Proj example:
simple language(~C)
-> made up assembly lang
-> scanner, parser, type checker, code gen

First stage :lexical analyzer, lexer, scanner
(reserved id)int (id)i2 (semicolon); (/*ignored*/) new line
i++2; -> i+ +2? i ++2? confusion. (greedy rule, take the longest valid)
char ->| scanner |->token ->stream
ignore parts
— new lines
— white space (but python)
— comments (one line; multiline)
(original fortran ignore all whitespace but not new lines
Do 10(<-#line) i=1,100) -> Do10I=1,100\\\==\==\\\Do 10I=1.100
look ahead (lexical) :for example, Do10I=1,100; Do 10I=1.100

What's the output?

for i:=1 to 10
begine
….
end

kw_for
ID (i)
assign
literal int(1)
key_to

(maybe store them into a token table)
int i;
X(type name) Y(variable name);

— — — — — — — -
regular expression: easy to describe tokens as regular expression
- a stands for litera "a"
- XY where x,,y regex
- X|Y x or y regex
- X*

id in C language:
(a|b|c|d|…|z|….|9|) = [_a-zA-Z][_a-zA-Z0-9]*
$A^4 = AAAA$
$A^+ = AA*$
— — — — — — — —

D =[0-9]
L=[_a-zA-Z]
L(D|L)* return (ID)
D+ return(LITERAL_INT)
if return(KW_IF) (match L(D|L)*)
+ …(PLUS)
+ ++
ws+ do nothing
dot error()
— — — — — — — —
C++
stack<int>
stack<stack<int>>
"lex"

scanner generator scanner.l, scanner.lex
yylex()->call repeatedly get the next token

flex<-lex

——————————

How does regEx matcher actually work

DFA = deterministic (finite automaton/finite-state machine)
difference between deter and indeter

DFA: S = set states , $\sum$ = finte alphabet
s: starting state

S = {1,2}
F = {2}
s = 1
$\delta : S * \Sigma$->S
$\sum$= {0-9a-zA-Z}

	else								0-9
1	2	2	2	2	2	2	2	2	XXXXXX
2	2	2	2	2	2	2	2	2	

s1 = $\delta(s0,x1)$
s2 = $\delta(s1,x2)$

if sn+1 belongs F accept else reject

L(A) = {all strings in ($\sum$)* that A accepts}
L is regular
if L= L(A) for some DFA A

——————————————————

NDFA = none-deterministic Finite Automaton
S = {1,2}
F = {2}
s = 1
$\epsilon(\lambda)$: empty string
$\delta : S * (\Sigma U \epsilon) -> 2^S$ all subsets of S

0(0|1)* dfa
(0|1)*0 ndfa
epsilon closure(X): (set of all sates the x can reach from state of x by 0 or more using epsilon closure
e-closure{s0)={s0,A,B}
e