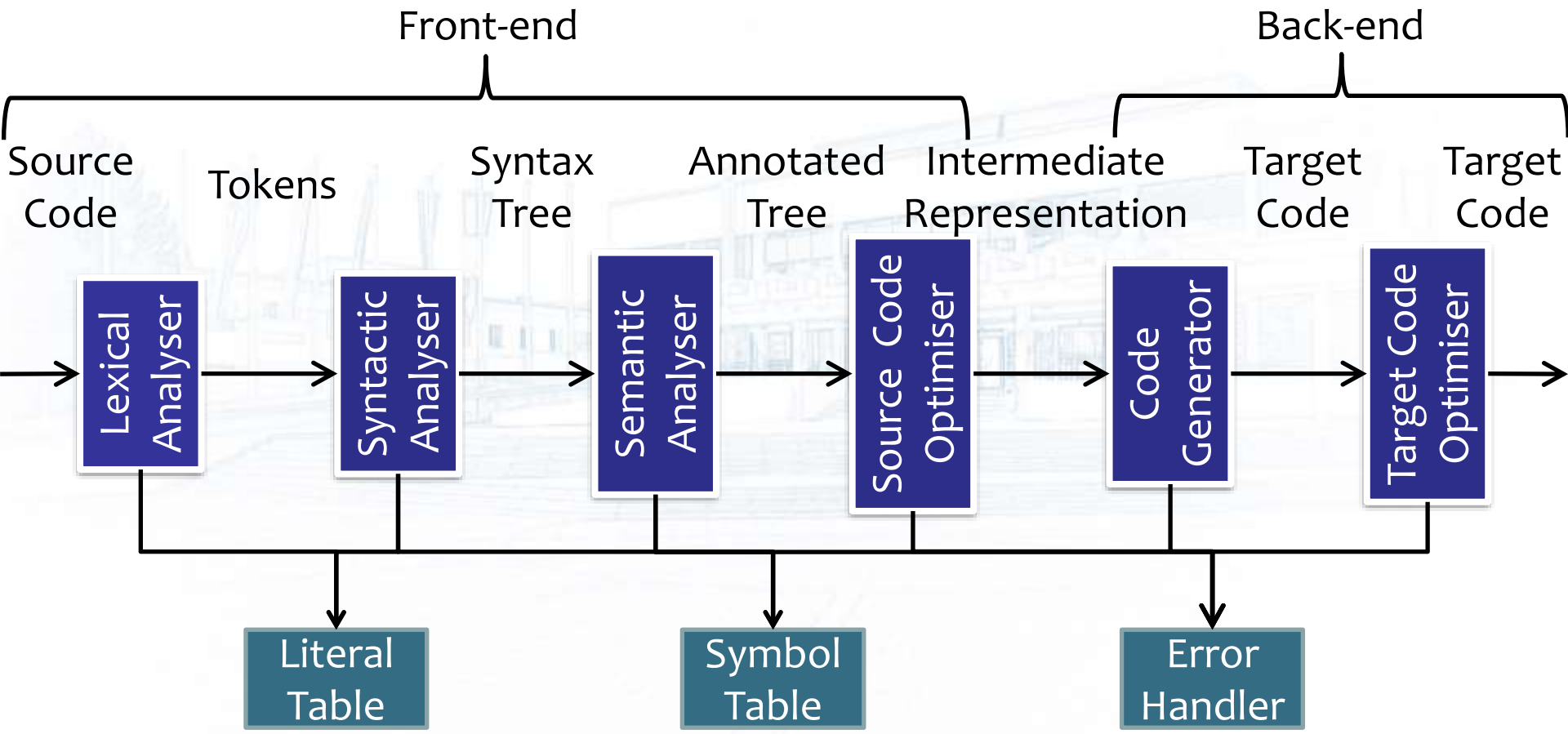
Prof. Radu Prodan

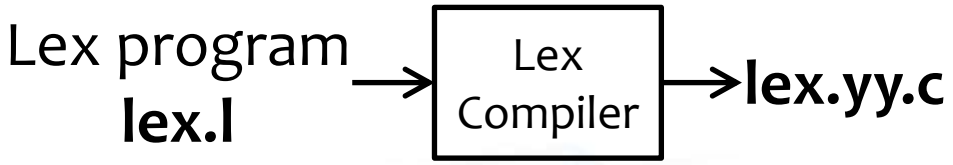
LEX
LEXICAL ANALYSER GENERATOR

Phases of a Compiler

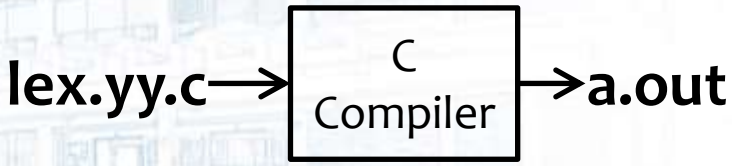


Lex

- Lexical analyser generator
  - **flex** (Fast Lex) is the most popular GNU implementation



- Input
  - A text file with regular expressions
  - Actions to be taken when each expression is matched
  - *Pattern { Action }*



- Output
  - Lexical analysis code in C
    - **`lex.yy.c`** or **`lexyy.c`**
  - **`yylex`** procedure
    - Table driven DFA implementation of the regular expressions

Lex Regular Expression Operators

Regular Expression	Meaning
x	The character x
“x”	An “x”, even if x is an operator
\x	An “x”, even if x is an operator
[xy]	The character x or y
[x-z]	The characters x, y or z
[^x]	Any character but x
.	Any character but newline
^x	An x at the beginning of a line
<y>x	An x when lex is in start condition y
x$	An x at the end of a line
x?	An optional x

Lex Regular Expression Operators

Regular Expression	Meaning
`x*`	zero or more instances of x
`x+`	one or more instances of x
`x|y`	an x or a y
`(x)`	an x
`x/y`	an x but only followed by a y
`{xx}`	the translation of xx from the definitions section
`x{m,n}`	*m* through *n* occurrences of x

Format of a Lex Input File

%{
C code external to any function
%}
Definition of names for regular expressions

%%
Pattern { Action } Rules

- Regular expressions
- C code to be executed upon matching

%%
Auxiliary routines (optional)

- Called in the previous section and not defined elsewhere
- Main function if stand-alone program

```
%{
/* This example counts and displays the
    number of words of an input file */

#include <stdio.h>

int words = 0;
%}

whitespace                [ \t\n]

%%

[^{whitespace}]+          words++;
[{whitespace}]+

%%

main(int argc, char **argv) {
  yylex();
  printf("\nNumber of words: %d\n", words);
  return 0;
}
```

Important Lex Internal Names

Lex Internal Name	Meaning / Use
`int yylex(void)`	Lex scanning routine
`char *yytext`	String matched on current action (lexeme)
`int yyleng;`	Length of the current token
`FILE *yyin`	Lex input file stream (default **`stdin`**)
`FILE *yyout`	Lex output file stream (default **`stdout`**)
`yyrestart(FILE *new_file)`	Point **`yyin`** to a new input file stream
`yyterminate()`	Terminate the scanner and return 0
`input()`	Reads the next character from input stream
`unput(char)`	Puts a character back into the input stream
`yymore()`	Append next token to **`yytext`** (instead of overwrite)
`yyless(n)`	Return all but first *n* character to input stream for rescanning
ECHO	Write **`yytext`** to **`yyout`**
INITIAL	Initial start condition
BEGIN condition	Switch start condition
REJECT	Match the “second best” rule
<<EOF>>	End of file

Exercise

- `lex -oyour_scanner.c your_scanner.l`
  - **`lex`** is an alias to **`flex`** on most UNIX systems
  - Produces by default **`lex.yy.c`** output file
- `gcc -o your_scanner your_scanner.c -lfl`
  - Needs to link against `libfl.a` runtime library
- Execute your program counter example
  - **`your_scanner input_file`**
- Documentation
  - **`man lex`**
  - http://flex.sourceforge.net/manual/