6 Constructing a Tokenizer for a Python Subset

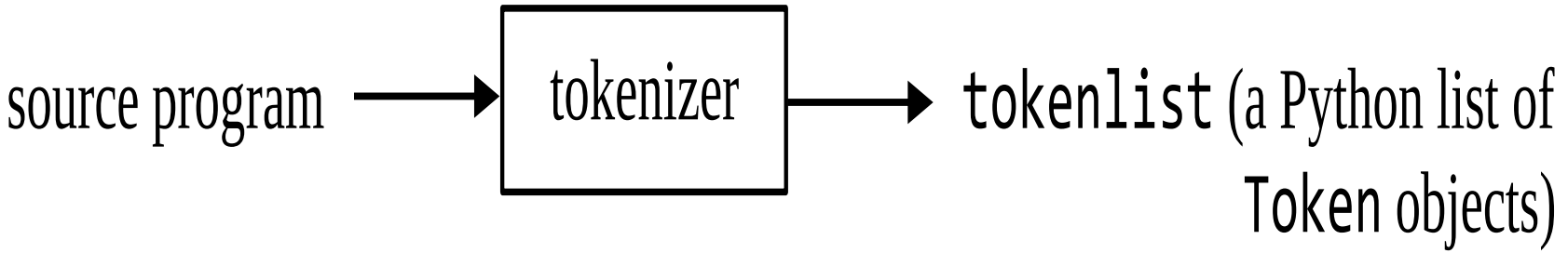

Figure 6.1

Source code	Line	Column	Category	Lexeme
a = (-59 + 20*3)	1	1	NAME	a
	1	3	ASSIGNOP	=
	1	5	LEFTPAREN	(
	1	6	MINUS	-
	1	7	UNSIGNEDINT	59
	1	10	PLUS	+
	1	12	UNSIGNEDINT	20
	1	14	TIMES	*
	1	15	UNSIGNEDINT	3
	1	16	RIGHTPAREN	)
	1	17	NEWLINE	
print(a)	2	1	PRINT	print
	2	6	LEFTPAREN	(
	2	7	NAME	a
	2	8	RIGHTPAREN	)
	2	9	NEWLINE	
	3	1	EOF	

Figure 6.2

```
 1 # t1.py tokenizer
 2 import sys          # sys needed to access cmd line args and exit()
 3
 4 class Token:
 5    def __init__(self, line, column, category, lexeme):
 6       self.line = line          # source prog line number of the token
 7       self.column = column      # source prog col in which token starts
 8       self.category = category # category of the token
 9       self.lexeme = lexeme      # token in string form
10
11 # global variables
12 debug = True             # tokenizer trace on
13 source = ''              # receives entire source program
14 sourceindex = 0          # index into source
15 line = 0                 # current line number
16 column = 0               # current column number
17 tokenlist = []           # list of tokens created by tokenizer
18 prevchar = '\n'          # '\n' in prevchar signals start of new line
19 blankline = True         # reset to False if line is not blank
20
```

```
21 # constants that represent token categories
22 EOF            = 0      # end of file
23 PRINT          = 1      # 'print' keyword
24 UNSIGNEDINT    = 2      # integer
25 NAME           = 3      # identifier that is not a keyword
26 ASSIGNOP       = 4      # '=' assignment operator
27 LEFTPAREN      = 5      # '('
28 RIGHTPAREN     = 6      # ')'
29 PLUS           = 7      # '+'
30 MINUS          = 8      # '-'
31 TIMES          = 9      # '*'
32 NEWLINE        = 10     # newline character
33 ERROR          = 11     # if not any of the above, then error
34
```

```
35 # displayable names for each token category
36 catnames = ['EOF', 'PRINT', 'UNSIGNEDINT', 'NAME', 'ASSIGNOP',
37             'LEFTPAREN', 'RIGHTPAREN', 'PLUS', 'MINUS',
38             'TIMES', 'NEWLINE','ERROR']
39
```

```
40 # keywords and their token categories}
41 keywords = {'print': PRINT}
42
```

```
43 # one-character tokens and their token categories
44 smalltokens = {'=':ASSIGNOP, '(':LEFTPAREN, ')':RIGHTPAREN,
45                '+':PLUS, '-':MINUS, '*':TIMES, '\n':NEWLINE, '':EOF}
46
```

```
47 # main() reads input file and calls tokenizer()
48 def main():
49     global source
50 
51     if len(sys.argv) == 2:   # check if correct number of cmd line args
52         try:
53             infile = open(sys.argv[1], 'r')
54             source = infile.read()  # read source program
55         except IOError:
56             print('Cannot read input file ' + sys.argv[1])
57             sys.exit(1)
58     else:
59         print('Wrong number of command line arguments')
60         print('format: python t1.py <infile>')
61         sys.exit(1)
62 
63     if source[-1] != '\n':         # add newline to end if missing
64         source = source + '\n'
65     if debug:                      # for token trace
66         print('Line  Col Category       Lexeme\n')
67 
```

```
68    try:
69       tokenizer()    # tokenize source code in source
70    except RuntimeError as emsg:
71      # output slash n in place of newline
72      lexeme = token.lexeme.replace('\n', '\\n')
73      print('\nError on '+ "'" + lexeme + "'" + ' line ' +
74         str(token.line) + ' column ' + str(token.column))
75      print(emsg) # message from RuntimeError object
76      sys.exit(1)       # 1 return code indicates an error has occurred
77
```

```
78 # tokenizer tokenizes tokens in source code and appends them to tokens
79 def tokenizer():
80     global token
81     curchar = ' '                          # prime curchar with space
82 
83     while True:
84         # skip whitespace but not newlines
85         while curchar != '\n' and curchar.isspace():
86             curchar = getchar() # get next char from source program
87 
88         # construct and initialize a new token
89         token = Token(line, column, None, '')
90 
```

```
91          if curchar.isdigit():           # start of unsigned int?
92              token.category = UNSIGNEDINT  # save category of token
93              while True:
94                  token.lexeme += curchar    # append curchar to lexeme
95                  curchar = getchar()        # get next character
96                  if not curchar.isdigit():  # break if not a digit
97                      break
98
```

```
99          elif curchar.isalpha() or curchar == '_':   # start of name?
100             while True:
101                 token.lexeme += curchar    # append curchar to lexeme
102                 curchar = getchar()            # get next character
103                 # break if not letter, '_', or digit
104                 if not (curchar.isalnum() or curchar == '_'):
105                     break
106
```

```
107             # determine if lexeme is a keyword or name of variable
108             if token.lexeme in keywords:
109                 token.category = keywords[token.lexeme]
110             else:
111                 token.category = NAME
112
```

```
113         elif curchar in smalltokens:
114             token.category = smalltokens[curchar]   # get category
115             token.lexeme = curchar
116             curchar = getchar()       # move to first char after token
117 
118         else:
119             token.category = ERROR    # invalid token
120             token.lexeme = curchar    # save lexeme
121             raise RuntimeError('Invalid token')
122 
```

```
123         tokenlist.append(token)      # append token to tokens list
124         displaytoken(token)
125         if token.category == EOF:
126             break
127
```

```
128 # getchar() gets next char from source and adjusts line and column
129 def getchar():
130     global sourceindex, column, line, prevchar, blankline
131 
132     # check if starting a new line
133     if prevchar == '\n':    # '\n' signals start of a new line
134         line += 1           # increment line number
135         column = 0          # reset column number
136         blankline = True    # initialize blankline
137 
```

```
138     if sourceindex >= len(source): # at end of source code?
139         column = 1                 # set EOF column to 1
140         prevchar = ''              # save current char for next call
141         return ''                  # null str signals end of source
142
```

```
143     c = source[sourceindex] # get next char in the source program
144     sourceindex += 1        # increment sourceindex to next character
145     column += 1             # increment column number
146     if not c.isspace():     # if c not whitespace then line not blank
147        blankline = False    # indicate line not blank
148     prevchar = c            # save current char for next call
149
```

```
150     # if at end of blank line, return space in place of '\n'
151     if c == '\n' and blankline:
152         return ' '
153     else:
154         return c              # return character to tokenizer()
155
```

```
156 def displaytoken(t):
157     if debug:
158         print("%3s %4s  %-14s %s" % (str(t.line), str(t.column),
159               catnames[t.category], t.lexeme))
160
161 main()                          # call main function
```

Figure 6.3