

~~unit~~ → special

“...|...” → special / specials

“sisa|;--” → SI, sai

“...|...” → special / specials

“sisa|;” → SI, sai

⌀[comment]

$\#_b$
 $\#_c \text{ } s1; \#_s \text{ } s2; s3;$

$\#_0 = "os"$
 $\#_c = \text{column}$
 $\#_s = [s/ ""]$

$s1 = \text{Bottle}$
 $s2 = \text{on the}$

$s3 = \text{wall}$
 $s4 = \text{beer}$

$\#_c \text{ } s1; \#_s \text{ } s2; s3;$

$\#_c \text{ } s1; \#_s \text{ } s2; s3; \Rightarrow \text{all chars}$

Strings to standard
(no uncompressing)

'... ' → Nothing, already standard

q...q → Add spaces after ; if unescaped & two chars before (that are valid see char)

Then replace q w/ ' (leading & trailing)

q ... q → Every two chars, add a ; & a space then replace leading & trailing q w/ ' ,

q3...q3 → Like above, but w/ no spaces

«...|...» → ① Split the format section into a list of expressions

② Every two chars, add an item from the format list & a space

③ « → ' ,

Normal strings - Takes standard
chars & SCCs as is

Q...Q
" " " "

Spaceful strings - Same as above
but spacey after SCCs

99...99

Spaceless compressed strings -
SCCs only, no spaces

'...'

Spaceful compressed strings -
like above, but with spaces

LL...LL

LL...LL

~~Space~~

~~Space~~

Special compressed strings

This needs a
new page.

Differences between Compressed & uncompressed strings



Points of contention:

- * Spaces between SCCs
- * String delimiters

* Length

✓ Hello, World! ✓
↓ ↓ ↓
x x x

`s1; "sa;!" → 9 + 2

New idea:
two types of strings

cells → ' ' → as is, no spaces between SCCs

cell → " " → spaces between SCCs

Conditional Params

~ to

@name []

@name - [else] | ... @

Tries to pop item from stack

& use that many Params

else, uses given data ..

Adds n items into new array

e.g. [1, 2, 3]

@A; → [1, 2, 3]

2 A_ [*] | `n []` (! 1 - | +) @

Returns 1 if all items in array are true

~~2 AA *~~

2 AA 1 | ! f c 0 ^ (! 1 - | 0 = [n 1 +]) * c = [1 | 0] @

Key + BF

@A; → @Z;

Section A - Array Functions

@A * | ^nLJi (+) @ # Convert all items into one big # array

~~@AA*~~

@AA 1 | (+) @ # Array add (reduction)

@AB

n

e

int i;

int i;

int i;

int i;

Integer
↓
Type

int i;

int i;

int i;

int i;

~~char~~
↓
String
Type

int i;

int i;

int i;

char
↓
Type

int i;

Array
↓
String

int i;

int i;

int i;

int i;

Object
↓
type

The Uncompressor

* Used when either printing, comparing or

* Uncompressed upon start

* All strings turned into standard strings (text only)

Eg $S1 = \text{"Hello"}$ $S2 = \text{"world"}$

Before $S3 = \text{"nice"}$ $XS = [S1][S2]$ → default S

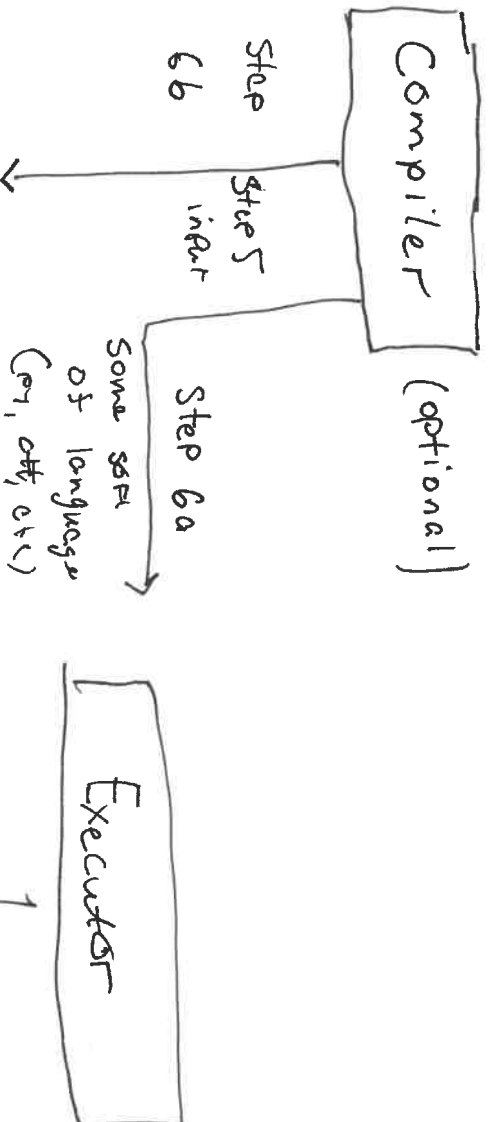
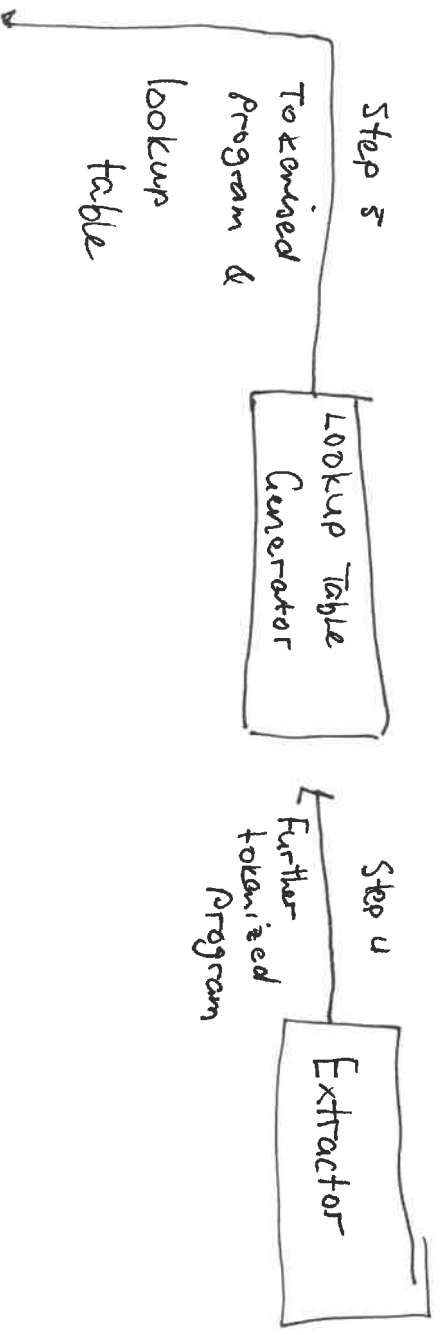
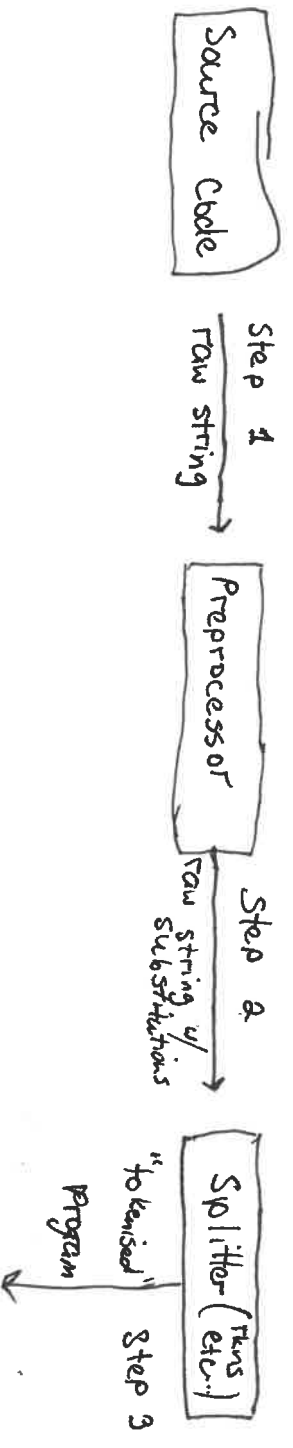
After "Hello"

$S1;$

"Hello"

$S1;XS$

SS



Output

A Library

Convert main stack items into an array

@ A -[*] | ^n [] ^ (: 1 - / +) @

Add all items in array

@ AA 1 | ^n (+) @

@ AA 1 | (: 1 - / +) @

Convert all items in array into binary representation
if possible

@ AA 1 |

[3, 4] → ✓

[2, 0] → ✗

[3] → ✓

NO.

Structure extractor - keg

```
def is_block(source):  
    """  
    Given a source, this function will try to extract  
    the data from an if statement "[expr-true|expr-false]"  
    and place it in a dictionary"  
    """
```

Interpreter Parts

Phase 1 → (Substitution)

- The Preprocessor
 - Comes along and replaces references to the SSL (&...) with the corresponding sequence
 - Doesn't replace:
 - Occurrences in strings
 - escaped &'s
 - Not too much of a Syntax check (very light strings & " ")

- The Uncompressor
 - Replaces SCCs in strings
- The Tokenizer
 - Splits source into "tokens"
 - Literals string command Sn...

⑤ The Preprocessor:

Begin Preprocess(source)

processed = ""

string-mode = escaped = false

For each character in source

~~If string mode~~

If character is

"\", \" or \"'\" or \"<\"
// or other
string
type,

If escaped

escaped = false

processed += character

Else

String-mode = toggle(string-mode)

processed += character

Else, If character is \" \"

A Second look @ string compression...

Examples

3 letter
SCCs

'e a x'; \rightarrow e a x

'e a x | i'; \rightarrow e a x;

'a b c d i'; \rightarrow a b c d

2 letter
SCCs

'| e a x'; \rightarrow e a x

'a x i'; \rightarrow a x

'a x | i'; \rightarrow a x;

1 letter
SCCs

'| e | a x'; \rightarrow e a x

'e a | x'; \rightarrow e a x;

'x i'; \rightarrow x

'e | a x'; \rightarrow e a x

'x | i'; \rightarrow x;

Object Strings

The ^{in the} undervalued backbone of keg + part

A brief overview:

is

* contained within ~~standard~~ strings (is)

* To be implemented in the interpreter

* Allows for non-keg constructs to easily be made

* not preprocessed in any way

* Allows for BFL to be 100% keg

* Needs own library/file

Design

data
; n | E I ;
; n | . . . ;

operative
command
; n | ;
; c | S > I ;

; f | r . 1 ;
; f | r . + ;

Test ambiguity

1 Begin Standard (String)

2 create var result & store an empty string

3 - - - If String → type is " " then

4 - - - - - Create var code & store an empty string

5 - - - - - Create var escaped & store False

6 Append " " to result ^{reversed}

7 - - - For each char in string → data, do

8 - - - - - If escaped then

9 Toggle escaped

10 - - - - - Continue

11 - - - - - Otherwise, If char is " " then

12 - - - - - If code → length is 2 then ^{reversed}

13 - - - - - Append suc → code to result

14 - - - - - Set code to an empty string

15 - - - - - otherwise, Is Char is " \" then

16 - - - - - Set escaped to True

17 continue

18 - - - - - ~~If~~ If code → length is 2 then

19 - - - - - Append code[0] to result

20 Set code to code[1] + char

21 ~~Return result~~ → reversed

22 Otherwise

Append Char to code

Form

;
[C] | . . . 1 ;

where:

[C] is the command

...1 is the body



stacks

↳ dicts

↳ strings

↳ files

in [C];
in \$3;
in " ";
in \$file;

C → cast (next page)

A → file
 ↳ open \$f/o; "ing" → text
 ↳ read \$f/r.1; ; \$f/r.t;
 ↳ write/append ; \$f/w; ; \$f/a;
 ↳ \$f/a.b;
 ↳ \$f/l.b;

S → system

etc...

js/platform; operating sys version

js/version;

des extract

"a8b" → [{"name": "cmd", "data": "a"}, {"name": "cmd", "data": "5"}, {"name": "cmd", "data": "b"}]

"[1|0]" → [{"name": "if", "data": {}}, {"name": "cmd", "data": "1"}],
False: [{"name": "cmd", "data": "0"}]]

"(0799|ip|Gip *ip)" → ~~[...]~~
[{"name": "for", "data": {}}, {"name": "cmd", "data": "0"}, {"name": "cmd", "data": "7"}, {"name": "cmd", "data": "9"}, {"name": "cmd", "data": "9"}]

structure

EXTRACTOR

Goal: A modular single function to return all data parts of a structure:

[onTrue|onFalse] (count|var|code) {cond|type|code}

Assumed
prog is balanced @name count[] | code @

But also, parse in a sense

def extract(source):

for

final = []

data = {}

temp = ""

structures = [] # structure stack
current = ""

for i in range(len(source)):

char = source[i]

if char not in "[](){}`" or escaped

OVERVIEW

Standard char

if

for

{ "name": "char",

"data": ... }

{ "name": "if",

data: {

"true": ...,

"false": ...,

}

}

Special strings

«515253341,1,1. LL

s_1, s_2, s_3, s_4

$$c_1 c_2 c_3 \dots c_n$$

$$S1C_1 \bigcap S2C_2 \bigcap S3C_3 \bigcap \dots \bigcap SNC_N$$

$$\begin{array}{c} \ll \\ \text{SIS23} \dots \text{SN} \mid c_1 c_2 c_3 \dots c_{N-1} \ll \end{array}$$

$$S1C_1 \quad S2C_2 \quad S3C_3 \quad \dots \quad S(N-1)_{C_{N-1}} \quad SN$$

$$L_1 \text{ SSSSS } \dots S_N \mid C_1 C_2 C_3 \dots L_N$$

$$SIC_1 \quad SAC_2 \quad SSC_3 \quad \dots \quad S_N$$

$$\mathbb{Z} \oplus \mathbb{Z} \oplus \mathbb{Z} \dots \oplus \mathbb{Z} \mid \mathbb{C} \mathbb{C} \mathbb{C} \dots \mathbb{C}_N \mathbb{Z}$$

$$S(C_N, S_2 C_{N-1}, S_3 C_{N-2}, \dots, S(N-2) C_3, S(N-1) C_2, S(N) C_1)$$

19

' $\text{\textyen}x$ S1; ~~$\text{\textyen}bs$~~ S2; S3; S4;'

1 2 3 4 5 6

' $\text{\textyen}x$ S1; $\text{\textyen}s$ S2; S3; S4;'
→ 22 (24 auto comp)

LL $\text{\textyen}x$ S1 S2 S3 S4 | ' $\text{\textyen}s$ os' LL
→ 20 (19 auto comp)

LL $\text{\textyen}x$ S1 $\text{\textyen}s$ S2 S3 S4 | 'os' LL → 19
(18 auto comp)

case	space
Var → SCC	✓
SCC → Var	✗
Var → Var	✓
SCC → SCC	✓

~~LL $\text{\textyen}x$ S1 $\text{\textyen}s$ S2 S3 S4 LL~~

LL $\text{\textyen}x$ S1 $\text{\textyen}s$ 0 S2 S3 S4 LL → 15
(14 auto)

SCC only

□ 515253 □

□ is the delimiter.

Simply start @ delimiter & move through string
until second delimiter. If the count
of char is odd - error.

If the char contain " | ; ' " « » , * > "
→ Error.

Otherwise, Split into pairs & do a
Simple loop

11 - Strings

Objects

Stacks \rightarrow `inl[];` \rightarrow pushed on empty stack
Strings \rightarrow `in/" ";` \rightarrow " " string
files \rightarrow `in/file;` \rightarrow creates a new file
ports \rightarrow `in/port(i);` \rightarrow creates a new internet conn
~~dictionary~~

Math obj \rightarrow `in/math[property];` \rightarrow creates a new math object

Stack obj's

`inl[];` \rightarrow empty

`inl[0,1,2];` \rightarrow `[3,2,3]`

`inl["hello"];` \rightarrow `["hello"]`

N3.

* Aster

Preprocessor,

Aster

1 Hello

Hells

nice \times s (no spaces)

Hello world

142110

Nice xs

Hello World

Hello world

Hello, world!

~~1st~~ nice's

- Hello, world!

nicely world

- Hello! world! -

OR

Simplify

change by

20

backticks

Extra

Processing
needed

abdeefgh 8

[0, 2, 4, 6]

[s[i:i+2] for i in range(0, len(s), 2)]

s[0:2]

s[2:4]

LL SCCS | split LL

~~LL SCCS | split LL~~

LL codes [] joins LL

The Preprocessor → Done!

* Implements the `SSL (&<char>)`

Draft 3

def preprocess(source):

ginal = ""

in_string = False

escaped = False

for char in source:

if char in '"\`' :

if escaped:

escaped = False

else:

in_string = toggle(in_string)

or Not a string

elif

END

"Failure because string type not stored"

else:

if char == "&":

if escaped;

escaped = False

elif string_type != "":

SSL-needed = True

Continue

else:

SSL-needed = False

final += char

⑦ is SSL-needed:

final += SSL[char]

SSL-needed = False

Draft 2

def preprocess(source):

final = ""

string_type = ""

escaped = False

SSL-needed = False

for char in source:

① if char in STRING_CHARS:

if string_type == char:

if escaped:

escaped = False

else:

string_type = None

Indent → elif ~~else~~ is string_type == "\":

if escaped:

escaped = False

outdent ← else:

string_type = char

~~else:~~

elif char == "\":

if escaped: Escaped = False

else: escaped = True

And look at Special compressed strings

Delimiter: \llcorner

~~Spaceless~~ by
Spaces of
delimiter

$\llcorner \text{sisal} \llcorner, \llcorner \llcorner \rightarrow \text{sl, s2}$

$\llcorner \text{slsal} \llcorner, \llcorner \llcorner \rightarrow \text{sl sa,}$

| means that args are ~~presented~~ applied
left to right

| means that args are applied
right to left

Form:

$\llcorner \dots 1 \Delta : \dots 2 \llcorner \left| \begin{array}{l} \dots 1 = \text{scs} \quad \dots 2 = \text{separators} \\ \vdots = | \text{ or } ! \end{array} \right.$

$\begin{array}{l} 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \\ \llcorner \text{slsal} \llcorner, ! \end{array} \rightarrow 8 \text{ byte "Hello world"}$

$\dots 1$ The sc part

Design:

$\#x \text{sls2}$

$\llcorner \#x \text{sls2sls2sls2sls2} \llcorner$

<< ... | ... >>
↑ ↑
Body Format

Body

↳

sccs only. No vars or other chars

As

Format

↳ List of expressions

↳ keywords → and go

↳ only standard strings remain

a keyword

↳ Escaping allowed

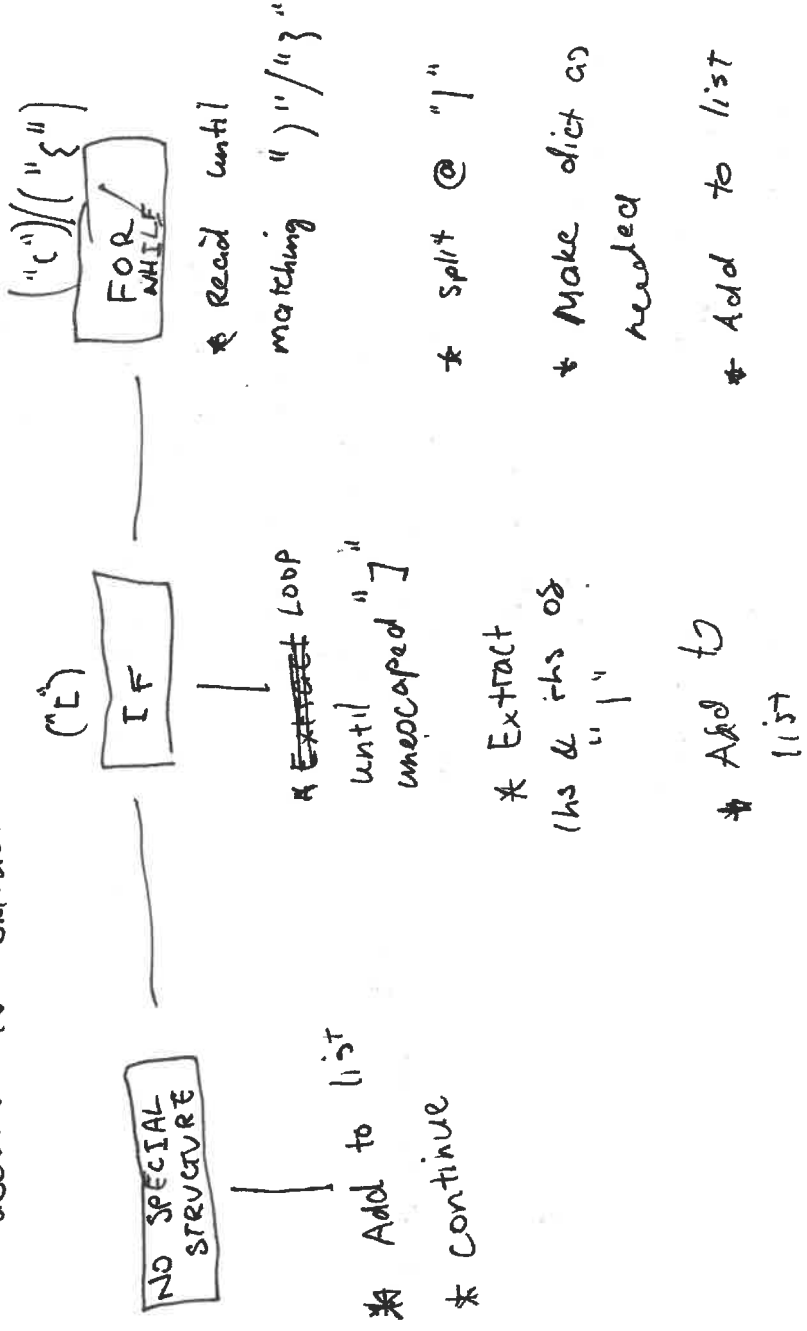
<< sss2 | 'hi' >> → sss hi s2

```

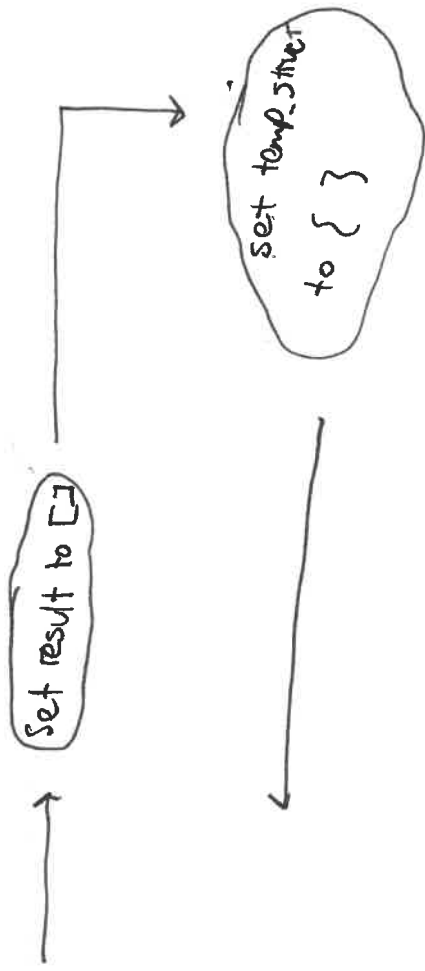
{
  name: . . .
  data: . . .
}

```

- * list of structures
- * store current section of data
- * temporary list of dictionaries
- * data dictionary
- * subcalls to extract



Function Extract
→ Source



Function Extract(source)

↳ Result = []; escaped = False

Temp_struct = {}; temp = ""

structures = []

For char in source:

 If structures is empty:

 If char in "[{(@" and not escaped:

 structures.append(char)

 Else:

 Results.append({name: "cmd", data: char})

 Else, if char matches opening bracket at pos -1 in struct:

 structures.pop()

 If char == "}" :

 lhs, rhs =

~~S1 = "Hello world"~~

S1 = Hello S2 = world Goal: "Hello, world!"

Uncompressed :

"Hello, world!" → 15 (14 a)

Compressed, spaceless:

"S1, S2!" → 11 (10 a)

Compressed, spaceful:

Impossible (no punctuation)

Compressed, special, structure:

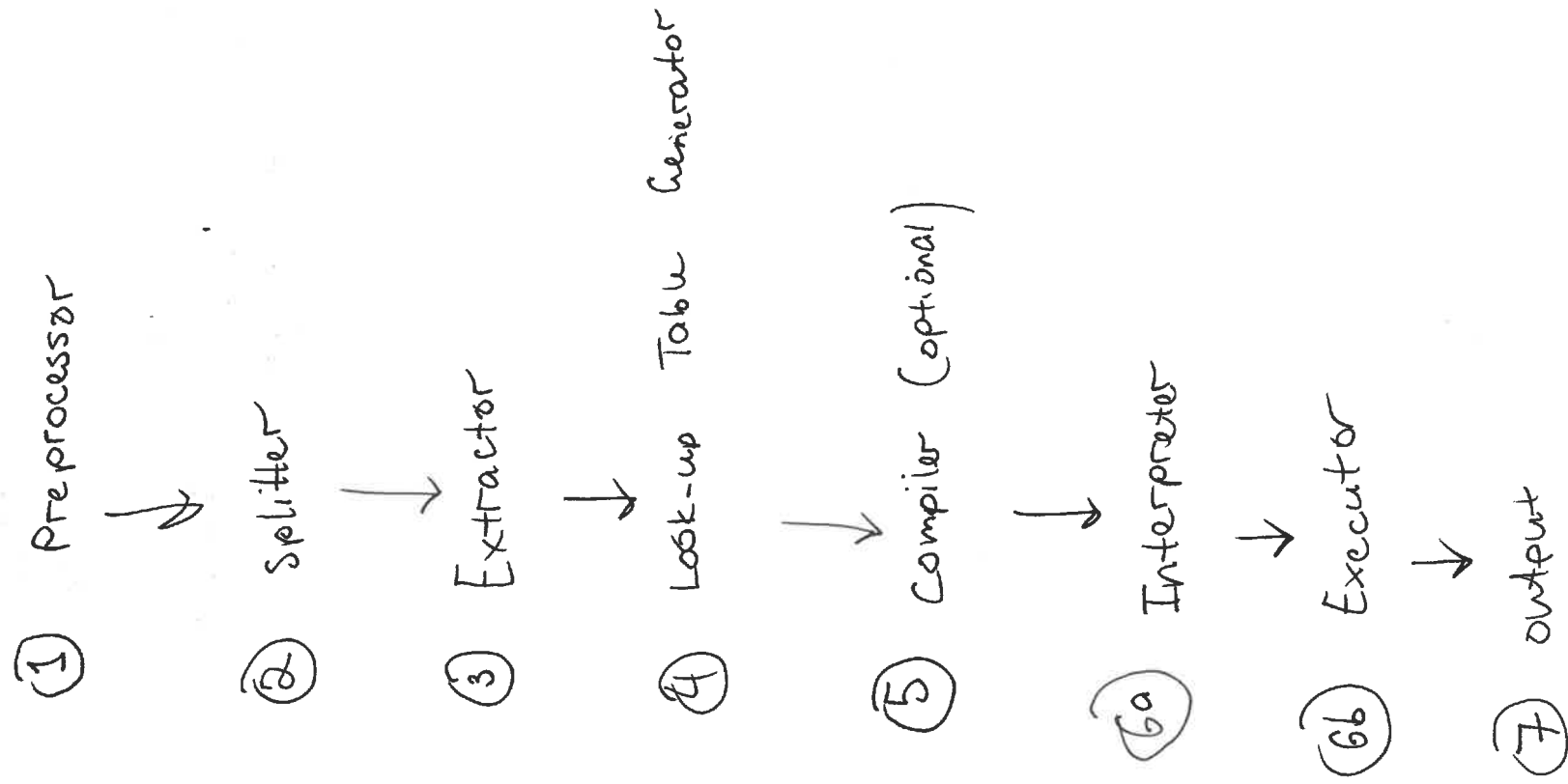
«S1S2!» 9 (8 a)

Compressed, special, constants

~~«S1S2!»~~

«S1S2!» 10 (9 a)

7 Steps



hello, world!
s1 s2

s1, s2! → 11

q s1; s2; q → 8 (but no punctuation)

~~>> s1 s2 <<~~

~~>> s1 s2 <<~~ → 10

<< s1 s2, ! → 8 chars!

Spaceful Strings

9 contents 9

like normal spaceless strings, but

spaces are placed between

secs:

Hello World!

`s1; sa;`

9 s1; sa; 9

1

hmm

→ 1 byte
shorter

Spaceful Special Compressed Strings

Delimited by <<

Usage:

<< sccs | Separators <<

✓
any series of sccs (no i)
↓
each separator is either a single char, or a variable and is placed after

Examples the corresponding scc.

s1 = Hello s2 = world s3 = Bottles s4 = bottle
%a = s %b = , %c = " "

<< s1s2, ! << → Hello, world!

<< s1s2[%b]! << → " " " "

<< s3 << → Bottles

<< s4 | %a << → Bottles

String types

'...' → standard/as-is / spaceless

's1;', sa;' → s1, sa!

q...q → standard/spaceful

's1;', sa;' s1, sa!

'...' → SCC only / spaceless

's1sa' → s1sa

''...'' → SCC only / spaceful

''s1sa'' → s1 sa