HAUS++

P-Code Magazine

P for Personal
P for Private
P for Popular
P for Poetic

P for Pretty
P for Playful
P for Pleasent
P for Peaceful

P for Primary
P for Proto
P for Post
P for Potential

P for Programming
P for Physical
P for Paper

001

目次

P-Code Magazine 創刊にあたって

プログラミングをもっとフィジカル（物理的／身体的）にするために、「実行できる本が作れないか？」というアイデアが、このプロジェクトの出発点となった。

2004 年、ニューヨーク大学の ITP（Interactive Telecommunications Program）教授のトム・アイゴは、ダン・オサリバンとの共著で『フィジカル・コンピューティング』を出版した。彼らは、コンピュータと人間のインターフェイスが指（キーボード）と目（ディスプレイ）に限られたままであることを指摘し、さまざまなセンサーやアクチュエーターを用いて、人間とコンピュータをよりフィジカルに結びつける方法を提示した。

このフィジカル・コンピューティングは、世界各地の美術デザイン系の大学や、メイカームーブメントなどの DIY 文化を通じて、急速に広がっていった。Processing や Arduino といった、オープンで使いやすいツールが生まれ、さまざまな教育、制作の現場で使われるようになっていった。フィジカル・コンピューティングは、その後のモノのインターネット（IoT：Internet of Things）の発展とも結びつき、私たちの生活を大きく変化させた。

しかしながらそこで、コーディング（プログラムを書くこと）は、未だ取り残されたままである。さまざまなデバイスやツールが生み出されたが、コーディングだけは、いまだに指と目だけで行っているものがほとんどだ。プログラミングという活動を、もっとフィジカルなものにできないか？それが最初の「実行できる本が作れないか？」という問いの根底にあった。

P-code の「P」はプログラミングの「P」、フィジカルの「P」であるが、さらに「Paper（紙の）」「Personal（個人の）」「Private（私的な）」「Popular（大衆の）」「Poetic（詩的な）」「Pretty（可憐な）」「Playful（遊び心のある）」「Pleasant（楽しい）」「Peaceful（穏やかな）」「Primary（最初の）」「Proto（始原の）」「Post（次の）」「Potential（潜在的な）」といったさまざまな意味を重ね合わせている。そうした「P」なコードを実験、実装、普及するための研究機関誌として、この『P-Code Magazine』を創刊する。

HAUS++

久保田晃弘 / 竹田大純 / 林洋介 / 稲福孝信

言語仕様

リファレンス

P-Code はリズムマシンのパターンをテキスト形式で記述するアイデアから発展し、プログラミングの要素を取り入れたライブ・コーディング用の言語である。 コードは左から右に解釈され、数値とそれ以外の記号に分けられ、実行される。 解釈できない記号は全てホワイトノイズとして扱われる。

記号	意味
~	サイン波
^	三角波
N	ノコギリ波
[	矩形波
=	ミュート
<CODE>	CODE 部分を繰り返す
NUMBER	周波数を設定する
+NUMBER	設定された周波数に NUMBER を足す
-NUMBER	設定された周波数から NUMBER を引く
*NUMBER	設定された周波数に NUMBER を掛ける
/NUMBER	設定された周波数を NUMBER で割る

アプリケーション

スマートフォンで P-Code を実行できるアプリケーションを用意しました。
以下のウェブサイトを参照してください。
https://p-code-magazine.github.io/

チュートリアル

数値と記号

数値は周波数を意味する。記号ひとつが時間の単位（1/30 秒）になる。

441Hz のサイン波

```
441~
```

11025Hz の矩形波

```
11025[
```

数値には小数も使うことができる。

123.456Hz の三角波

```
123.456^
```

同じ音を鳴らし続けるには、同じ記号を連続させる。

882Hz のノコギリ波の持続（0.5 秒）

```
882NNNNNNNNNNNNNNN
```

繰り返し

同じことが繰り返しでも書ける。< > で挟んだ部分を繰り返す。

< > のネスト（入れ子）で 2^（ネストの数）の繰り返し

```
882N
```

```
882<N>
```

```
882<<N>>
```

```
882<<<N>>>
```

```
882<<<<N>>>>
```

このように、音の長さが 2 倍になっていく。また、カッコが足りない場合は最後に補完される。

```
882<<<<N>
```

予約されていない文字（記号）以外はすべてホワイトノイズに解釈される。

ノイズの断続（ノイズとミュート（休符））の繰り返し

```
<<<<#=>>>>
```

四則演算

周波数は加減乗除の演算ができる。

1550Hz の矩形波

```
44100/7/5/2/3*5+1000-500[[[[[
```

演算記号を伴わない数値は絶対値とみなされる。

441Hz と 882Hz のサイン波の断続の繰り返し

```
<<<441~~==+441~~==>>>
```

上のコードは、以下のようにも書くことができる。

```
882<<<-441~~==+441~~==>>>
```

```
<<<441~~==*2~~==>>>
```

繰り返しを使っていろいろなリズムをつくれる。

```
<<<441~~==*2~~==>>>
```

周波数を連続的に変化させることもできる。

サイン波のスイープ（線形）

```
100<<<<<<<<+10~>>>>>>>>
```

サイン波の逆スイープ（線形）

```
10000<<<<<<<<-10~>>>>>>>>
```

サイン波のスイープ（対数）

```
20<<<<<<*1.1~~~~>>>>>>
```

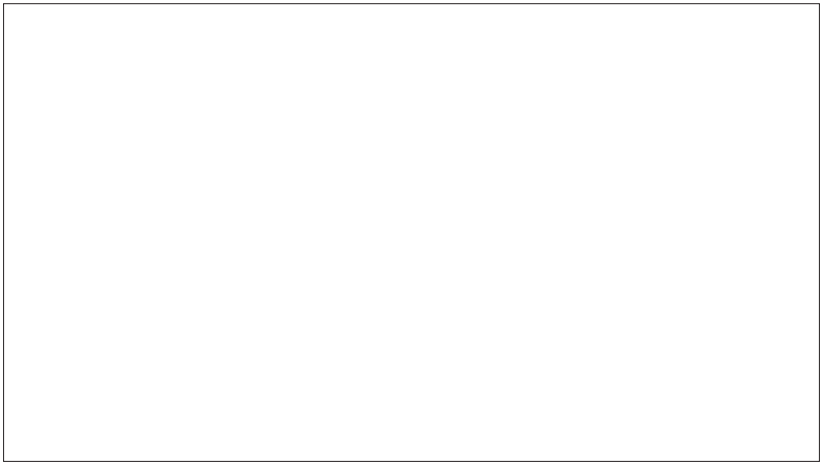
サイン波の逆スイープ（対数）

```
20000<<<<<</1.1~~~~>>>>>>
```

サンプルコードを書いてみよう

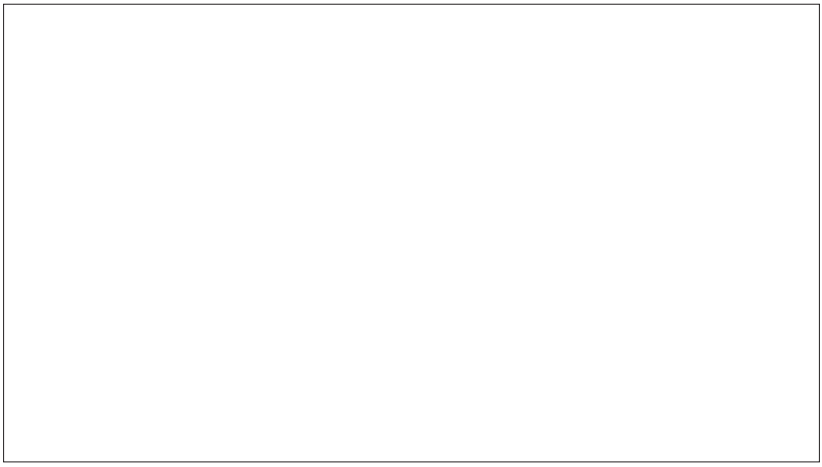
繰り返しとミュートの組み合わせによるリズムパターン

```
<200N==<<<=50^==><<=800~>2000[>>=*==>>
```



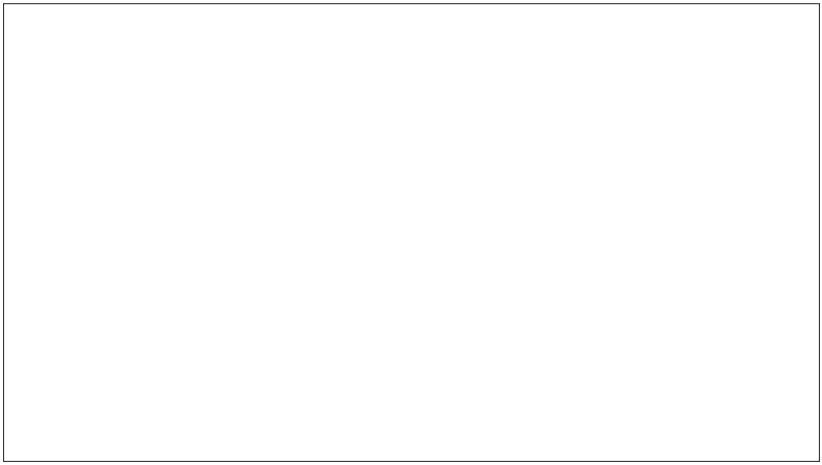
数式のように見えるコードもかける

```
<<<100+100=200~>>>
```



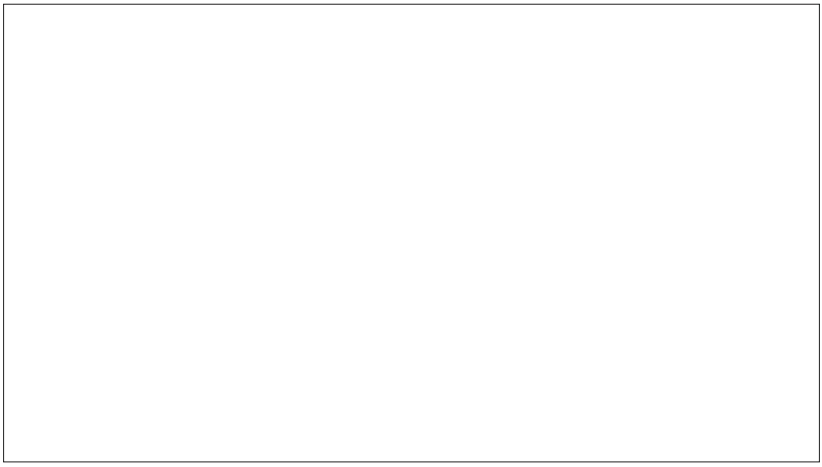
どんなコードを書いてもエラーにはならない

```
Hello, world!
```



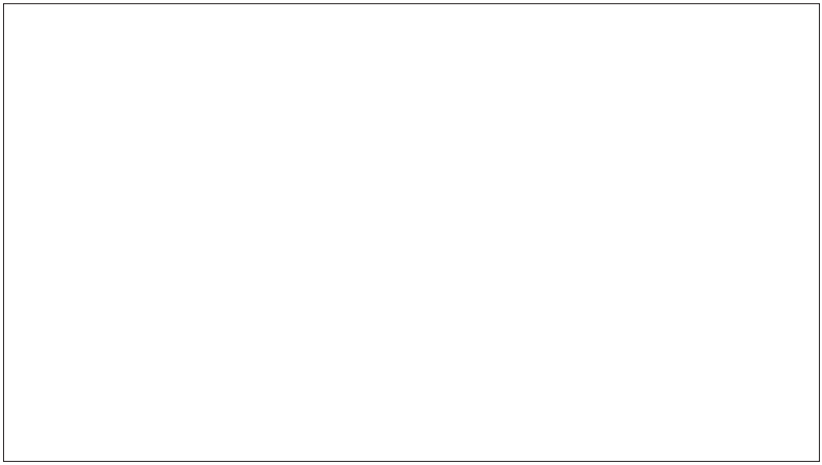
沈黙のコード

```
@<<<<<<<<<<=>>>>>>>>>>@
```



遭難信号 (SOS) も打てる

```
<<[[==[[==[[==[[[[[[==[[[[[[==[[[[[[==[[==[[==[[========>>
```



P-Code はどんなコードを書いてもエラーにはなりません。
恐れずにコードを書き、僕らの想像を超える~~ひどい~~すごい音を聴かせて下さい。

We need a Pan-Hacker movement.
私たちにはパンハッカー運動が必要だ。

Some decades ago, computers weren't nearly as common as they are today. They were big and expensive, and access to them was very privileged. Still, there was a handful of people who had the chance to toy around with a computer in their leisure time and get a glimpse of what a total, personal access to a computer might be like. It was among these people, mostly students in MIT and similar facilities, where the computer hacker subculture was born.

数十年前、コンピュータは今日ほど一般的ではありませんでした。それは大きくて高価であり、アクセスできる人はとても限られていました。それでも、余暇にコンピューターをいじり回して、コンピュータを個人的にアクセスすることが、一体どのようなものかを垣間見ることができた一握りの人々がいました。こうした、主に MIT（マサチューセッツ工科大学）のような大学の学生たちから、コンピューターハッカーのサブカルチャーが生まれました。

The pioneering hackers felt that computers had changed their life for the better and therefore wanted to share this new improvement method with everyone else. They thought everyone should have an access to a computer, and not just any kind of access but an unlimited, non-institutionalized one. Something like a cheap personal computer, for example. Eventually, in the seventies, some adventurous hackers bootstrapped the personal computer industry, which led to the the so-called "microcomputer revolution" in the early eighties.

先駆的なハッカーたちは、コンピュータが自分たちの生活をより良い方向に変えるに違いないと感じたので、こうした新たな改善のための方法を、他の人たちと共有したいと考えました。彼らは、誰もがコンピュータにアクセスできるべきであり、あらゆる種類のアクセスだけでなく、無制限で自由なものであるべきだと考えました。それは例えば、安価なパーソナル・コンピュータのようなものです。70 年代になると、何人かの冒険的なハッカーが、パーソナルコンピュータ会社を立ち上げて、80 年代初頭のいわゆる「マイクロコンピュータ革命」をもたらしました。

The era was filled with hopes and promises. All kinds of new possibilities were now at everyone's fingertips. It was assumed that programming would become a new form of literacy, something every citizen should be familiar with -- after all, using a computer to its fullest potential has always required programming skill. "Citizens' computer courses" were broadcasted on TV and radio, and parents bought cheap computers for their kids to ensure a bright future for the next generation. Some prophets even went far enough to suggest that personal computers could augment people's intellectual capacities or even expand their consciousnesses in the way how psychedelic drugs were thought to do.

時代は希望と期待に満ち溢れていました。あらゆる種類の新しい可能性が、今や皆の手の届くところにありました。プログラミングはすべての市民が慣れ親しむべき、新しい形のリテラシーになると考えられていました — というよりもむしろ、コンピュータの能力を最大限に引き出すためには、プログラミングのスキルがどうしても必要でした。「市民のためのコンピュータ授業」がテレビやラジオで放送され、次の世代の明るい未来を確実なものにするために、親は子供たちに安いコンピュータを買い与えました。一部の極端な人は、パーソナルコンピュータが人々の知的能力を増強し、さらにはサイケデリックな薬物のように意識を拡大することさえできる、と預言しました。

In the nineties, however, reality stroke back. Selling a computer to everyone was apparently not enough for automatically turning them into superhuman creatures. As a matter of fact, digital technology actually seemed to dumb a lot of people down, making them helpless and dependent rather than liberating them. Hardware and software have become ever more complex, and it is already quite difficult to build reliable mental models about them or even be aware of all the automation that takes place. Instead of actually understanding and controlling their tools, people just make educated guesses about them and pray that everything works out right. We are increasingly dependent on digital technology but have less and less control over it.

しかし 90 年代になると、現実は逆行してしまいます。万人にコンピュータを販売しても、人々をもれなく超人的な生き物に変えることはできませんでした。実際のところ、デジタル技術は多くの人々を失望させ、人々を解放するのではなく、無力で依存的にしてしまいました。ハードウェアもソフトウェアもますます複雑になり、その確かなメンタルモデルを構築することも、さらにはそこで行われている自動操作を意識することさえ、困難になりました。コンピュータのツールを実際に理解して使いこなすのではなく、人々はコンピュータを自分の経験の中だけで想像し、すべてが正しくうまくいくことを祈るだけです。私たちはますますデジタル技術に依存するようになり、それをコントロールすることはますます少なくなっています。

So, what went wrong? Hackers opened the door to universal hackerdom, but the masses didn't enter. Are most people just too stupid for real technological awareness, or are the available paths to it too difficult or time-consuming? Is the industry deliberately trying to dumb people down with excessive complexity, or is it just impossible to make advanced technology any simpler to genuinely understand? In any case, the hacker movement has somewhat forgotten the idea of making digital technology more accessible to the masses. It's a pity, since the world needs this idea now more than ever. We need to give common people back the possibility to understand and master the technology they use. We need to let them ignore the wishes of the technological elite and regain the control of their own lives. We need a Pan-Hacker movement.

一体何が間違っていたのでしょうか？ ハッカーは万人のためのハッカー世界への扉を開きましたが、そこに大衆は入って来ませんでした。多くの人が、単に愚かすぎて実際の技術を認識でき

なかったのか、そのための道のりが難しすぎたのか、あるいは時間がかかり過ぎたのでしょうか？コンピュータ業界が意図的に、複雑にし過ぎることよって、人々を騙そうとしているのでしょうか、それとも最先端のテクノロジに対する真の理解を単純にすること自体が、不可能なのでしょうか？ いずれにせよ、ハッカー運動は、デジタル技術を大衆にとってより身近なものにすることを、おろそかにしていたことは確かです。世界は今まで以上に、こうしたハッカーの考え方を必要としているので、それはとても残念なことです。私たちは一般の人々に、ハッカーが使う技術を理解してもらい、それを習得する可能性をお返ししていく必要があります。ハッカーに「技術エリートになりたい」という願望を捨ててもらって、自分たち自身の生活に対するコントロールを取り戻す必要があります。私たちには、パンハッカー運動が必要なのです。

What does "Pan-Hacker" mean? I'll be giving three interpretations that I find equally relevant, emphasizing different aspects of the concept: "everyone can be a hacker", "everything can be hacked" and "all hackers together".

「パンハッカー」とは一体どういう意味でしょうか？私はそこに、それぞれに異なる意味があるものの、等しく関連があると思われる 3 つの説明があると考えています。それは

- 誰もがハッカーになれる
- あらゆるものがハックできる
- すべてのハッカーがいっしょに

です。

The first interpretation, "everyone can be a hacker", expands on the core idea of oldschool hackerdom, the idea of making technology as accessible as possible to as many as possible. The main issue is no longer the availability of technology, however, but the way how the various pieces of technology are designed and what kind of user cultures are formed around them. Ideally, technology should be designed so that it invites the user to seize the control, play around for fun and gradually develop an ever deeper understanding in a natural way. User cultures that encourage users to invent new tricks should be embraced and supported, and there should be different "paths of hackerdom" for all kinds of people with all kinds of interests and cognitive frameworks.

最初の説明は「誰もがハッカーになれる」ことです。それは、オールドスクールハッカーの基本的な考え方、つまりテクノロジーを可能な限りたくさんの人が利用できるものにする、という考えを拡張したものです。しかし、主な問題はもはや技術の利用可能性ではなく、さまざまな技術がどのようにデザインされているか、そしてそれらの周囲にどのようなユーザー文化が形成されているかということです。理想的には、テクノロジーはユーザー自身がコントロールでき、楽しみのために遊ぶことができ、自然な方法で深い理解に少しずつ到達できるようにデザインされるべきです。ユーザー自身が新しいトリック（うまいやり方）を発明することを奨励するユーザー

文化を受け入れ、支援しなければなりません。あらゆる種類の興味と認知的な枠組みを持つ人々に対して、さまざまな「ハッカーの道」があるべきです。

The second interpretation, "everything can be hacked", embraces the trend of extending the concept of hacking out of the technological zone. The generalized idea of hacking is relevant to all kinds of human activities, and all aspects of life are relevant to the principles of in-depth understanding and hands-on access. As the apparent complexity of the world is constantly increasing, it is particularly important to maintain and develop people's ability to understand the world and all kinds of things that affect their lives.

第二の説明は「あらゆるものがハックできる」です。それは、ハッキングの概念を技術の領域からさらに外へ広げることを意味しています。ハッキングの一般的な考え方は、あらゆる種類の人間の活動に関連していて、生活のあらゆる面に対する深い理解と実践的なアクセスの原則に結びついています。世界の見かけの複雑さは絶えず増大しているので、世界とそこでの生活に影響を与えるあらゆる種類のものごとを理解するための能力を、維持し発展させることが特に重要です。

The third interpretation, "all hackers together", wants to eliminate the various schisms between the existing hacker subcultures and bring them into a fruitful co-operation. There is, for example, a popular text, Eric S. Raymond's "How To Become A Hacker", that represents a somewhat narrow-minded "orthodox hackerdom" that sees the free/open-source software culture as the only hacker culture that is worth contributing to. It frowns upon all non-academic hacker subcultures, especially the ones that use handles (such as the demoscene, which is my own primary reference point to hackerdom). We need to get rid of this kind of segregation and realize that there are many equally valid paths suitable for many kinds of minds and ambitions.

3 番目の説明は「すべてのハッカーがいっしょに」です。既存のハッカーのサブカルチャー間のさまざまな分裂を取り除いて、それらを実りある協力に導きたいと考えています。たとえば、エリック・レイモンドの「How To Become A Hacker（ハッカーになろう）」というテキストがあります（https://cruel.org/freeware/hacker.htm）。これは、フリー / オープンソースのソフトウェアカルチャーを、唯一の貢献する価値のあるハッカーカルチャーと見なす、やや狭義の「正統派ハッカーの世界」です。それはすべての学術的でないハッカーのサブカルチャー、特にハンドルネームを使うようなもの（私自身にとってハッカー共同体の第一の事例であるデモシーンのようなもの）を認めません。私たちはこうした分裂を解消し、さまざまな考えや目標に応じた道のりが、等しくあることを理解しなければなりません。

Now that I've mentioned the demoscene, I would like to add that all kinds of artworks and acts that bring people closer to the deep basics of technology are also important. I've been very glad about the increasing popularity of chip music and circuit-bending, for example. The Pan-Hacker movement should actively look for new ways of "showing off the bits" to different kinds of

audiences in many kinds of diverse contexts.

デモシーンについて述べたので、テクノロジーの奥深い基礎に人々を近づけてくれる、あらゆる種類の芸術作品と活動が重要であることも付け加えたいと思います。例えば、私はチップミュージックやサーキットベンディングの人気が高まっていることをとてもうれしく思います。パンハッカー運動は、多種多様な文脈の中で、さまざまな種類の観客に「ちょっとした魅力」を披露するための新しい方法を、積極的に模索していくべきです。

I hope my writeup has given someone some food of thought. I would like to elaborate my philosophy even further and perhaps do some cartography on the existing "Pan-Hacker" activity, but perhaps I'll return to that at some later time. Before that, I'd like to hear your thoughts and visions about the idea. What kind of groups should I look into? What kind of projects could Pan-Hacker movement participate in? Is there still something we need to define or refine?

私の記事が、誰かの思考の糧になることを願っています。私は私の哲学を、さらに詳しく述べたいと思いますし、すでにある「パンハッカー」活動についての地図作成もしたいのですが、それにはもう少し時間が必要です。その前に、このアイデアについての、みなさんの考えやビジョンを聞きたいです。どのようなグループを調査すればいいですか。パンハッカー運動はどのようなプロジェクトに参加できますか？まだ他に定義や改良が必要なものはありますか？

（訳：久保田晃弘）

countercomplex
bitwise creations in a pre-apocalyptic world

Viznut a.k.a. Ville-Matias Heikkilä
FRIDAY, 17 JUNE 2011
http://viznut.fi/

https://countercomplex.blogspot.com/2011/06/we-need-pan-hacker-movement.html

スペシフィック・コード《Specific Codes》

久保田晃弘

1 行のコード

ソフトウェアと、それを記述するコンピュータ・プログラムは、今日もっとも身の回りに溢れ、私たちの生活の中に偏在するメディアとなった。コンピュータはプログラムに書かれたコードを翻訳、解釈、実行することで、大量のデータを処理し、その結果を表示したり、コンピュータ同士でやりとりする。人々は、コンピュータのプログラムを作成するだけでなく、ソフトウエアを日常的に使用することで、ものごとの見方や考え方を形作っていく。私たちは、プログラム・コードをつくるだけでなく、プログラムによってつくられている。

プログラムは、コンピュータと人間の双方が理解可能な、人工言語によって記述されている。プログラムはコンピュータと人間のコミュニケーションの記述であり、より一般的には（分析や解釈の対象となる）「テクスト」である。そこには、アルゴリズムだけでなく、プログラム制作の前提、目的、過程、改良の過程が埋め込まれている。さらに変数や関数の名付け方や、アルゴリズムの注釈や解説といったプログラムのアルゴリズム以外の部分にも、さまざまなものごとを書き込むことができるし、逆に人間にとって理解しにくいよう難読化することもできる。プログラムは数学 / 数値的、論理 / 構造的な言語であると同時に、文化 / 人文的、思想 / 哲学的なテクストでもある。

実用的な、あるいは有用なプログラムの多くは、正しく、効率よく実行すること、あるいは保守や改良、再利用がし易いことが求められている。それらを巧みなバランスで実現したコードはしばし、コンピュータ・プログラミングの「ART（技芸）」と呼ばれる。しかし、そうした自明な目的や有用な機能がないプログラム・コードでありながら、多くの人に共有されているものも存在する。そのひとつの（代表的であり、極めて早い時期の）例が、「10 PRINT」と呼ばれる 1 行の BASIC プログラムである。

```
10 PRINT CHR$(205.5+RND(1));:GOTO 10
```

Commodore 64 という、米コモドール社が 1982 年に発売開始した 8 ビットの家庭用コンピューターで、この極めてシンプルなコードを実行すると、斜め線、または逆斜め線のグラフィック記号がランダムに 1 つづつ、左から右、そして上から下へと表示されていく（図 1）。画面が一杯になると 2 行ずつスクロールし、このプログラムは（中断するまで）永遠にこの表示を繰り返す。当時のコンピュータの速度は遅かったので、記号が画面を埋め尽くすのには約 15 秒かかる。し

図 1「10 PRINT」プログラムを VirtualC64（Commodore 64 のエミュレータ）上で実行している様子

かしこのたった 1 行のコードが、当時のさまざまなパーソナルコンピュータに移植され、さまざまなバリエーションが生みだされた。そして、Commodore 64 の誕生から 30 年を経た 2012 年、この文化的工芸品としてのプログラムを、ソフトウェア・スタディーズの視点から詳細に分析した本（ https://10print.org/ ）が The MIT Press から出版された。本のタイトルも『10 PRINT CHR$(205.5+RND(1));:GOTO 10』である。

ジェネリックでないコード

今日の、使いやすく、わかりやすく、役に立つ、そして技芸に優れたソフトウェアからみれば、こんなちっぽけで単純な迷路生成プログラムは、取るに足らないもののように見える。しかしそんな、40 年近く前のマイクロコンピュータのための 1 行のプログラムに対して、今なお多くの人が関心を持ち、議論し、さらには今日のコンピュータにも移植され、さまざな修正版が実行さ

れているのはなぜなのだろう。

まず重要なのは、このプログラムが非常に短いことである（おそらくこれ以上に短いものは、”hello, world” くらいのものだ）。今日の大規模データを活用した計算論的（computational）な文化分析手法（cultural analytics）が、対象そのものに触れることなく、そのマクロな傾向を把握しようとするのに対し、このコードは私たちに、ミクロな「精読」を要求する。しかしその精読は、限られた専門家による精読ではなく、（コードが短いがゆえの）万人に開かれた、そして対象に自由に手を加えることができるような精読である。

伝統的なプログラミングにおいては、プログラムで記述しようとする対象を分析し、それをなるべくシンプルなかたちで抽象し、（人間にとって）わかりやすい形で表現することが推奨されてきた。こうした機能の抽象化によって、コードの汎用性、再利用性を高めたコードを「ジェネリック・コード」と呼ぶとすれば、「10 PRINT」のように、（キャラクター文字を使って迷路のような模様を生成し続けるという）ある限られた目的のために作られたプログラムは、「スペシフィック・コード」と呼べるだろう。

「スペシフィック・コード」という呼び名は、「ジェネリック・コード」の対義語であるだけでなく、米国の美術家ドナルド・ジャッドが 1964 年に提示した「スペシフィック・オブジェクト」という概念にも由来している[1]。ジャッドのこの概念は、50 年代から 60 年代に制作された、アメリカ美術の新しい傾向の特徴分析から生まれたものである。「彫刻でも絵画でもない」このスペシフィック・オブジェクトのように、「スペシフィック・コード」はプログラム・コードであるだけでなく「テクストでもポエトリーでもある」。一般的で汎用のコードは、コードそのものよりも、コードが記述しているアルゴリズムとその実行結果が重要なことが多い。それに対して、スペシフィックなコードは、コードそのもの、そこに何がどのように記述されているのかが重要である。つまり、コードの実行結果（例えば「10 PRINT」が描く迷路自体）ではなく、コードを実行した主体がその実行をどのように受容（観賞）したのか、コードの実行中に何が生み出されているのか、そしてそれらが指し示しているものごとは何なのか、ということに思いを馳せなければならない。

ミニマルであるが故に、固有のものであると同時に拡張的でもあるスペシフィック・コードは、ユーザーにシステムやツールを提供するのではなく、ユーザーとしての、つまり個人の使用から見たコードの内在的な可能性を探求し、それを限りなく拡げていこうとする。通常のプログラミングにおける有用性や再利用性のような、客観的な価値や機能を実現するのではなく、その意味や価値は状況や文脈（コード以外の環境）に大きく依存している。スペシフィックであるという

[1]テキストは、例えばここで読むことができる。https://juddfoundation.org/artist/writing/

ことは、ジェネリックではないだけでなく、それが異質であり、強いインパクトを持っている、ということでもある。

テクストとしてのコード

スペシフィック・コードが現れる代表的な場として、「コード・ポエトリー」と「ライヴ・コーディング」の 2 つがあげられる。コード・ポエトリーとは、その名前の通り、コードを用いて詩を書くことである。その代表的なサイトの一つである、Source Code Poetry（https://www.sourcecodepoetry.com/）には、このような規範が書かれている。

- どんな言語でもいい：あなたが一番好きな言語で書いてください。
- コンパイルできること：とはいえ、インタープリタ言語で書かれたものでも受け付けます。
- 韻を踏むこと：とはいえ、現代の名作は規範を逸脱しています。

このサイトには、さまざまなコード・ポエトリーの作例が掲載されている。中でも、Python 言語で書かれたこの Mike Heaton の詩はもっとも短いものである。

```
t = 0
while True:
    print("Nothing lasts forever.")
    t += 1
```

「10 PRINT」と同じように、単にテキスト出力を無限に繰り返す（だけの）ものであるが、時間を表す変数名の「t」と出力を繰り返す文の間には、意味の詩的な結びつきがある。

2012 年に刊行された「code {poems}」（http://code-poems.com）というアンソロジーには、55 のコード・ポエトリーが掲載されている。これらは、プログラム・コードとして実行するよりもむしろ、テクストとして読まれることを意図している。例えば、Daniel Bezerra の「UNHANDLED LOVE（処理されない愛）」は、C++ のプログラムではあるが、その実行結果ではなく、プログラムのエラー管理機能としての「例外処理」が持つ意味を用いた詩である。

```
class love {};

void main()
{
    throw love();
}
```

Richard Littauer の「Import Soul」も同様に、Python 言語としての意味とテクストそのもの意味が重ね合わせられている。

```
# This script should save lives.

import soul

for days in len(life):
  print "happiness"
```

Daniel Holden と Chris Kerr による、「./code--poetry」（http://code-poetry.com/）は、逆にコードの実行結果に着目したものである。そこにはコードの実行時に具体詩、あるいはアスキーアートのような視覚的出力（図 2）が生まれるコード・ポエトリーが数多く収録されている[2]。

パロールとしてのコード

ライヴ・コーディングは、プログラム・コードを直接操作しながら行うライヴ・パフォーマンスの総称である。その起源は、21 世紀初頭のラップトップ・ミュージック、さらには 80 年代の FORTH 言語の音楽制作への使用に遡ることができる。今日のライヴ・コーディングは、Algorave（https://algorave.com/ ）というクラブ文化との融合や、プログラミング教育への応用（例えば https://sonic-pi.net/ ）など、多様な文化と結びついている。

---

[2]久保田晃弘「コード・ポエトリー：算法詩から自己複製芸術まで」も参照のこと（「遙かなる他者のためのデザイン―久保田晃弘の思索と実装」（BNN 新社 , 2017）

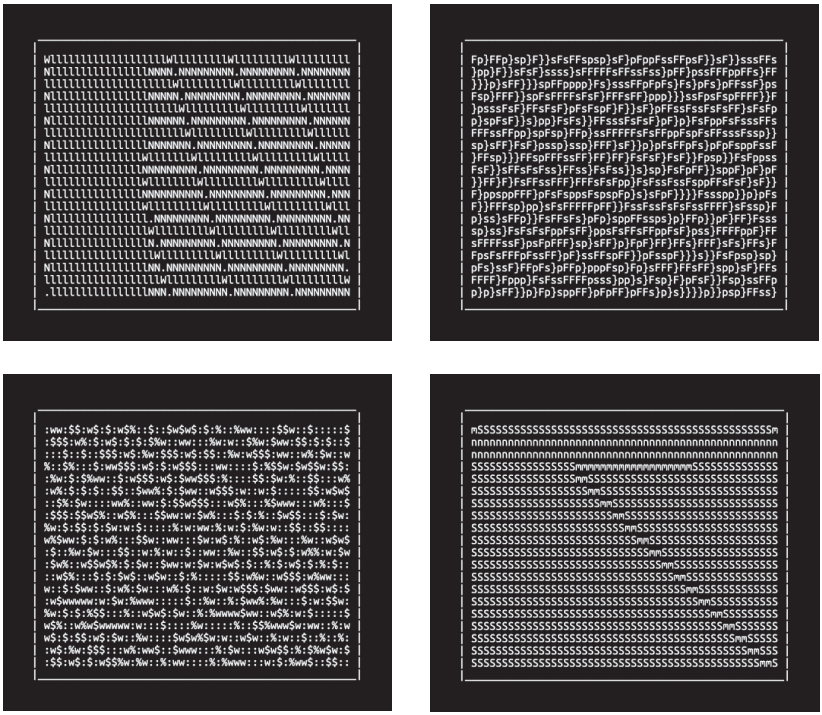


図 2 ./code --poetry トップページの「turing.poem」のスナップショット

ライヴ・コーディングのスタイルには、白紙のエディターから始めて、その場で一からすべてのコードを書く人から、事前に用意していたコードのフラグメントを、DJ のようにその場で組み合わせていくスタイル（CJ: コード・ジョッキー）まで、さまざまなやり方がある。しかし、基本的にライヴ・コーディングにおけるコードは、その場で生まれ、その場で消える。コード・ジョッキーのように、事前に準備しておくことはあっても、パフォーマンスの場で一度実行されたコードを、そのまま別のライヴで再利用することはない。

ライヴ・コーディングの場におけるコードの現れ方を、もっとも特徴的に体現しているのが、ライヴ・コーディング運動の創始者の一人であり、自らが生み出した TidalCycles（ https://tidalcycles.org ）というライヴコーディング言語を用いて行われる、アレックス・マクリーンのパフォーマンスだろう。彼が 2018 年 11 月に来日した際に出演した、DOMMUNE でのライヴの様子が以下に記録されている（図 3）。

TidalCycles という言語が、少ない入力で迅速に出力（パターン）を変化できるように設計されているため、アレックスが書くプログラムは決して長くはなく、どんなに長くても数行で、1 画面の中に全体が収まる程度である。ライヴ・コーディングにおいては、ディスプレイが身体であり、この身体をパフォーマーと観客が共有することから出発していることを考えれば、常にコード全体を一望できることは、ライヴ・コーディングの演奏者にとっても、リスナーにとっても、重要な意味を持っていることがわかる。

しかし何よりアレックスのコーディングで印象的なのは、コードの入力のしなやかさと、書いたコードを実行し終わるとすぐに消し、新たなコードを次々と書き続けていく、エフェメラルな姿勢である。プログラム言語というと、どうしてもアルゴリズムを正確に記述するラング（規則体系としての言語）を思い浮かべがちだが、Alex のライヴ・コーディングにおけるコードは、話しことばとしてのパロール（個人的な発話）である。コード入力の行き来はアレックスの思考の構造を垣間見せ、カーソルの揺らぎはアレックスの思考の状態を反映している（ように見える）。このパフォーマンスでは、映像のちょうど 18 分のところで突然コンピュータがクラッシュし、再びそこからライヴを再開するのだが、そんなハプニングも決してエラーやミスには感じられない。日常の会話においては、いい直したり、中断する（させられる）ことは茶飯事である。パロールとしてのコードは、スペシフィック・コードのもうひとつの重要な特徴である。

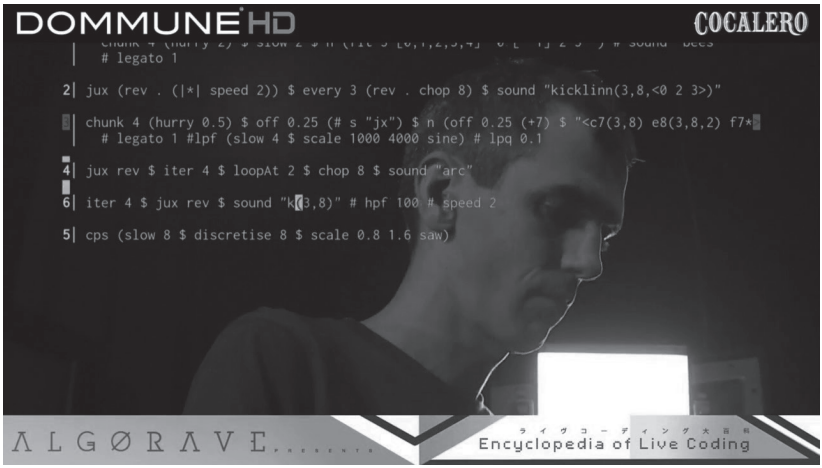


図 3 Alex McLean (Yaxu) live on DOMMUNE Tokyo, 14 Nov 2018
(https://youtu.be/dIpzU71LAQQ)

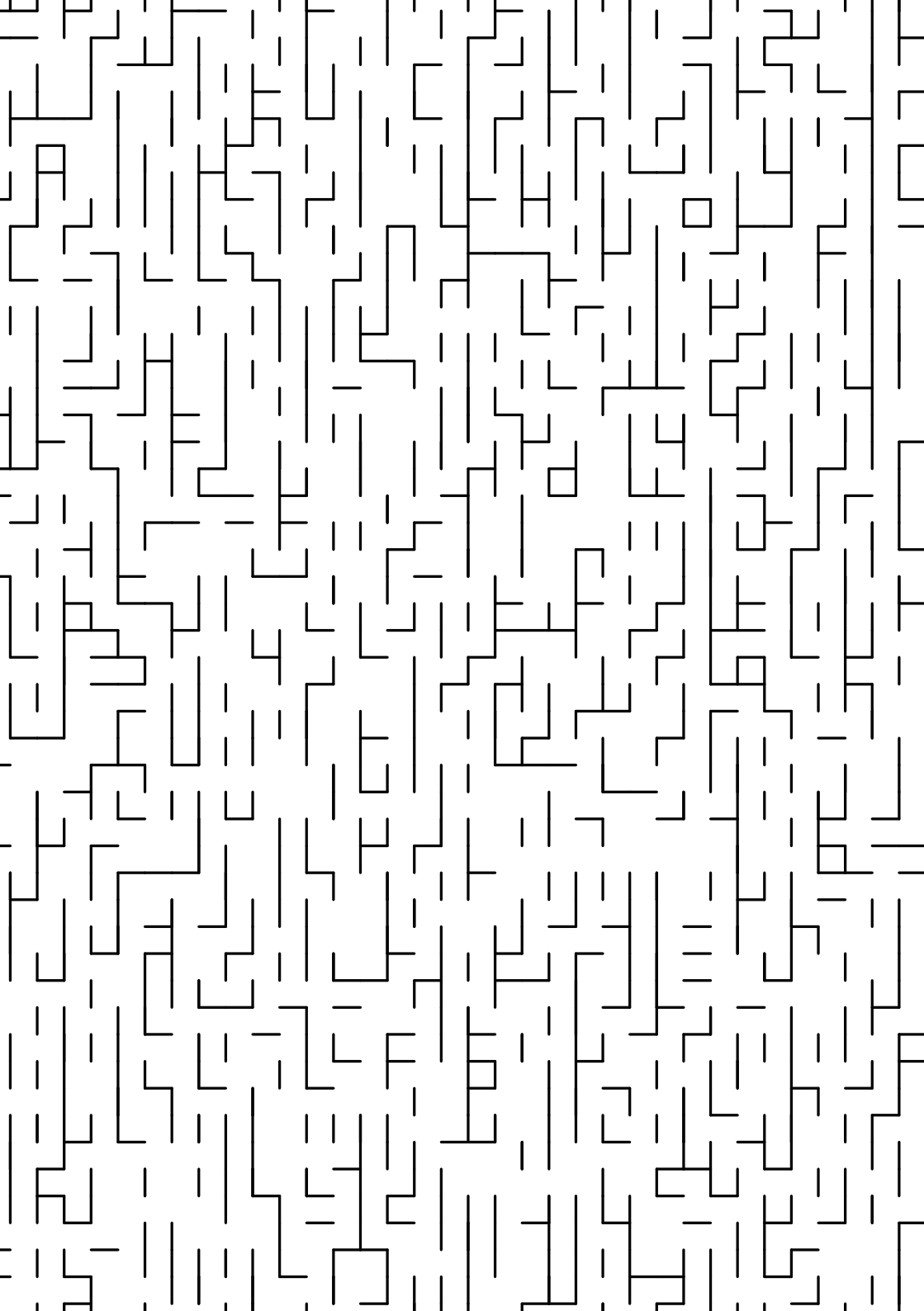
通気口としてのコード

ソフトウェアが日常のインフラストラクチャーとなり、スマートフォンが生活の日用品となった今、それらは生活の中で、ますます見えなくなっている。人間は、スマートフォンを運ぶメディアとなり、人々のものの考え方や行動は、暗黙のうちにソフトウェアによって操作管理されている。冒頭で述べたように、プログラム・コードは確かに人間がつくったものであるが、逆にプログラムによって人間なるものがつくられている。

そうした状況の中、個人、あるいは市民としてのエンドユーザーが、自らの手でプログラムを書き、それを実行することに、一体どんな意味が残っているのだろうか。本稿で取り上げたいくつかの「スペシフィック・コード」は、

- 極めて短いミニマルなコード
- 正しさよりも大切なものがあるコード
- アルゴリズム以外の部分も重要なコード
- 環境や文脈に依存するコード
- 実行する必要のないコード
- 話し言葉のように生成され消滅するコード

といった特徴のいくつかを持つ。それはいずれも、IT/SNS 企業やエリートハッカーのように超越的に見える何者かが提供してくれる、使いやすく、わかりやすく、役に立つものとは違うかもしれない。しかしスペシフィック・コードは、自分でつくり実行するものであり、変更できるものであり、他の人たちと共有できるものでもある。その意味で、スペシフィック・コードは、究極のエンドユーザー・プログラミングであり、今日の資本主義と監視社会の中で、個人が自由に息をするための、通気口のひとつにもなる。だから僕自身、個人が生き延びるための通気口としてのスペシフィック・コードを書きながら、そのことについて、もうしばらく考えていきたいと思っている。



P-Code Magazine 001

発行日 : 2019 年 11 月 23 日
発行人・企画・編集 : HAUS++（久保田晃弘 / 竹田大純 / 林洋介 / 稲福孝信）
価格：¥500（税込）

https://h4us.jp
i@h4us.jp