14 Constructing a Tokenizer Level 2

Introduction

```
print(-59 + 20*3)
a = 2
bb_1 = -(a) + 12
print(a*bb_1 + a*3*(-1 + -1 + -1))

# start of code that tests level 2 features
print()               # print with no args
if a <= 2:            # if, no else
   if a == 2:         # equal operator
      if a > 1:       # greater than operator
         pass         # does nothing
         print(3)
if a != 2:            # if-else
   print('err#or')    # this print should be skipped
else:                 # else
   print(4,)          # this print should be executed
while a >= 3:         # greater or equal operator
   print('error')     # this print should be skipped
a = ----+--+--5       # multiple unary operators
while a < 10:         # while, less than operator
   print(a)
   a = +a + .5 + 0.5  # floating-point computation
if True:              # True is a boolean constant
   pass               # pass does nothing
print('10\n11')       # escape sequence
print('1' + '2')      # string concatenation
print(-a + 23)        # unary minus operation
if a <= 10:           # less than or equal operator
   print('#14\'\\')   # pound, quote, backslash in string
print(15, 16)         # one space between numbers
if (None != False):   # None, False, not equal operator
   print(51/(5 - 2))  # subtraction
```

Figure 14.1

Required Modifications

- No modification required

  subtraction operator

- Trivial modification:

  `pass`
  `if`
  `while`
  `True`
  `False`
  `None`
  `/` (floating-point division)

- Requiring a little thinking

  `==` (equal)
  `!=` (not equal)
  `<` (less than)
  `<=` (less than or equal)
  `>` (greater than)
  `>=` (greater than or equal)
  floating-point constants
  source code comments
  strings delimited by single quotes

- Requiring serious head scratching

  Python indentation

Figure 14.2

Adding Support for Two-character Tokens

```
smalltokens = {'=':ASSIGNOP, '==':EQUAL, '<':LESSTHAN,
               '<=':LESSEQUAL, '>':GREATERTHAN, '>=':GREATEREQUAL,
               '!':ERROR, '!=':NOTEQUAL, '(':LEFTPAREN,
               ')':RIGHTPAREN, '+':PLUS, '-':MINUS, '*':TIMES,
               '\n':NEWLINE, '':EOF, ',':COMMA, ':':COLON, '/':DIV}
```

Value of "!" is ERROR

```
elif curchar in smalltokens:
   save = curchar
   curchar = getchar()  # curchar might be '' (end of source code)
   twochar = save + curchar           # two chars if curchar != ''
   if twochar in smalltokens:
      token.category = smalltokens[twochar]  # get category
      token.lexeme = twochar                 # get lexeme
      curchar = getchar()                    # move past end of token
   else:                                     # one-char token
      token.category = smalltokens[save]     # get category
      token.lexeme = save                    # get lexeme
```

Adding Support for Floating-Point Constants

Change category if decimal point detected.

Adding Support for Comments

```
c = source[sourceindex]
sourceindex += 1
```

If c is # (i.e., the start of a comment), then advance until the newline character is reached.

Adding Support for String Data

Must record that string processing in progress.

Adding Support for Python Indentation

Using indentation

```
if a == 1:
   print('hello')
   if b == 2:
      print('goodbye')
      if c == 3:
         print('up')
print('down')
# end of program
```

Using braces (illegal in Python)

```
if a == 1:
{
   print('hello')
   if b == 2:
   {
      print('goodbye')
      if c == 3:
      {
         print('up')
      }
   }
}
print('down
```

Figure 14.3

Using indentation

```
1 if a == 1:
2    print('hello')
3    if b == 2:
4       print('goodbye')
5       if c == 3:
6          print('up')
7    print('down')
8 # end of program
```

Using braces (illegal in Python)

```
 1 if a == 1:
 2 {
 3    print('hello')
 4    if b == 2:
 5    {
 6       print('goodbye')
 7       if c == 3:
 8       {
 9          print('up')
10       }
11    }
12    print('down')
13 }
```

Figure 14.4

```
indentstack = [1]
```