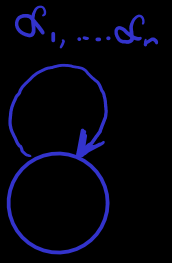
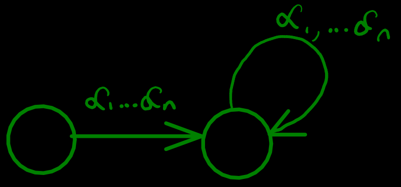
316 9-24-09

$\{\alpha_1|\ldots|\alpha_n\}$ where α_i are terminals



$\{\alpha_1|\ldots|\alpha_n\}^+$



<u>2nd example</u> = Java floating-point numerals

$\langle digits\rangle \longrightarrow \{\langle digit\rangle\}^+$ $\qquad$ $\langle suffix\rangle \longrightarrow \overbrace{f\,|\,F}^{\text{single precision}}\,|\,\overbrace{d\,|\,D}^{\text{double precision}}$

$\langle exponent\rangle \longrightarrow (E|e)\,[+|-]\,\langle digits\rangle$

$\langle float\rangle \longrightarrow \langle digits\rangle\,.\,[\langle digits\rangle][\langle exponent\rangle][\langle suffix\rangle]\;|$

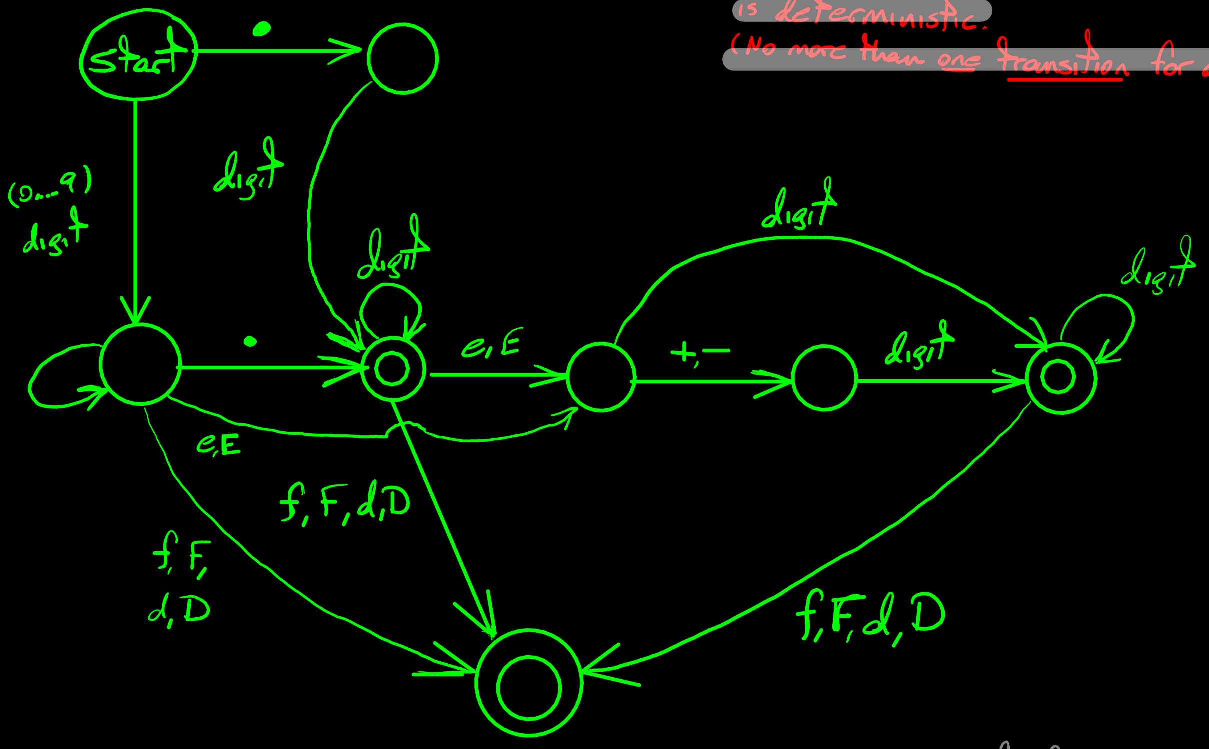
$\qquad .\langle digits\rangle[\langle exponent\rangle][\langle suffix\rangle]\;|$

$\qquad \langle digits\rangle\langle exponent\rangle[\langle suffix\rangle]\;|$

$\qquad \langle digits\rangle[\langle exponent\rangle]\langle suffix\rangle$

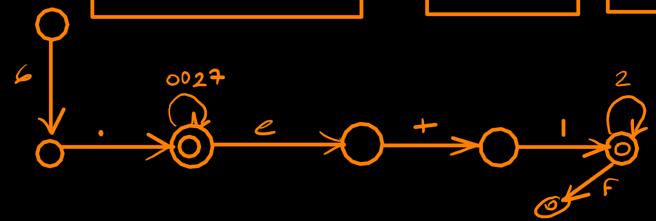
Construct a DFA to accept <float>

Make sure your automaton is deterministic.
(No more than one <u>transition</u> for any state-input pair)



When DFA's are used for lexical analyzers, the <u>longest-taken rule</u> is used: the DFA should run until it gets stuck.

<u>ex=</u> $\boxed{6.0027e+12F}$ $\boxed{3E-9D}$ $\boxed{3e3f}$ $\boxed{2.}$ $\boxed{.3}$

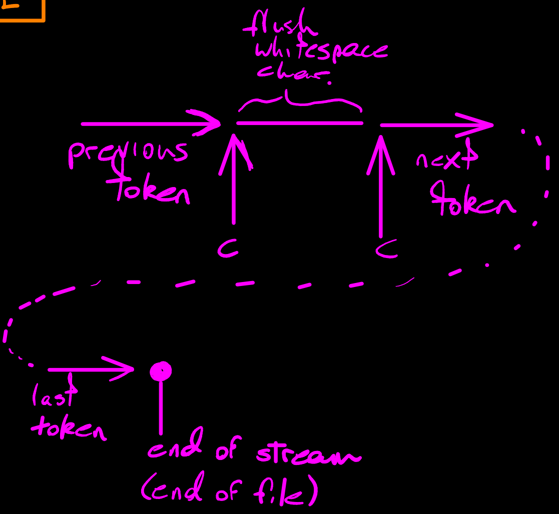


If the DFA gets stuck in a final state, a valid token is extracted

If the DFA gets stuck in a non-final state, an invalid token is extracted

ex: 12.3E␣   123␣   .␣   12.3E-

DFA driver algorithm



t holds the string extracted so far
c holds the current input character
state holds the current state

```
t = empty string;
state = START;
if(c is a whitespace char)
{
   advance c to the next non-whitespace character
}
if(EndOfStream)
     return -1;
while (not EndOfStream)
{
   nextState = δ(state, c);
   if (nextState == UNDEFINED)   //DFA will halt
   {
      if(state is a final state)
           return 1;    //valid token extracted
      else                     //c is an unexpected char, invalid token extracted
      {
         t = t + c;
         c = next character on the input stream;
         return 0;
      } //else
   } //if
   else          //DFA will continue
   {
      state = nextState;
      t = t + c;
      c = next char on the input stream
   } //else
} //while
//end of stream reached while last token is being extracted
if (state == a final state) return 1; //the last token is valid
else return 0;   //the last token is invalid
```

This driver can't be applied to all DFA's.

3 ways to implement state-transition function δ :

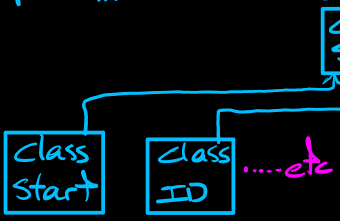
① Nested conditions or nested switch statements

```
if (state == start)
{
   if(c == a letter) return id state;
  else if (c == a digit) return int state;
  else if(c== '+') return plus state;
}
else if (state == id)
{
  enumerate transitions for id state;
}
```

② Arrays

- Create 2D array of [state, inputChar]
- The array's values are states.
- See example Java programs for 1 & 2, and also for the driver

③ Implement states as a class.



generic (or virtual) function δ(character input) with empty body

create a subclass for each state

```
state s;
   ⋮
nextState = s.δ(c);
```

Redefine body of δ in each of the subclasses