LEX

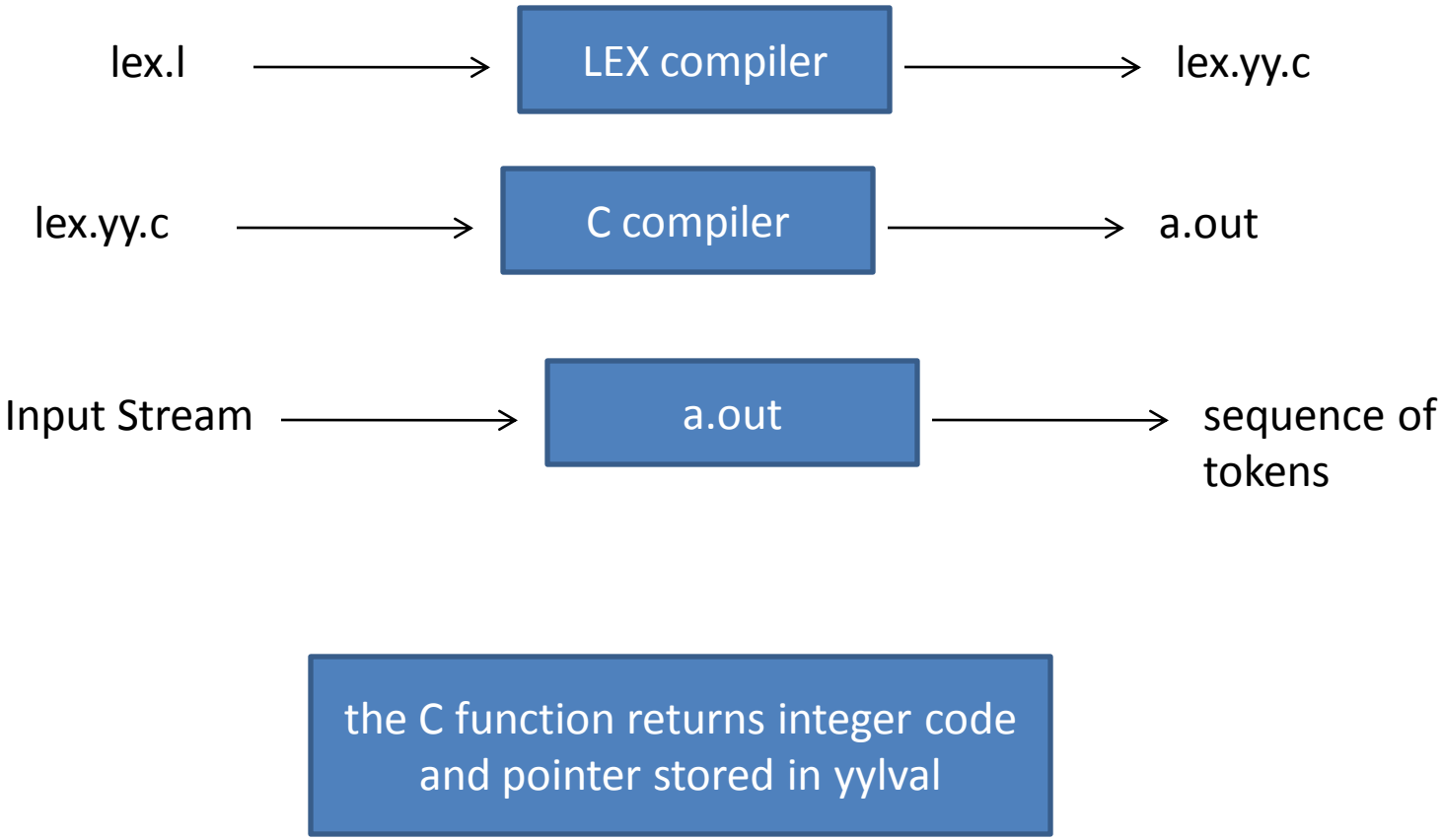
A Language for specifying Lexical Analyzers

- A lexical analyzer can be built using a specification of tokens in the form of a list of regular expressions
- A LEX source program is a specification of a lexical analyzer
  - Consisting of a set of regular expression together with action for each regular expression
  - The action is a piece of code which is to be executed whenever a token specified is recognized
  - An action pass an indication of the token found to the parser(making a symbol table entry)

A Language for specifying Lexical Analyzers

- The output of a LEX is a lexical analyzer program constructed from the LEX source specification
- LEX provide a program that simulates a finite automaton
  - This program takes a transition table as data
  - The transition table is that portion of LEX's output that are derived from the LEX's input

Creating a lexical analyzer with Lex



LEX specification

- Any LEX program consists of three parts

  d*eclarations*
  %%
  rules
  %%
  sub routines

LEX specification

- declarations- creates an environment in two areas
  - First for the Lexer which is a C code
    - separated by %{ and %}
    - contains C statements such as global declarations, commands, library files, variables, constants etc
    - LEX copies this section as it is
  - Secondly for regular definitions and manifest constants (identifiers declared to stand for a constant. e.g the name of a token)

Regular Definitions/Auxiliary Definitions

- They are statements of the form

$$
\begin{aligned}
D_1 &= R_1 \\
D_2 &= R_2 \\
&\vdots \\
D_n &= R_n
\end{aligned}
$$

Regular Definitions

- where each D_i is a distinct name
- each R_i is a regular expression whose symbols are chosen from $\Sigma \cup \{D_1, D_2, \ldots D_{i-1}\}$, i.e characters or previously defined names

Example of a Auxiliary Definitions

- Letter = A|B|C…|Z|a|b|…|z
- Digit = 0|1|…|9
- Id = letter(letter|digit)*

Translation Rules

- They are of the form

P_1 $\{A_1\}$
P_2 $\{A_2\}$
.
.
.
P_m $\{A_m\}$

Translation Rules cont…

- where each P_i is a RE called a *pattern*, over the alphabet of Σ and the auxiliary definition names
- the patterns describe the form of tokens
- Each A_i is a program fragment describing what action the lexical analyzer should take (C code) when token P_i is found

Behavior of LEX

1. L reads its input, one character at a time until it has found the longest prefix of the input which matches one of the regular expression say P_i
2. L removes it from the input and places it in a buffer called TOKEN
    1. TOKEN may be a pair of pointers to the beginning and end of the matched string
3. L executes the action A_i
4. L returns to the parser

Behavior of LEX

- If none of the regular expression matches the prefix, then
  - an error is detected amd L transfer the control to the Error handler
- If two or more pattern match the same longest prefix, then
  - L takes the first one that occurs in the list of Translation Rules

Lookahead Operator

- It is required since in certain programming language require a lookahead feature in order to specify their lexical analyzer correctly
- In FORTRAN
- DO 10 I =1.24
  - not a do statement but actually an assignment statement