Chapter 2

Scanning

Objective

Today's primary objective:

- Present and explore pseudo code for an ad-hoc scanner and table-driven scanner.

Lexeme & Tokens

- Lexeme – lowest-level syntactic unit of a language
- Token – category of lexemes

The following program has the following lexemes and tokens.

sum := sum + 10;

Lexeme	Token
sum	identifier
:=	assign_op
+	add_op
10	int_literal
;	semicolon

Scanning

Three approaches to writing a lexical analyzer:

- Using a tool: Express the acceptable patterns in a formal language such as regular expressions and input this into a software tool that automatically generates the lexical analyzer. (Unix lex tool is an example.)
- Ad hoc: Draw a state transition diagram (DFA) that describes the patterns and write a program that implements the diagram.
- Table driven: Draw a state transition diagram as above and construct a table-driven implementation of the state diagram.

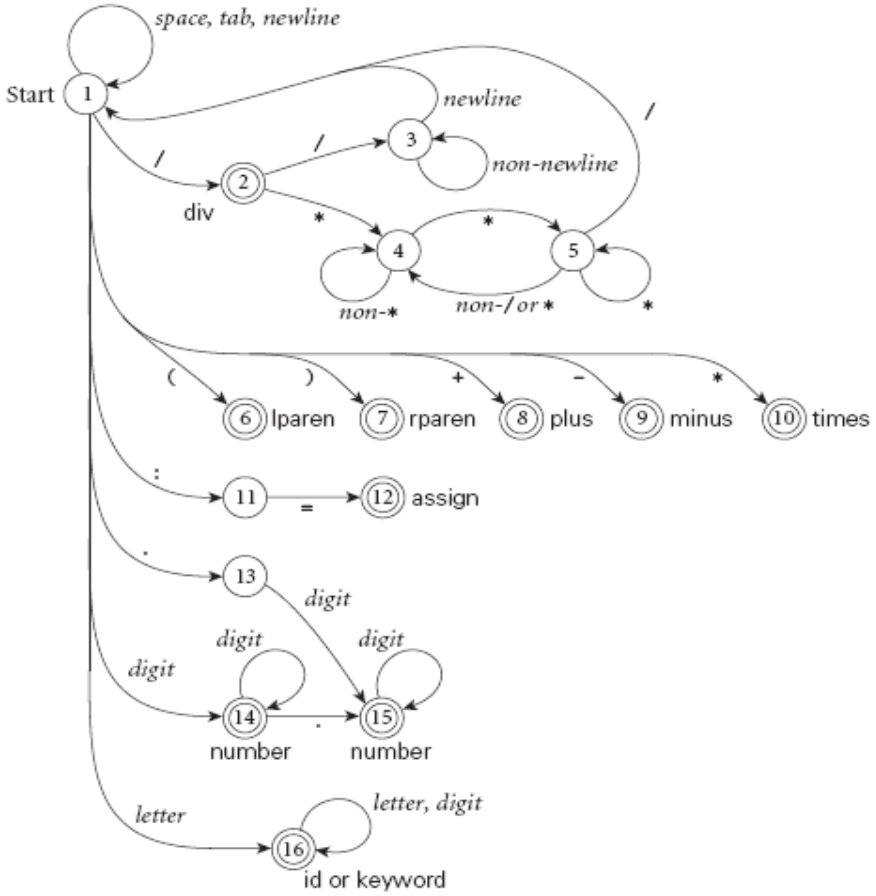
Minimal DFA for Scanning

The last two approaches require having a minimal deterministic finite automaton (DFA) for the tokens in the language.

Calculator Language (page 54)

assign → :=
plus → +
minus → -
times → *
div → /
lparen → (
rparen →)
id → letter (letter | digit)* except for read and write
number → digit digit* | digit* (.digit | digit .) digit*
comment → / * (non-* | * non=/)* |
// (non-newline)* newline

Minimal DFA for Calculator Language



Ad Hoc Scanner for Calculator Language

```
skip any initial white space (spaces, tabs, and newlines)
if cur_char ∈ {'(', ')', '+', '-', '*'}
    return the corresponding single-character token
if cur_char = ':'
    read the next character
    if it is '=' then return assign else announce an error
if cur_char = '/'
    peek at the next character
    if it is '*' or '/'
        read additional characters until "*/" or newline is seen, respectively
        jump back to top of code
    else return div
if cur_char = .
    read the next character
    if it is a digit
        read any additional digits
        return number
    else announce an error
if cur_char is a digit
    read any additional digits and at most one decimal point
    return number
if cur_char is a letter
    read any additional letters and digits
    check to see whether the resulting string is read or write
    if so then return the corresponding token
    else return id
else announce an error
```

Build Scanner Table for Language

1. Using the minimal DFA create a table showing the transitions from each state on each symbol.
2. Write code which reads characters from the source code and uses the table to determine if the characters make up a valid token.

(See Figure 2.11 & 2.12 on pages 66 and 67)

Scanner Tables

State	Current input character														
	space, tab	*newline*	/	*	(	)	+	−	:	=	.	*digit*	*letter*	*other*	
1	17	17	2	10	6	7	8	9	11	–	13	14	16	–	
2	–	–	3	4	–	–	–	–	–	–	–	–	–	–	div
3	3	18	3	3	3	3	3	3	3	3	3	3	3	3	
4	4	4	4	5	4	4	4	4	4	4	4	4	4	4	
5	4	4	18	5	4	4	4	4	4	4	4	4	4	4	
6	–	–	–	–	–	–	–	–	–	–	–	–	–	–	lparen
7	–	–	–	–	–	–	–	–	–	–	–	–	–	–	rparen
8	–	–	–	–	–	–	–	–	–	–	–	–	–	–	plus
9	–	–	–	–	–	–	–	–	–	–	–	–	–	–	minus
10	–	–	–	–	–	–	–	–	–	–	–	–	–	–	times
11	–	–	–	–	–	–	–	–	–	12	–	–	–	–	
12	–	–	–	–	–	–	–	–	–	–	–	–	–	–	assign
13	–	–	–	–	–	–	–	–	–	–	–	15	–	–	
14	–	–	–	–	–	–	–	–	–	–	15	14	–	–	number
15	–	–	–	–	–	–	–	–	–	–	–	15	–	–	number
16	–	–	–	–	–	–	–	–	–	–	–	16	16	–	identifier
17	17	17	–	–	–	–	–	–	–	–	–	–	–	–	white_space
18	–	–	–	–	–	–	–	–	–	–	–	–	–	–	comment

```
state = 0 . . number_of_states
token = 0 . . number_of_tokens
scan_tab : array [char, state] of record
        action : (move, recognize, error)
        new_state : state
token_tab : array [state] of token              -- what to recognize
keyword_tab : set of record
        k_image : string
        k_token : token
-- these three tables are created by a scanner generator tool

tok : token
cur_char : char
remembered_chars : list of char
```

Scanner

```
repeat
    cur_state : state := start_state
    image : string := null
    remembered_state : state := 0        -- none
    loop
        read cur_char
        case scan_tab[cur_char, cur_state].action
            move:
                if token_tab[cur_state] ≠ 0
                    -- this could be a final state
                    remembered_state := cur_state
                    remembered_chars := ϵ
                add cur_char to remembered_chars
                cur_state := scan_tab[cur_char, cur_state].new_state
            recognize:
                tok := token_tab[cur_state]
                unread cur_char          -- push back into input stream
                exit inner loop
            error:
                if remembered_state ≠ 0
                    tok := token_tab[remembered_state]
                    unread remembered_chars
                    remove remembered_chars from image
                    exit inner loop
                -- else print error message and recover; probably start over
        append cur_char to image
    -- end inner loop
until tok ∉ {white_space, comment}
look image up in keyword_tab and replace tok with appropriate keyword if found
return ⟨tok, image⟩
```

Scanner Tables

State	Current input character														
	space, tab	*newline*	/	*	(	)	+	−	:	=	.	*digit*	*letter*	*other*	
1	17	17	2	10	6	7	8	9	11	–	13	14	16	–	
2	–	–	3	4	–	–	–	–	–	–	–	–	–	–	div
3	3	18	3	3	3	3	3	3	3	3	3	3	3	3	
4	4	4	4	5	4	4	4	4	4	4	4	4	4	4	
5	4	4	18	5	4	4	4	4	4	4	4	4	4	4	
6	–	–	–	–	–	–	–	–	–	–	–	–	–	–	lparen
7	–	–	–	–	–	–	–	–	–	–	–	–	–	–	rparen
8	–	–	–	–	–	–	–	–	–	–	–	–	–	–	plus
9	–	–	–	–	–	–	–	–	–	–	–	–	–	–	minus
10	–	–	–	–	–	–	–	–	–	–	–	–	–	–	times
11	–	–	–	–	–	–	–	–	–	12	–	–	–	–	
12	–	–	–	–	–	–	–	–	–	–	–	–	–	–	assign
13	–	–	–	–	–	–	–	–	–	–	–	15	–	–	
14	–	–	–	–	–	–	–	–	–	–	15	14	–	–	number
15	–	–	–	–	–	–	–	–	–	–	–	15	–	–	number
16	–	–	–	–	–	–	–	–	–	–	–	16	16	–	identifier
17	17	17	–	–	–	–	–	–	–	–	–	–	–	–	white_space
18	–	–	–	–	–	–	–	–	–	–	–	–	–	–	comment