Creating a Lexical Scanner

COMP360

“Nothing ever comes to one, that is worth having, except as a result of hard work.”

Booker T. Washington

Scanning Assignment

- Lexical Scanning questions (Regular Expressions and FSA) have been posted to Blackboard
- Due 2:00pm on Monday, February 6
- You can submit the assignment on paper or on Blackboard using any readable media

Lexical Analysis

- Lexical Analysis or scanning reads the source code (or expanded source code)
- It removes all comments and white space
- The output of the scanner is a stream of tokens
- Tokens can be words, symbols or character strings
- A scanner can be a finite state automata (FSA)

Syntactic Analysis

- Syntactic Analysis or parsing reads the stream of tokens created by the scanner
- It checks that the language syntax is correct
- The output of the Syntactic Analyzer is a parse tree
- The parser can be implemented by a context free grammar stack machine

Simple Program

```
/*  This is an example program */
bull = (cow + cat)    *    dog;
goat = “quote”;
```

After Lexical Scan

bull	=	(	cow	+	cat	)
*	dog	;	goat	=	quote	;

- The lexical scanner creates a list of all tokens in the program
- Comments, line divisions and whitespace have been removed
- Words, numbers and quote strings are single tokens instead of individual characters
- The list could be an array, a linked list or an ArrayList

Token Object

A token created by the lexical scanner should contain:

- String value; // text of the token
- int type; // type of value
  - name
  - number
  - punctuation
- int line number, column // position in source code

Machines of a Compiler

- Lexical Analysis – Finite State Automata
- Syntactic Analysis – Push Down Automata

- Since these simple theoretical machines can recognize the grammars we are using, compilers are based on these simple models

State Tables

- An FSA graph can be converted to a table
- The table has cells for each state and each input symbol
- In the cell goes the next state if the DFA is in that state and receives that input symbol
- You can consider the state table to be an adjacency table for the graph

Moving Everything into the Code

- I once needed to set the parity of a byte in assembler

```
                load  R1, byte    // byte to set
                load  R2, 7       // loop counter
                load  R3, 0       // result
again:          xor   R3, R1      // XOR byte with result
                shr   R1,1        // move to next bit
                sub   r2, 1       // decrement loop counter
                jnz   again       // repeat
                shl   R3,7        // move result to parity
position
                or    R1, R3      // set parity
```

- 33 instructions executed per byte

Table Lookup

- To make the program faster, I created a table of 128 bytes, each with the proper parity
- The program now shortens to:

```
load R1, byte          // byte to set
load R1, table[R1]     // R1 has byte with parity
```

Data Driven Code

- Instead of having lots of IF statements in your program, it may be easier and clearer to use a table to make the decision

```
symbol = symTab.get(instruction);
length = symbol.len;
```

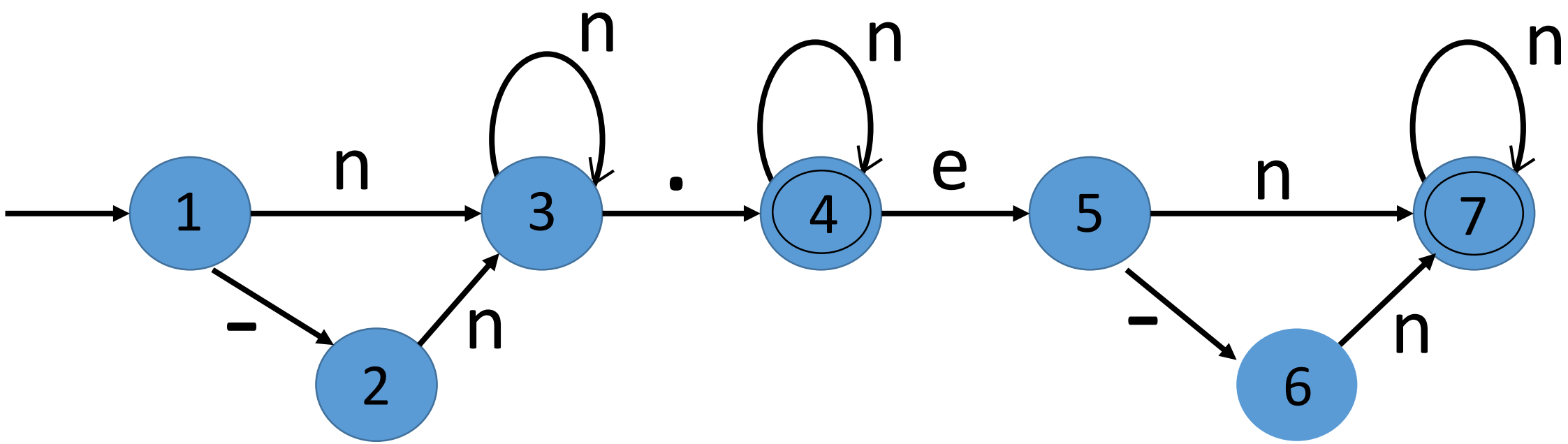
```
public static int calcSize(int instruction){
   if(instruction >= 0 && instruction <= 3){
      return 4;
   }else if(instruction >= 4 && instruction <= 7){
      return 8;
   }else if(instruction == 8){
      return 2;
   }else if(instruction >= 9 && instruction <= 11){
      return 8;
   }else if(instruction >= 12 && instruction <= 15){
      return 2;
   }else{
      return 8;
   }
}
```

Public only if it is Public

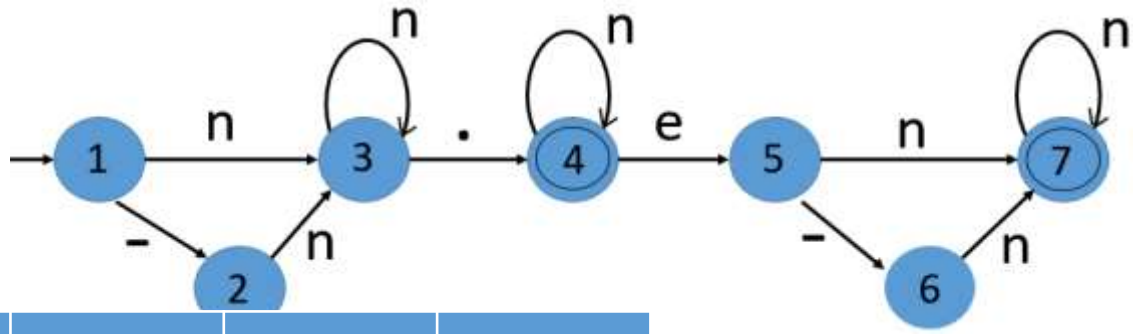
- Whenever you define a method or a class instance variable, make it private unless there is a real need to be public
- If it is private, it will not be used by any other part of the program
- The compiler can optimize if it knows nothing else will use the private method or variable
- Humans will know it is not used elsewhere

Example FSA

- Consider a FSA to recognize Java or C++ double constants of the form $-^{0..1}\ n^{+}\ .\ n^{*}\ (e\ -^{0..1}\ n^{+})^{0..1}$
- Examples 12.34 1. -1.23e-4 0.5



Convert the Graph to a Table

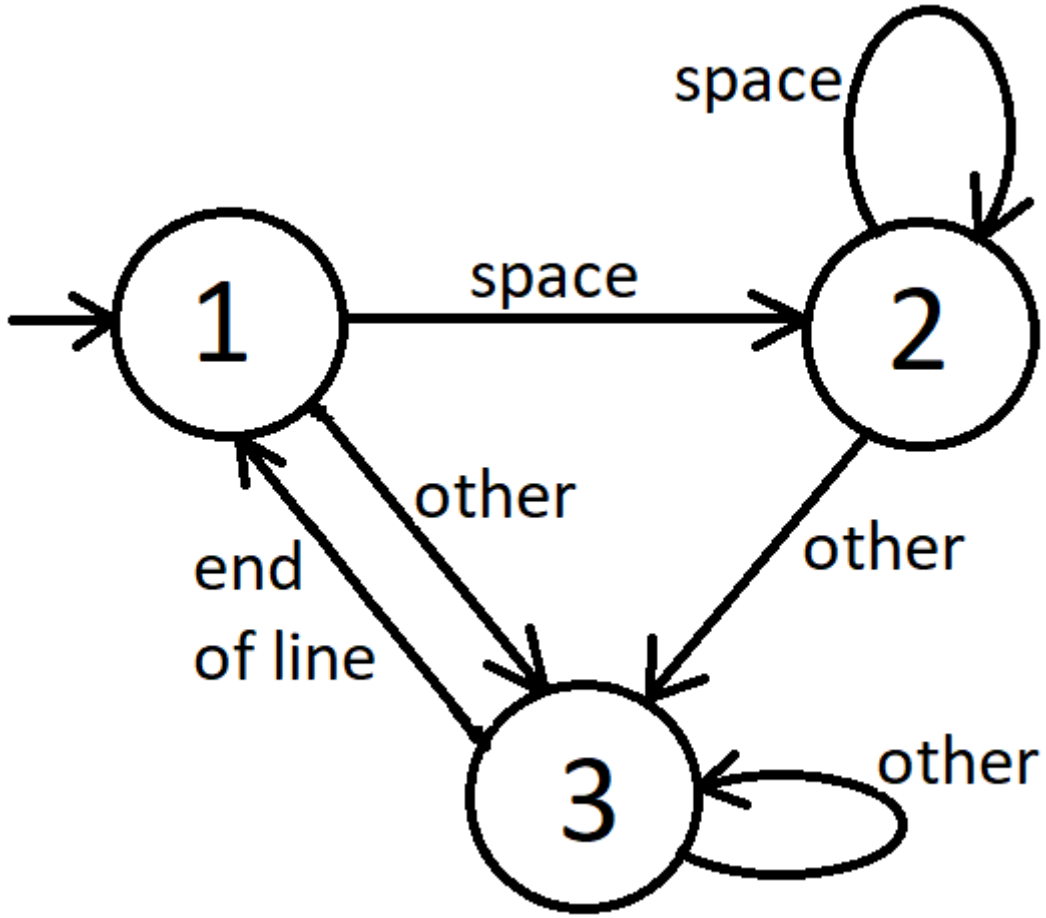


	1	2	3	4	5	6	7
n	3	3	3	4	7	7	7
-	2	0	0	0	6	0	0
.	0	0	4	0	0	0	0
e	0	0	0	5	0	0	0

- The states are listed along the top
- The input symbols are along the side
- For that symbol while in that state, the DFA will go to the new state given in the table
- State zero represents a final error state

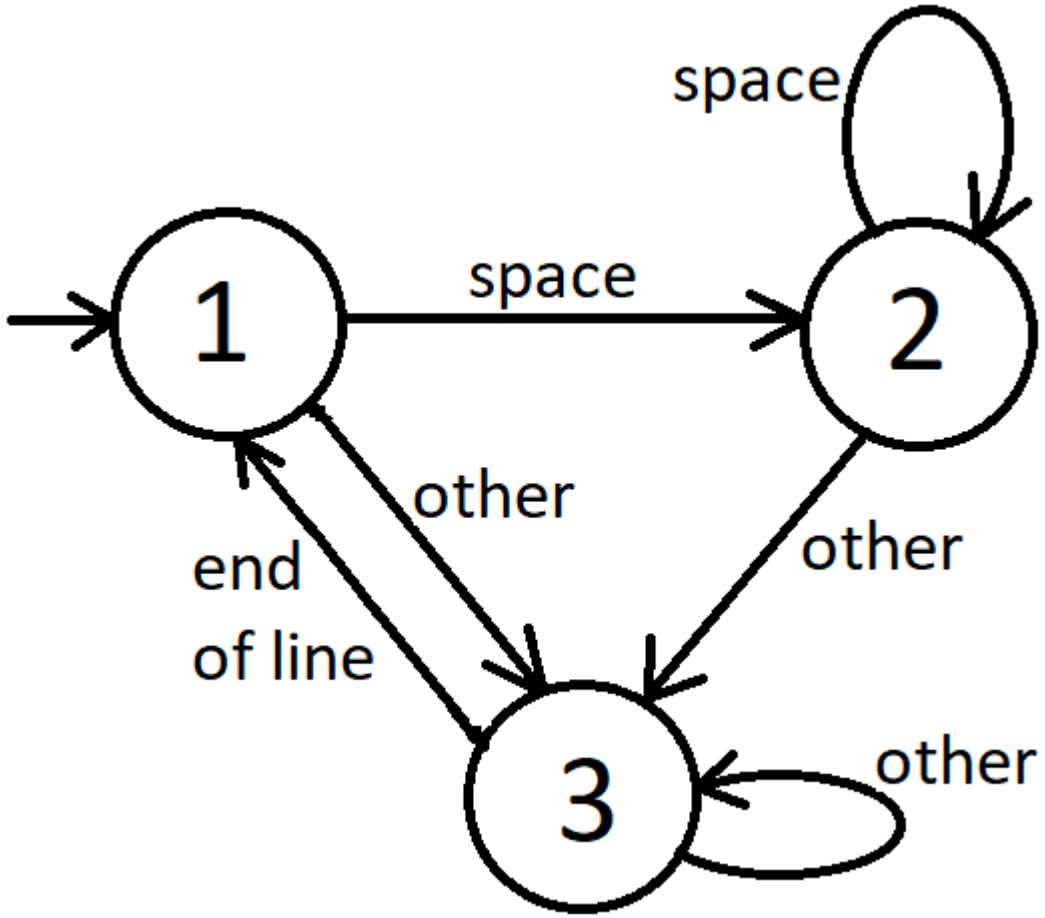
Example FSA

- In Python, leading spaces are important to the syntax
- Other spaces are unimportant
- Create an FSA that captures the leading spaces



Convert the Graph to a Table

	1	2	3
Space	2	2	3
Other	3	3	3
New Line	1	1	1

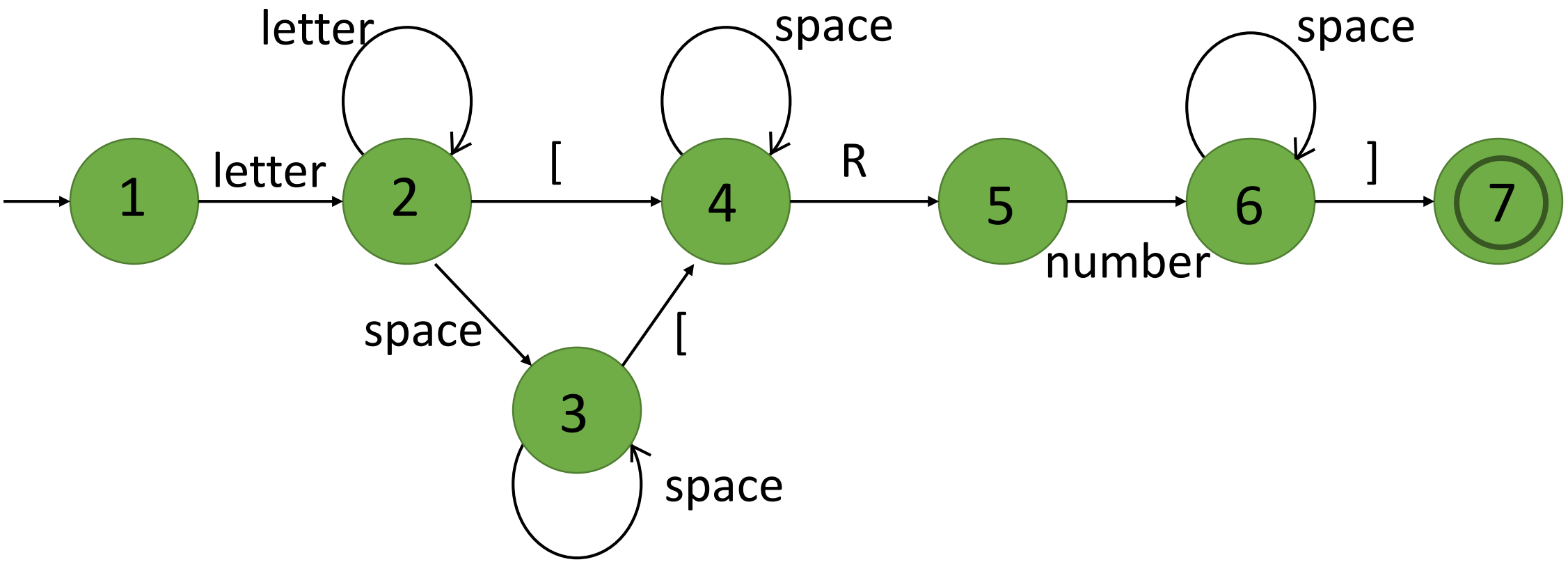


Write an FSA

- Create an FSA to recognize a name followed by a register (R*number*) in square brackets, e.g. **`stuff[R8]`**

Possible Solution

- Create an FSA to recognize a name followed by a register (R*number*) in square brackets, e.g. **`stuff[R8]`**



Programming a FSA

- It is relatively simple to implement a Finite State Automata in a modern programming language
- This program can be used to recognize if a string conforms to a regular language

FSA Program

```
state = 1
while not end of file {
        symbol = next input character
        state = stateTable[ symbol, state ]
        if state = 0 then error
}
if state is a terminating state, success
```

Grouping Input Symbols

- For many FSA programs, it is useful to create an index value for the input symbols, e.g. index = char – ‘A’
- Often you can have groups of symbols use the same index value, e.g. all letters have the index 2 and all numbers have the index 3

The assembler program was difficult

A. because I didn’t know what it was supposed to do
B. because I didn’t understand symbol tables
C. because I didn’t know how to implement it
D. because I didn’t have enough time
E. but doable

The program that I submitted

A. Works correctly
B. Works with a few minor bugs
C. Doesn’t work at all
D. I didn’t submit the assignment

Before doing this assignment, I rate my understanding of what an assembler does as

A. Good
B. Moderate
C. A little fuzzy
D. No idea

My last class that required significant programming was

A. Last semester (Fall 2019)
B. two semesters ago (Spring 2019)
C. three semesters ago (Fall 2018)
D. Longer

Beyond reading the assignment, I started working on the assembler program by

					17	18 A
19	20 B	21	22	23	24 C	25
26	27 D	28	29			

E. I did not do the assignment

It would have helped to do this assignment in teams of two students

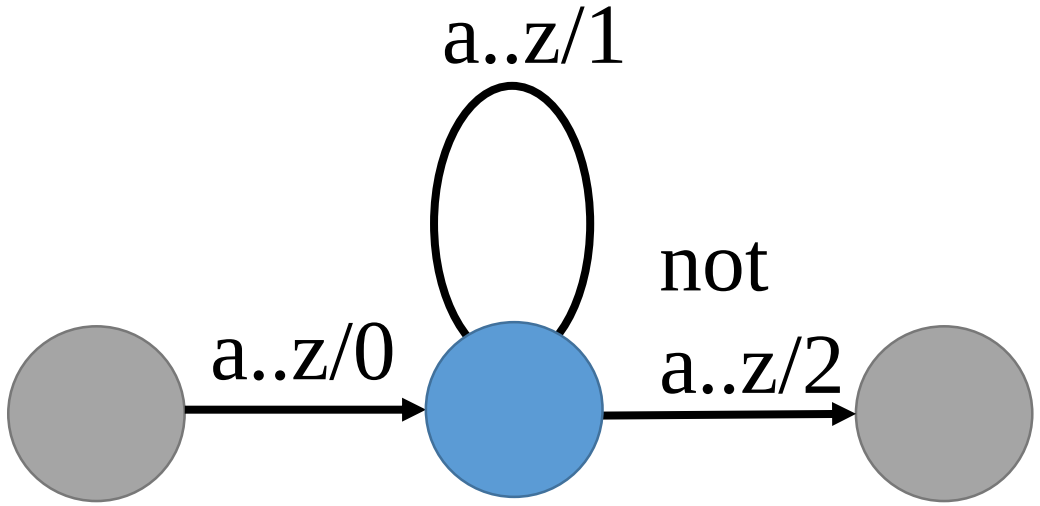
A. A lot
B. A little
C. Not really
D. It would be no help at all

Mealy and Moore Machines

- The FSA we have discussed so far simply determine if the input is valid for the specified language
- An FSA can also produce an output
- A Mealy machine has an output or function associated with each transition or edge of the graph
- A Moore machine has an output or function associated with each state or node of the graph

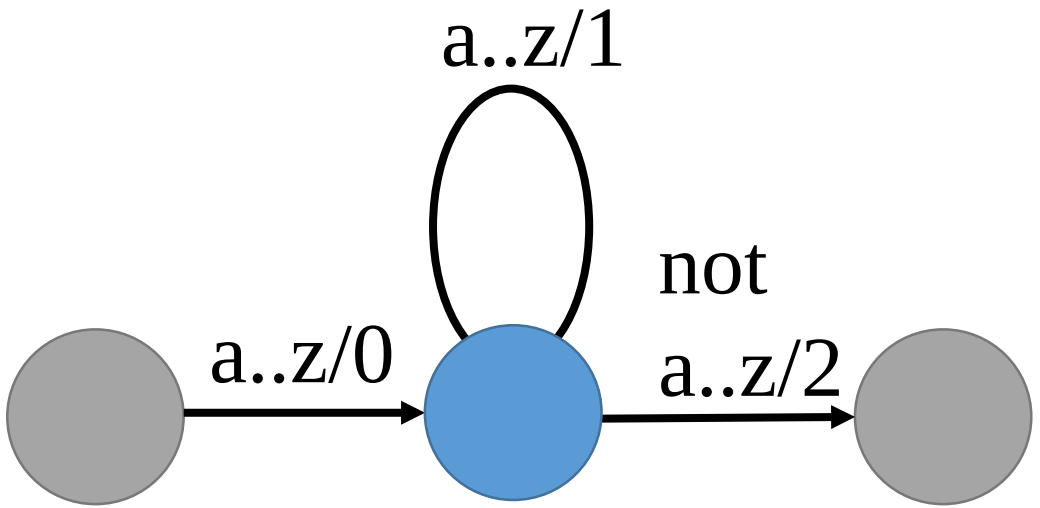
Using Mealy Machines to Create Tokens

- Consider an FSA reading text and creating token of words separated by spaces or punctuation



- 0 = Save character as first letter in a string
- 1 = Save character as next letter in a string
- 2 = Save the string as a token, handle other symbol

Mealy Machine State Table

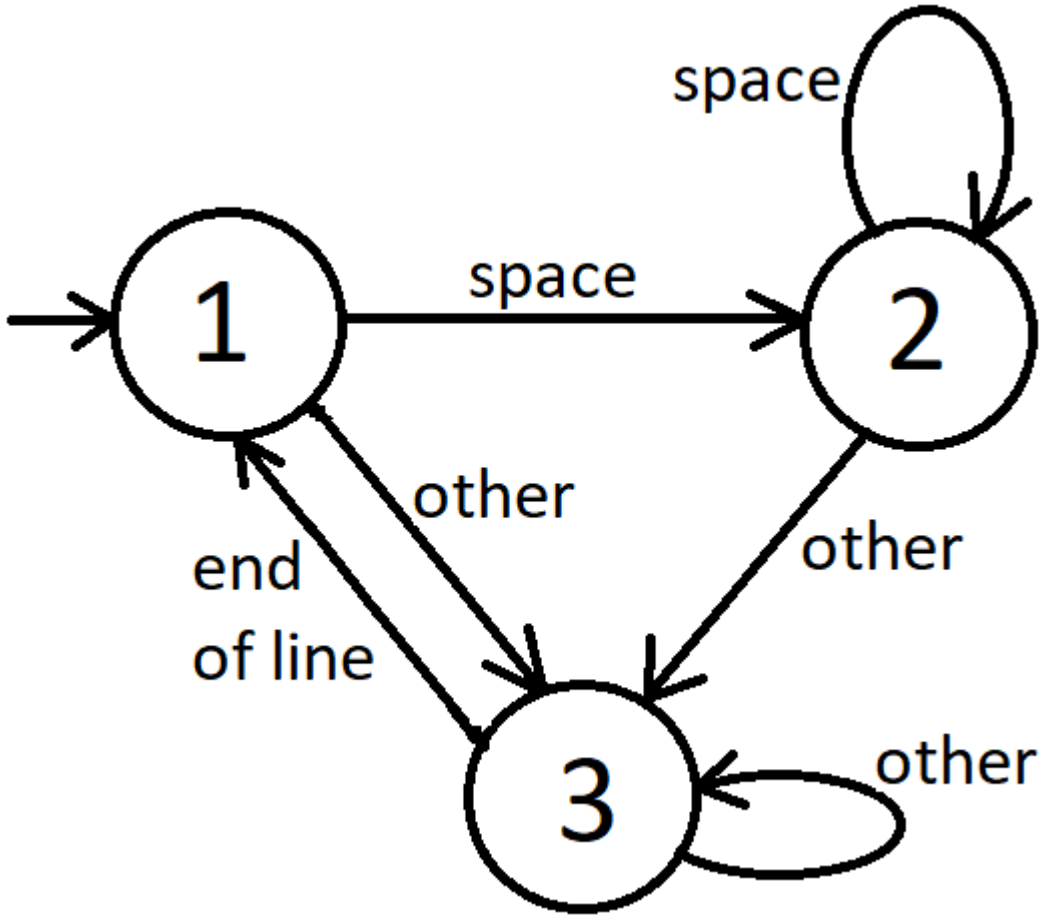


	1	2	3
a .. z	2/0	2/1	2/0
not a .. z	1/x	3/2	3/x

New state / Output function

Creating Mealy Machine to Make Python Token

	1	2	3
Space	2/1	2/2	3/3
Other	3/3	3/3	3/3
New Line	1/0	1/0	1/0



Actions

0 = do nothing

1 = create new token

2 = append space to end of existing token

3 = do something else

Lexical Analysis with a Mealy Machine

- Compilers can use a Mealy machine to scan the source code
- The FSA recognizes and discards comments and white space
- Names, numbers, strings and punctuation are each output as a list of tokens
- A token is an object that contains one unit of the input specifying the value and type

Coding a Mealy Machine

- You can use two 2D arrays of integers indexed by input symbol type and current state OR
- You can use a 2D array of objects indexed by input symbol type and current state
- One value of each array cell is the new state
- The other value tells the program what action to take at this transition

Common Transition Actions

- Create a token with the saved character
- Save another character in the token
- Do nothing

Hints for Creating a DFA

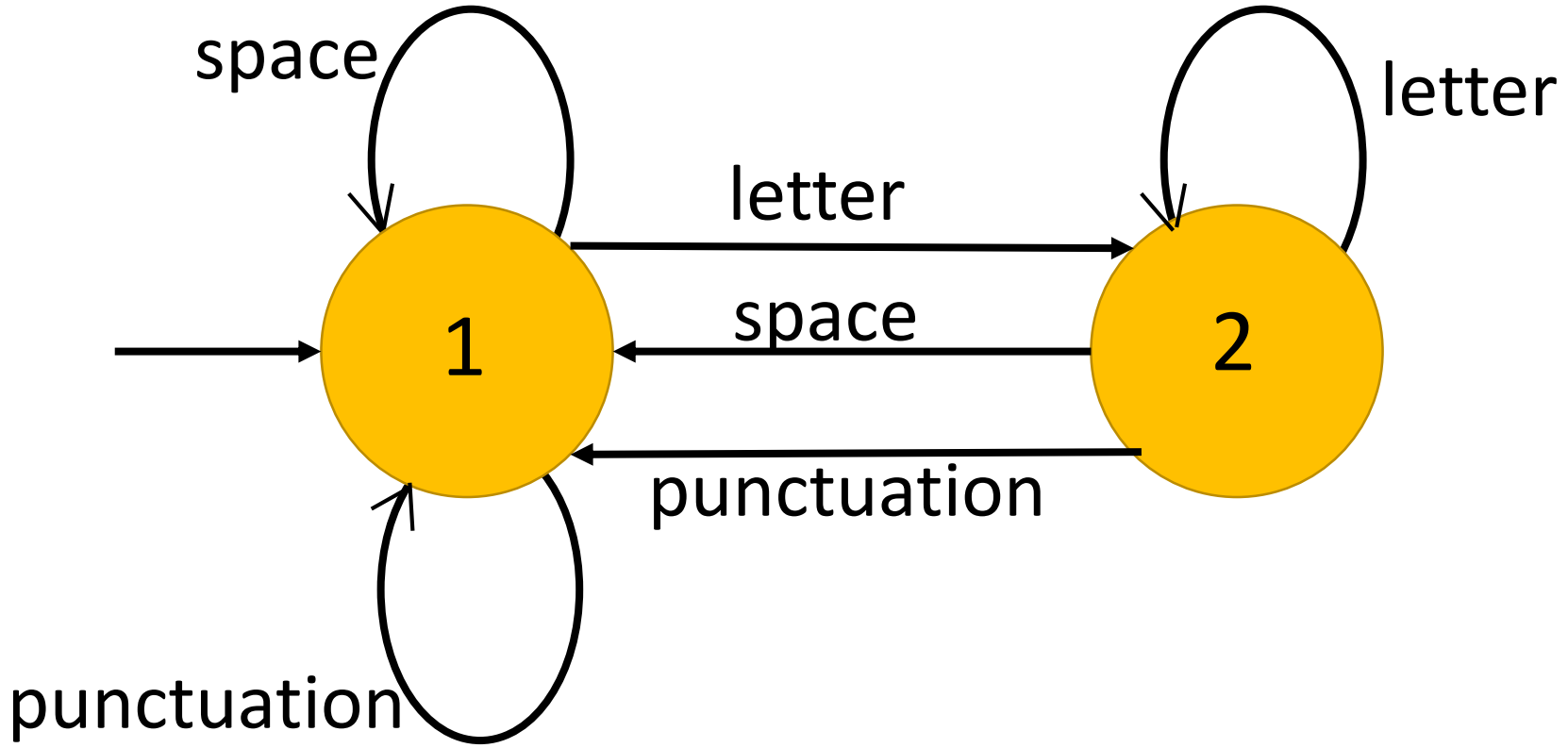
- It usually helps to think of the input one character at a time
- You can make a DFA for a subset of the language and then merge the results

Design a DFA

- Create a DFA that recognizes strings of words separated by punctuation while ignoring spaces

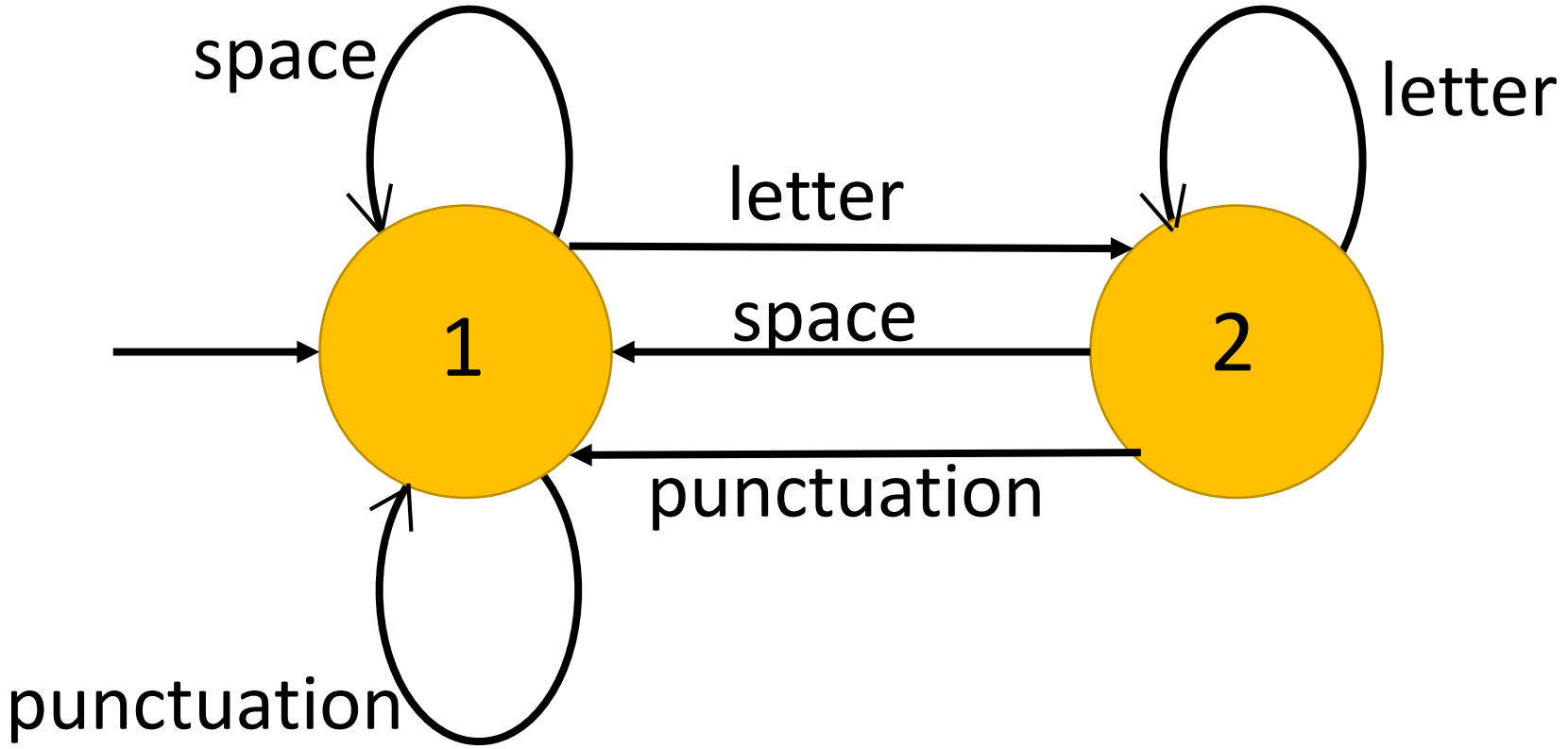
Possible Solution

- Create a DFA that recognizes strings of words separated by punctuation while ignoring spaces



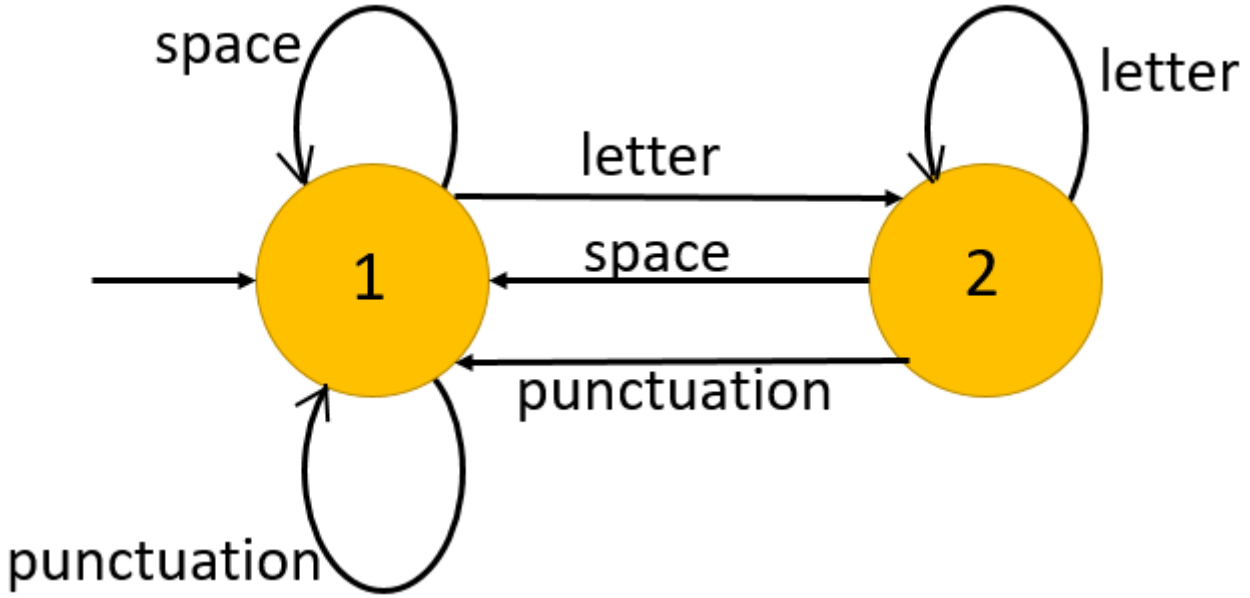
Create a State Table

- Create a state transition table for the DFA



Possible Solution

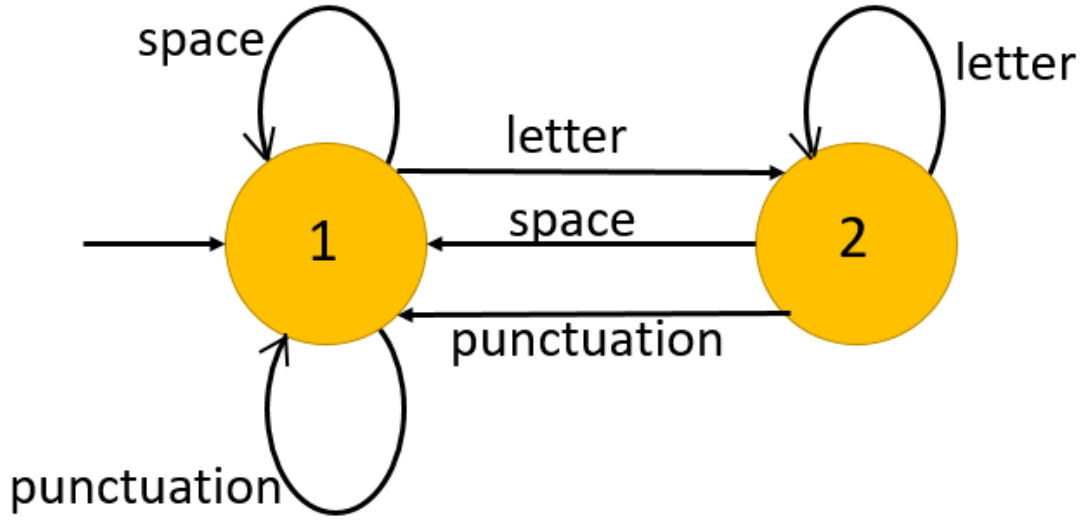
	1	2
letter	2	2
punctuation	1	1
space	1	1



Add Actions to State Transitions

- Make a Mealy machine that will create tokens of the words and punctuation

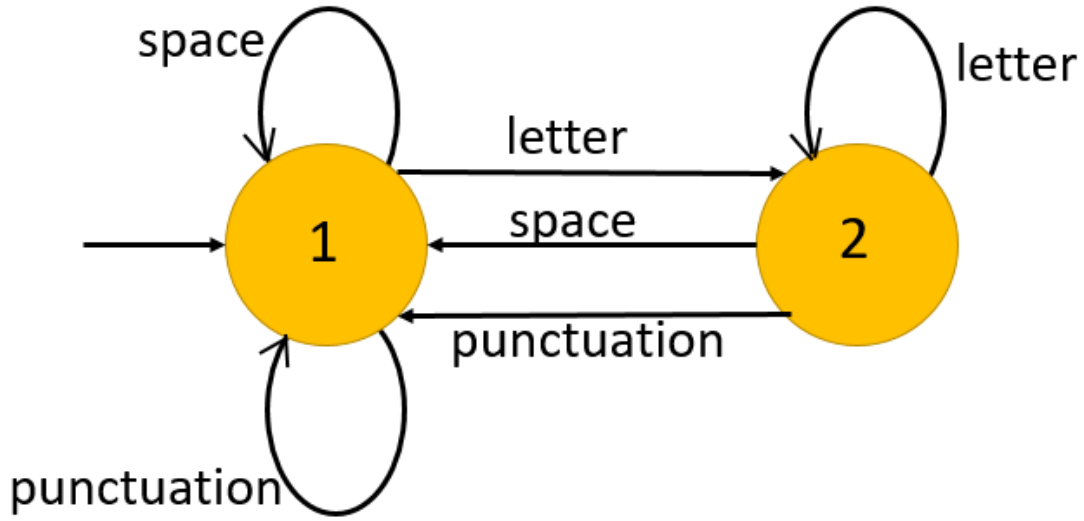
	1	2
letter	2/	2/
punctuation	1/	1/
space	1/	1/



Possible Solution

- Make a Mealy machine that will create tokens of the words and punctuation

	1	2
letter	2/1	2/2
punctuation	1/1	1/1
space	1/0	1/0



0 = do nothing

1 = create token with input character

2 = add input character to current token

Scanning Assignment

- Lexical Scanning questions (Regular Expressions and FSA) have been posted to Blackboard
- Due 2:00pm on Monday, February 6
- You can submit the assignment on paper or on Blackboard using any readable media