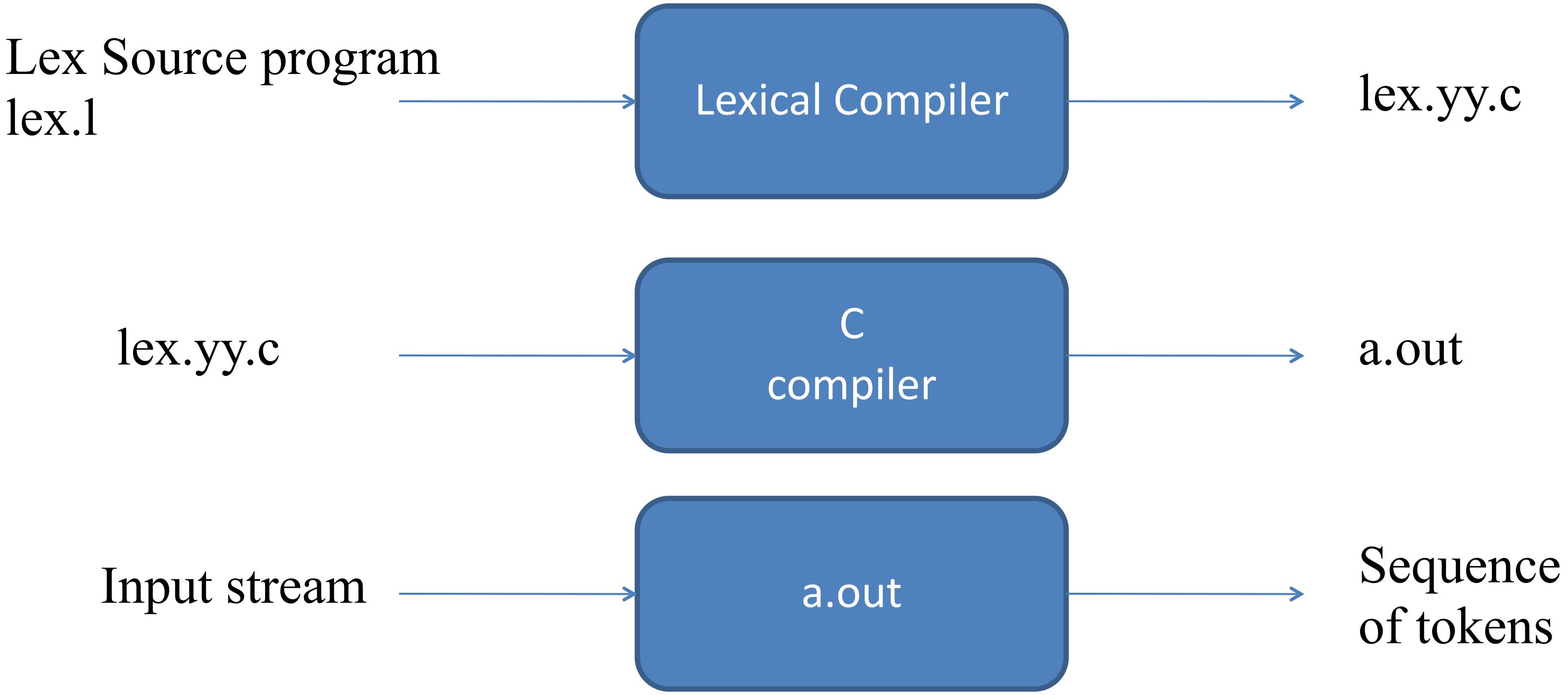
Lex -- a Lexical Analyzer Generator

Given tokens specified as regular expressions, Lex automatically generates a routine that recognizes the tokens.

Lexical Analyzer Generator - Lex



Lex Source program
lex.1

Lexical Compiler

lex.yy.c

lex.yy.c

C
compiler

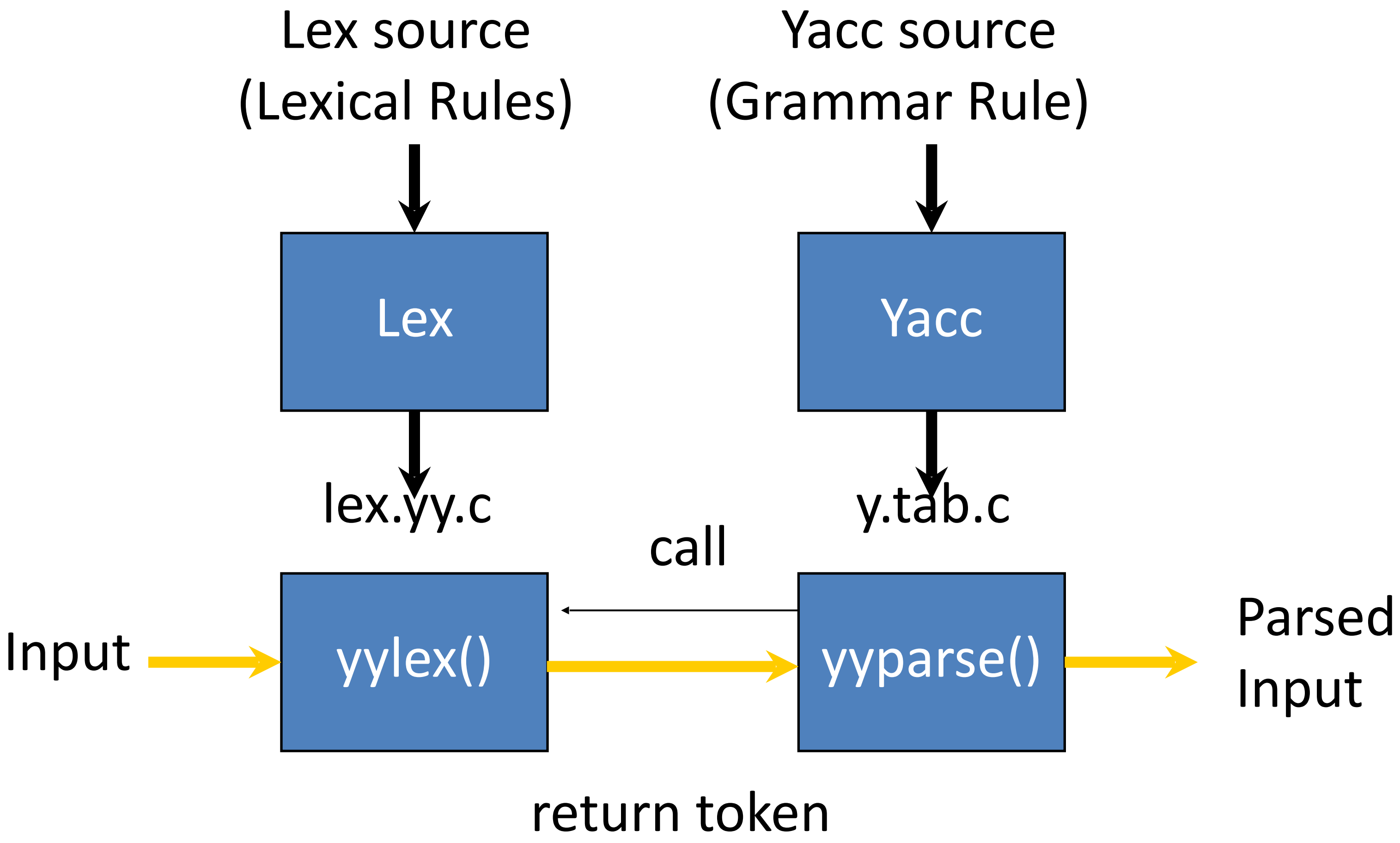
a.out

Input stream

a.out

Sequence
of tokens

Lex with Yacc



Structure of Lex programs

The lex input file consists of three sections, separated by a line with %% in it:

```
declarations
%%
translation rules          →          Pattern    {Action}
%%
auxiliary functions
```

Definitions Section

- The definitions section contains declarations of simple name definitions to simplify the scanner specification.
- Name definitions have the form:

```
name definition
```

- Example:

```
DIGIT    [0-9]
ID       [a-z][a-z0-9]*
```

Rules Section

Rules: <regular expression> <action>

Each regular expression specifies a token.

Default action for anything that is not matched: **copy to the output**

- The rules section of the lex input contains a series of rules of the form:

```
pattern action
```

- Example:

```
{ID} printf( "An identifier: %s\n", yytext );
```

- The *yytext* and *yylength* variable.
- If action is empty, the matched token is discarded.

Action

Action: C/C++ code fragment specifying what to do when a token is recognized.

- If the action contains a `{`, the action spans till the balancing `}` is found, as in C.
- The *return* statement, as in C.
- In case no rule matches: simply copy the input to the standard output (A default rule).

User Code Section

- The user code section is simply copied to *`lex.yy.c`*
- The presence of this section is optional; if it is missing, the second `%%` in the input file may be skipped.
- In the definitions and rules sections, any indented text or text enclosed in `%{` and `%}` is copied exactly to the output (with the `%{}`'s removed).

- lex program examples:
  - ‘lex ex1.l’ produces the lex.yy.c file that contains a routine yylex().
    - The *int yylex()* routine is the scanner that finds all the regular expressions specified.
      - yylex() returns a non-zero value (usually token id) normally.
      - yylex() returns 0 when end of file is reached.
  - Need a drive to test the routine. Main.c is an example.
    - Have a *yywrap()* function in the lex file (return 1)
      - Something to do with compiling multiple files.

yylex() is a function of return type int. LEX automatically defines *yylex()* in *lex.yy.c* but does not call it. The programmer must call yylex() in the Auxiliary functions section of the LEX program. LEX generates code for the definition of yylex() according to the rules specified in the Rules section.

- LEX declares the function yywrap() of return-type int in the file *lex.yy.c* . LEX does not provide any definition for yywrap(). yylex() makes a call to yywrap() when it encounters the end of input. If yywrap() returns zero (indicating *false*) yylex() assumes there is more input and it continues scanning from the location pointed to by yyin. If yywrap() returns a non-zero value (indicating true), yylex() terminates the scanning process and returns 0 (i.e. “wraps up”). If the programmer wishes to scan more than one input file using the generated lexical analyzer, it can be simply done by setting yyin to a new input file in yywrap() and return 0.
- As LEX does not define yywrap() in lex.yy.c file but makes a call to it under yylex(), the programmer must define it in the Auxiliary functions section or provide %option noyywrap in the declarations section.

Review of Lex Predefined Variables

Name	**Function**
`char *yytext`	pointer to matched string
`int yyleng`	length of matched string
`FILE *yyin`	input stream pointer
`FILE *yyout`	output stream pointer
`int yylex(void)`	call to invoke lexer, returns token
`char* yymore(void)`	return the next token
`int yyless(int n)`	retain the first n characters in yytext
`int yywrap(void)`	wrapup, return 1 if done, 0 if not done
`ECHO`	write matched string
`REJECT`	go to the next alternative rule
`INITAL`	initial start condition
`BEGIN`	condition switch start condition

Usage

To run Lex on a source file, type

```
lex scanner.l
```

- It produces a file named lex.yy.c which is a C program for the lexical analyzer.
- To compile lex.yy.c, type

```
cc lex.yy.c -ll
```

- To run the lexical analyzer program, type

```
./a.out < inputfile > output
file
```

lex1.l

```
%{int count=0;
%}
chars [A-Za-z]
number [0-9]
delim [" "\n\t]
ws {delim}+
words {chars}+
numbers {number}+
%%
if printf("%s\n",yytext);
then printf("%s\n",yytext);
else printf("%s\n",yytext);
"<" printf("%s\n",yytext);
{words} {count++;}
{numbers} printf("digits %s\n",yytext);
%%
void main()
{
yylex();
printf("There are total %d words\n", count);
}
int yywrap()
{return 1;}
```

e1.l

```
%{
int count=0;
%}
chars [A-Za-z]
numbers [0-9]
delim [" "\n\t]
ws {delim}+
words {chars}+
%%{words} {count++;}
%%void main()
{
extern FILE* yyin;
yyin=fopen("input.txt","r");
yylex();
printf("%d",count);
}
int yywrap()
{return 1;}
```

input.txt

```
we are students from g if < else 3333
```

```
psg@psg-OptiPlex-3060:~$ flex lex1.l
psg@psg-OptiPlex-3060:~$ cc lex.yy.c -ll
psg@psg-OptiPlex-3060:~$ ./a.out <input.txt
   if
 <
 else
 3333
There are total 5 words
psg@psg-OptiPlex-3060:~$ ./a.out <input.txt >
out.txt
```

Example

```
%{
        /* definitions of manifest constants
        LT, LE, EQ, NE, GT, GE,
        IF, THEN, ELSE, ID, NUMBER, RELOP */
%}

/* regular definitions
delim           [ \t\n]
ws              {delim}+
letter          [A-Za-z]
digit [0-9]
id              {letter}({letter}|{digit})*
number          {digit}+(\.{digit}+)?(E[+-]?{digit}+)?

%%
{ws} {/* no action and no return */}
if              {return(IF);}
then{return(THEN);}
else  {return(ELSE);}
{id}  {yylval = (int) installID(); return(ID); }
{number}        {yylval = (int) installNum(); return(NUMBER);}
…
```

```
Int installID() {/* funtion to install the
     lexeme, whose first character is
     pointed to by yytext, and whose
     length is yyleng, into the symbol
     table and return a pointer thereto
     */
}

Int installNum() { /* similar to
     installID, but puts numerical
     constants into a separate table */
}
```