8 Constructing a Pure Interpreter

Structures Needed for a Pure Interpreter

```
symtab = {}


symtab['x'] = 1
```

creates an entry for `'x'` and associates with it the value `1`. If we then execute

```
symtab['x'] = 7


print(symtab['x'])

if 'x' in symtab:
   ...
```

```
operandstack = []

operandstack.append(symtab[token.lexeme])

symtab[token.lexeme] = operandstack.pop()
```

Modifications to Our Parsing Functions

```
1 def assignmentstmt():
2     left = token.lexeme # save lexeme of the current token
3     advance()
4     consume(ASSIGNOP)
5     expr()                # leaves value of expr on stack
6     symtab[left] = operandstack.pop()
```

```
1 def printstmt():
2     advance()
3     consume(LEFTPAREN)
4     expr()          # leaves value of expr on the stack
5     print(operandstack.pop())
6     consume(RIGHTPAREN)
```

```
1 def expr():
2     term()                # pushes value of term onto top of stack
3     while token.category == PLUS:
4         advance()
5         term()            # pushes value of term onto top of stack
6         rightoperand = operandstack.pop()
7         leftoperand = operandstack.pop()
8         operandstack.append(leftoperand + rightoperand)
```

```
 1 def term():
 2     global sign
 3     sign = 1          # initialize sign before calling factor()
 4     factor()          # leaves value of term on top of stack
 5     while token.category == TIMES:
 6         advance()
 7         sign = 1      # initialize sign before calling factor()
 8         factor()      # leaves value of term on top of stack
 9         rightoperand = operandstack.pop()
10         leftoperand = operandstack.pop()
11         operandstack.append(leftoperand * rightoperand)
```

```
 1 def factor():
 2    global sign
 3    if token.category == PLUS:
 4       advance()
 5       factor()
 6    elif token.category == MINUS:
 7       sign = -sign                    # flip sign for each unary minus
 8       advance()
 9       factor()
10    elif token.category == UNSIGNEDINT:
11       operandstack.append(sign*int(token.lexeme))
12       advance()
13    elif token.category == NAME:
14       if token.lexeme in symtab:
15          operandstack.append(sign*symtab[token.lexeme])
16       else:
17          raise RuntimeError('Name ' + token.lexeme + ' is not defined')
18       advance()
19    elif token.category == LEFTPAREN:
20      advance()
21      # save sign because expr() calls term() which resets sign to 1
22      savesign = sign          # sign is global but savesign is local
23      expr()                   # value of expr is pushed onto operandstack
24      if savesign == -1:       # use the saved value of sign
25         operandstack[-1] = -operandstack[-1] # change sign of expr
26      consume(RIGHTPAREN)
27    else:
28       raise RuntimeError('Expecting factor')
```