COMPILER DESIGN

TEXT BOOK:

1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, "Compilers: Principles, Techniques and Tools", Second Edition, Pearson Education Limited, 2014.

Language Specification

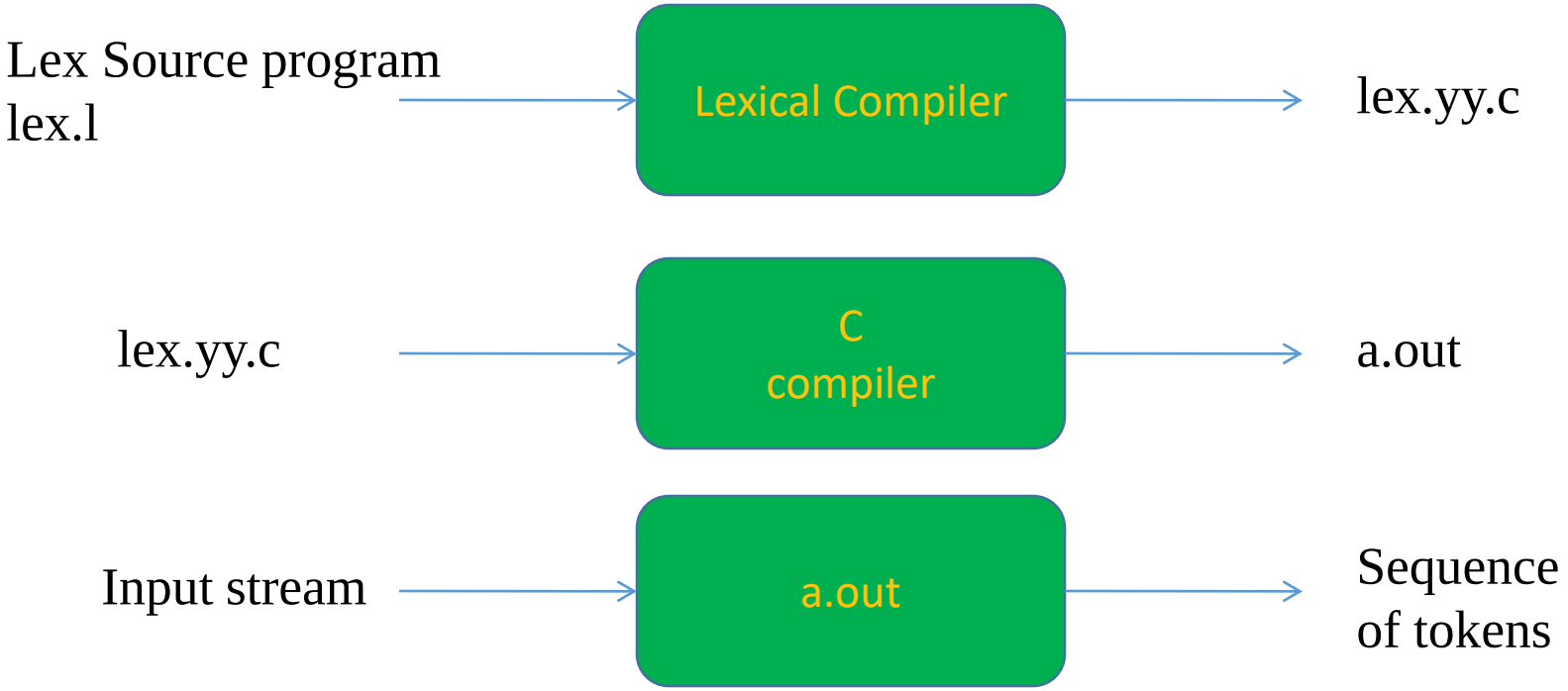
- Lex
- Yacc

Language Specification

- Lex is a scanner generator
  - Input is description of patterns and actions
  - Output is a C program which contains a function yylex() which, when called, matches patterns and performs actions per input
  - Typically, the generated scanner performs lexical analysis and produces tokens for the (YACC-generated) parser

Lexical Analyzer Generator - Lex

Lexical Analyzer Generator - Lex



Structure of Lex programs

declarations
%%
translation rules
%%
auxiliary functions

Pattern {Action}

Structure of Yacc programs

Definitions
%%
Rules
%%
Supplementary Code

Simple Lex Example

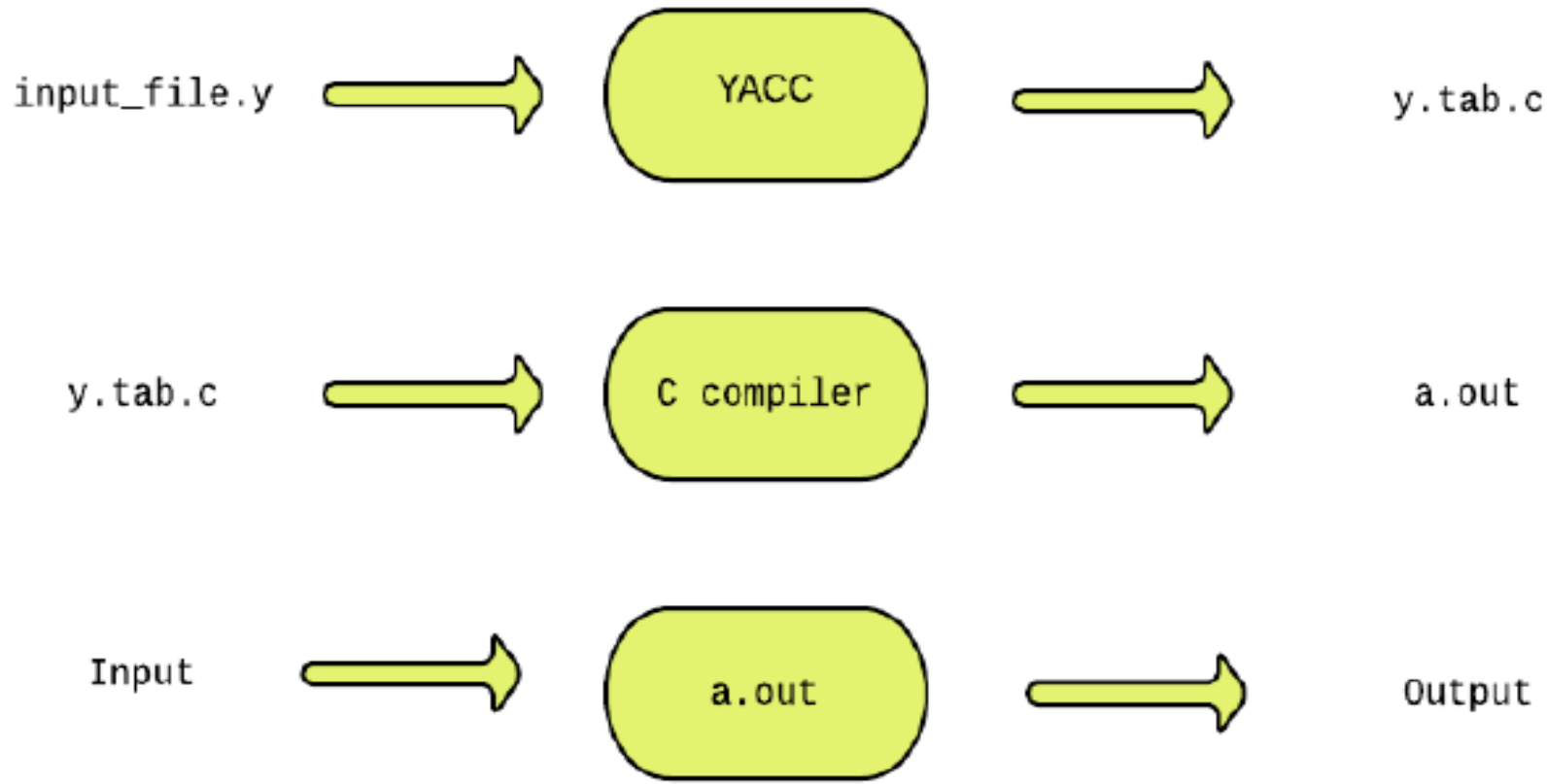
```
        int num_lines = 0, num_chars = 0;
%%
\n      ++num_lines; ++num_chars;
.       ++num_chars;
%%
main()
{
        yylex();
        printf( "# of lines = %d,
                # of chars = %d\n",
                num_lines, num_chars );
}
```

MODULE VI:	L	T	P	EL
	3	0	4	3
LALR Parser – CALR Parser – Parser Generators – Design of a parser generator				

Language Specification

- YACC
  - Tool which will produce a parser for a given grammar.
  - YACC (Yet Another Compiler Compiler) is a program designed to compile a LALR(1) grammar and to produce the source code of the syntactic analyzer of the language produced by this grammar
  - Input is a grammar (rules) and actions to take upon recognizing a rule
  - Output is a C program and optionally a header file of tokens

Parser Generators



Yacc

- yacc -dy filename.y
- gcc y.tab.c
- a.exe

Lex & YACC

- yacc -dy filename.y or bison -dy filename.y
- lex filename.l
- gcc lex.yy.c y.tab.c
- a.exe

Lex & YACC

- yacc -dy filename.y or bison -dy filename.y
- lex filename.l
- gcc lex.yy.c y.tab.c -o filename.exe
- filename.exe

Parser Generators

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

Figure 4.58: Yacc specification of a simple desk calculator

Parser Generators

```
%{
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double   /* double type for Yacc stack */
%}
%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
      ;
%%
yylex() {
     int c;
     while ( ( c = getchar() ) == ' ' );
     if ( (c == '.') || (isdigit(c)) ) {
          ungetc(c, stdin);
          scanf("%lf", &yylval);
          return NUMBER;
     }
     return c;
}
```

Figure 4.59: Yacc specification for a more advanced desk calculator.

THANK YOU