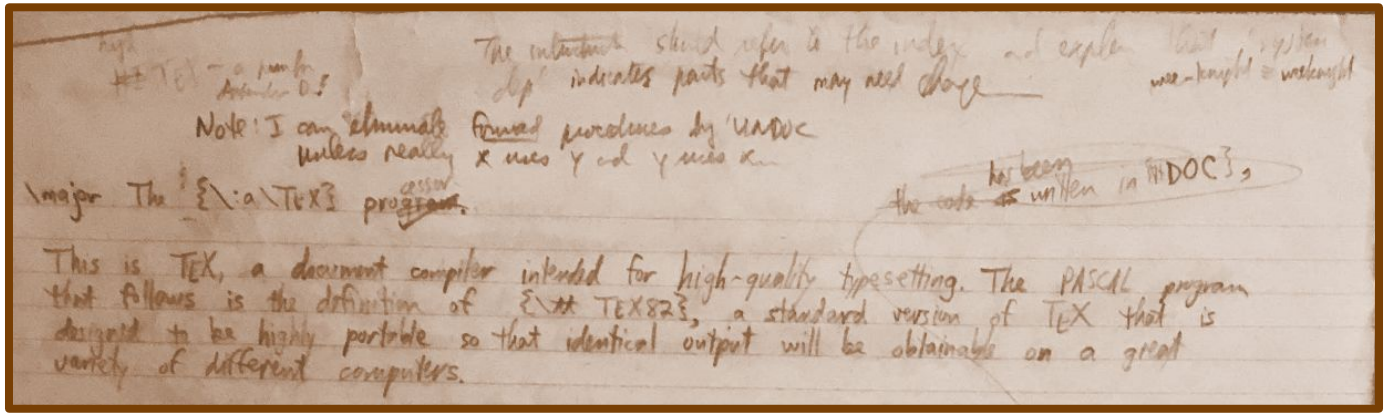
Holon Programming Regained

precursors to literate programming

Mitch Gerrard



This is TEX, a document compiler intended for high-quality typesetting. The PASCAL program that follows is the definition of {\tt TEX82}, a standard version of TEX that is designed to be highly portable so that identical output will be obtainable on a great variety of different computers.

(from 1982 draft)

tex

tex.

tex.web

tex.web

web =

tex.web

web = pascal

tex.web

web = pascal +

tex.web

web = pascal + tex?

tex.web

web = pascal + tex?

web

tex.web

web = pascal + tex?

web.

tex.web

web = pascal + tex?

web.web

P. A. de Marneffe, *Holon Programming*. Univ. de Liège, Service D’Informatique (December, 1973).


"That's all Folks!"

April 1, 1974

Prof. Pierre-Arnoul de Marneffe
Universite de Liege
Service D'Informatique
Avenue des Tilleuls 59
B-4000 Liege, Belgium

Dear Prof. de Marneffe:

Thank you very much for sending me your survey of Holon Programming. I especially enjoyed your references to the non-computer literature (Koestler, Bernard-Shaw, Shanley, Mount Vernon, etc.) since computer scientists need to avoid insularity.

December, 1973

HOLON PROGRAMMING:

A

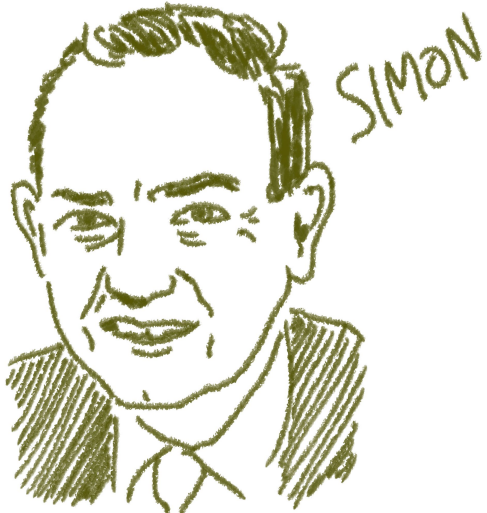
SURVEY

By Pierre-Arnoul de Marneffe

Universite de Liege
Service D'Informatique
Avenue Des Tilleuls, 59
B-4000 LIEGE
Belgium.

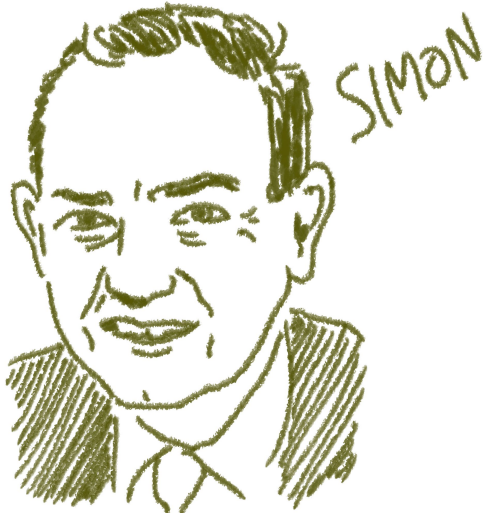

KOESTLER


KOESTLER


SIMON

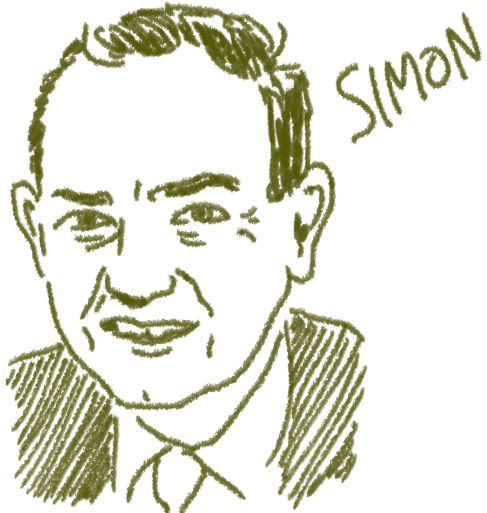

KOESTLER

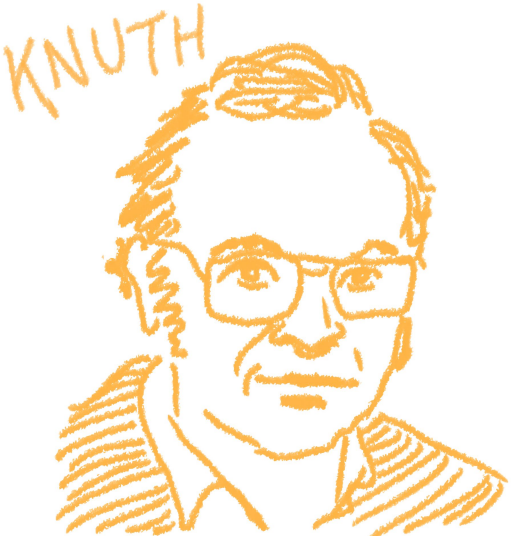

DIJKSTRA

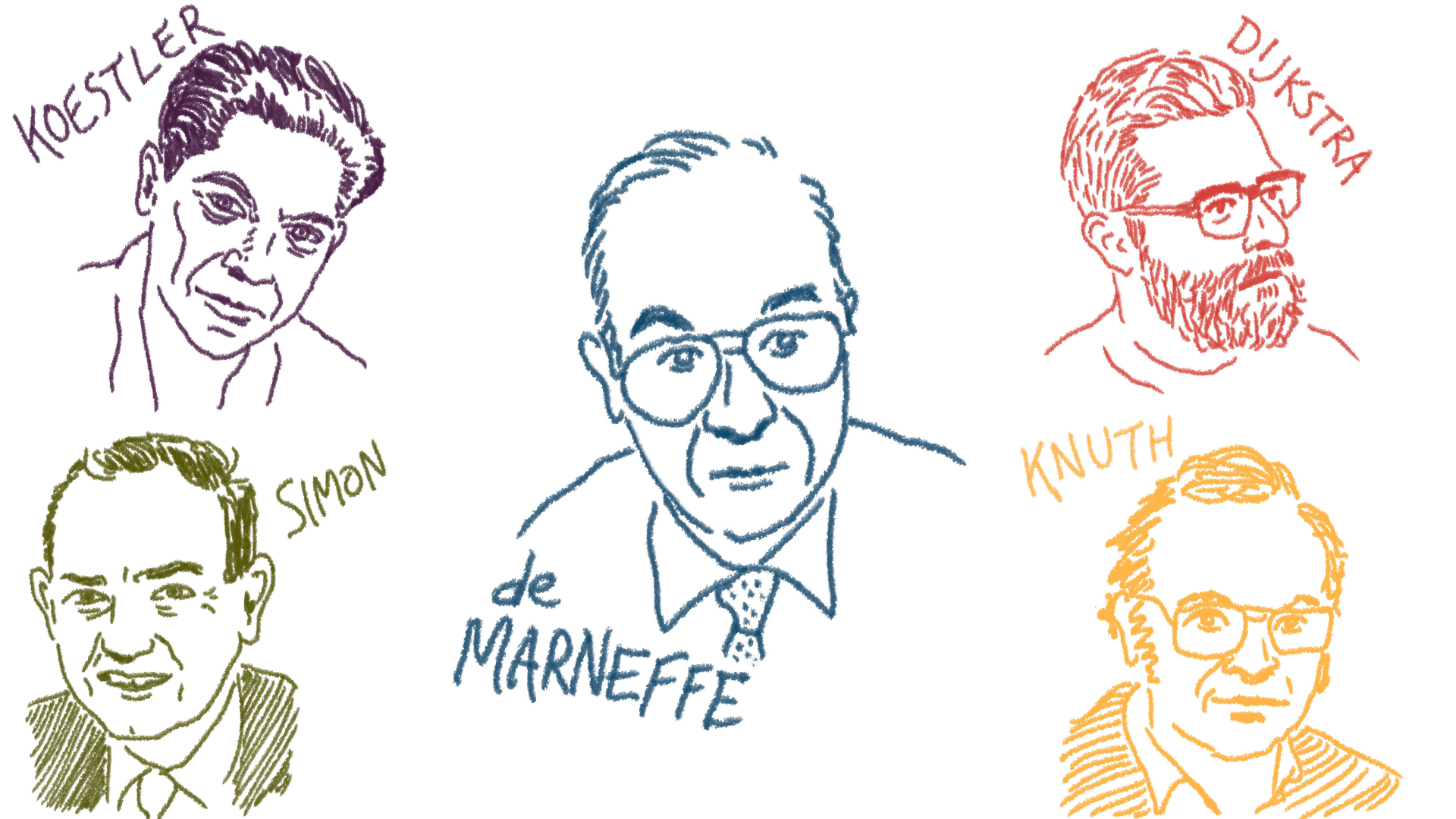

SIMON


KOESTLER

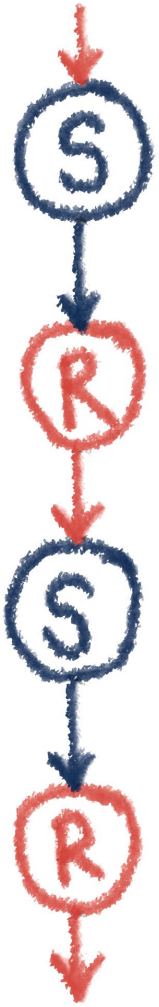

DIJKSTRA

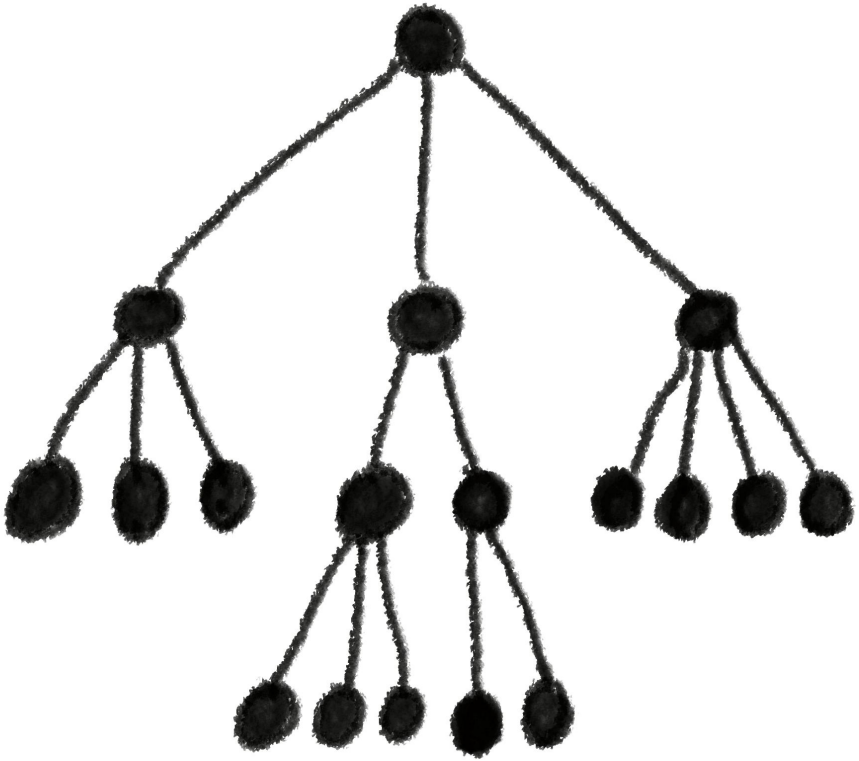

SIMON

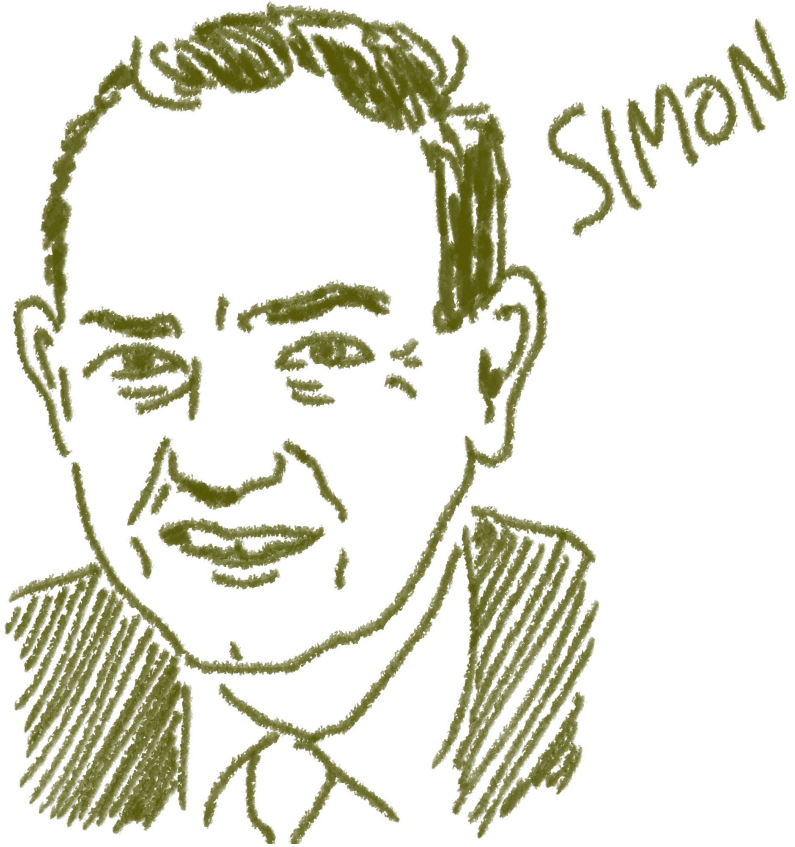

KNUTH


KOESTLER
DIJKSTRA
SIMON
de
MARNEFFE
KNUTH

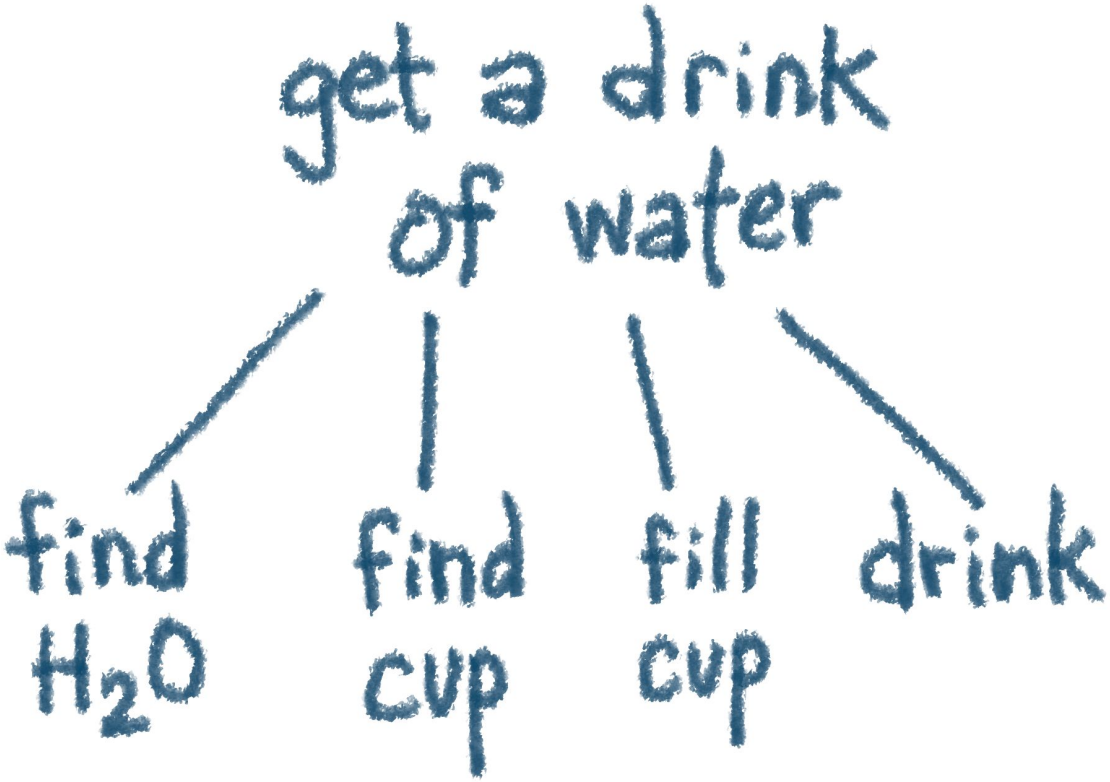

KOESTLER

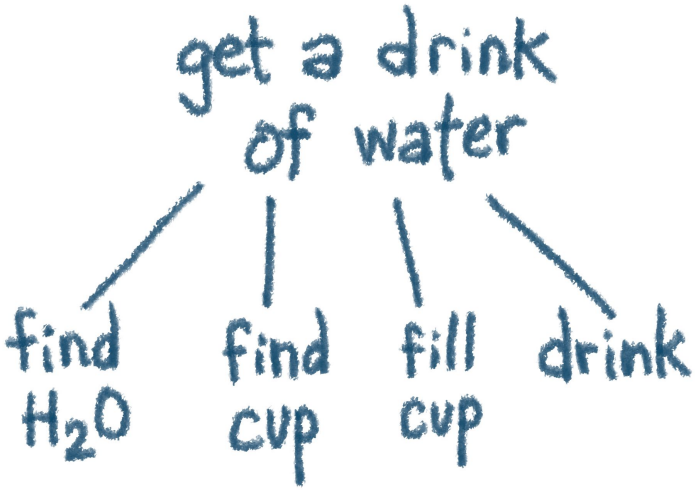

S
R
S
R




SIMON

get a drink
of water

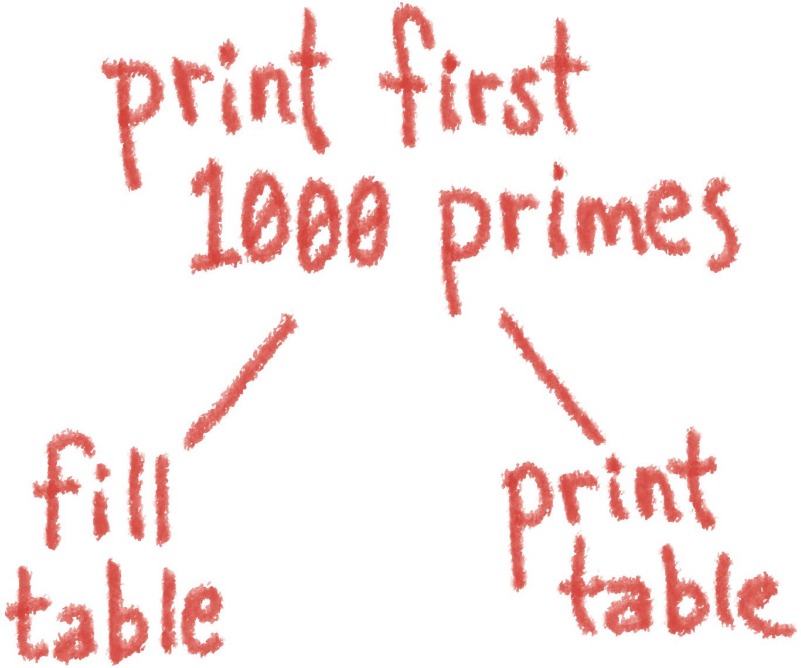

get a drink of water
find H_2O
find cup
fill cup
drink

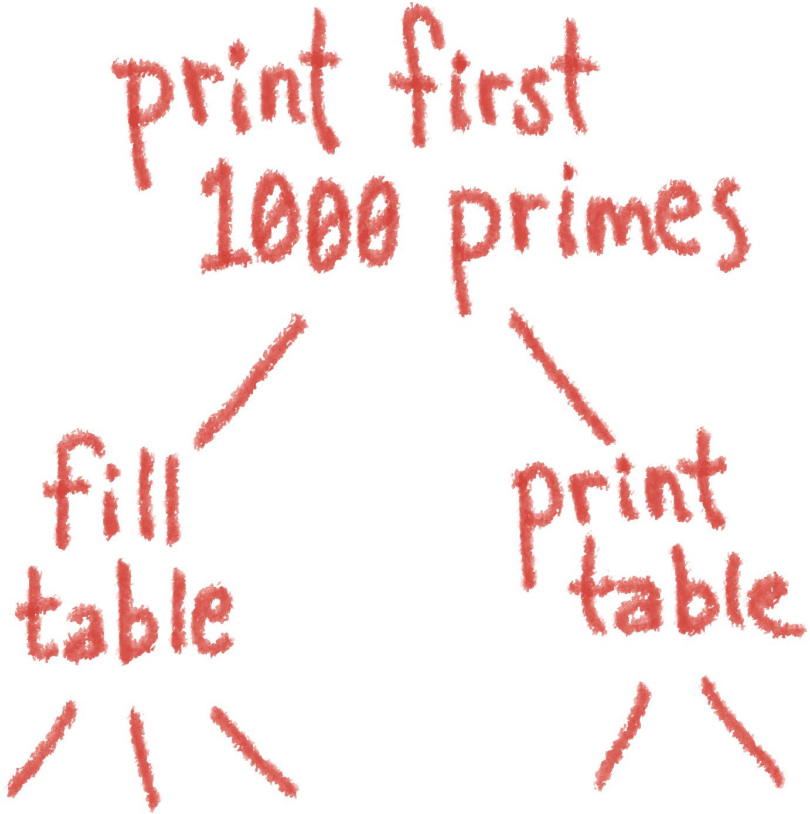


“It seems preferable to coin a new term to designate these nodes on the hierarchic tree which behave partly as wholes or wholly as parts, according to the way you look at them. The term I would propose is ‘holon’, from the greek holos (meaning whole), with the suffix ‘on’, which, as in proton or neutron, suggests a particle or a part.”


DIJKSTRA

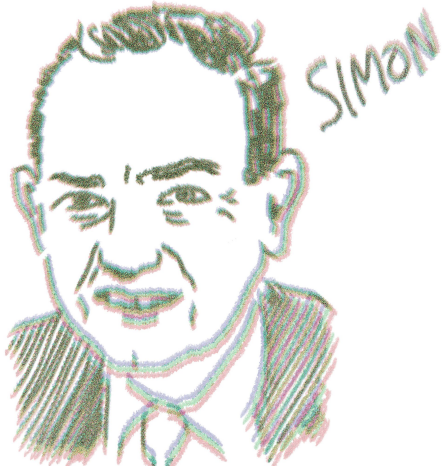
print first
1000 primes


print first
1000 primes
fill
table
print
table

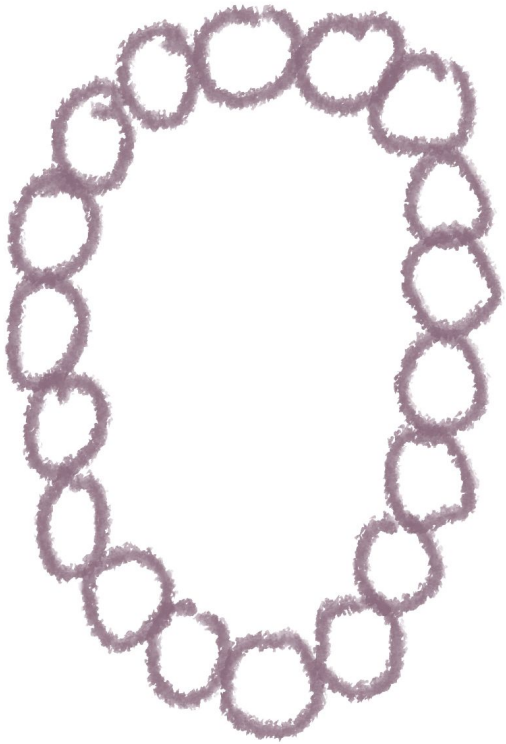

print first
1000 primes
fill
table
print
table

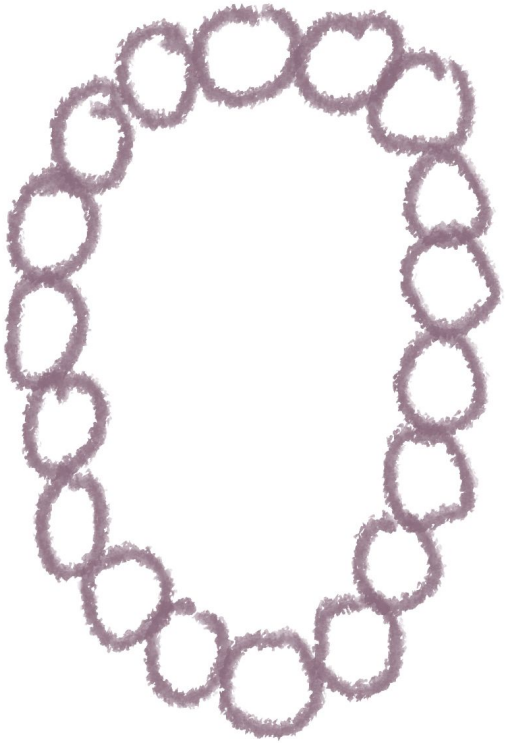

DIJKSTRA

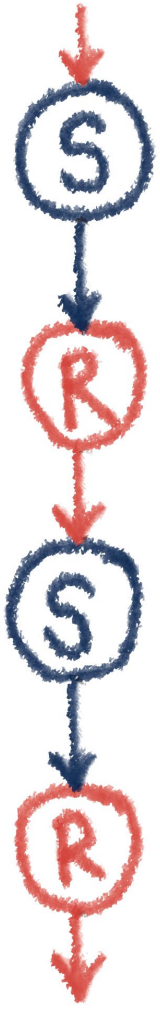

DIJKSTRA


SIMON


DIJKSTRA






S
R
S
R


de
MARNEFFE

```
odd inversion program
   begin find first word starting character;
      repeat find a word and print correctly;
      until end of useful file
   end

find first word etc
   begin read first symbol;
      while last read symbol is a space;
      do read next symbol
   end
 ⋮

read first symbol
   begin declare lrs: character at
        odd inversion program level;
      # lrs ← RNC ##;
   end
 ⋮
```

```
odd inversion program
   begin find first word starting character;
      repeat find a word and print correctly;
      until end of useful file
   end

find first word etc
   begin read first symbol;
      while last read symbol is a space;
      do read next symbol
   end

⋮

read first symbol
   begin declare lrs: character at
         odd inversion program level;
      # lrs ← RNC ##;
   end

⋮
```

```
begin
 declare even: boolean, lrs: character, EOF: boolean;
 even ← true; EOF ← false;
 lrs ← RNC; while lrs ≡ '␣' do lrs ← RNC;
 repeat
  begin
   declare buffer: character(20), pointer: integer;
   pointer ← 0;
   repeat
    if pointer ≥ 20 then (datatest exit);
    pointer ← pointer + 1;
    buffer(pointer) ← lrs;
    lrs ← RNC;
   until (lrs ≡ '␣') or (lrs ≡ '.');
   if even ≡ true
    then
     begin declare point2: integer; point2 ← 1;
     repeat PNC(buffer(point2)); point2 ← point2 + 1;
     until point2 > pointer;
     end
    else
     begin while pointer ≠ 0 do
      begin PNC(buffer(pointer)); pointer ← pointer − 1; end
     end
   even ← not even;
   if lrs ≡ '␣'
    then begin repeat lrs ← RNC until lrs ≠ '␣'; end
   if lrs ≡ '.'
    then begin PNC('.'); EOF ← true; end
    else PNC('␣');
  end
 until EOF ≡ true;
end
```

odd inversion program
  begin find first word starting character;
    repeat find a word and print correctly;
    until end of useful file
  end

find first word **etc**
  begin read first symbol;
    while last read symbol is a space;
    do read next symbol
  end

⋮

read first symbol
  begin declare lrs: **character at**
      odd inversion program **level**;
    # lrs ← RNC ##;
  end

⋮

1. This program is one possible solution to the problem posed in section 16 of Dijkstra's *Notes*.

program *odd_inversion*;
  var ⟨ Global variables 4 ⟩
  begin ⟨ Find first word starting character 2 ⟩;
    repeat ⟨ Find a word and print correctly 3 ⟩;
    until ⟨ End of useful file 13 ⟩
  end.

2. ⟨ Find first word starting character 2 ⟩ ≡
  begin ⟨ Read first symbol 11 ⟩;
    while ⟨ Last read symbol is a space 12 ⟩;
    do ⟨ Read next symbol 6 ⟩
  end

This code is used in section 1.

⋮

10. ⟨ Global variables 4 ⟩ +≡
lrs: *char*; { last-read symbol }

This code is used in section 1.

11. ⟨ Read first symbol 11 ⟩ ≡
  begin *lrs* ← *RNC*; { read next character }
  end

This code is used in section 2.

⋮

RNC: **procedure**, §15

```
find first word etc
  begin read first symbol;
    while last read symbol is a space;
    do read next symbol
  end
```

2. ⟨ Find first word starting character 2 ⟩ ≡

```
  begin ⟨ Read first symbol 11 ⟩;
    while ⟨ Last read symbol is a space 12 ⟩;
    do ⟨ Read next symbol 6 ⟩
  end
```

This code is used in section 1.

odd inversion program
  begin find first word starting character;
    repeat find a word and print correctly;
    until end of useful file
  end

find first word **etc**
  begin read first symbol;
    while last read symbol is a space;
    do read next symbol
  end

⋮

read first symbol
  begin declare lrs: **character at**
      odd inversion program **level**;
    # lrs ← RNC ##;
  end

⋮

1. This program is one possible solution to the problem posed in section 16 of Dijkstra's *Notes*.

program *odd_inversion*;
  var ⟨ Global variables 4 ⟩
  begin ⟨ Find first word starting character 2 ⟩;
    repeat ⟨ Find a word and print correctly 3 ⟩;
    until ⟨ End of useful file 13 ⟩
  end.

2. ⟨ Find first word starting character 2 ⟩ ≡
  begin ⟨ Read first symbol 11 ⟩;
    while ⟨ Last read symbol is a space 12 ⟩;
    do ⟨ Read next symbol 6 ⟩
  end

This code is used in section 1.

⋮

10. ⟨ Global variables 4 ⟩ +≡
lrs: *char*; { last-read symbol }

This code is used in section 1.

11. ⟨ Read first symbol 11 ⟩ ≡
  begin *lrs* ← *RNC*; { read next character }
  end

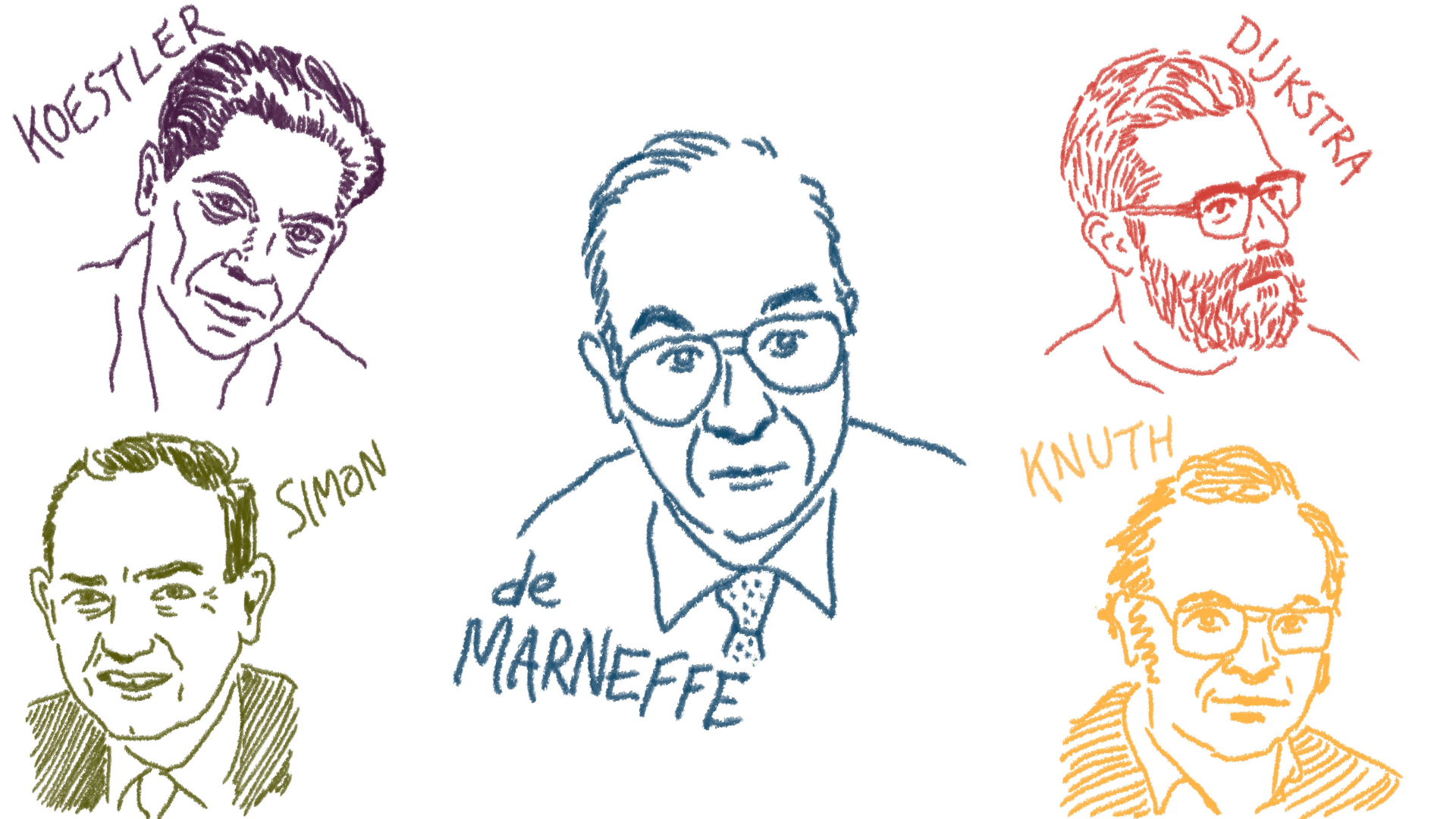
This code is used in section 2.

⋮

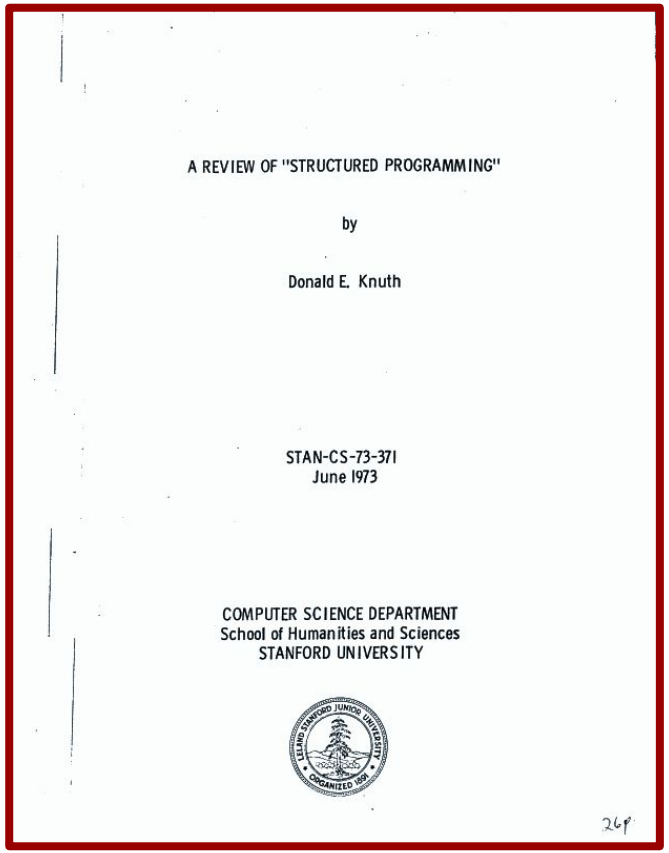
RNC: **procedure**, §15


de
MARNEFFE


KNUTH


KOESTLER
DIJKSTRA
SIMON
de
MARNEFFE
KNUTH

D. E. Knuth, Computer-drawn flowcharts. *Communications of the ACM*, *6*(9), 555–563 (1963).



A REVIEW OF "STRUCTURED PROGRAMMING"

by

Donald E. Knuth

STAN-CS-73-371
June 1973

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

269

Donald E. Knuth.
A review of "Structured Programming".
Tech. Report. Stanford University (1973).

Peter Naur,
Myrtle Kellington,
Derek Oppen,
Ole-Johan Dahl,
Jim Dunlap,
Edwin Towster,
Robert Snowdon,
...

Preliterate Programming

(an anthology)

forthcoming!

Holon Programming: A Survey

github.com/holon-scribe/holon-programming