Hints on Tokenization and Parsing

6 "READ" **2** "+" **8** "x" **4** "--"

The tokenizer reads in characters, "chunks" them and spits Tokens out the back end.

R E A D + x - - * 1 2 7

Tokens are just a container class holding a code (int) and content (String).

Tokenizer

state: seeking op or arg?

(switches on token code)

Parser

| △ |
| △ |
| △ |

| ☐ |
| ☐ |
| ☐ |

Arg stack   *Op stack*

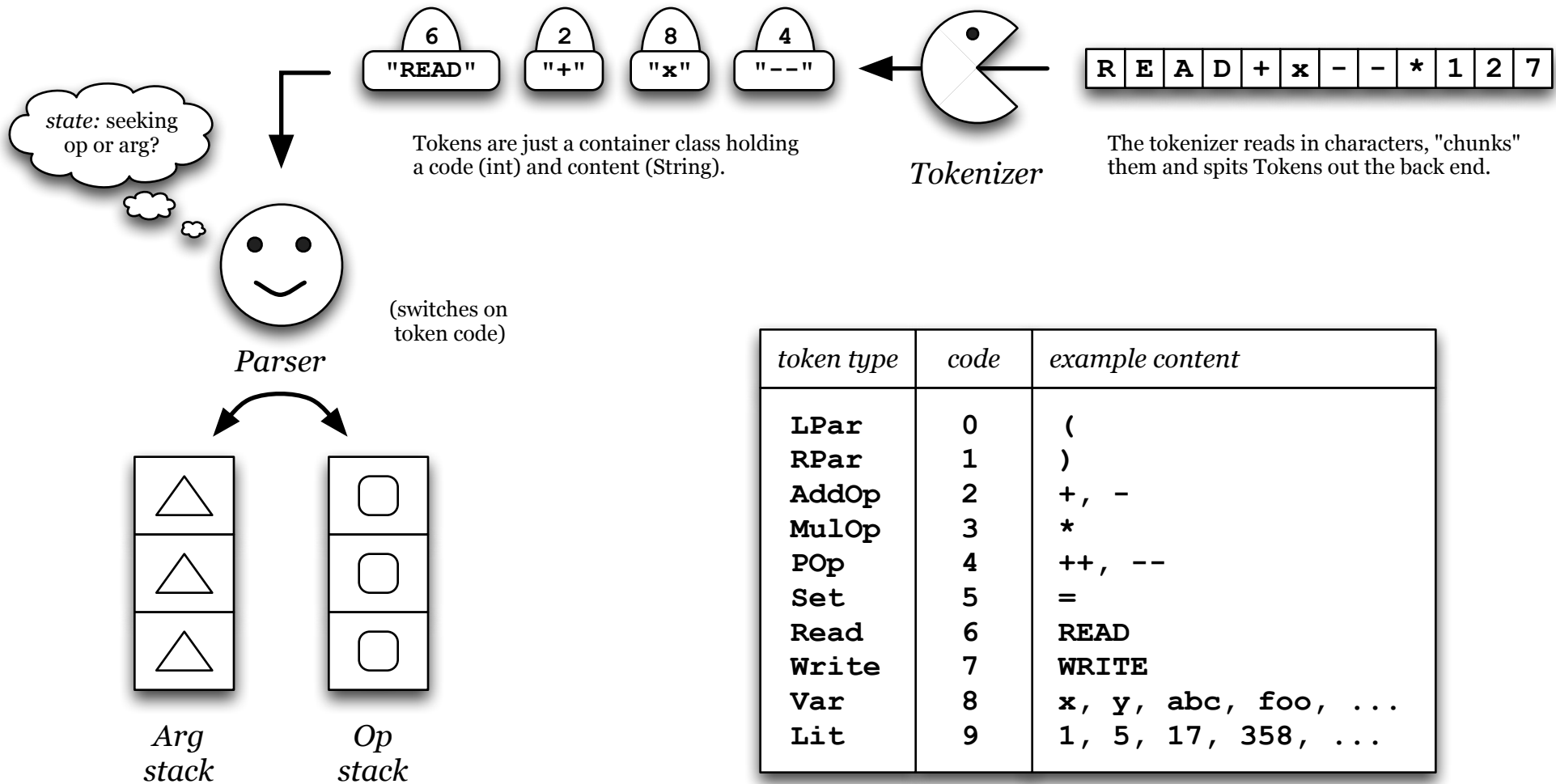| token type	code	example content
LPar	0	**(**
RPar	1	**)**
AddOp	2	**+, -**
MulOp	3	*****
POp	4	**++, --**
Set	5	**=**
Read	6	**READ**
Write	7	**WRITE**
Var	8	**x, y, abc, foo, ...**
Lit	9	**1, 5, 17, 358, ...**

Depending on which type of token we get, we either:
• create a leaf expression and push it on the arg stack;
• push an operator token on the op stack;
• reduce an op token and its (popped) sub-exprs into a larger expression and push it on the arg stack.

Depending on precedence, we may hold an op token while we reduce some ops below it on the op stack.

(see on-line write-up for more details)

We group the tokens into "token types" depending on how they will be handled: + and - have the same precedence, ++ and -- are handled in exactly the same way.

Different variables and literals are all treated the same as far as parsing is concerned (but we keep their content for use later).