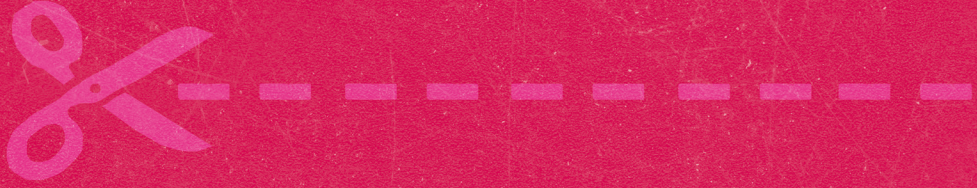
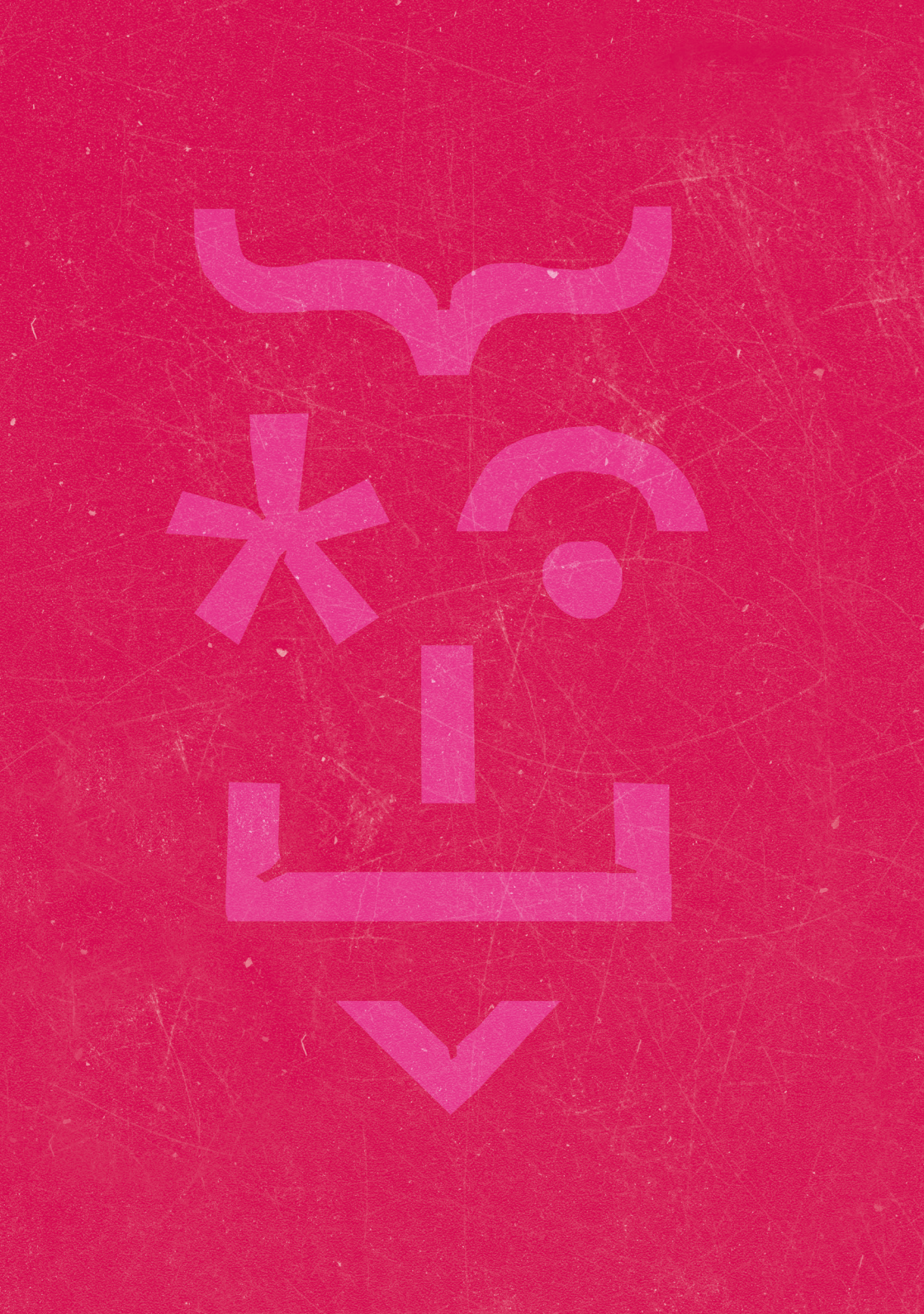
BREAKING AND MAKING CODE POEMS

ETHOS LAB
IT UNIVERSITY
OF COPENHAGEN

EDITING BY
MARISA LEAVITT
COHN, RACHEL
DOUGLAS-JONES
AND MERETHE
RIGGELSEN
GJØRDING





<DOCTYPE BOOK>
<BOOK>
<HEAD>
<TITLE>
BREAKING AND MAKING CODE POEMS — CODE POETRY
</HEAD>

CONTENTS
CONTENTS
CONTENTS

CONTENTS
CONTENTS
CONTENTS

{PREFACE}

Code poetry is a form of experimental writing and coding. Both writing and coding pay attention to the poetics of syntax, grammar, and punctuations, considering natural and computer languages as playful and aesthetic materials. This mix of languages is called "codework", a genre coined by poet-theorist Alan Sondeim which he describes as "the computer stirring into the text, and the text stirring the computer" (2001). Though textuality is an important element in language that contains meaning yet it requires interpretation, I want to point to the aspect of performativity in computer code and natural languages. One might draw upon John Langshaw Austin's Speech-Act theory in thinking about the things that you can do with performative speech, in which the utterance of words carries inherent actions beyond simply making statements. However, code execution is different from the speech-act, because computer code, especially high-level programming language, is written for both humans and machines. When a piece of source code is executed, the computer is doing something immediately, for example, the translation of high-level code to binary code, or to display certain visual and textual materials on a web page. In this way, there are different kinds of readers beyond the human.

QUEER CODE | Figure 1: A code snippet of Vocable Code (2017) by Winnie Soon

```
33  function SpeakingCode(iam, makingStatements) {
34    let getVoice = "voices/" + iam + makingStatements + ".wav";
35    speak = loadSound(getVoice, speakingNow);
36  }
```

By Winnie Soon

Code poetry becomes highly interesting and challenging because the piece of code deals with the poetics of two different formalities and systems. In this zine, many entries use specific computational syntax, such as conditional if-else statements, boolean values of true and false, function blocks, while/for loops among others. But we also see some conceptual computational terms are in use, for example, “Include”, “Exit”, “Break”, “String”, “Retry”, “Null”, etc. In other words, the aesthetics of code poetry lies in the material and linguistic tensions of writing and reading code (Soon 2018). It is important to note that such an experimental form of writing is more than making things functionally work. When source code and critical writing operate together, the codework embodies “queer code,” refusing the normativity of coding practice and queering code beyond the solution-orientation as a dominant frame. In the words of Karen Barad, to queer something is to engage with “political imaginaries,” exploring “new forms of becoming, new possibilities of kinship, alliance, and change” (2015, 410). As such, queer code focuses on the performativity of code, subjectivity and language to create new forms of embodiment through reading, writing, coding and speaking practices.

REFERENCES:

Austin, John Langshaw. *How to Do Things with Words*. Oxford: Clarendon Press, 1975.
Barad, Karen. “Transmaterialities: Trans*/matter/realities and queer political imaginings.” *GLQ: A journal of lesbian and gay studies* 21.2–3 (2015): 387–422.
Sondheim, Alan. “Introduction: codework.” *American Book Review* 22, no. 6 (2001): 1–+.
Soon, Winnie. “Vocable Code.” *MAI Feminism & Visual Culture* Autumn Issue (2) (2018).

{INTRODUCTION}

By Merethe Riggelsen Gjørding and Edith Terte

Code has become a ubiquitous and intertwined part of our everyday lives. Many of us, however, don't encounter code as language and commands in a terminal. Not knowing how to code or having limited abilities in understanding code language, can hinder our ability to relate to code. Yet, we live in and experience code in action.

Code language as part of a poem works by other logics. It does not need to comply to any syntax, and you do not need coding skills in Java, Python or R to appreciate code poems or work with codes in a poem creation. This opens code to new interpretations and forms of participation.

A code poem is a play of integrating languages. It engages us with code in a poetic manner: Noticing code's qualities and how these affect us as a composer or a reader of the poem.

This praxis of reading and doing code poems can be part of leaning into the world of code and getting more familiar with the various languages and ways of code, which are affecting us in so many ways.

In ETHOS Lab we continuously work with matters of technology; how they work in practice and how we relate to them, and these thoughts and desires led us to decide to host a workshop on code poetry in the Lab on Ada Lovelace Day

Weaving languages together

Augusta Ada King (1815–1852), Countess of Lovelace, known as Ada Lovelace, has during the last two decades been recognized and honored for her important contributions and philological reflections on computing.

In 1843, she was the first person to publish what we now would call a computer programme. The programme was for mathematician Charles Babbage's Analytical Engine – the first mechanical computer – which only partly was built and never got fully developed. Lovelace wrote a set of instructions of how to calculate the numbers of Bernoulli[1] on the Analytical Engine, encoding punch cards. While Lovelace's contributions to the Analytical Engine have been recog-

[1] A sequence of rational numbers which occur frequently in number theory. The Bernoulli numbers were discovered independently by the Swiss mathematician Jacob Bernoulli, and the Japanese mathematician Seki Takakazu. Seki's discovery was posthumously published in 1712 in his work Katsuyō Sanpō; Bernoulli's, also posthumously, in his Ars Conjectandi of 1713. Perhaps, not surprisingly, the discovery took the name after Jacob Bernoulli.

nized to some degree, often left out are her contributions to sketch out and envision the creative potential of computers.

Lovelace imagined the Analytical Engine to be much more than just a smart calculator. She saw that a computer could be making music or graphics, given the right inputs. Lovelace realized that codes can create art. They can be artful.

In her now famous article *Sketch of the analytical engine invented by Charles Babbage* from 1842, she noted: *"(...) the Analytical Engine weaves algebraical patterns just as the Jacquard-loom weaves flowers and leaves."*[2]

Correspondingly, in our code poetry workshop we were weaving machine and human readable language together. In doing so, we came to recognize how machine-readable language is always already made up by human-created signs and letters, but nonetheless exploring poetically brought new patterns and meanings to appear. Code poems when running in a terminal are fast composers of a fabric and form.

Engaging with code as a poetic language

Ada Lovelace Day is a day for celebrating women and their achievements in STEM (Science, Technology, Engineering & Mathematics). It is widely known that these fields face a great gender inequality both historically and currently, which takes many forms; cultural narrations, payment gap, acknowledgement gabs, hiring gaps etc.

In ETHOS, we wanted to further unfold the diversity issue, remembering that the category of women intersects with a lot of other social categories and possible marginalizations, and expanding the gendered perspective to include non-binary and trans people.

We actively worked with that in our way of framing our event on code poetry and by including quotes from a diverse section of feminist scholars and non-cismen working within STEM. The quotes were used as materials in the poem creation. Participants could find inspiration in the quotes or use full sentences.

[2] Lovelace in: Menabrea, L. F., & Lovelace, A. (1842). Sketch of the analytical engine invented by Charles Babbage.

We related to codes not in terms of their functionality as a programming language but as a poetic language. We encouraged people to create a code poem from what spoke to them in that current moment, in relationship to their abilities, voice, and desire.

It feels great to think about that from a poem creation session in less than an hour, beautiful and thought-provoking pieces can be generated.

We ate a delicious cake with a laser printed code poem in marzipan; looked at each other's creations and shared experiences of engaging with codes in this way.

When we later on mailed with people, asking if they would like to feature in this very collection, we also asked them to send us a reflection note. Some of them are short, and others give more context to the composer's thoughts around their poem.

This collection is also an extension of the invitation to make poetry with us, to you.

If and when you feel inspired, put the book away and get going with your own poem. In the end of the book, you will find a page ready for your poem to be featured.

We hope you will enjoy reading and relating to codes and code poetry.

IT UNIVERSITY OF COPENHAGEN


ETHOSLAB

When she does find

< matrix of power > already built, on the * faces of men *

in { technological developments
governments.
social movements.
universities }

she observes, she feels, she records

{ role THEY play IN domination. }

she does act,

she seeks from (relations * collective values)

within her shared burden and responsibility.

she relishes with < \ love. >

{STAGES OF FEMINIST ACTIVISM}

By Lara Tatjana Reime

The poem maps the stages of feminist activist engagement. She, the curious and critical feminist explores the matrix of power in different institutions and socio-technical systems. Through collective action, nurtured from relations and shared experiences, she acts against those mechanisms of domination. With love and care, for oneself and others as well as non-human companions, the matrix of power can potentially be broken (hence the <matrix of power> </love>).

{PUBLIC CLASS SURVEILLANCE}

By Edith Terte

Using my knowledge of programming in Java, I decided to make a code poem focused on surveillance. The Surveillance class represents in general terms the actions of a surveillance camera. It moves, it observes by passers etc. By instantiating a surveillance object, given it a specific person to act towards, the poem becomes personal - It is not just random people who are followed, it is you! Executing the code, you are presented with the quote “privacy is any rights you have to control your personal information and how it is used”. The quote is printed 1.500.000 times - one for each CCTV registered in a public space in Denmark back in 2019.

```
/**
 * @author Edith Terte, 2021
 */

public class Surveillance {
  boolean monitorBehavior = false;
  Person person;
  boolean withinReach = false;


  public Surveillance(Person whoEver) {
    person = whoEver;
  }

  public void registerPerson() {
    withinReach = true;

      if (withinReach) {
        monitorBehavior = true;
      }
  }

  public void getIdentity() {
    Person personRegistered = person;
  }

  public void followsYouEveryWhere() {
    String forever = "privacy is any rights you have to control your
personal information and how it is used";

    System.out.println(forever);
  }

  public void drawAttention() {
    boolean attentionCaught = true;
    boolean motion = false;

    while (!motion) {
      registerPerson();

      if (!motion) {
        registerPerson();

        motion = true;
      } else {

        break;
      }
    }

    attentionCaught = false;
    withinReach = false;
  }
```

```
  public static void main(String[] args) {
    int id = 0000000000;
    Person you = new Person(id);
    Surveillance surveillance = new Surveillance(you);

    surveillance.registerPerson();
    surveillance.getIdentity();
    surveillance.drawAttention();
    surveillance.followsYouEveryWhere();

    for (int CCTVinDK = 1499999; CCTVinDK > 0 ; CCTVinDK--) {
      surveillance.followsYouEveryWhere();
    }

  }
}
```

Feminism

{

String forever =

"

an ongoing discursive practice

a constructing

a term in process,

a becoming. ";

System.out.println(forever);
}

/**
 * @author
 */

Gender Trouble *

public class Surveillance

{GENDER TROUBLE X PUBLIC CLASS SURVEILLANCE}

By Anonymous

This poem is inspired by the use of repetition in Edith Terte Andersen's poem 'Public class surveillance'. This use made me think about how practicing feminism is also in itself a continuing process, one that involves doing things again and again, as well as reflecting about what has been done and what is to come. Judith Butler writes on the ongoing performativity of gender, and I hope putting Butler's words into this new context can inspire thought about practicing feminist intervention and thinking not as something contained to a single day such as Ada Lovelace Day, but rather something that continues and is ongoing.

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY

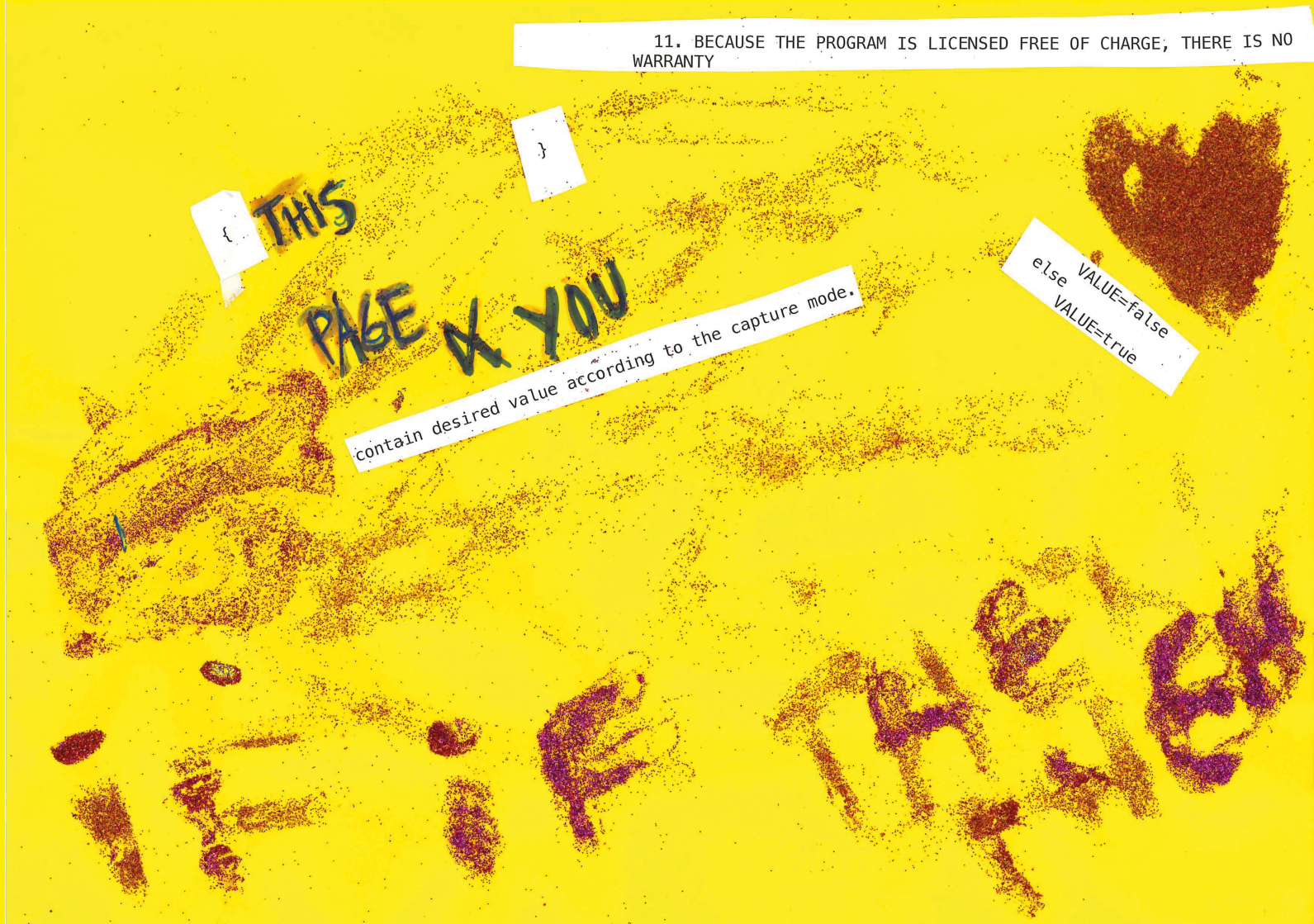
{

}

THIS PAGE IS FOR YOU

contain desired value according to the capture mode.

VALUE=false
else
VALUE=true



{IF, THEN!}

By Christopher Gad

Recursivity and loops are important in coding and in some strands of Anthropology. According to Marily Strathern Euro-Americans have many different values, but what they really value, are their values. This glimmery ‘poem’ has no particular value, but may be valued, or not, in any way you please, but likely according to capture mode and your appreciation of glimmery punk vibes.

```
if (!motion) {
  registerPerson();
```

```
}
```

And how is this socially organized?"

```
}
```

"Classifying, defining, sorting, ranking things by value,

```
  motion = true;
} else {
```

we all have a role to play in asking

{ASKING}

By Merethe Riggelsen Gjørding

My poem is mostly an empty space where surveillance occur.
The space is intervened by asking: how?
The surveillance is not stopped, but maybe it could?

{DROP TABLE;}

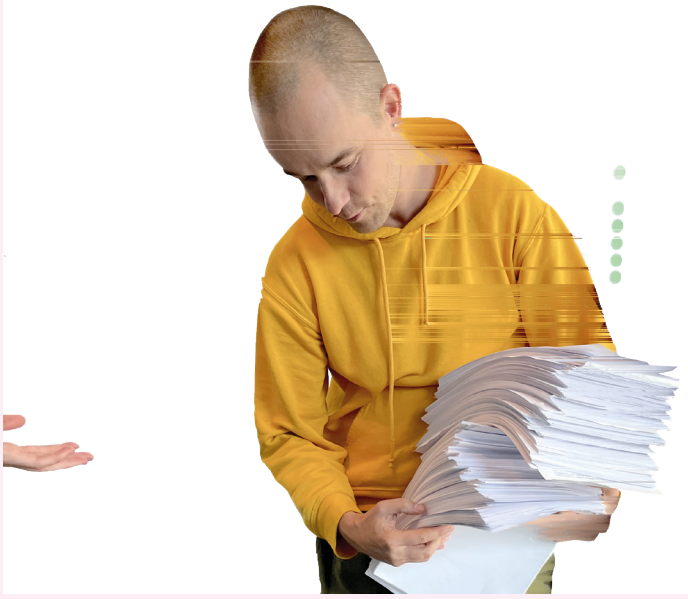
By Mace Ojala

I was first introduced to the thought of printing absurd amounts of computer source code in the late 1990s; I heard lore that GNU developers had printed the entire BSD operating system to study its code and re-implement the functionality from scratch, thus avoiding copyright and licenses. Or was it BSD developers studying AT&T Unix? The legend is lost in time. Anyhow, the idea of tens or hundreds of thousand of pages of code printout has captivated me since.

Our beloved research tool TCAT, or Twitter Capture and Analysis Toolset, is a server-side software which periodically queries Twitter for selected keywords, and stores them in a local database (Borra and Rieder 2014). We run it for research and teaching purposes. I am not sure what went into me on a Tuesday not long ago; I couldn't help myself. The day started off as any other day but by lunchtime it had turned out TCAT's source code of is modest 2100 A4 pages, with 12 point font. Warm, heavy, imposing pile of pages flowing mostly with PHP, SQL and JavaScript plus whatever one finds in a body of code. Printing itself was satisfying and I took sound recordings too.

It intuitively felt appropriate to bring this 10 kilogram object with me wherever I went on the university campus.

```
for ing in layering:
    lightness = mean(ing)
    what is not left of lightness:
        sort()
```



A colleague remarked that printing the source code was a kind of a *breaching experimen*t a research technique attributed to ethnomethodologist Harold Garfinkel (1967), which interrogate everyday cultural assumptions and norms about what should and should not be done with source code, printers and sheets of paper.

Three common reactions to walking around campus with an unwieldy 2100–page printout were following: “why?”, “isn’t that a huge carbon footprint?”, and “do you need hand?”. They suggest that printing code is an exceptional behaviour out of bounds, an unexpected exception to environmental values, or the pile will overflow onto the floor. The first is to be dismissed by throwing the exception back at the fool, the second verifying the assertion of the print’s environmental burden in comparison. The third, however, warrants a consideration.

```
<?php
try {
   self::carry( $all_the_printouts );
} catch ( OutOfBoundsException $e ) {
   throw $e;
} catch ( UnexpectedValueException $e ) {
    assert( co2( $all_the_printouts ) < co2( $a_lunch,
  $vegetarian=false ));
} catch ( OverflowException $e ) {
   assert( $e->goodwill, “Thanks” );
   return $this->careHandler;
}
?>
```

A helping hand reaches in just as prints are about to drop.

Code was on the table at the workshop. Having materials at the table helped drop the bar. Many of the printed source code pages were undone, cut, copied and pasted into poems you find now on these pages in your hands. Re-implementing; dropping bars.

TCAT collects data droppings of Twitter users in a database tables. At the end of each semester, with some *hiraeth*, we drop them.

Ah new semester –
DROP TABLE STUDENT_PROJECTS_%;
rest, humble server.

REFERENCES:

– Harold Garfinkel. 1967. *Studies in Ethnomethodology*. Prentice-Hall.
– Erik Borra, Bernhard Rieder, (2014) "Programmed method: developing a toolset for capturing and analyzing tweets", *Aslib Journal of Information Management*, Vol. 66 Iss: 3, pp.262 - 278.

keep intact

works, which are not by their nature

code;

"keep intact

background-color:

else { implode

keep intact

$changes =

$newDiff =

$newKey =

foreach compilation of you

lookbehind

under

the source code has-props

{UNTITLED}

By Anonymous

The poem is composed by pieces of the 1000+ pages long code of TCAT (Twitter Capture and Analysis Toolset); a digital tool used to scrape social media data from Twitter.

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Light & Dark</title>
        <meta charset="utf-8">
    </head>
    <body>
    <h1>For a research worker</h1>
    <p> Science<br></p>
    <p> the unforgotten moments<br></p>
    <p> life is rare<br></p>
    <p> product of plodding work<br></p>
    <p> nature reveals itself<br></p>
    <p> the secret is within oneself<br></p>
    <p> obfuscation & shrouds<br></p>
    <p> clarity, daybreak and release from clouds
<br></p>
    <p> glycogen to glucose<br></p>
    <p> a pattented symbiosis<br></p>
    </body>
</html>
```

For a research worker

Science

the unforgotten moments

life is rare

product of plodding work

nature reveals itself

the secret is within oneself

obfuscation & shrouds

clarity, daybreak and release from clouds

glycogen to glucose

a pattented symbiosis

{LIGHT AND DARK}

By David Søbæk Olsen

I took inspiration of Gerti Cory :) She discovered the conversion of glycogen to glucose and that is something very relevant to understanding our internal composition. So in that sense I wanted to convert her statements of the life of a researcher into something of my own. Taking a couple of lines and making it more 'present' to my context and time as a learning student and researcher.

{ELSE INCLUDE}

By Marisa Leavitt Cohn

```
<?php

/*
 * Database credentials
 */
$dbuser = "";
$dbpass = "";
$database = "twittercapture";
$hostname = "localhost";

/*
 * Capturing role(s) for DMI-TCAT
 * Here you can define which types of capturing you would like to do
 * Possible values are "track", "follow", "onepercent".
 * Note that you can only do one of track, follow or onepercent per
IP address and capturing key.
 */
define('CAPTUREROLES', serialize(array('track')));

/*
 * The user(s) who can add and modify query bins.
 * All users should exist in your htaccess authentication
configuration.
 * Leave the array empty if you do not want to restrict access to
the query manager - which, of course, is a security risk.
 */
define('ADMIN_USER', serialize(array('admin', 'admin2')));

/*
 * *Super advanced and currently undocumented feature, leave
settings as they are*
 * We have made it possible to tunnel Twitter API connections
through other hosts (obtaining a different source IP address), and
use multiple keysets for multiple streaming queries.
 * Each capture script should define its role, see
define('CAPTUREROLES',serialize(array()))
 * Every distinct role should then get a different network path
below
 *
 */
$GLOBALS["HOSTROLE"] = array(
    'track' => "https://stream.twitter.com/",
    'follow' => "https://stream.twitter.com/",
    'onepercent' => "https://stream.twitter.com/",
);

/*
 * Mail address to report critical errors to
 */
$mail_to = "";

/*
 * Twitter API keys (basic configuration)
```

Ada Lovelace Day causes me to reflect on the many efforts of inclusion made on my behalf as a woman proximate to STEM fields over the years. I have been included many times and I have learned incompletely, inadequately, slowly to code. I am non-proficient in a handful of languages. I have had my capacity to theorize code as a medium in my anthropological research questioned if I am myself not a proficient coder. For my poem, I had in mind to see if Ahmed's work on diversity and inclusion, might itself be expressed in code. Ahmed has helped me reflect on these many efforts of inclusion I have taken part in as well as my hesitations and resistances to them. I chose to begin from the language in our TCAT application in part because it represented a kind of monolith of code, something that appears impenetrable and yet was so evocatively easy to break by stealing some of its pages. Working from lines of existing code turned out to provide ample opportunity to reflect upon Ahmed's ideas. Code is full of language about powers, to grant or deny access, to start or end a process, to control or grant permission. Working with the comments in the code felt at moments like cheating, a back door to natural language for writing poems, but within the comments are expressions of desire about what we want to enable or empower or limit. Eliminating lines of code felt empowering in a way that trying to write code never has. Removing what executes to leave something that resides in the meanings of the words. The imperative tones of code were an interesting material to work with, where writing poetry through deletion could break or stop that imperative by leaving it impotent, without its object. The enjambment of code's existing line breaks I sometimes left as is, to expose the repetitiveness of the code in a way that reminds me of Ahmed's ideas on the tedium and never ending ness of diversity work, of just existing so others can continue to exist without growing weary. At other times I wanted to stitch the lines back together to concatenate what was once a serial set of commands. If else statements meet in "if else", becoming a kind of threat. Binaries come back together again. This and not this come together as internal contradiction.

Left is safe Right is secret,
This request
Left and right
is empty.
We always replace this request;
This request implodes.

Set time for each,
For each is set:
Nodespell time time,
Edgespell time time.

You can only follow The us who can exist,
Leave empty if you want to.

We made it possible
Each capture, See
Every role below
Follow.
Withheld
Withheld.

Create, if not null
Not null, create places
Prepare, create places
Create, prepare, execute
Create if not null
Create not,
Not null.

Withheld from us
Withheld from us
Possibly withheld from u.

Exit Already.
I will not exit.
Attempt to recognize if exiting.

Exit,
If else
Attempt to recognize.
We are greedy.
You r not recognized as containing exit.

Exit.
Include once or all
If else you did not specify enough to parse
If exists, does not exist, or is not exiting.

Retry
Retry
else include
~~Include~~
~~Include~~
~~Include~~
Include
Only run from each.

If empty
Die
If empty
Die not.
Query new for each
Else get time.
If we want ~~the future~~
We must set the end times to now
~~To start It starts the users who share the most~~

Another seems to exit
Sending a signal
Waiting for a graceful exit.
We need some time to allow sleep.
Sleep
if unable
Break.

<html>
<head>Net Day</head>

<body>
<img_src>6d6a8816-1989-4303-b0e2-6307eb0604dd-golden_primary.jpg</
img>

<h1>Net Day</h1>
<h2>get some cheesecake, gather around</h2>
Picture it... 1997, suburban Perth, Australia. It's a Saturday morning and there's excitement at the local primary school. It's Net Day!

there was excitement...
about infrastructure!
People were going to lift up some paving bricks and some manhole covers to install some new, fast cables so that we could get the internet at school. Yours truly, aged 12 decided to volunteer to be a guide for Net Day.

But infrastructure isn't super exciting, despite its capabilities.

What do you show people?

Bricks

Cables

Yellow, grainy sand?

<h2>And then</h2>
So Net Day came and went
And geocities
And ICQ
(Uh-oh!)
And blogger
And MySpace
And that RSS reader that you mourn the loss of.

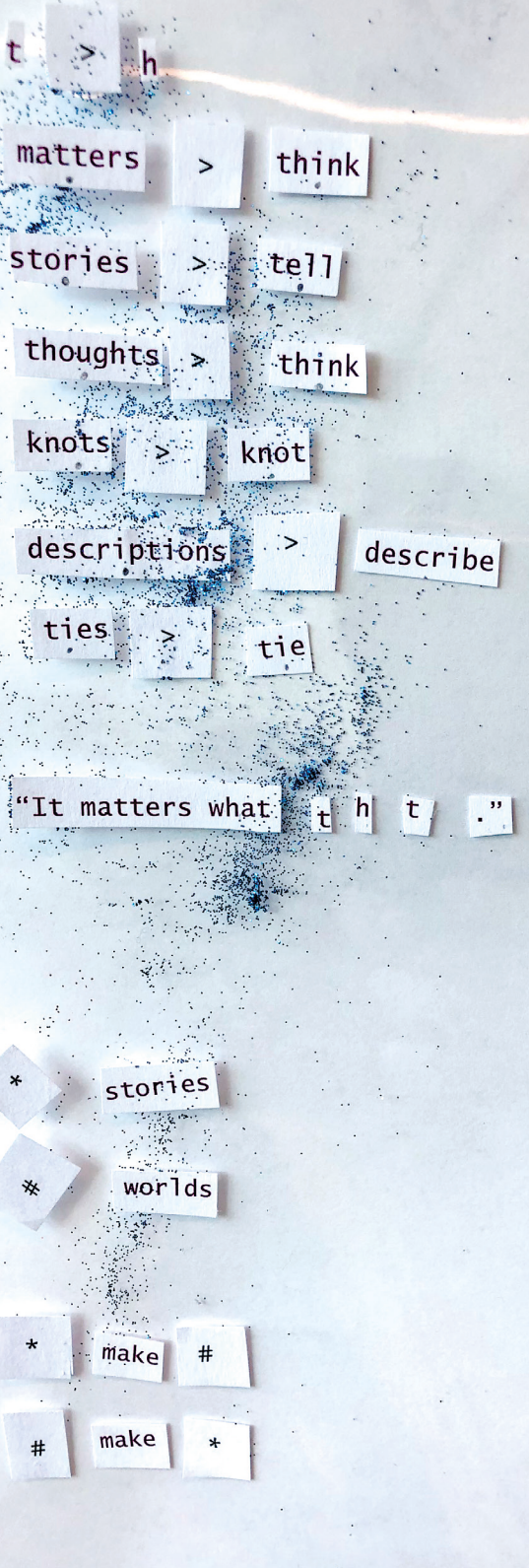
Now everyday is Net Day.
Net Day is everywhere
Net Day is coming along for the ride
Telling you where to go.
Allowing you to share ephemera from a pre Net Day era.
Gifs of the Golden Girls being sassy af.
You are Net Day.

but what do you show people?

{NET DAY}

By Jessamy Perriam

I really don't mean this to sound like a "Back in my day..." kind of a poem, but inevitably it will. But I wanted to capture a moment in the mid 90s where we were encountering internet infrastructure as it was being constructed, guiding us online (many) for the first time. Of course, infrastructure isn't that exciting so it was just reflections of cables being laid in the sandy soil of a West Australian primary school. It's written in HTML because... well, CSS and JS and other coding languages that visually make up the internet simply weren't around back then (it was the geocities era, after all). What's with the Golden Girls reference? I don't know, it was a big deal for me back then. I was a weird kid.


t > h
matters > think
stories > tell
thoughts > think
knots > knot
descriptions > describe
ties > tie
“It matters what t h t .”
* stories
※ worlds
* make #
# make *

{STORIES TELL STORIES}

By Imke Grabe

The poem decomposes words from Donna Haraway's Staying with the Trouble: Making Kin in the Chthulucene. I broke the original text down into code-like structure as an attempt to revel its underlying logic, which often helps me to grasp the content when reading natural language.

{YOUR
CODE
POEM}

{AFTERWORD}

In an interview twenty years ago, Janice J. Heiss asked Richard Gabriel, a computer scientist with a Master in Fine arts in Poetry, how writing poetry had influenced the way he wrote code. His response:

– *Writing code certainly feels very similar to writing poetry. When I'm writing poetry, it feels like the center of my thinking is in a particular place, and when I'm writing code the center of my thinking feels in the same kind of place (Heiss and Gabriel 2002)*

Through the event that produced these poems, we explored this 'centre of thinking', together, exploring the question of how code and poetry work. The poems in this collection are the printed result. But why bring code and poetry together here, now, and why in this (printed) format? A similar question could be asked of poetry and regulation, a combination of genres with which we have previously experimented in ETHOS Lab. A partial answer comes from our role in convening people within the Lab, and experimenting with methods that open up the world, showing us how they do so as we put them to work.

Like the erasure poetry we created from the General Data Protection Regulation (Douglas-Jones and Cohn 2018, Douglas-Jones, Blønd and Bluum 2020), code poetry can be considered method as much as intervention, as much an analytical exercise as a site of collaboration and meaning making. However, the poems and conversations resulting from code poetry lead us in different directions. In this afterword, we describe these directions, their implications in our present moment, and what it means to encounter code poetry in the early 2020s.

Writing

Code poems have likely existed as long as code, and, some scholars, argue, since before computation as we know it today (Freeman 2019, Chetcuti 2014, Aarseth 1997)[1]. On an archived Usenet site of the 1990s, a

[1] Chetcuti analyses ee. Cummings poem "I will be" alongside Stefan's "The Dreamlife of Letters" to show how typewriter code illustrates the 'instrumental role of the machine in the poem's creation' (2014:176).

By Rachel Douglas-Jones
and Marisa Leavitt Cohn

query posted by a Daniel Rosenberg about "forking a bunch of processes" led, over the course of a couple of weeks, to a discussion about poetry written in Perl (Practical Extraction and Report Language), the programming language developed by Larry Wall in 1987. As the replies go back and forth, posters discuss how 'bare word mode', also known as 'perl poetry mode' came to be, with Randal Schwartz claiming that 'the real reason this was added was to better support Perl poetry', tagging "Sharon". "Sharon" was Sharon Hopkins, who joined the thread, confirming:

– Yup. :-) There was even a paper on Perl poetry at the 1992 Winter Usenix conference, with examples by myself, Larry Wall, and Craig Counterman, plus a FORTRAN example by David Mar. (Hopkins 1993).

Through this exchange, we can see some of the aesthetic and practical considerations that early Perl code poets were discussing, and how the poetic and aesthetic was also given space in professional settings. By asking participants at our event to read as much as write, we were inviting them to get familiar with elements of code, the visual and the actionable. We were inviting them into this space where the poetic and the professional can occupy the same environment.

Reading and Compiling

In what ways can this collection be read? *Reading* immediately raises the question: who or what is the imagined reader? You, human reader, can access some (aspects) of these poems. To borrow the poet John Cayley's formulation, there is an 'ambiguous address', potentially both 'human reader and machinic processor' can be considered readers. For Chetcuti if code is executable interface text, it raises the question 'will the human reader remain its primary addressee?' (2014:177). Critically, Cayley asks what a 'code-naïve reader' learns about the 'characteristics and power of code' from code poems? (Cayley 2002). It

is a question we might ask for our own settings. Perhaps Adrian Mackenzie would reply that code, 'as the site of social negotiations that structure and organize human agency, behavior and intention' (Mackenzie, summarized in Hayles 2006), necessitates an encounter with its power. In our event, by providing snippets of feminist scholarship, and bringing them into conversation with code elements, we required our poets to also be readers. Yet the question remains for you as a reader of this collection: do 'code and language require distinct strategies of reading'? (Cayley 2002).

One reason Cayley argues that they might require distinct strategies is the breadth of genres of code poetry, some of which we find in this collection. In Edith Terte's poem, for example, this tension between possible readers comes most to light: printed here in these pages, we as readers cannot make the code run. As Cayley puts it, 'for the code to function as generator, as programmaton, as manipulator of the text, it must, typically, be a distinct part of the global textual system; it must be possible to recompile the code as operative procedures, as aspects of live-art textual practice' (2002). A printed collection is not where that happens, but we have, to take another meaning of compilation, compiled a collection.

Including?

On a day like Ada Lovelace Day, the celebration of women and nonbinary people in STEM, we are called to acts of recognition and inclusion. But what does inclusion mean in STEM fields, where the work of women who have contributed to these fields' foundations has historically been erased, (Hicks 2017), and where 'being included' often privileges those doing the including? (Ahmed 2012). Being able to code continues to be seen as an 'accessible' entry into STEM fields that is available as an already open door. Code camps and academies highlight testimonials of career switches made possible by learning to code. (Imagine such an accessible entry to other STEM fields, just learn the syntax of astrophysics and go!)

If code is promised as a door that

is open, or which we attempt to prop open through efforts like Ada Lovelace Day, then maybe we can reflect upon that door through poetry[2]. As Jane Abbate has noted, coding is offered as a source of empowerment through slogans like Anyone can code! when in fact there continue to be many barriers to participation in computing fields beyond making programming itself more accessible (Abbate). "Superficial claims that learning to code will automatically be empowering can mask a lack of commitment to structural change" (Abbate 2021). Those efforts to teach people to code are based on a false assumption that mastery and meritocracy will work together to lift anyone who can gain access to the tools of programming languages.

What then does code poetry offer? Is it meant to be a way into code, open a different sort of door? Perhaps replacing slogans like *Anyone can code* with *Anyone can write code poetry*, we seek to open a different kind of space. Not so much to include and make code appealing to non-coders, but to consider the medium of code together, to encounter the privileges accrued to writing code, and consider the role of code in structuring power relations of which we are a part.

Breaking with code

By providing elements to participants, such as a very large pile of existing code and printed out syntax, we aimed to open a writerly space to work with code as part of poetry making and feminist thinking. Using methods of paper collage, cutting, pasting, deleting, we begin not only from writing code but also breaking it apart, allowing us to use its elements without the retribution of a compiler that says Does Not Compute. By breaking code, we also work with the metaphor of breaking bread together, offering a table spread of code elements which we consume together. Cutting and collaging can be a means not only of writing but also reading, dissembling,

[2] Drawing on Ahmed we can ask how code as an already open door "can tell us something not only about who can get in but who can get by or who can get through" and how keeping the door open is not only to create a pipeline to empowerment but also provide exits. (Ahmed 2020).

reassembling, and partaking of code where the outcome is undetermined.

Working with existing code to write poetry also allows us to embrace the extensive collaborative nature of coding work – the code is not only what runs, but the comments, the errors, the considerations and suggestions embedded into code. In the comments there are not only explanations but also reminders of how to use or not use the code, what will break if not used properly. There are limits on these powers of the code – it can be misused. At times composing code poetry using elements of "natural language" from comments felt like cheating. Likewise, others expressed a feeling that a collage that works freely with code syntax ignoring how those elements function was "not proper" code poetry. This challenges us to consider breaking with notions of purity in how code performs in world making.

The collection comprises running code poems as well as poetry that embrace code as a 'center of thinking'. We find poems that break code syntax apart to repurpose it for feminist thinking, as well as poems that seek to find code-like structure within feminist thought to grasp them differently or make them one's own. Writing code poetry in this way we suggest, offers ways into code via feminist thought as well as ways into feminist thought via code. We find poetry reflecting overlaps of thought across diverse fields, or drawing out personal histories with code, or drawing us into reflections on the way code inhabits our everyday lives through the infrastructures around us.

Our workshop was not, then, a space of mastery over code by the individual authors writing the poems. It was a collective confronting together of "what is code" anyway – is it syntax, is it execution, is it volume, is it monolith, is it words? Encountering our own ambivalences towards mastery of code in making poetry provides both a way in and a way out of ideas of what code is.

{REFERENCES}

Aarseth, Espen. 1997. *Cybertext: Perspectives on Ergodic Literature*. Baltimore and London: John Hopkins University Press.

Abbate, Jane. 2021. "Coding Is Not Empowerment", Your Computer Is on Fire, Thomas S. Mullaney, Benjamin Peters, Mar Hicks, Kavita Philip.

Ahmed, Sara. 2012. On Being Included: Racism and Diversity in Institutional Life. Durham, NC: Duke University Press.

Ahmed, Sara. 2020. "Slammed Doors: Diversity and/as Harassment", Paper presented for Thinking of Leaving: Racism and Discrimination in British Universities Panel, March 6, University of York.

Cayley, John. "The Code is not the Text (Unless It Is the Text)", *Electronic Book Review*, September 10, 2002.

Chetcuti, Clara. 2014. "Enc0d1ng Poetry" *antae* 1(3): 166–180.

Freeman, John. 2019. "Code Poetry in Motino: E.E. Cummings and his Digital Grasshopper" *Postmodern Culture* 29(2).

Heiss, Janice J. and Richard Gabriel. 2002. The Poetry of Programming. https://www.dreamsongs.com/PoetryOfProgramming.html

Hicks, Mar. 2017. *Programmed Inequality. How Britain Discarded Women Technologists and Lost Its Edge in Computing*. Boston, MA: MIT Press

Hopkins, Sharon. 1993. Reply: Forking a bunch of processes https://groups.google.com/g/comp.lang.perl/c/jVu7Zjn9JcY/m/yPMby6sCyyEJ?hl=en [accessed December 22nd 2021]

Hayles, N. Katherine. 2006. Traumas of Code, *Critical Inquiry* 33(1): 136–157 https://criticalinquiry.uchicago.edu/traumas_of_code_by_n._katherine_hayles

#listen (a perl poem)
Sharon Hopkins

```
#!/usr/bin/perl
APPEAL:
listen (please, please);
open yourself, wide,
join (you, me),
connect (us,together),
tell me.
do something if distressed;

@dawn, dance;
@evening, sing;
read (books,poems,stories)
until peaceful;
study if able;

write me if-you-please;
sort your feelings, reset goals, seek
(friends, family, anyone);
do not die (like this)
if sin abounds;

keys (hidden), open locks, doors,
tell secrets;
do not, I-beg-you, close them, yet.
accept (yourself, changes),
bind (grief, despair);
require truth, goodness if-you-will,
each moment;

select (always), length(of-days)
# Sharon Hopkins, Feb. 21, 1991
# listen (a perl poem)
# {for jimmy (and tom)}
```

<link rel="book design"
href="anagramdesign.no">

</CODE POETRY></BOOK