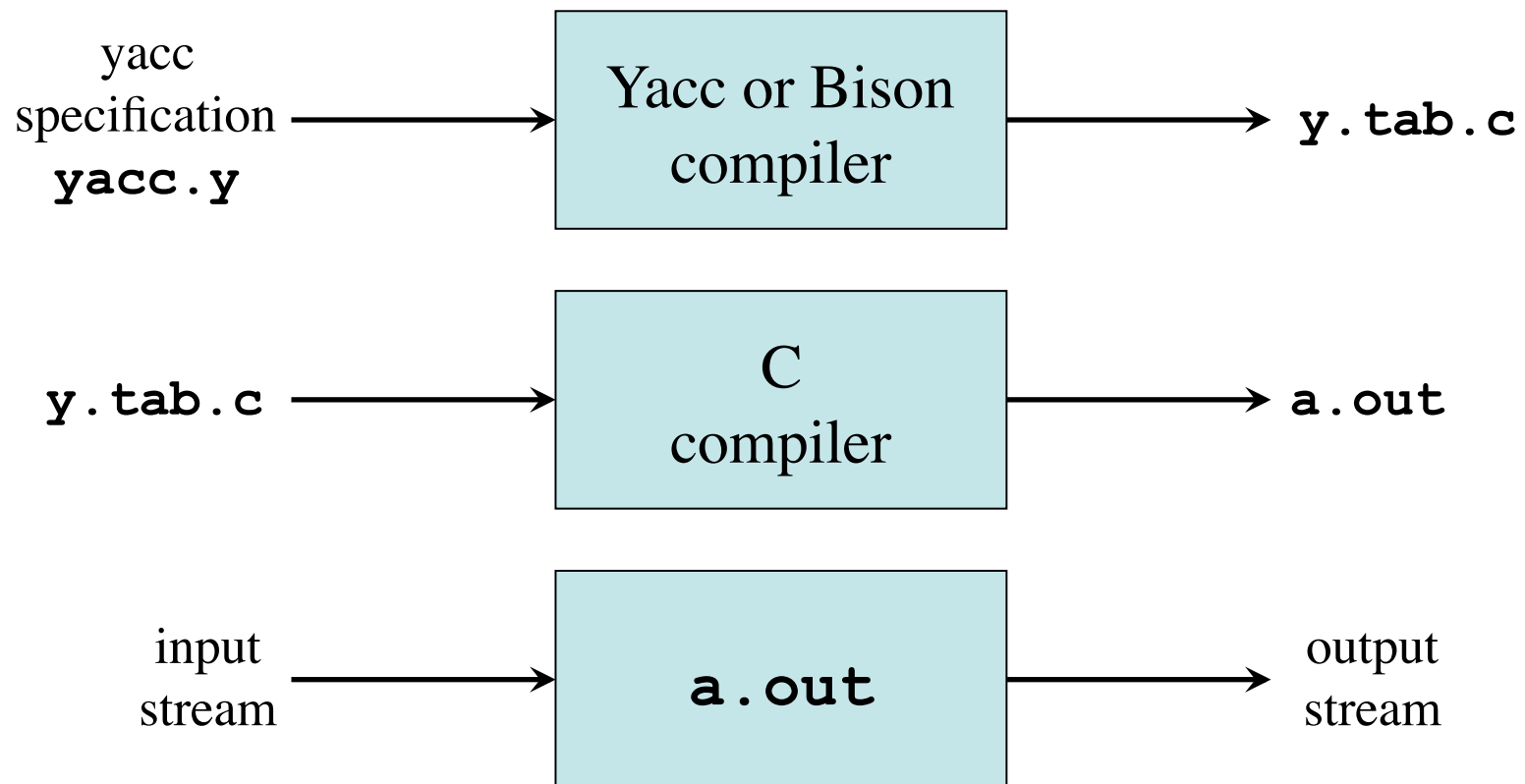
Part III

Chapter 4

ANTLR, Yacc, and Bison

- *ANTLR* tool
 - Generates LL(k) parsers
- *Yacc* (Yet Another Compiler Compiler)
 - Generates LALR(1) parsers
- *Bison*
 - Improved version of Yacc

Creating an LALR(1) Parser with Yacc/Bison



Yacc Specification

- A *yacc specification* consists of three parts:
 - yacc declarations, and C declarations within* `% { % }`
 - `%%`
 - translation rules*
 - `%%`
 - user-defined auxiliary procedures*
- The *translation rules* are productions with actions:
 - production*₁ { *semantic action*₁ }
 - production*₂ { *semantic action*₂ }
 - ...
 - production*_n { *semantic action*_n }

Writing a Grammar in Yacc

- Productions in Yacc are of the form

```
Nonterminal : tokens/nonterminals { action }  
              | tokens/nonterminals { action }  
              ...  
              ;
```
- Tokens that are single characters can be used directly within productions, e.g. `'+'`
- Named tokens must be declared first in the declaration part using

```
%token TokenName
```

Synthesized Attributes

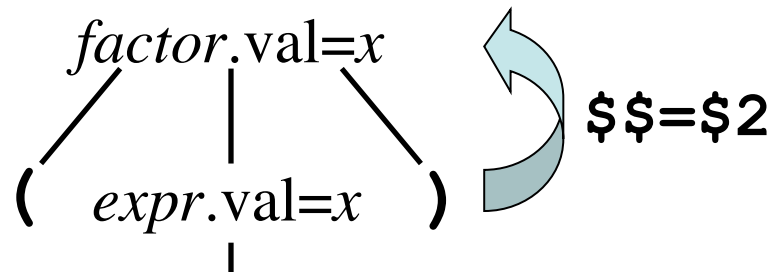
- Semantic actions may refer to values of the *synthesized attributes* of terminals and nonterminals in a production:

$$X : Y_1 Y_2 Y_3 \dots Y_n \{ \textit{action} \}$$

- $\$\$$ refers to the value of the attribute of X
- $\$i$ refers to the value of the attribute of Y_i

- For example

factor : `' (' expr ') ' { $\$\$=\$2$; }`



Example 1

```

%{ #include <ctype.h> %}
%token DIGIT
%%
line      : expr '\n'          { printf("%d\n", $1); }
          ;
expr      : expr '+' term     { $$ = $1 + $3; }
          | term              { $$ = $1; }
          ;
term      : term '*' factor   { $$ = $1 * $3; }
          | factor           { $$ = $1; }
          ;
factor    : '(' expr ')'     { $$ = $2; }
          | DIGIT            { $$ = $1; }
          ;
%%
int yylex()
{ int c = getchar();
  if (isdigit(c))
  { yylval = c-'0';
    return DIGIT;
  }
  return c;
}

```

Also results in definition of **#define DIGIT xxx**

Attribute of **term** (parent)

Attribute of **factor** (child)

Attribute of token (stored in **yylval**)

Example of a very crude lexical analyzer invoked by the parser

Dealing With Ambiguous Grammars

- By defining operator precedence levels and left/right associativity of the operators, we can specify ambiguous grammars in Yacc, such as

$E \rightarrow E+E \mid E-E \mid E * E \mid E/E \mid (E) \mid -E \mid \mathbf{num}$

- To define precedence levels and associativity in Yacc's declaration part:

```
%left '+' '-'
```

```
%left '*' '/'
```

```
%right UMINUS
```

Example 2

```

%{
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double
%}

%token NUMBER
%left '+' '-'
%left '*' '/'
%right UMINUS
%%

lines  : lines expr '\n'      { printf("%g\n", $2); }
      | lines '\n'
      | /* empty */
      ;

expr   : expr '+' expr       { $$ = $1 + $3; }
      | expr '-' expr       { $$ = $1 - $3; }
      | expr '*' expr       { $$ = $1 * $3; }
      | expr '/' expr       { $$ = $1 / $3; }
      | '(' expr ')'        { $$ = $2; }
      | '-' expr %prec UMINUS { $$ = -$2; }
      | NUMBER
      ;


%%

```

Double type for attributes and `yylval`

Example 2 (cont'd)

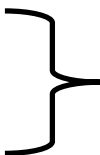
```
%%  
int yylex()  
{ int c;  
  while ((c = getchar()) == ` `)  
    ;  
  if ((c == `.`) || isdigit(c))  
  { ungetc(c, stdin);  
    scanf("%lf", &yylval);  
    return NUMBER;  
  }  
  return c;  
}  
int main()  
{ if (yyparse() != 0)  
  fprintf(stderr, "Abnormal exit\n");  
  return 0;  
}  
int yyerror(char *s)  
{ fprintf(stderr, "Error: %s\n", s);  
}
```



Crude lexical analyzer for
fp doubles and arithmetic
operators

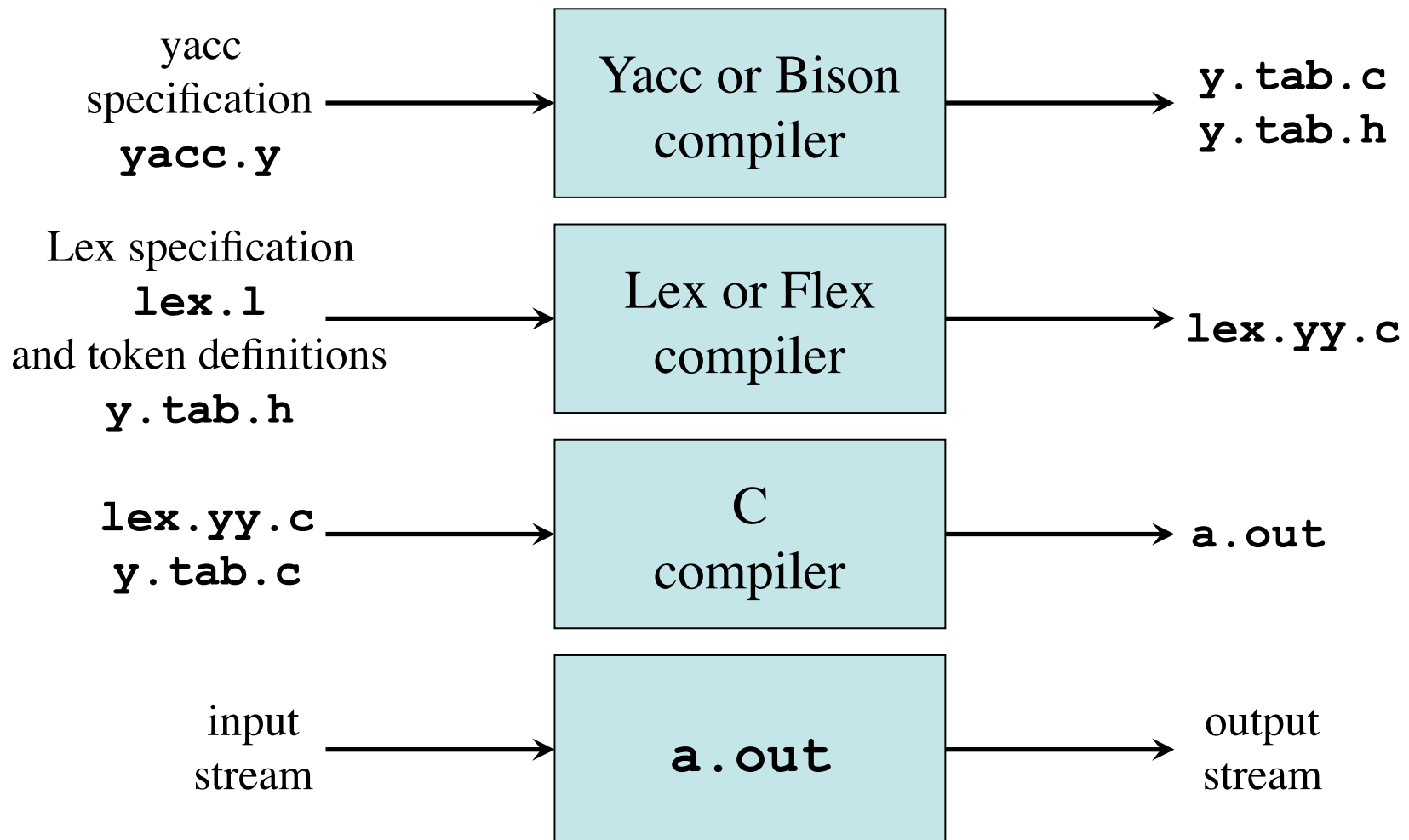


Run the parser



Invoked by parser
to report parse errors

Combining Lex/Flex with Yacc/ Bison



Lex Specification for Example 2

```

%option noyywrap
%{
#include "y.tab.h"
extern double yylval;
}%
number [0-9]+\.|[0-9]*\.[0-9]+
%%
[ ]          { /* skip blanks */ }
{number}    { sscanf(yttext, "%lf", &yylval);
              return NUMBER;
            }
\n|.        { return yttext[0]; }

```

Generated by Yacc, contains **#define NUMBER xxx**

Defined in **y.tab.c**

```

yacc -d example2.y
lex example2.l
gcc y.tab.c lex.yy.c
./a.out

```

```

bison -d -y example2.y
flex example2.l
gcc y.tab.c lex.yy.c
./a.out

```

Error Recovery in Yacc

```

%{
...
%}
...
%%
lines : lines expr '\n'      { printf("%g\n", $2; }
      | lines '\n'
      | /* empty */
      | error '\n'
;
...

```

Error production:
set error mode and
skip input until newline

Reset parser to normal mode