

$\ll \dots \ll \rightarrow \text{special}$

$\ll \dots | \dots \ll \rightarrow \text{special / spaceless}$

$\ll s_1 s_2 |, \dots \ll \rightarrow s_1, s_2!$

$\ll \dots | \dots \ll \rightarrow \text{special / spaceless}$

$\ll s_1 s_2 |, ! \ll s_1, s_2!$

$\emptyset[\text{comment}]$

$\#b$
 $\#c \ S1; \#s \ S2; \ S3;$

$\#0 = "0s"$

$\#c = \text{com}$

$\#s = [s/ ""]$

$S1 = \text{Bottle}$

$S2 = \text{on the}$

$S3 = \text{wall}$

$S4 = \text{beer}$

~~$\#c \ S1; \#s \ 0s \ S4; \ S2; \ S3;$~~

$\#c \ \square \ S1; \#s \ \square \ 0s \ \square \ S4; \ \square \ S2; \ \square \ S3; \ \square \Rightarrow 26 \text{ chars}$

$\square \ S1$

strings to standard
(no uncompressing)

`...` → Nothing, already standard

q...q → Add spaces after q IF unescaped & two chars before (that are valid sec chars)

Then replace q w/ ` (leading & trailing)

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

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

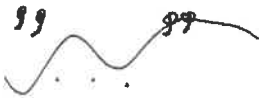
«...|...» → ① split the format section into a list of expressions

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

③ « → `

\ . . . \

9 . . . 9

99  99

99 . . . 99

9 . . . 9

LL . . . LL

LL . . . LL

Normal strings - Takes standard
chars & SCCs as is

Spaceful strings - Same as above
but spaces after SCCs

Spaceless compressed strings -
SCCs only, no spaces

Spaceful compressed strings -
like above, but with spaces

~~Space~~

~~Smr~~

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;', 's2;!' → 9 + 2

New idea:

two types of strings

celess → ' — ' → as is, no spaces between SCCs

ce sul → ¶ — ¶ → spaces between SCCs

Conditional params

~ to

@ name [-]

@ name [-else] | ... @

Tries to pop item from stack

& use that many paramse

else, uses given data!!

Adds n items into new array

e.g. [1, 2, 3]

@A; → [[1, 2, 3]]

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

Returns 1 if all items in array are true

~~AA *~~

@AA 1 | !{c 0^(!1-1 0=[1+]) %c = [1|0]}@

keg+ BFL
@A; → @z;

Section A - Array Functions

@A * |`n[]i`(+)@ # Convert all items into one big
array

~~@AA *~~

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

@AB

n	c
; n C I;	; c I > S;
; n ...;	; c I > C;
	; c I > A;
	Integer ↓ Type
	; c S > I;
	; c S > C;
	; c S > A;
	Char String ↓ Type
	; c C > I;
	; c C > S;
	; c C > A;
	Char ↓ Type
	; c A > S;
	Array ↓ String
	; c O > I;
	; c O > C;
	; c O > S;
	; c O > A;
	Object ↓ type

The Uncompressor

* used when either printing, comparing or

* uncompressed upon start

* All strings turned into standard strings (text only)

S1 = "Hello" S2 = "World"

Eg

Before

'Hello'

'S1;'

'S1; %s'

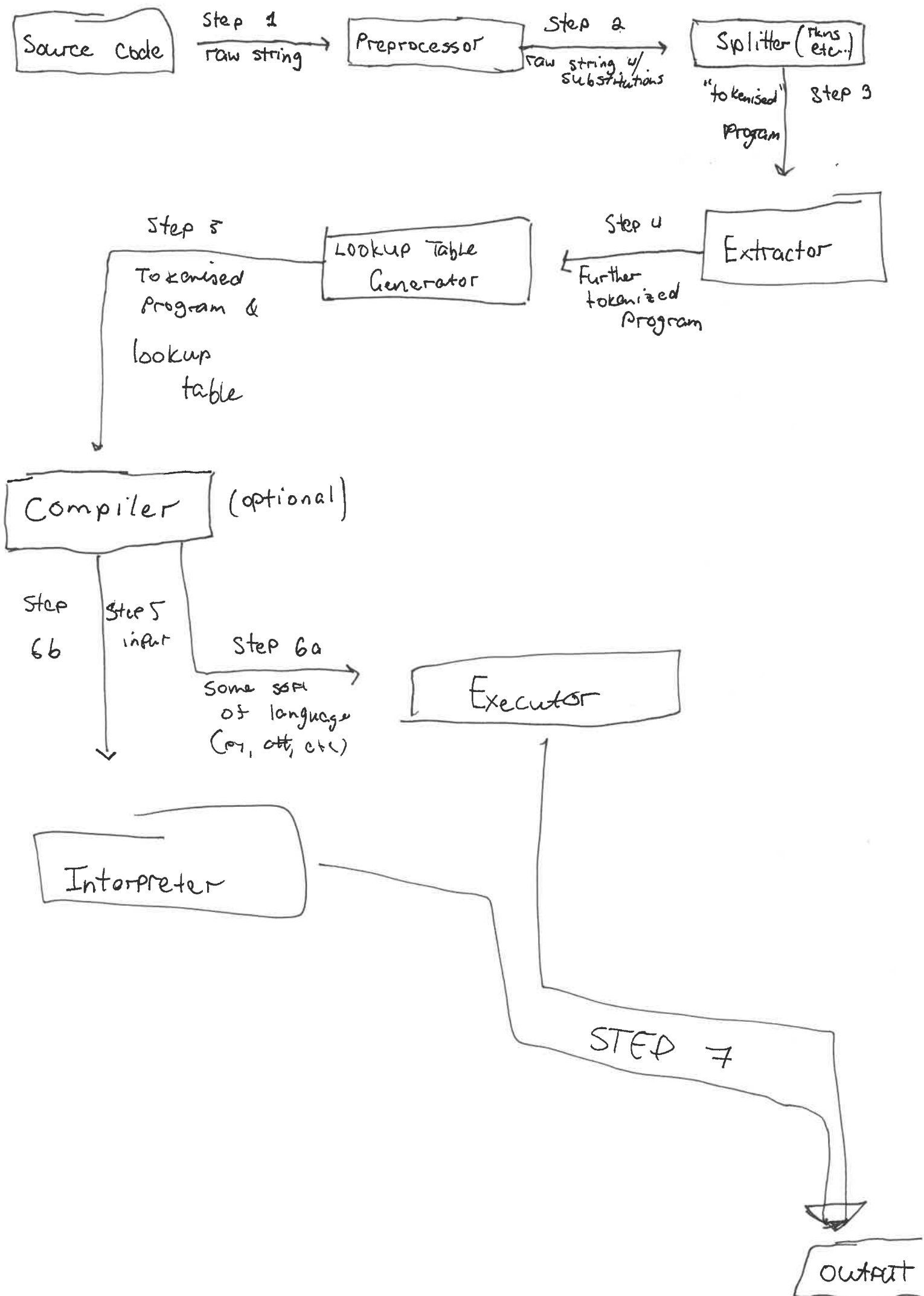
S3 = "nice" %s = [s1""] → default
s

After

'Hello'

'Hello'

~
S1



A Library

Convert main stack items into an array

@ A -[*] | 'n[]' (!1-/+) @

Add all items in array

@ AA 1 | " (+) @

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

Convert all items in array into binary representation
is possible

@ AB 1 |

[3, 4] → ✓

[2, 0] → x

[3] → ✓

NO.

Structure extractor - kept

```
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

Not too much of a
Syntax check (very light strings
& "&")

Doesn't replace:

- occurrences in strings
- escaped &'s

- The uncompressor



Replaces SCCs
in strings

- The Tokenizer

Splits source
into "tokens"

↓
Literal
string
Command
Sn...

① The Preprocessor:

Begin Preprocess(source)

processed = ""

string-mode = escaped = false

For each character in source

~~If~~ string mode

If character is "\", " or " or "<" // or any 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	2 letter SCCs	1 letter SCCs
'eax;' → <u>eax</u>	'\eax;' → e <u>ax</u>	'\e\ax;' → ea <u>x</u>
'eax\j' → eaxj	'axj' → <u>ax</u> j	'ea\x;' → ea <u>x</u> ;
'abcdj' → a <u>bcd</u>	'ax\j' → axj	'x;' → <u>x</u>
		'\e\ax;' → ea <u>x</u>
		'x\j' → xj

~~<<S1S2S3>~~

Goal: s1 s2; s3 s4

<<S1S2S3S4| ; <<

<<S|S2S3S4' ; <<

~~; * ; <<S1S2* ; S3S4 <<~~

<<S1S2; ; S3S4 <<

NEW IDEA!!!

Pre-defined Constants!

character command: ;

Usage:

; name

a-z → as is

A-Z → as is

0 → 100

1 → 10

2 → 256

3 → 32

4 → 64

5 → 25

6 → 1000

7 → 500

8 → 128

9 → 10,000

10

all other chars
→ as is

well, maybe
not...

hmm...

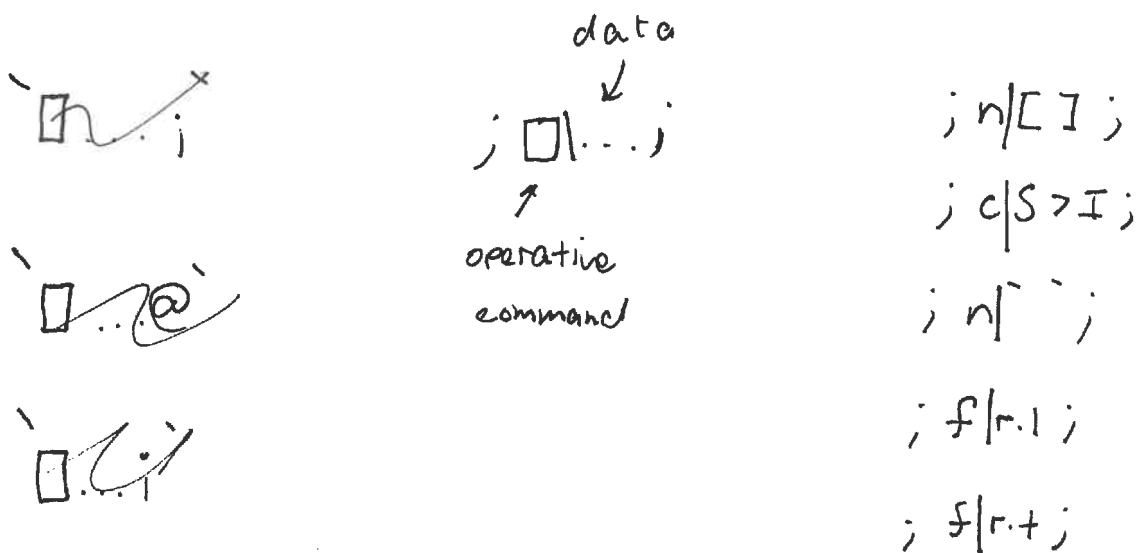
Object Strings

The ^{in the} undervalued backbone of key+

A brief overview:

- * contained within ~~standard~~ ^{is} strings (-)
- * To be implemented in the interpreter
- * Allows for non-key constructs to easily be made
- * NOT preprocessed in anyway
- * Allows for BFL to be 100% key+
- * Needs own library/file

Design



To be ambitious

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
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
13 Append ^{reversed} code → code to result
14 Set code to an empty string

15 Otherwise, If char is " \" then
16 Set escaped to True
17 Continue

18 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

23 Append char to code

Form

$j[\boxed{c}] \dots 1 ;$

Where:

\boxed{c} is the command $\dots 1$ is the body



$n \rightarrow$ new object

- stacks $in[\boxed{c}];$
- dicts $in[\{ \};$
- strings $in[" "];$
- files $in[file];$

$c \rightarrow$ cast (next page)

$f \rightarrow$ file

- open $is/o;$ is/o \rightarrow text
- read $is/r.l;$ $is/r.t;$
- write/appended $is/w;$ $is/a;$
- close $is/lb;$

$s \rightarrow$ system

- etc...
- operating system $is/platform;$
- version $is/version;$

des extract

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

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

"(0-99|ip|Gip *ip)" → ~~if~~

[{name: for, data: {
count: [{name: cmd, data: "0"},
{name: cmd, data: "9"},
{name: cmd, data: "i"},
{name: cmd, data: "99"}]}

structure

EXTRACTOR

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

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

Assumes 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

$$\ll s_1 s_2 s_3 s_4 | , ! , ! \ll$$

$$s_1, s_2! s_3, s_4!$$

$$\ll s_1 s_2 s_3 \dots s_N | c_1 c_2 c_3 \dots c_N \ll$$

$$s_1 c_1 \square s_2 c_2 \square s_3 c_3 \square \dots s_N c_N$$

$$\ll s_1 s_2 s_3 \dots s_N | c_1 c_2 c_3 \dots c_{N-1} \ll$$

$$s_1 c_1 s_2 c_2 s_3 c_3 \dots s_{N-1} c_{N-1} s_N$$

$$\ll s_1 s_2 s_3 \dots s_N | c_1 c_2 c_3 \ll$$

$$s_1 c_1 s_2 c_2 s_3 c_3 \dots s_N$$

$$\ll s_1 s_2 s_3 \dots s_N | c_1 c_2 c_3 \dots c_N \ll$$

$$s_1 c_N s_2 c_{N-1} s_3 c_{N-2} \dots s_{N-2} c_3 s_{N-1} c_2 s_N c_1$$

29

' $\$x$ S1; ~~$\$s$~~ S2; S3; S4;'

1 2 3 4 5 6

' $\$x$ S1; $\$s$ S2; S3; S4;'
 → 22
 (21 auto comp)

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

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

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

LL ' $\$x$ S1 $\$s$ S2 S3 S4 LL

LL ' $\$x$ S1 $\$s$ $\$o$ S2 S3 S4 LL → 15
 (14 auto comp)

SCC only

□ S1S2S3 □

□ is the delimiter.

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

If the chars contain " | ; ' < > , * ~ " "

→ Error.

Otherwise, Split into pairs & do a
Simple loop

N - strings

objects

Stacks $\rightarrow ; n | [] ; \rightarrow$ pushes an empty stack

Strings $\rightarrow ; n | " " ; \rightarrow$ " " string

Files $\rightarrow ; n | file ; \rightarrow$ creates a new file

ports $\rightarrow ; n | p(ip) ; \rightarrow$ creates a new internet conn

~~ictionaries~~

Math obj $\rightarrow ; n | m[Property] ; \rightarrow$ creates a new
math object

Stack objs

$; n | [] ; \rightarrow$ empty

$; n | [0, 1, 2] ; \rightarrow [3, 2, 3]$

$; n | ["hello"] ; \rightarrow ["hello"]$

N2.

The Uncompressor

* Aster

Preprocessor:

S1 = Hello

S2 = World

S3 = nice

Before

'Hello'

'S1;'

'S3;#S'

Q S1; S2; Q

Q Hello Q

Q S3; #S Q

' S1#S2 '

' S1 S2 ''

// S1S2|,!,

,, S3|#S,,

. S1 S2|, ! <<

S3 S2|#S <<

S1S2|, ! <<

Add
spaces

(OR)

Simply
change to
use
backticks

Extra

processing
needed

Aster

'Hello'

'Hello'

'nice#s' (no spaces)

'Hello world'

'Hello'

'Nice #s'

'Hello World'

'Helloworld'

'Hello,world!'

'-S3 'nice#s'

'Hello, World!'

'nice#s World'

'Hello! World!'

abcdefgh 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 [] joiners LL

The Preprocessor → Done!

* Implements the SSL (<char>)

Draft 3

```
def preprocess(source):
```

```
    final = ""
```

```
    in_string = False
```

```
    escaped = False
```

```
    for char in source:
```

```
        if char in "'\"":
```

```
            if escaped:
```

```
                escaped = False
```

```
            else:
```

```
                in_string = toggle(in_string) # or Not in string
```

```
        elif
```

```
    END
```

"Failure because string type not stored"

else:

if char == "&":

if escaped:

escaped = False

elif string_type != "u":

SSL_needed = True

Continue

else:

SSL_needed = False

final += char

⑦ if SSL_needed:

final += SSL[char]

SSL_needed = False

def preprocess(source):

final = ""

string_type = ""

escaped = False

SSL-needed = False

for char in source:

if char in STRING_CHARS:

if string_type == char:

if escaped:

escaped = False

else:

string_type = None

if char == "\":

if escaped:

escaped = False

else:

string_type = char

else:

if char == "\":

if escaped: Escaped = False

else: escaped = True

Standard/spaces - spaceful

word @ pos
word @ pos

Spore
after
of
not - valid
new

$\langle \psi | \hat{A} | \psi \rangle = \langle \hat{A} | \psi \rangle$

~~{char | (src, ...)}~~ ~~$\left(+ \left(\frac{1}{n} \cdot \dots \right) / p \right)$~~

Option w/ 98% space

$$1. \quad \left\{ \left(\left| \langle \text{name} \rangle \right| \left| \langle \text{src} \rangle \right| \left| \langle \text{char} \rangle \right| \right) \right\}$$
$$\left\{ \left(\langle \text{char} \rangle \mid \langle \text{name-char} \rangle \right) \mid \left(\langle \text{char} \rangle \mid \langle \text{name} \rangle \right) \right\}$$

2nd look at Special compressed Strings

Delimiter: \llcorner

~~Spaceless~~ by
Spaceful by
absent

es 'rs ← [77] ' | esrs [77]

as is \leftarrow $\boxed{77}$ | as is $\boxed{77}$

applied
args are processed
means that
left to right

means that eggs are applied right to left

Form:

$$\begin{bmatrix} \Delta \dots \Delta \\ \vdots \\ 1 \dots 1 \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ 1 \dots 1 \end{bmatrix} \text{ or } \begin{bmatrix} \vdots \\ \vdots \\ 1 \dots 1 \end{bmatrix}$$

11
 1000 91011, 2109 8 ← 1'1011011011
 10101011011

... The SC Part

∴ $\frac{1}{\sqrt{2}}$ is a

2515 1/2

50% / 15 25 35 45 55 65 75 85 95

<< ... | ... >>
Body
Format

Body
↳ SCCs only. No vars or other chars

✱

Format

↳ List of expressions
 ↳ and are treated as standard chars
 ↳ only standard strings remain
 ↳ a literal
 ↳ Escaping allowed

Q: s1s2 | 'h' 'l' → s1 h' s2

}

data : . . .

name : . . .

}

* list of structures

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

NO SPECIAL
STRUCTURE

* Add to list
* continue

* Extract
lhs & rhs of
" | "
* Add to
list
* Extract loop
until
unescaped] "
* Add to list

* Read until
matching ") " / " } "
* split @ " | "
* Make dict as
needed
* Add to list

FOR
WHILE
("c") / ("{")

I ← ("c")

Function Extract
→ source

Set result to []

set temp struct
to { }

```

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:
        Result+.append({ name: "cmd", data: char })
    Else, if char matches opening bracket at pos -1 in struct:
      structures.pop()
    If char == "\":
      If 3 rks =
  
```

~~SI = "Hello world"~~

SI = Hello S2 = word Goal: "Hello, world!"

Uncompressed:

"Hello, world!" → 15 (34 a)

Compressed, spaces:

"SI, sa!" → 11 (10 a)

Compressed, special:

Impossible (no punctuation)

Compressed, special, structure:

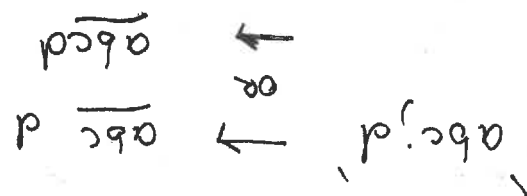
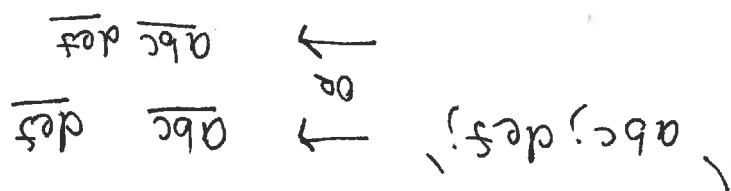
"Sisa!, i" 9 (8 a)

Compressed, special, constants

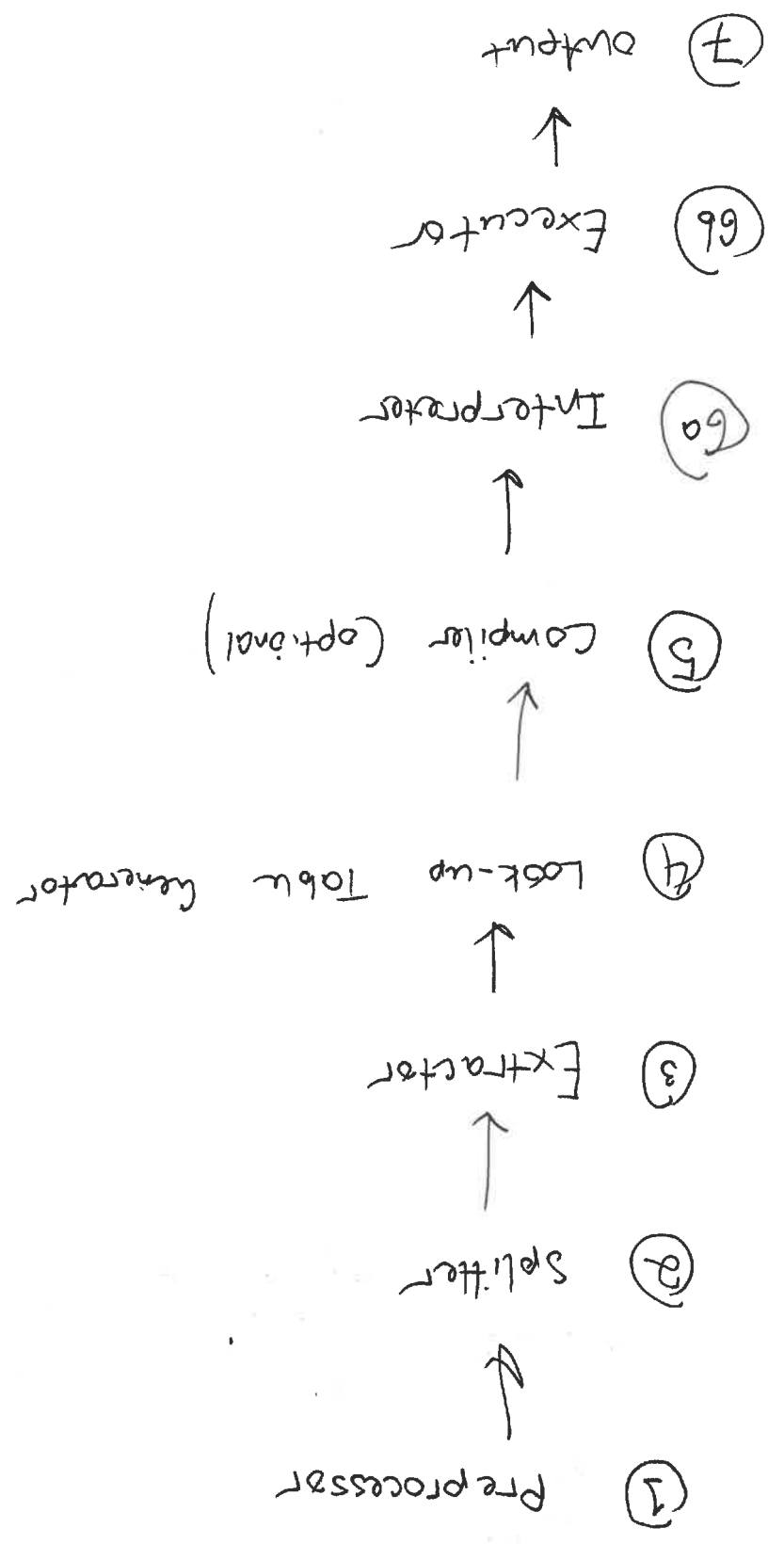
~~SI =~~

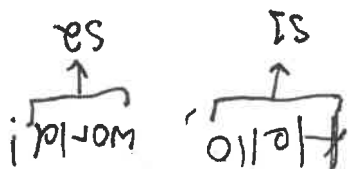
~~"SI, sa!"~~ 10 (9 a)

Spans & SCCs



7 Steps





's1, sa, i' → 11

q s1, sa, q → 8 (but no punctuation)

~~77 s1sa 77~~

~~[>7 s1sa |, i >7] → 10~~

[<7 s1sa |, i <7] → 8 chars

Strings

Spaceful

q contents q

like normal spaceless strings, but

spaces are placed between

secs :

Hello □ World!

`s1; □ sa; !`

q s1; sa; i q

↓

hmm

1 byte
shorter



Spaceful Special Compressed Strings

Delimited by <<

usage:

<< sccs / separators >>

any
Series of
sccs
(no !)

↑
each separator is
either a single
char, or a variable
and is placed after
the corresponding
scc.

Examples

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 / spaces

→ $\overline{SI}, \overline{sa}; i$ SI, sa; i

→ standard/spaces

→ $\overline{SI}, \overline{sa}; i$ SI, sa; i

→ SCC only (spaces)

→ $\overline{SI}, \overline{sa}; i$ SI sa

→ SCC only / spaces

→ $\overline{SI}, \overline{sa}; i$ SI sa