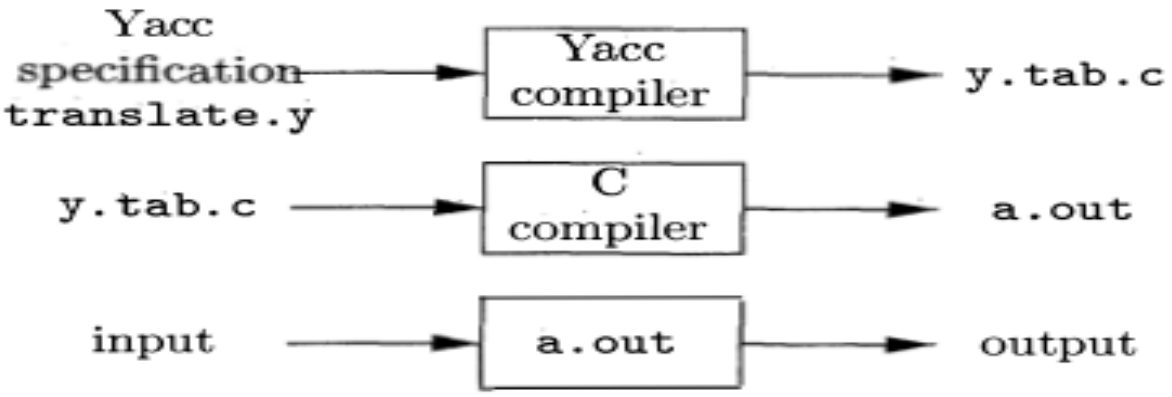
Yet Another Compiler Compiler(YACC)



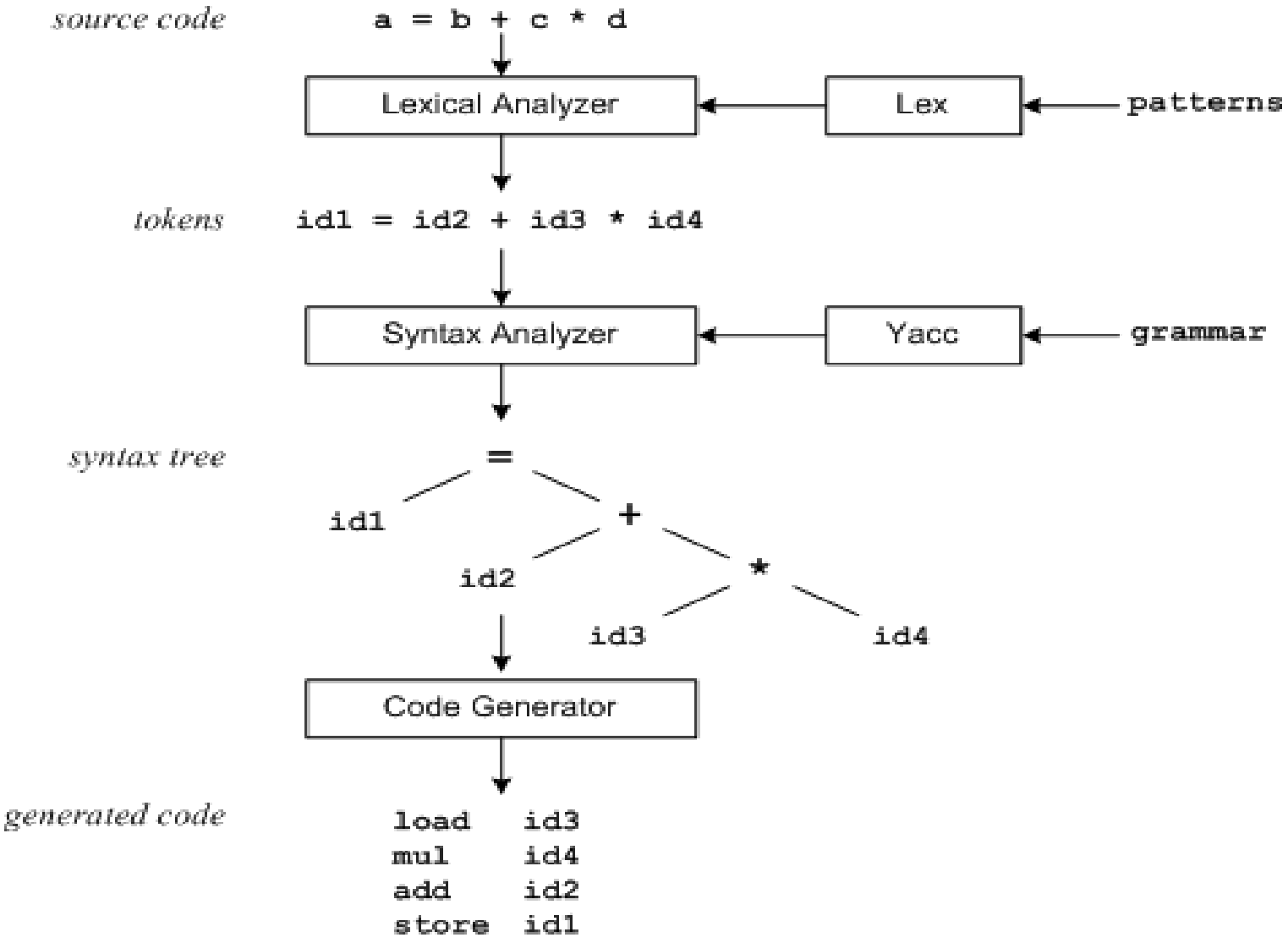
Dr. Bibhudatta Sahoo

Communication & Computing Group

CS215, Department of CSE, NIT Rourkela

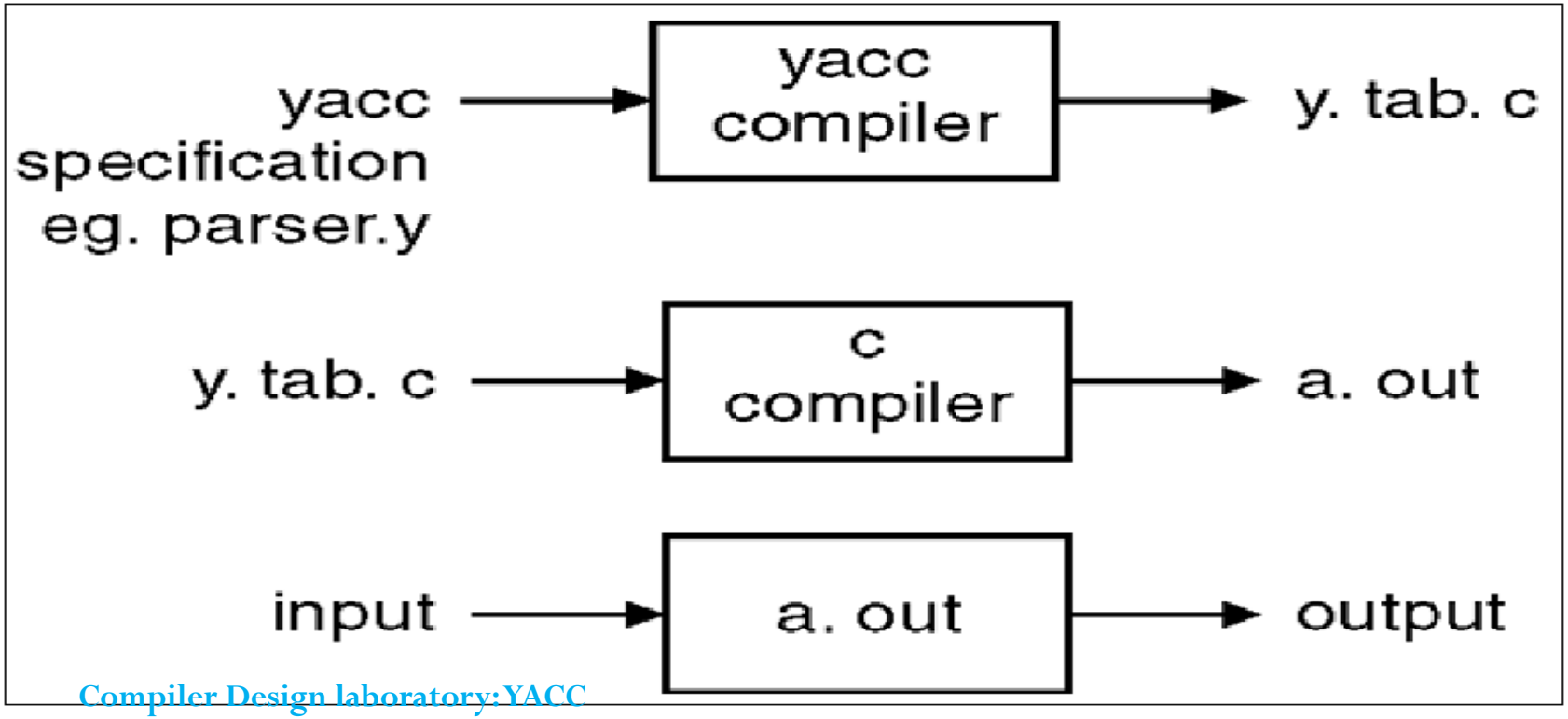
Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

Compilation Sequence

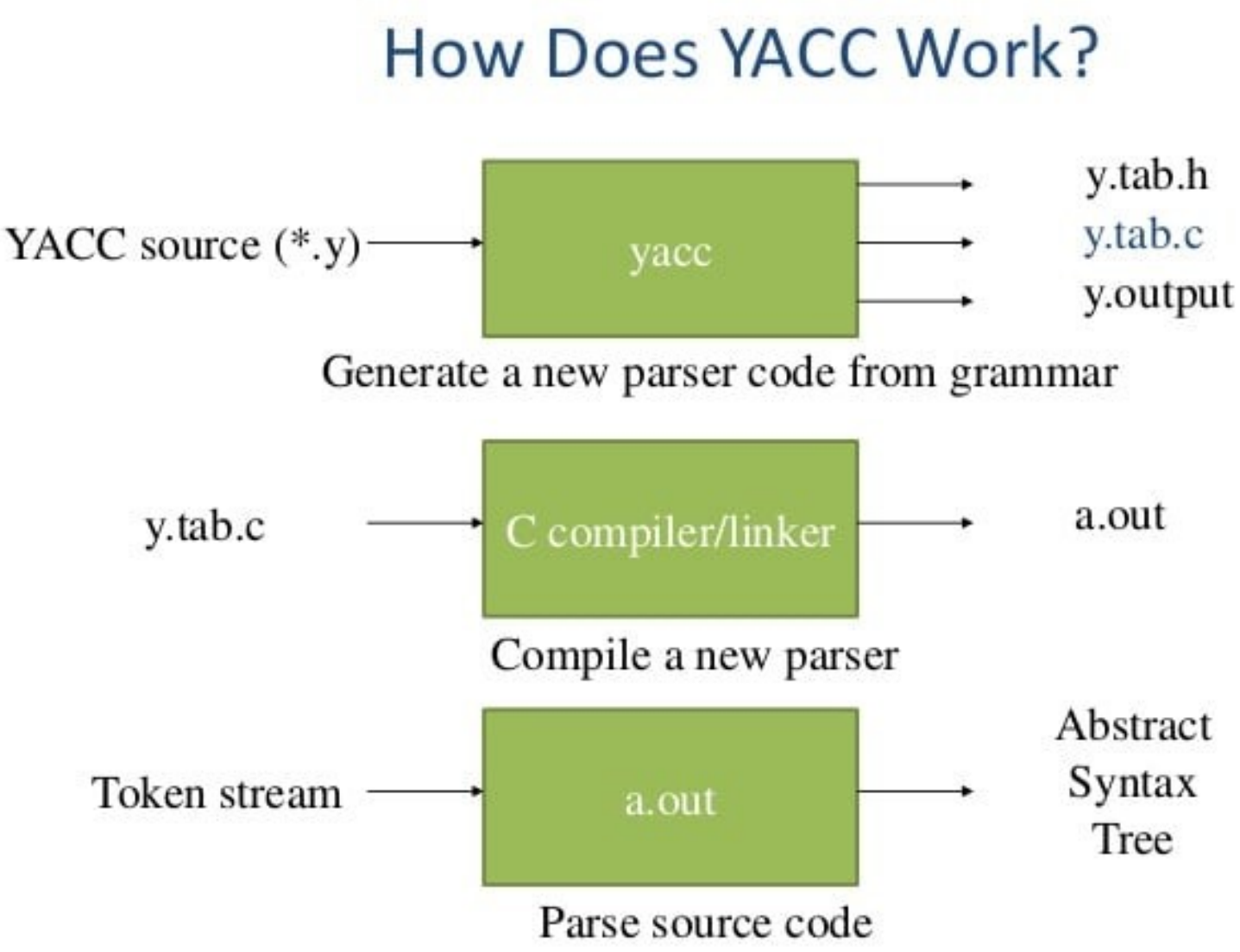


YACC

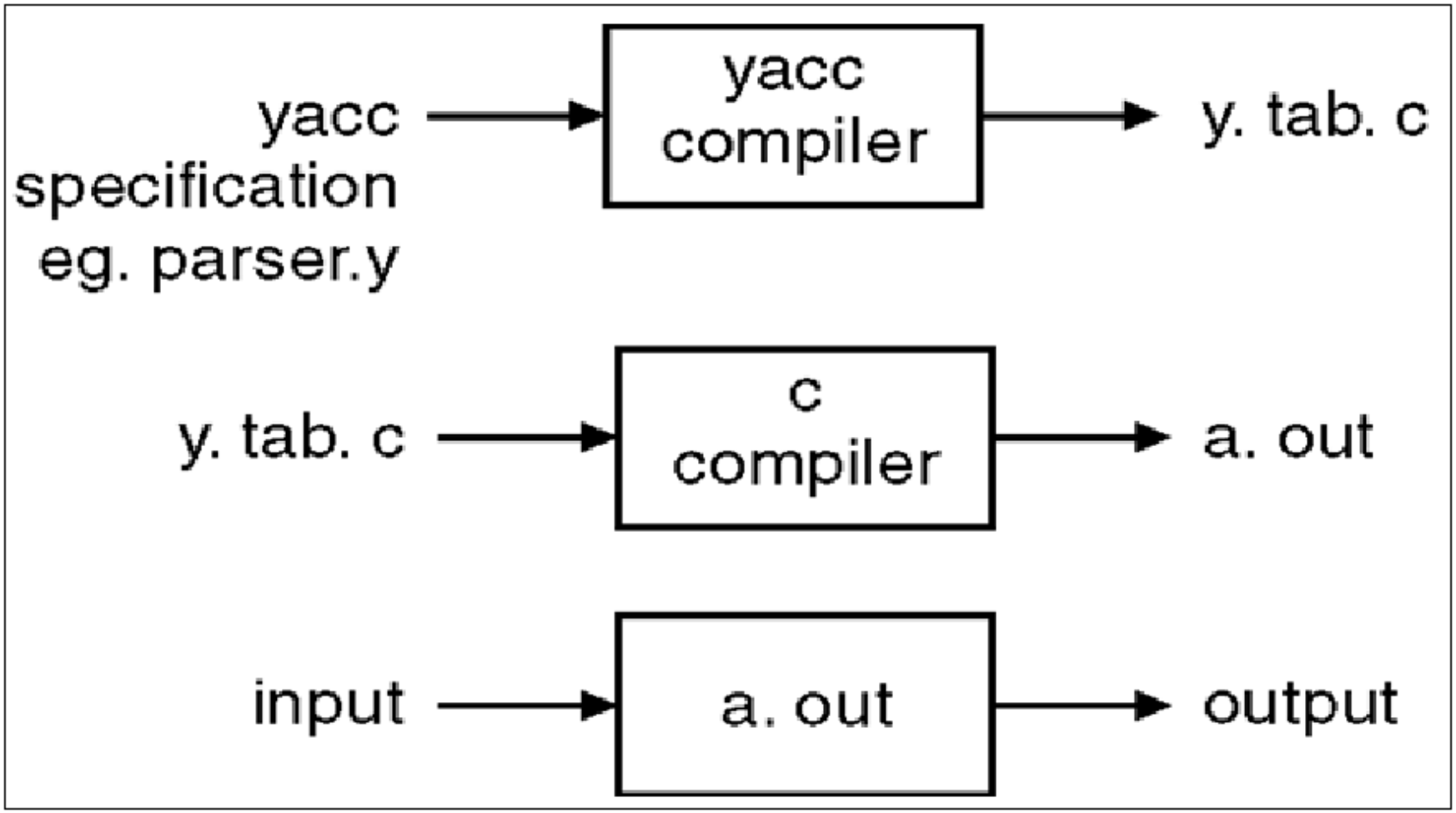
- YACC stands for **Yet Another Compiler Compiler**. YACC provides a tool to produce a parser for a given grammar.
- YACC is a program designed to compile a **LALR (1)** grammar.
- The input of YACC is the rule or grammar and the output is a C program.



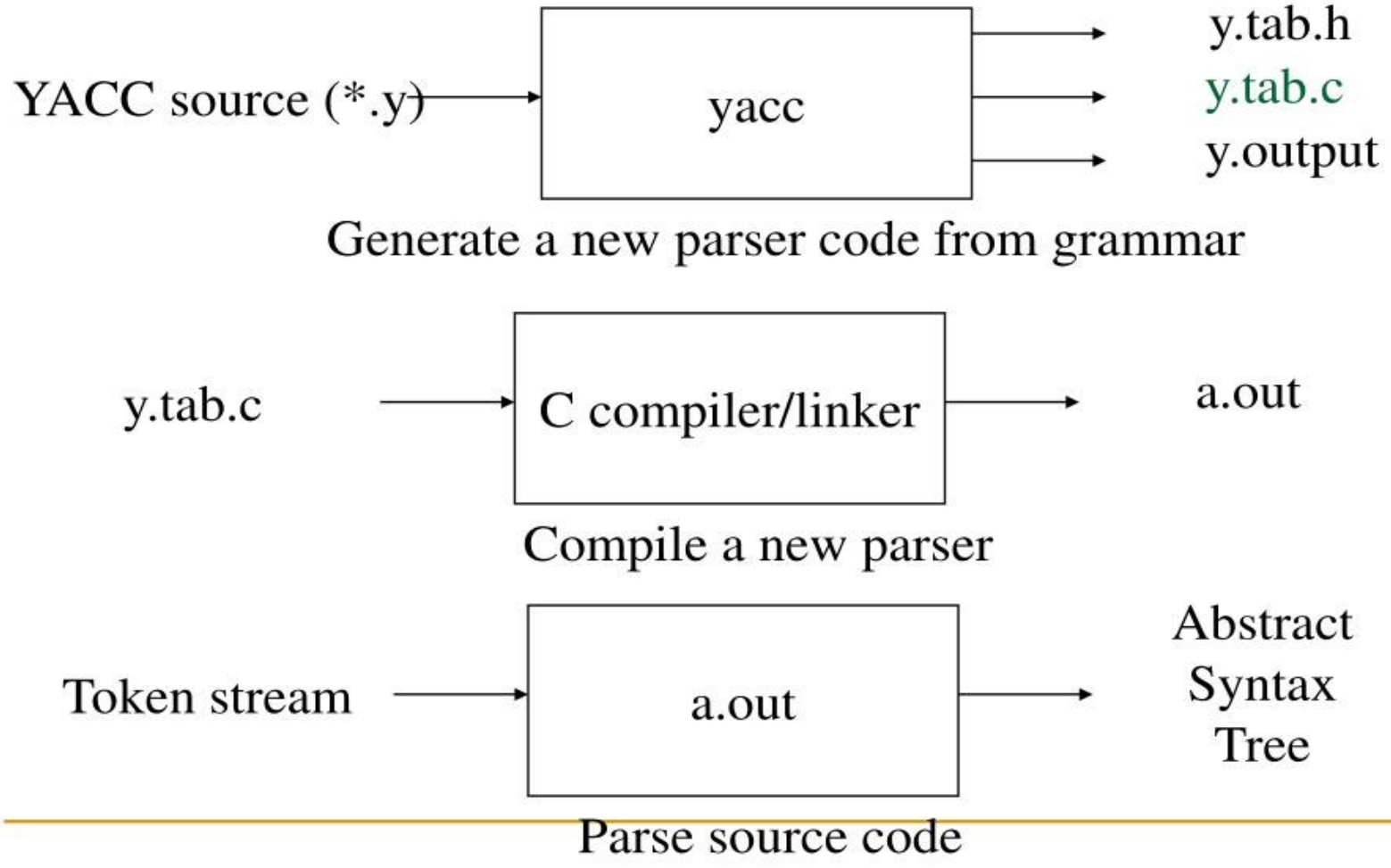
Yet Another Compiler Compiler



Yet Another Compiler Compiler

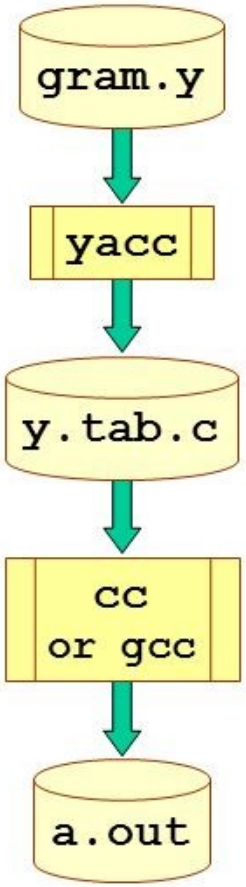


How Does YACC Work?



3

How YACC Works



File containing desired grammar in yacc format

yacc program

C source program created by yacc

C compiler

Executable program that will parse grammar given in gram.y

How yacc works

- The input to **yacc** describes the rules of a grammar. **yacc** uses these rules to produce the source code for a program that parses the grammar.
- You can then compile this source code to obtain a program that reads input, parses it according to the grammar, and takes action based on the result.
- The source code produced by **yacc** is written in the C programming language. It consists of a number of data tables that represent the grammar, plus a C function named **yyparse()**.
- By default, **yacc** symbol names used begin with **yy**. This is an historical convention, dating back to **yacc**'s predecessor, UNIX **yacc**. You can avoid conflicts with **yacc** names by avoiding symbols that start with **yy**.

How yacc works

- The yacc (yet another compiler compiler) utility provides a general tool for imposing structure on the input to a computer program.

Before using yacc, you prepare a specification that includes:

- A set of rules to describe the elements of the input
- Code to be invoked when a rule is recognized
- Either a definition or declaration of a low-level scanner to examine the input

yyparse() and yylex()

- **yyparse()** returns a value of 0 if the input it parses is valid according to the given grammar rules. It returns a 1 if the input is incorrect and error recovery is impossible.
- **yyparse()** does not do its own lexical analysis. In other words, it does not pull the input apart into tokens ready for parsing. Instead, it calls a routine called **yylex()** everytime it wants to obtain a token from the input.
- **yylex()** returns a value indicating the *type* of token that has been obtained. If the token has an actual *value*, this value (or some representation of the value, for example, a pointer to a string containing the value) is returned in an external variable named **yylval**.
- It is up to the user to write a **yylex()** routine that breaks the input into tokens and returns the tokens one by one to **yyparse()**

For Compiling YACC Program:

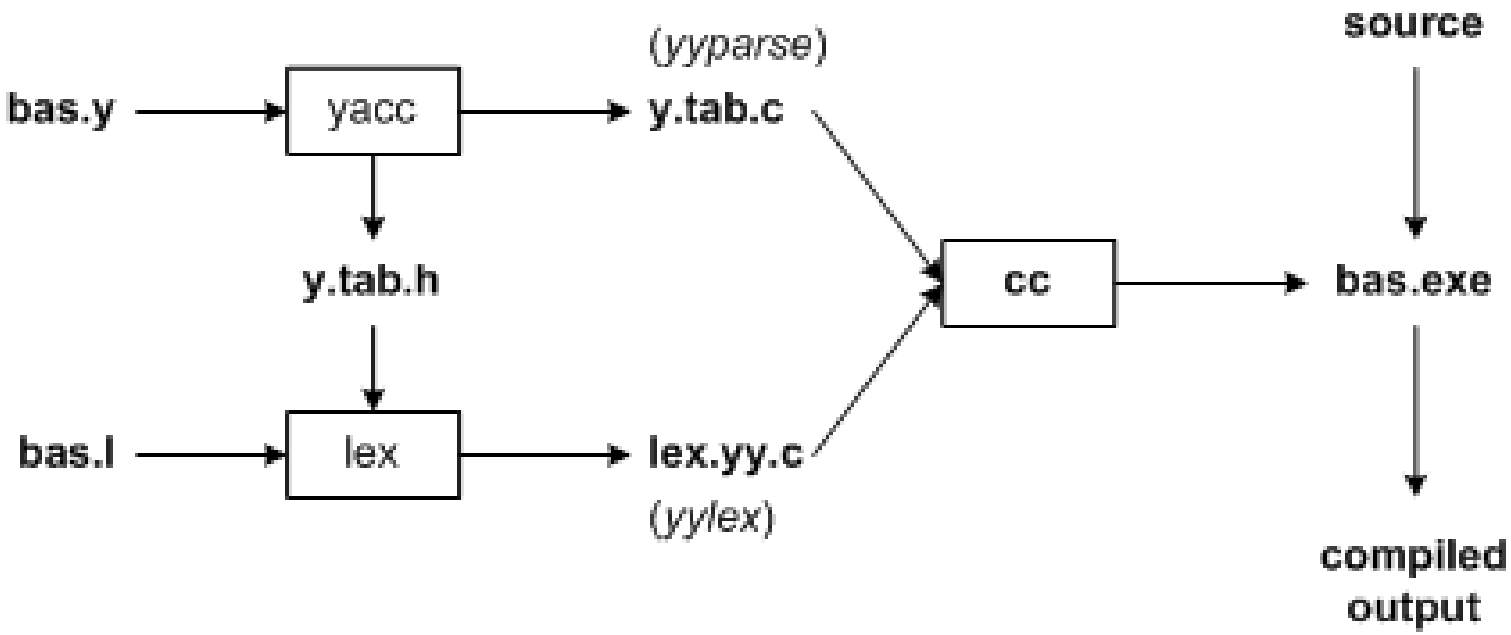
- Write lex program in a file **file.l** and yacc in a file **file.y**
- Open Terminal and Navigate to the Directory where you have saved the files.
- type lex file.l
- type yacc file.y
- type cc **lex.yy.c** **y.tab.h** -ll
- type ./a.out

YACC input file is divided into three parts

YACC input file is divided into three parts.

```
/* definitions */
 ....
%%
/* rules */
 ....
%%
/* auxiliary routines */
....
```

Building a Compiler with Lex & Yacc



Commands to create our compiler, bas.exe, are listed below:

```
yacc -d bas.y # create y.tab.h, y.tab.c

lex bas.l # create lex.yy.c

cc lex.yy.c y.tab.c -obas.exe # compile/link
```

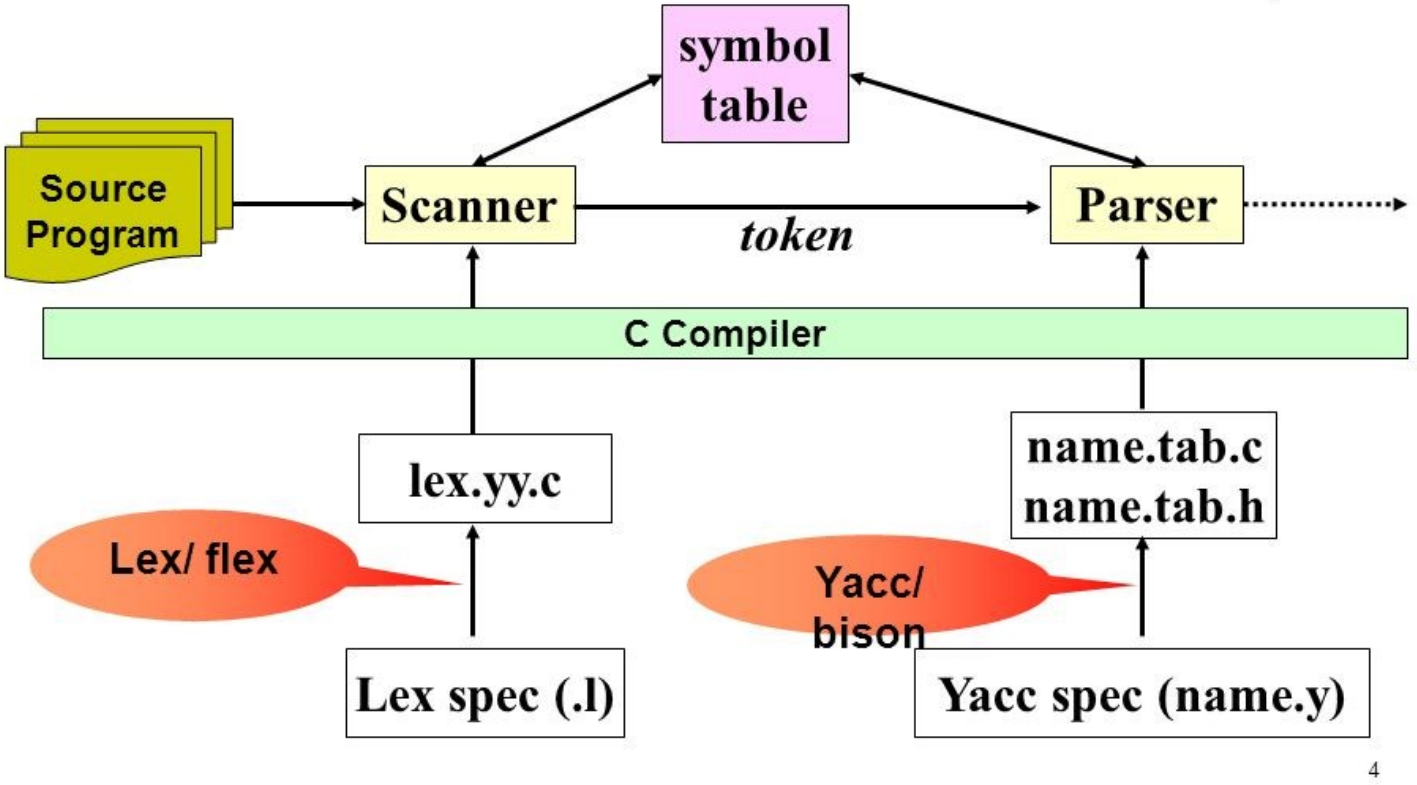
Lex File (.l)

```
%{
%}


%%
[ \t]       { /* skip blanks and tabs */ }
\n|.        { return yytext[0]; }
%%
```

Yacc File (.y)

```
%{
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double /* double type for yacc stack */
%}

%%
Lines : Lines S '\n' { printf("OK \n"); }
                | S '\n'
                | error '\n' {yyerror("Error: reenter last line:");
                                                                    yyerrok; };
S               : '(' S ')'
                | '[' S ']'
                | /* empty */ ;
%%

#include "lex.yy.c"

void yyerror(char * s)
/* yacc error handler */
{
fprintf (stderr, "%s\n", s);
}

int main(void)
{
return yyparse();
}
```

Scanner, Parser, Lex and Yacc



Grammar for arithmetic expression

Problem 1 *Given the following grammar for simple integer arithmetic expressions:*

$$
\begin{aligned}
&expr \rightarrow expr \ + \ term \ \mid \ term \\
&term \rightarrow term \ * \ factor \ \mid \ factor \\
&factor \rightarrow (\ expr \ ) \ \mid \ number \\
&number \rightarrow number \ digit \mid digit \\
&digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

draw parse trees and abstract syntax trees for the arithmetic expressions:

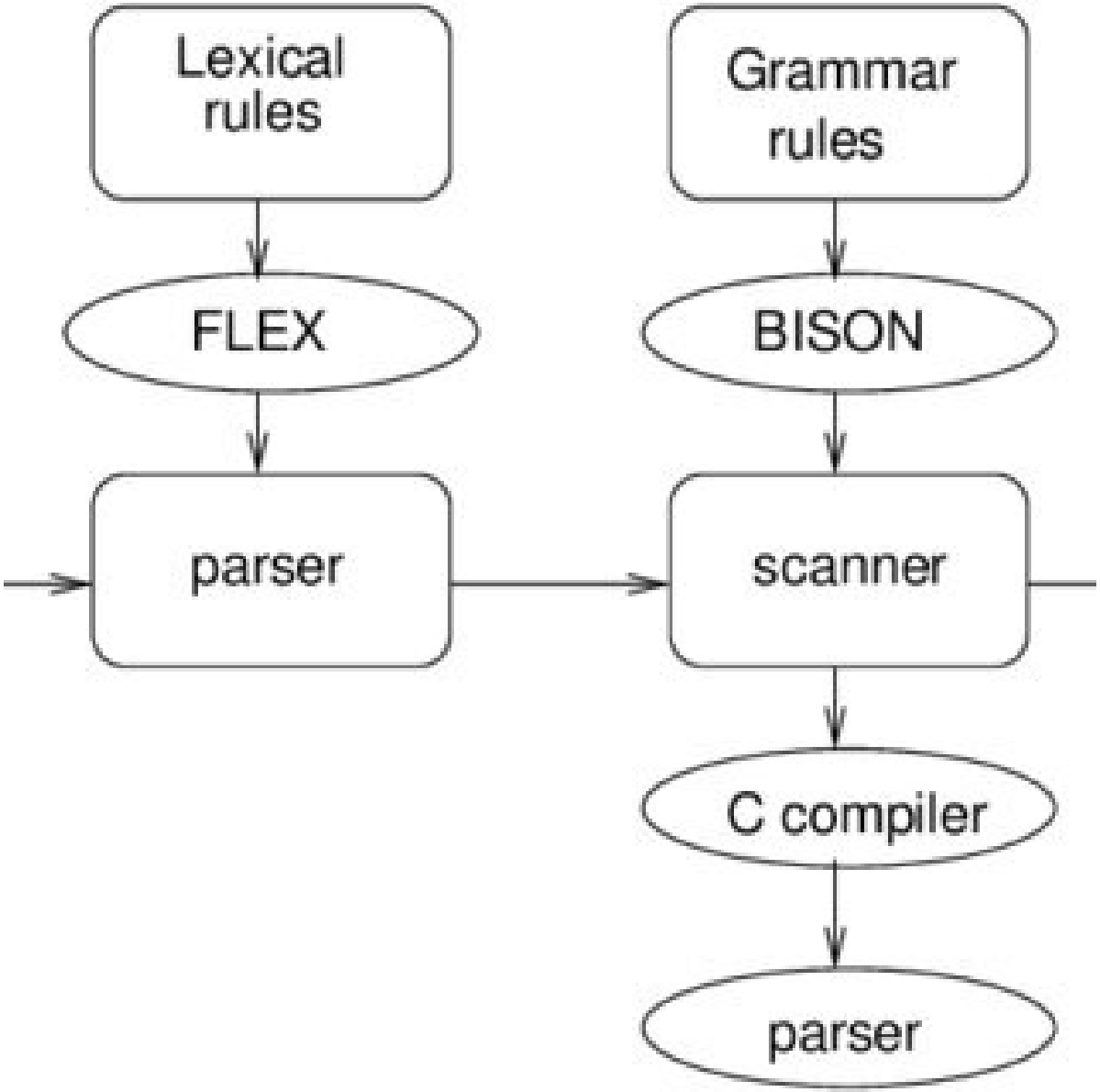
(A) 1+2*3+6*7
(B) 1*2+3*4+5
(C) 1*(2+3)*(4+5)
(D) (1+(2+(3+4)))

- When LEX and YACC work together lexical analyzer using yylex () produce pairs consisting of a token and its associated attribute value.
- If a token such as DIGIT is returned, the token value associated with a token is communicated to the parser through a YACC defined variable yylval.
- We have to return tokens from LEX to YACC, where its declaration is in YACC. To link this LEX program include a y.tab.h file, which is generated after YACC compiler the program using –d option.

Steps to Execute the program

- $ lex filename.l (eg: cal.l)
- $ yacc -d filename.y (eg: cal.y)
- $gcc lex.yy.c y.tab.c
- $./a .out

Structure of the parser developed with BISON and FLEX.



Lex VERSUS Yacc

Lex	Yacc
Computer program that operates as a lexical analyzer	Parser that is used in Unix Operating System
Developed by Mike Lex and Eric Schmidt	Developed by Stephan C. Johnson
Reads the source program one character at a time and converts it into meaningful tokens	Takes the tokens as input and generates a parse tree as output

Exercises

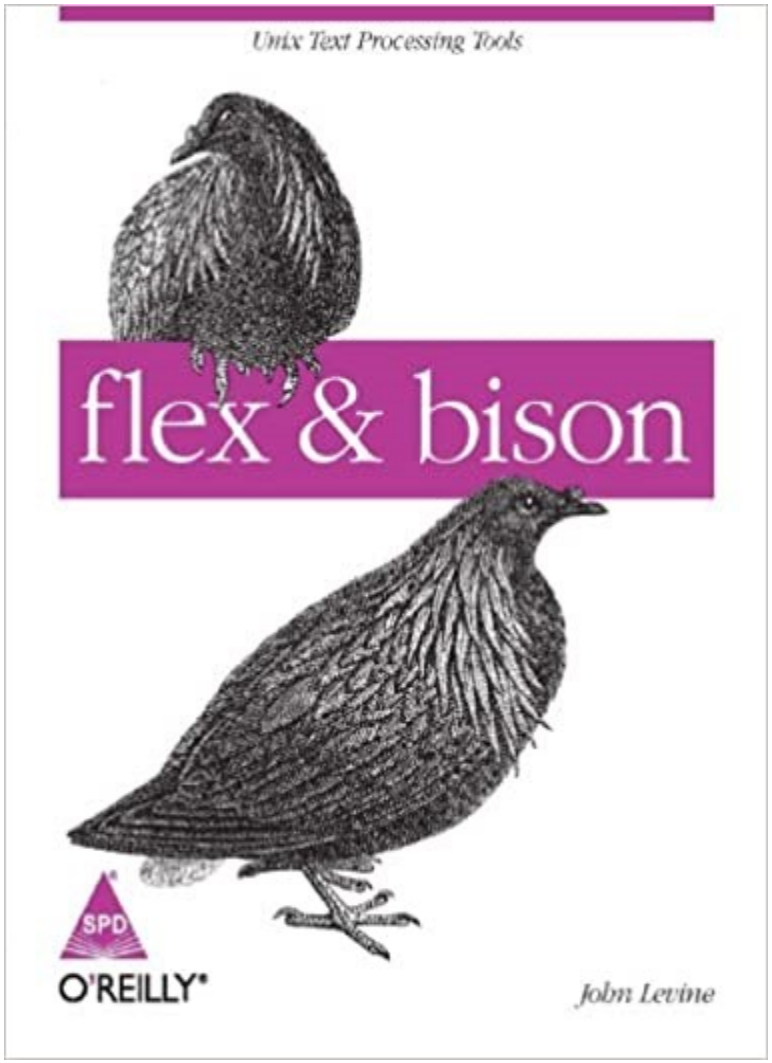
1. Write a YACC program to check whether the given grammar is valid or not. Consider an input expression and convert it to postfix form.

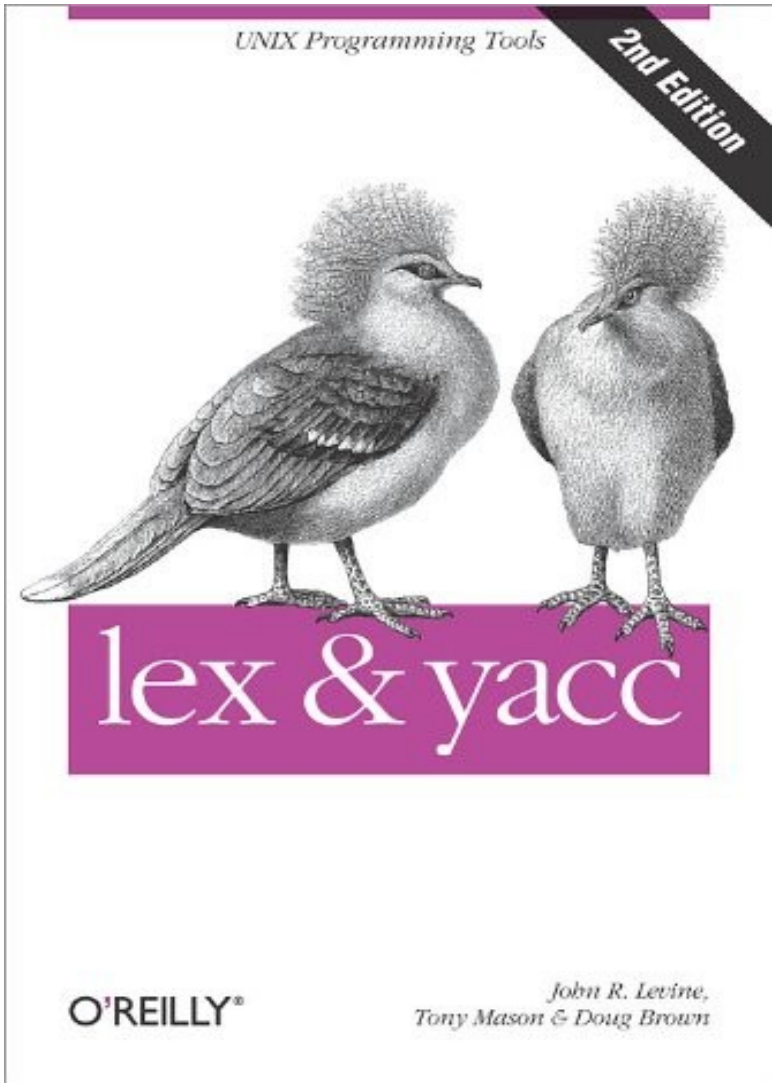
2. Write a program to Implement YACC for Subset of C (for loop) statement.

Reference

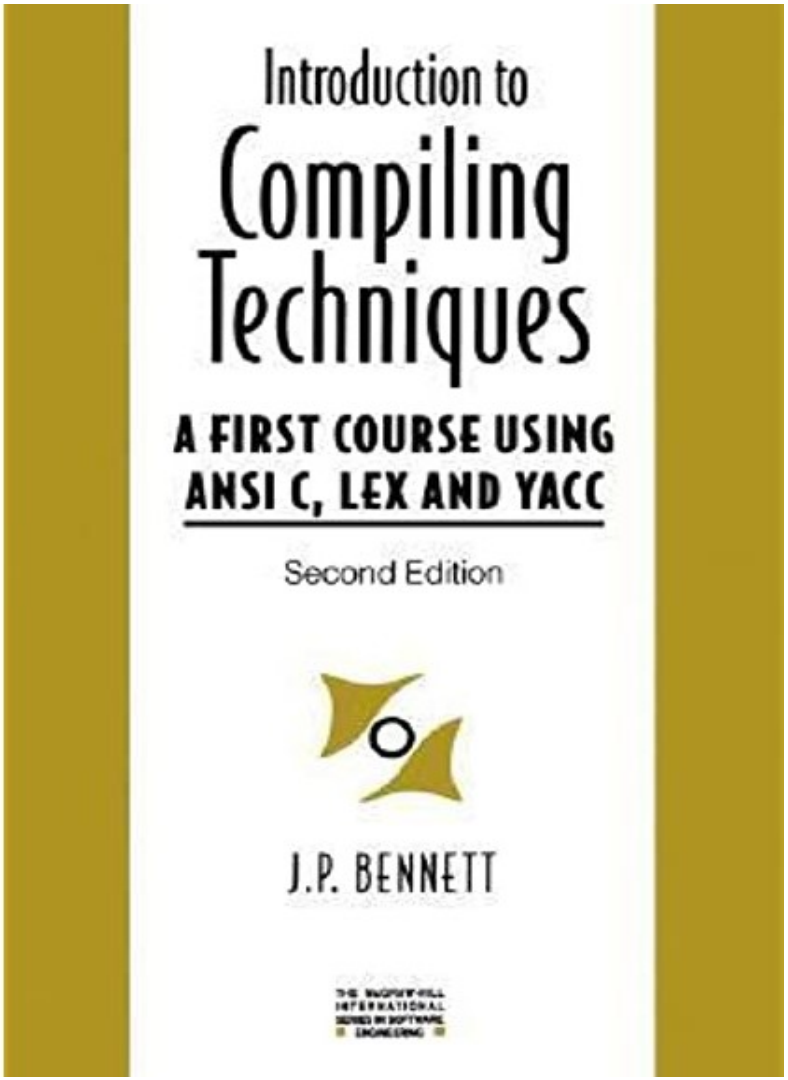
- https://milylike.wordpress.com/syntax-analyzer/lex/
- https://www.epaperpress.com/lexandyacc/intro.html
- https://www.epaperpress.com/lexandyacc/pry1.html
- https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dgt/index.html
- https://www.epaperpress.com/lexandyacc/calc1.html

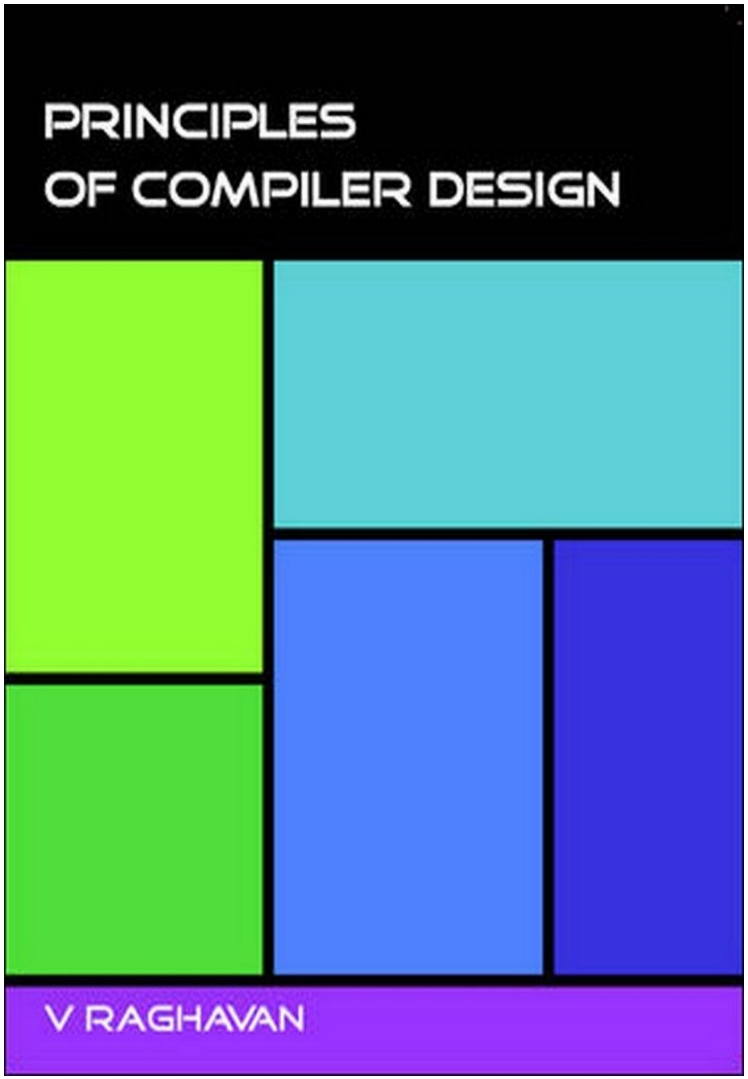
Reference Book



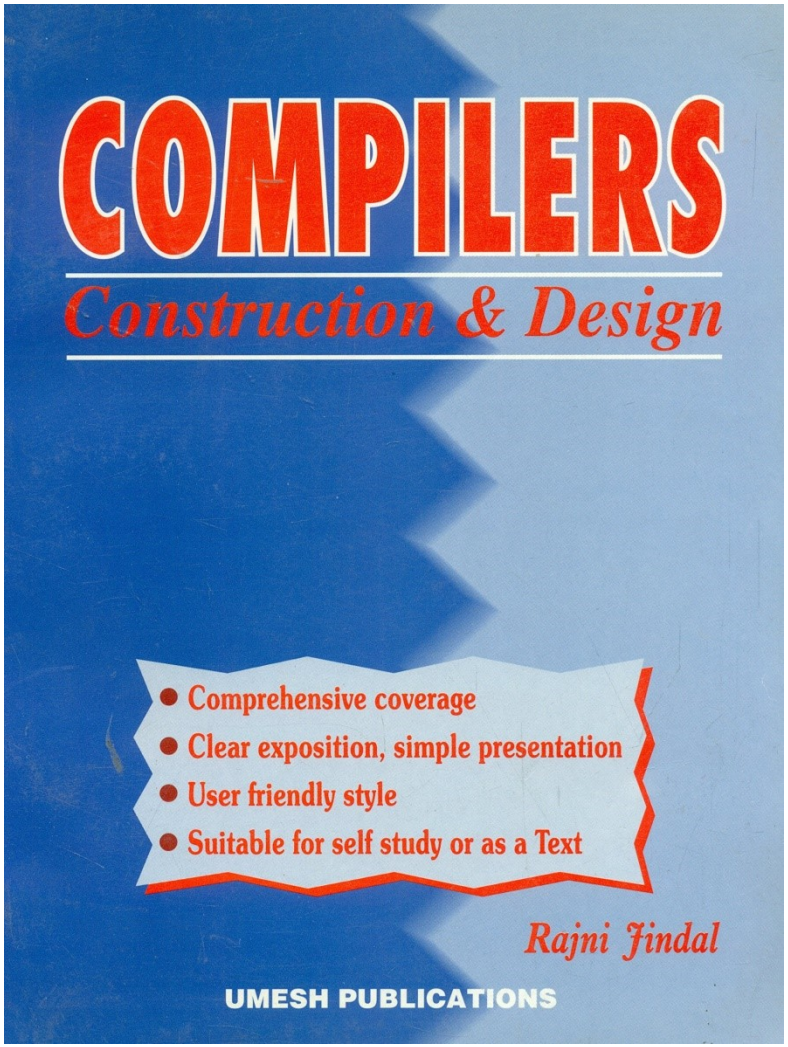


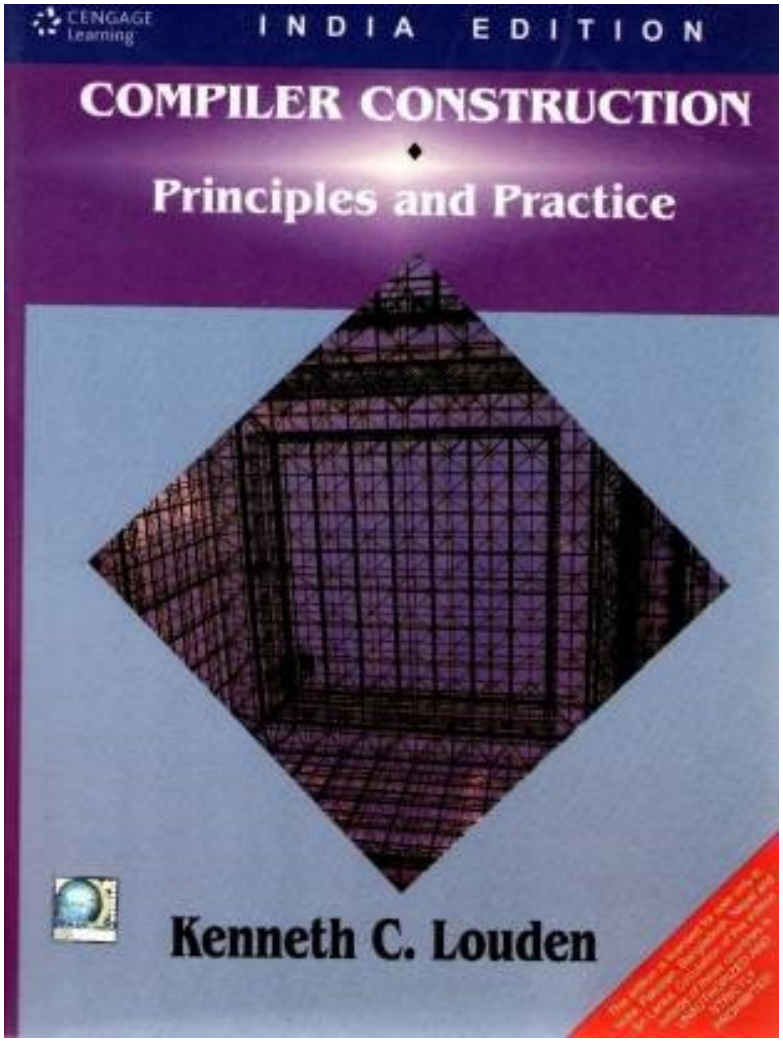
Reference Book



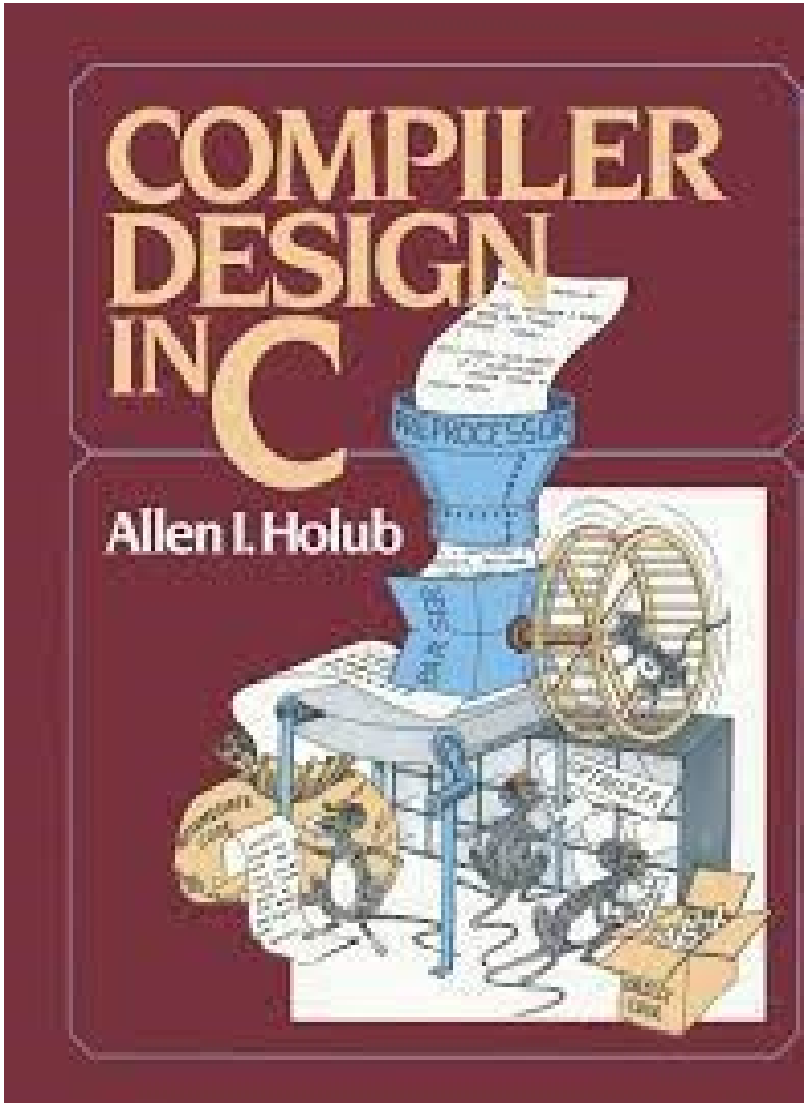


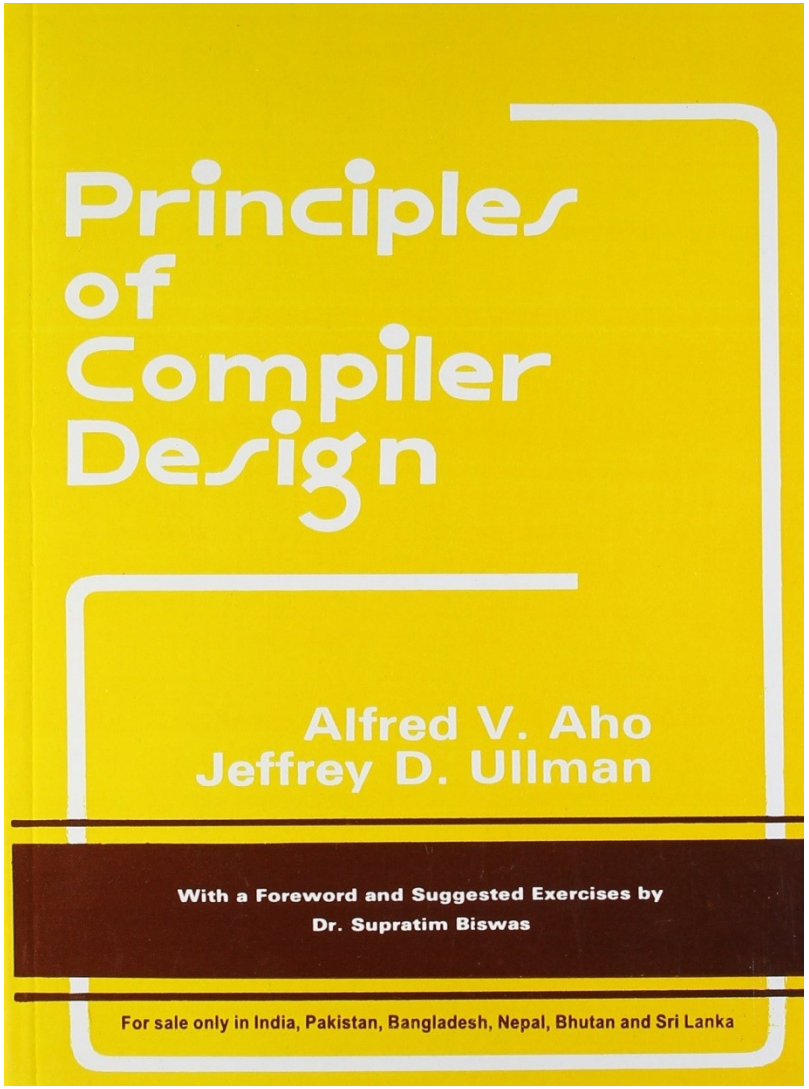
Reference Book



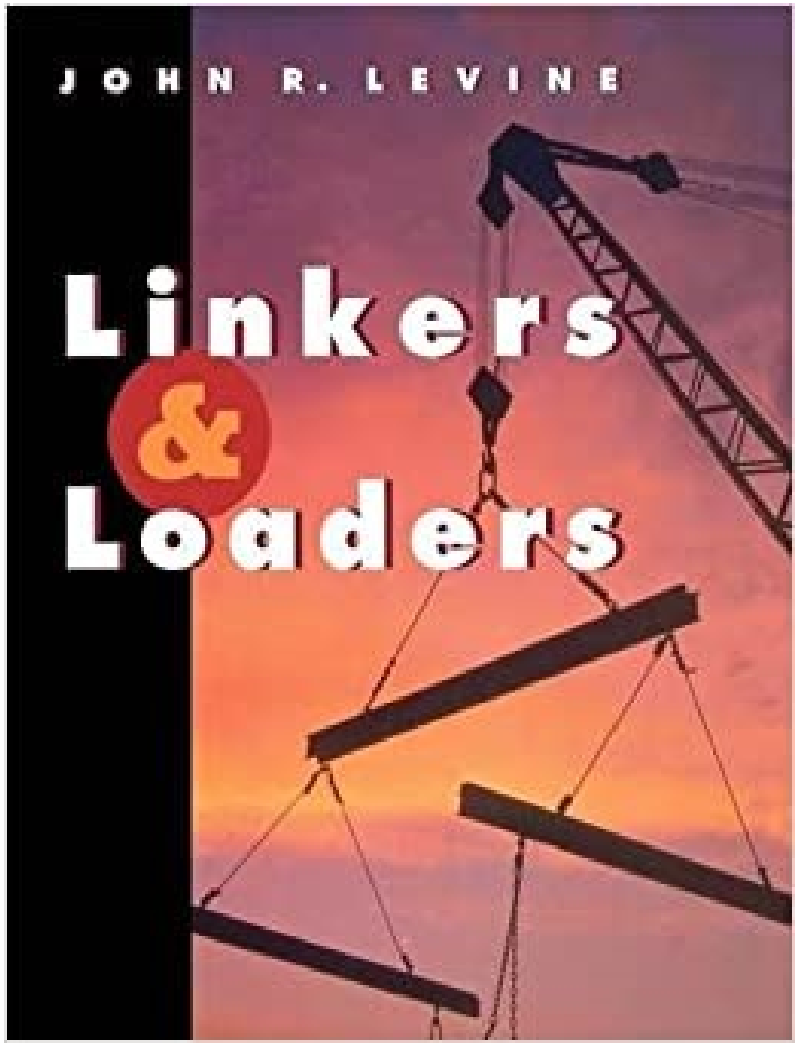


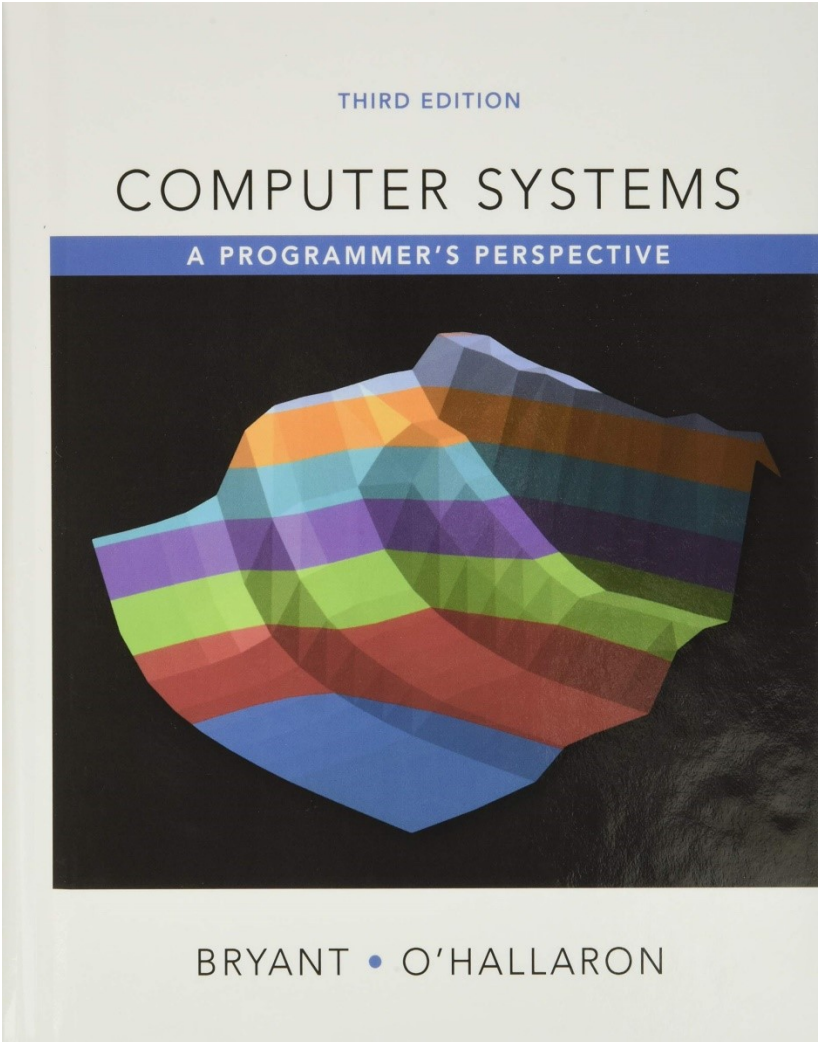
Reference Book



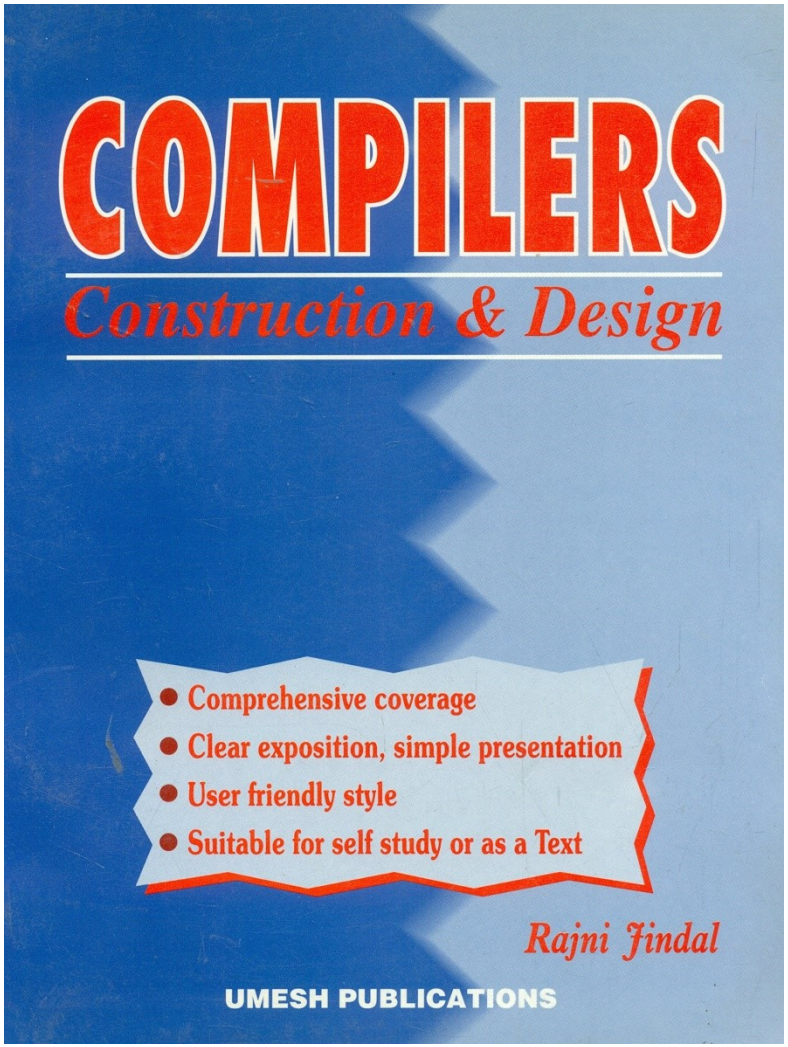


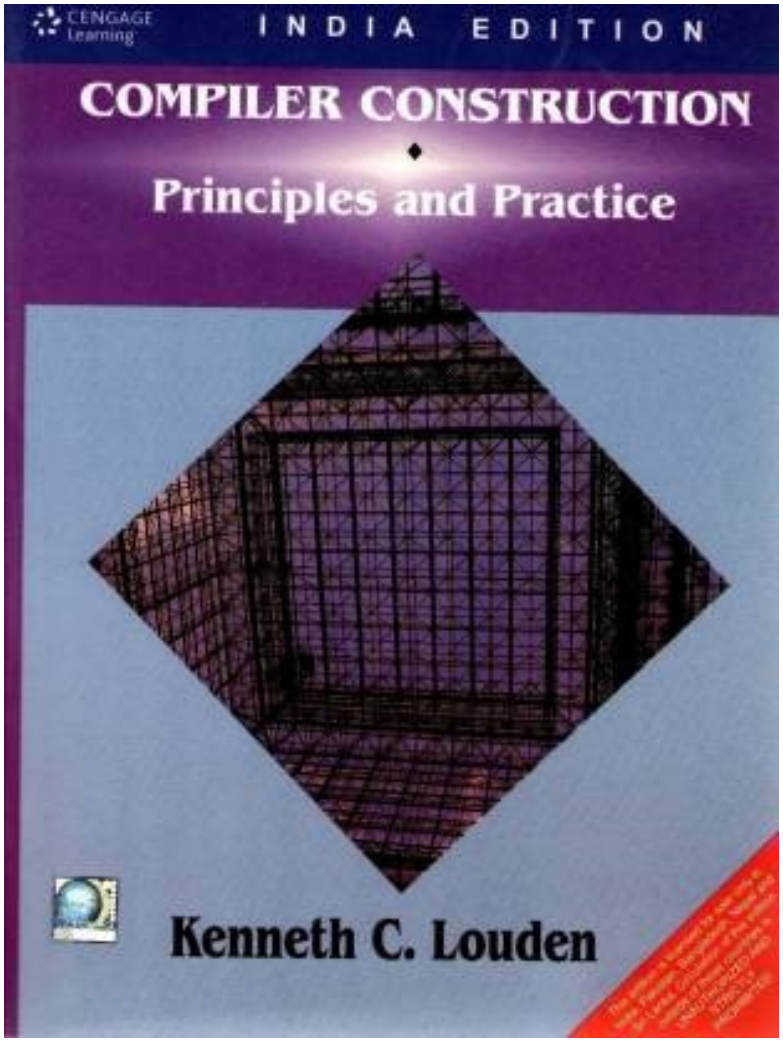
Reference Book



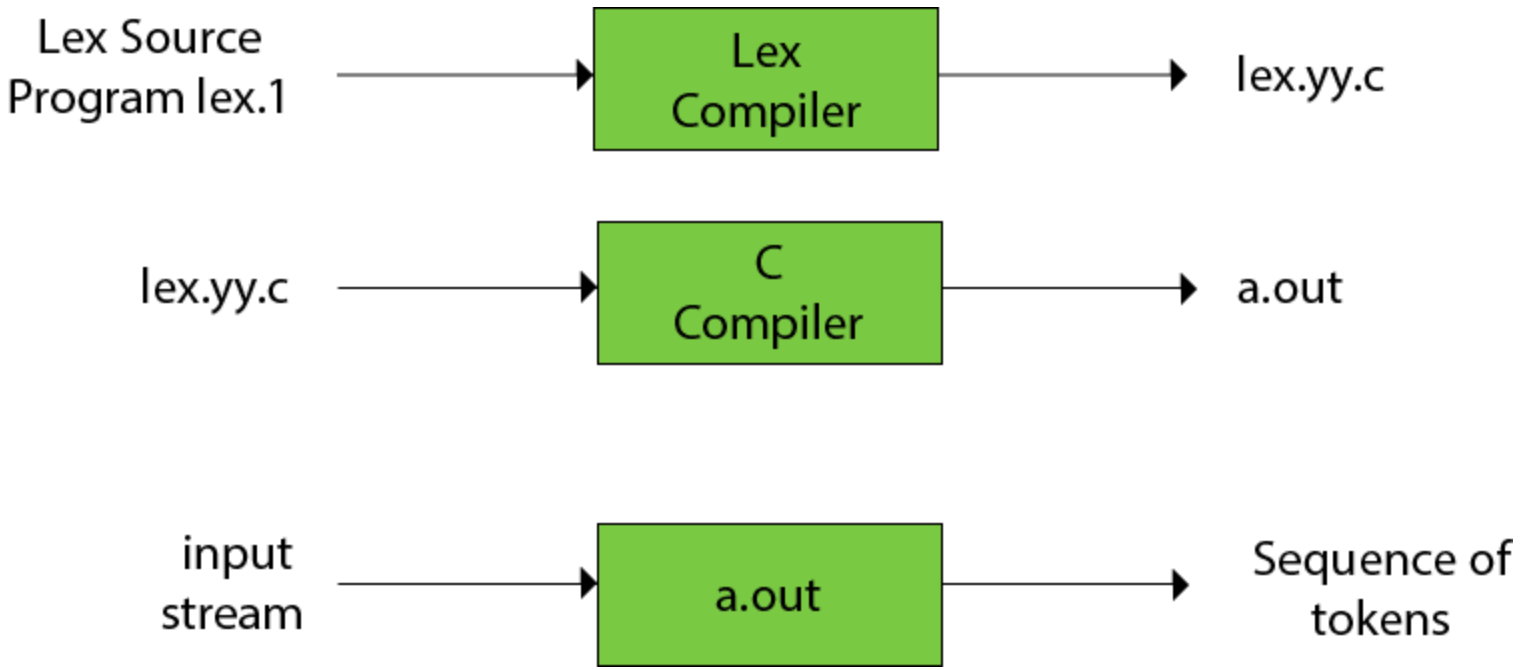


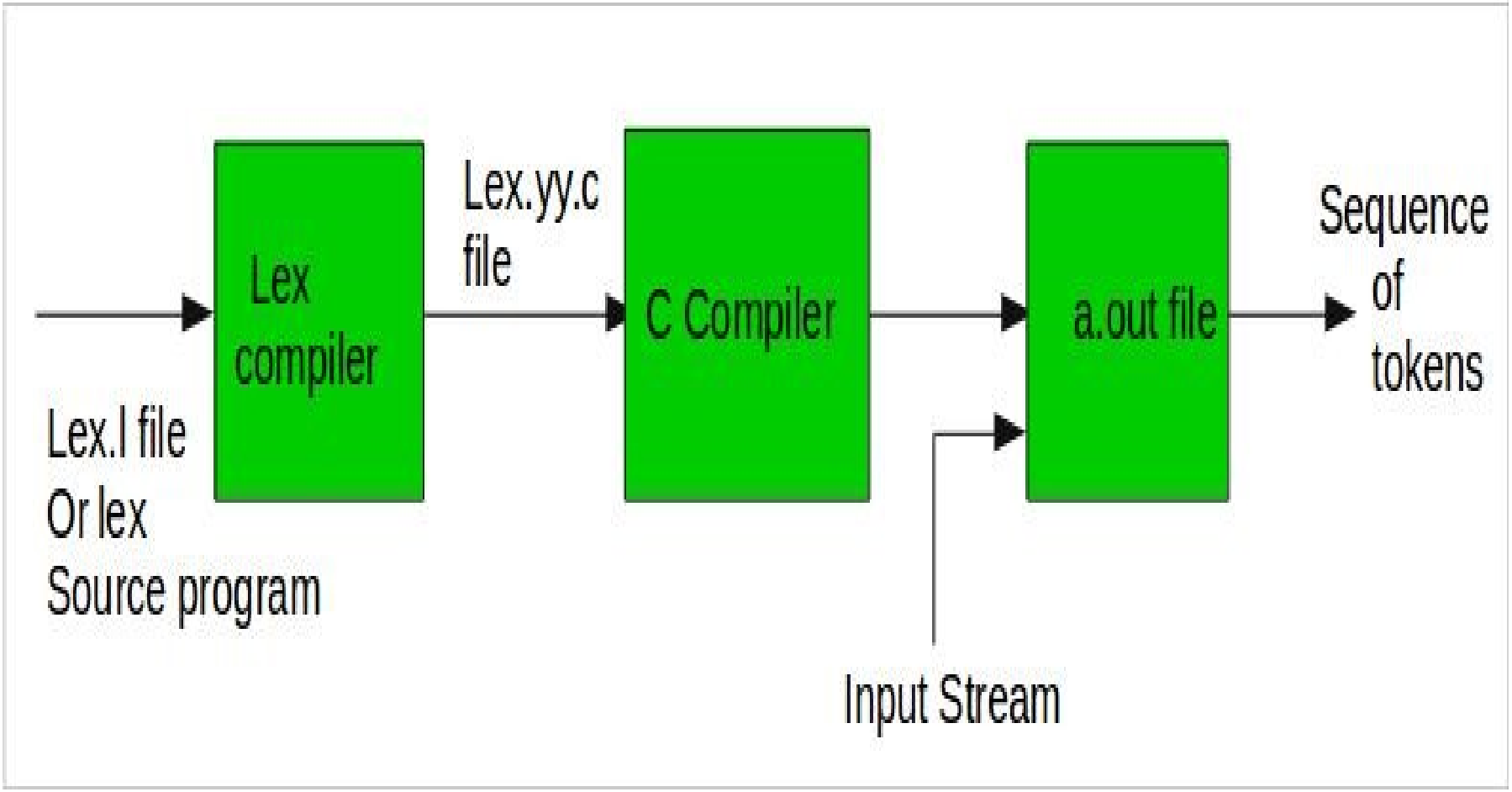
Reference Book





Working with Lex




Lex
compiler
Lex.yy.c
file
C Compiler
a.out file
Sequence
of
tokens
Lex.l file
Or lex
Source program
Input Stream