What it does, How it works, How to implement one.
Break stream of characters into *tokens*, return token + attribute (lexeme/pointer to symbol table entry) to parser

**Tokens**: atomic units or “words”
e.g. keywords, operators, identifiers, numerical constants, strings, punctuation, etc.

**Lexeme**: actual string forming a token

- Strip white space, comments
- Keep track of line number (for error messages)
Patterns

**Pattern:** rule specifying the set of strings corresponding to a given token
  e.g. identifier rule

**Regular expressions:** formal way of specifying patterns

Notation:
- $\cup$ union
- $*$ Kleene closure
- $+$ positive closure
- $rs$ concatenation of expressions $r$ and $s$
- $?$ zero or one instance
- $[abc]$ one character from specified class

**Regular definitions:** Sequence of definitions of the form

$$d_i \rightarrow r_i$$

where $r_i$ – regular expression over $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$

Examples: *letter, digit, identifier, string, number, float*
Token recognition

- Use finite automata / transition diagram
- Try automata for various patterns in turn
- Issues:
  - input retraction should be possible (⇒ buffering)
  - ordering of transition diagrams important
  - Qn: How to handle keywords?
while (1) {
    switch (state) {
    case 0:
        c = nextchar();
        if (isletter(c)) state = 1;
        else state = retry();
        break;
    case 1:
        c = nextchar();
        if (isletter(c) || isdigit(c))
            state = 1;
        else state = 2;
        break;
    case 2:
        pushbackchar();
        /* process token */
        . . .
    case 3:
        . . .
    default:
        error();
        break;
    }
    return state;
}
Buffering

- Buffer \(\equiv\) 2 disk blocks
- Maintain `current` and `token_beginning`
- When one half finishes, get other half
- **Sentinels:** eliminates checking at every step whether current pointer is at end of buffer

```
if (*current == EOF) read new block
```

**NOTE:** lookahead is limited
Errors:

- handle illegal characters
- *Panic mode* error recovery – skip successive characters until a valid token is found
- Example: \(i = 2j;\)

Problems:

- Fortran, Algol68: blanks not significant except in literal strings
  
  ```fortran
  DO 5 I = 1.25  vs.  DO 5 I = 1, 25
  ```

- PL/I: keywords not reserved (?)
  
  ```pli
  if then then then = else; else else = then;
  DECLARE (A1, A2, A3)
  ```
Usage: $ lex myfile.l  (generates lex.yy.c)
lex.yy.c contains yylex()

Format:
declarations (variables, constants, regular definitions)
%%
translation rules ($p_i \{ \text{action}_i \}$)
%%
auxiliary procedures (C functions)

Operation:
1. scan remaining input
2. find longest matching prefix
3. execute corresponding action (return to caller/ continue matching)
%{
    /* 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} {yyval = install_id(); return(ID);}
    {number} {yyval = install_num(); return(NUMBER);}
        "<" {yyval = LT; return(RELOP);}
        "<=" {yyval = LE; return(RELOP);}
        "=" {yyval = EQ; return(RELOP);}
        "><" {yyval = NE; return(RELOP);}
        ">" {yyval = GT; return(RELOP);}
        ">
    install_id() {
        /* procedure to install the lexeme, whose
        first character is pointed to by yyltext and
        whose length is yyleng, into the symbol table
        and return a pointer thereto */
    }
    install_num() {
        /* similar procedure to install a lexeme that
        is a number */
    }
Disambiguation: pattern listed earlier has precedence

Global variables: ytext, yleng, ylval

Errors: any input that does not match a pattern is copied to output by default (suppress using -s)

Lex vs. Flex: documentation, libraries, …