# Compiler Design Laboratory (CS 753)

#### Samit Biswas

samit@cs.iiests.ac.in



Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Shibpur

August 20, 2018



lex/flex

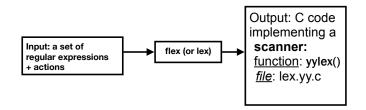


We refer to the tool as Lex compiler, and to its input specification as the Lex language.

#### flex (and lex): Overview

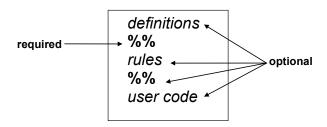
## Scanner generator

Helps write programs whose control flow is directed by instances of regular expressions in the input stream.



## flex input format

A Lex program (the lex.l file ) consists of three parts:



Shortest possible legal flex input:



#### **Definitions**

name definitions, each of the form

name definition

e.g.:

name definitions, each of the form

stuff to be copied verbatim into the flex output enclosed in %{ ..... }%, or (e.g., declarations, #includes):

#### **Lex Rules**

- The rules portion of the input contains a sequence of rules.
- Each rule has the following form Regular\_Expression Action where:
  - Regular\_Expression describes a pattern to be matched on the input.
  - action must begin on the same line.

# **Example of Lex Rules**

Pattern	Action
int	printf("Keyword: INTEGER");
[0-9]+	printf("Number ");

#### **Patterns**

- Essentially, extended regular expressions.
  - Syntax: similar to grep (see man page).
  - << EOF >> to match "end of file".
- Character classes:
  - [:alpha:], [:digit:], [:alnum:], [:space:], etc. (see man page).
- ▶ {name} where name was defined earlier.

```
%{
                                                       Definition for a digit
definitions
      #include <stdio.h>
                                                       (could have used builtin definition [:digit:] instead)
      #include <stdlib.h>
       %}
      dgt
            [0-9]
                                                       Rule to match a number and return its value to
                                                       the calling routine
      {dgt}+ return atoi(vvtext):
       %%
      void main()
        int val. total = 0. n = 0:
user code
        while ((yal = yylex()) > 0) {
                                                               Driver code
          total += val:
                                                               (could instead have been in a separate file)
          n++:
        if (n > 0) printf("aye = %d\n", total/n);
```

```
%{
definitions
       #include <stdio.h>
                                                                    defining and using a name
       #include <stdlib.h>
              0-9
rules
                return atoi(yytext);
       void main()
         int val, total = 0, n = 0;
user code
         while ( (val = yylex()) > 0 ) {
           total += val:
           n++:
         if (n > 0) printf("ave = %d\n", total/n);
```

```
%{
definitions
       #include <stdio.h>
       #include <stdlib.h>
             [0-9]
                                                                 char * yytext;
.nles
               return atol(yyte)
                                                                        a buffer that holds the input
                                                                        characters that actually match the
                                                                        pattern
       void main()
         int val, total = 0, n = 0;
user code
         while ((val = vvlex()) > 0) {
           total += val:
           n++:
         if (n > 0) printf("ave = %d\n", total/n);
```

```
%{
definitions
       #include <stdio.h>
       #include <stdlib.h>
             [0-9]
rules
               return atoi(yytext);
       void main()
         int val, total = 0, n = 0
ser code
         while ( (val = vvlex()) > 0 ) {
                                                                Invoking the scanner: yylex()
           total += val:
                                                                       Each time yylex() is called, the
           n++:
                                                                       scanner continues processing
                                                                       the input from where it last left
         if (n > 0) printf("ave = %d\n", total/n);
                                                                       off.
                                                                       Returns 0 on end-of-file.
```

## Lex library routines

- yylex() The default main() contains a call of yylex()
- yymore() return the next token.
- yyless(n) retain the first n characters in yytext.
- yywrap()
  - is called whenever Lex reaches an end-of-file.
  - The default yywarp() always returns 1.

#### Lex Predefined Variables

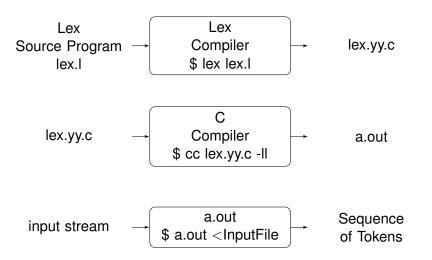
- yytext a string containing the lexeme
- yyleng the length of the lexeme
- yyin the input stream pointer
   the default input of default main() is stdin
- yyout the output stream pointer
   -the default output of default main() is stdout

# In lex program, a main() function is generally included as:

```
main(){
yyin = fopen(filename, "r");
while(yylex());
}
```

► Here **filename** corresponds to input file and the *yylex*() routine is called which returns the tokens. *yyin* is *FILE* pointer declared by Lex part.

## **Lexical Analyser Generators** — **Lex**



# **Assignment**

Implement a lexical analyzer using the tool: lex/flex for the following types of tokens:

- Arithmetic, Relational, Logical, Bitwise and Assignment Operators of C.
- Reserved words: int, float, char, for, while, if and else
- Identifier.
- Integer Constants.
- Parentheses, Curly braces

Take a complete C program as input and generate the above-mentioned tokens.

#### tokendef\_LEX.h

```
/* Single caharacter lexemes */
#define LPAREN_TOK '('
#define GT TOK '>'
#define RPAREN_TOK ')'
#define EQ_TOK '='
#define MINUS_TOK '-'
#define SEMICOLON_TOK ';'
/* Reserved words */
#define WHILE TOK 256
/* Identifier, constants..*/
#define ID TOK 350
#define INTCONST 351
.*/
```