02/17/2025.

quiz 3

Assignment: scanner

02/19/2025

2.5 Scanner.

So far. RE $\longrightarrow$ NFA $\longrightarrow$ DFA $\longrightarrow$ minimaDFA

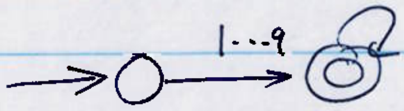
1. A DFA is not scanner.

DFA: tries to recognize a single word 0...9

input: 0 1

1 2

1 2 ␣ 12 ␣ 23



reads all input.

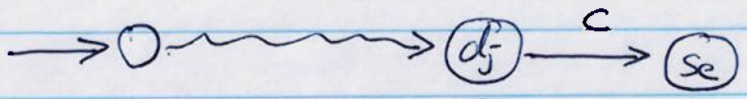
returns a binary answer.

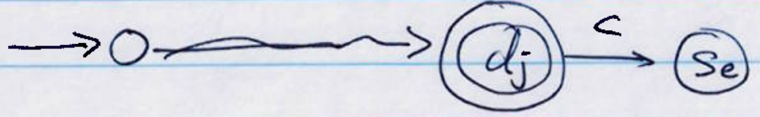
Scanner: takes entire input stream, breaks into individual words.

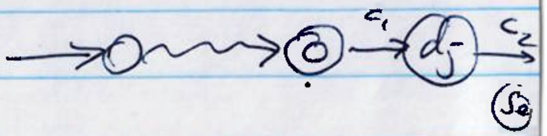
output: a seq. of tokens <lexeme, category>

repeatedly read enough input to find next word. leaves the input stream in a state from which it can find the next work read enough input. until hits an error

$\delta(d_j, c) = S_e$







if d_j is accepting

found a lexeme.

else.

back up. one char at a time

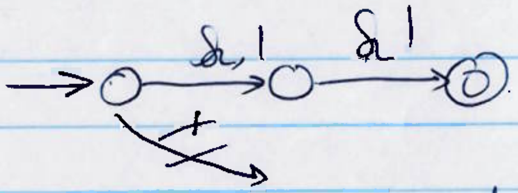
until it ~~is~~ either reaches accepting state or it exhausts ~~the~~ lexeme

Roll.back: as scanner backs up. it pushes char back into the input stream.

12 +

&& ||

find category.



NFA → DFA → min DFA

join NFA. final states. are merged.

Hoprocraft:

modify Hopcroft: separate final/accepting states of each catagory / partition initially

2. implement. scanners.

① . hard code scanner.

flexible. easy to make ~~mistake~~.

② . automated scanner construction.

direct~~ed~~ coded

table driven.

direct coded scanner.

$$\begin{tikzcd} {} \arrow[r] & s_0 \arrow[r, "r"] & s_1 \arrow[r, "0\ldots9"] & \boxed{s_2} \arrow[loop above, "0\ldots9"] \end{tikzcd}$$

```
s0 :   clear stack.
       push bad.
       char ← nextchar( ).
       lexeme ← char   ~~lexeme =~~
       push s0
       if char = 'r'
            goto s1
       else
            goto sout

s1 :   char ← ~~a~~nextchar( ).
       lexeme ← lexeme + char
       push s1
       if char ≤ "9" and char ≥ "0"
            goto s2
       else goto sout
```

```
S2 :  char ← nextchar( ) .
      lexeme ← lexeme + char .
      clear stack( ) , push bad .
      push (S2).

      if char ≥ "1"  and  char ≤ "9" "
            goto S2
      else
            goto sout

sout:  ~~wh~~ state ← Se .
      while ( state ≠ SA  and  state ≠ bad )
      {
            ~~Statk~~ state ← pop
                  truncate ← lexeme
                  roll back ( )
      }
      if  state ∈ SA
            return   token ( lexeme, type[ state])
      else
            return  invalid .
```

*. faster .
few memory op.
code . not . good style

3. table driven scanner

$$\begin{tikzcd} \arrow[r] & s_0 \arrow[r, "r"] & s_1 \arrow[r, "0\ldots9"] & s_2 \arrow[loop above, "0\ldots9"] \end{tikzcd}$$

{ transition table.
skeleton code : update current state base on next char.

	r	0	1	2	....	9	other.
S_0	S_1	S_e	S_e	.	- - - -	S_e	
S_1							
S_2							
S_e							

S_e : roll back.

table size can grow very large. shrink table.

char classification by combining identical col.

δ.	r	digits	other
S_0			
S_1			
S_2			
S_e			

Scanner skeleton.

```
//init.  state ← $s_0$
         lexeme ← ""
         clear stack( )
         push(bad)

//scan.  while ( state ≠ $S_e$ ).
           {    if state. ∈ $S_A$
                {    clear stack
                     push bad.
                }
                push state
                ~~Re~~ char ← nextchar( )
                lexeme ← lexeme + char.
                col ← charclass[char]
                state ← $\delta$(state, col)
           }

// roll back.
         while ( state ≠ $S_A$ ... state ≠ bad
         {    :
              :
//       }, ro
         report : ______
```

more memory. ~~that~~ direct-coded scanner

chapter 3 parsers

scanner : spelling.

parser · phrase structure. determines if the steam of tokens form a <u>sentence</u> in the source lange. program.

① describe the structure of sentence using (CFG. context free gramar)

② . parse : top down.

bottomup

parse tree.