第2章　プログラム :: コード

2.1　エゾテリスム (Esoterism/Esotericism)

難解プログラミング言語 (Esoteric programming language, esolong)[1] と呼ばれるプログラミング言語がある。C++や Java、Perl や Ruby などの実用プログラミング言語とは違って、読みやすさやわかりやすさなどの通常必要とされる機能や目的とは異なる価値によってデザインされたプログラミング言語である。

難解プログラミング言語としては、スペース、タブ、改行という空白に相当する文字だけで記述される見えない言語 Whitespace[2]や、`< > + - . , [ ]` のわずか 8 つの命令だけからなる Brainf*ck[3] といったミニマルな言語が有名だが、それ以外にも Brainf*ck の派生型である A[4]や Ook![5]、Chef[6]のようにプログラムが料理のレシピのように見える言語、あるいは `W, w, v` の 3 つの文字だけしか使わないことでプログラムが草原のように見える grass 言語[7]、さらにはダンテの神曲地獄篇における地獄の第 8 圏にちなんで名付けられ、人類が設計しうる最も邪悪な言語と呼ばれる Malebolge[8] などまで、実に様々のものが考案されている。

Whitespace による Hello World プログラム[9](S:スペース、T:タブ)

```
SSSTSSTSSS
T
SSSSSTTSSTST
T
SSSSSTTSTTSS
T
SSSSSTTSTTSS
T
SSSSSTTSTTTT
T
SSSSSTSTTSS
```

[1] http://esolangs.org/
[2] http://compsoc.dur.ac.uk/whitespace/
[3] http://www.muppetlabs.com/ breadbox/bf/
[4] http://www.sksk.info/a.html A の文字しか使わない言語
[5] http://www.dangermouse.net/esoteric/ook.html オランウータンの鳴き声しか使わないオランウータン向け言語
[6] http://www.dangermouse.net/esoteric/chef.html
[7] http://www.blue.sky.or.jp/grass/
[8] http://en.wikipedia.org/wiki/Malbolge 最初のプログラムが書かれるまでに 2 年を要し、しかもそれは人間ではなく LISP プログラムによって書かれたものだった。
[9] 画面上に "Hello, World!" を表示するだけのプログラム。多くのプログラミング言語の入門書が、このプログラムを最初の例題としていることから「世界一有名なプログラム」と呼ばれている。

```
T
SSSSSTSSSSS
T
SSSSSTTTSTTT
T
SSSSSTTSTTTT
T
SSSSSTTTSSTS
T
SSSSSTTSTTSS
T
SSSSSTTSSTSS
T
SSSSSTSSSST
T
SSSSSTSTS
T
SS
```

実際にはこのプログラムはすべて空白なので、画面上には何も見えない。

Brainf*ck による Hello World!プログラム

```
+++++++++[>++++++++>++++++++++>+++++<<<-]>.>++.+++++++..+++.>-.
------------.<++++++++.--------.+++.------.--------.>+.
```

Ook!による Hello World!プログラム

```
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook. Ook?
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook. Ook.
Ook? Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook! Ook! Ook? Ook!
Ook. Ook? Ook. Ook. Ook. Ook. Ook! Ook. Ook. Ook? Ook. Ook.
Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook. Ook. Ook? Ook. Ook. Ook. Ook. Ook! Ook.
Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook.
Ook. Ook? Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook.
```

```
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook. Ook. Ook? Ook. Ook.
Ook! Ook. Ook. Ook? Ook! Ook.
```

Chef による Hello World!プログラム

```
Hello World Souffle.

Ingredients.
72 g haricot beans
101 eggs
108 g lard
111 cups oil
32 zucchinis
119 ml water
114 g red salmon
100 g dijon mustard
33 potatoes

Method.
Put potatoes into the mixing bowl.
Put dijon mustard into the mixing bowl.
Put lard into the mixing bowl.
Put red salmon into the mixing bowl.
Put oil into the mixing bowl.
Put water into the mixing bowl.
Put zucchinis into the mixing bowl.
Put oil into the mixing bowl.
Put lard into the mixing bowl.
Put lard into the mixing bowl.
Put eggs into the mixing bowl.
Put haricot beans into the mixing bowl.
Liquify contents of the mixing bowl.
Pour contents of the mixing bowl into the baking dish.

Serves 1.
```

grass による Hello World!プログラム

```
wWwwwWwwwwwWWWWvwvwvwwWWWWwWWWWwwwvwwWWwwWwwvwwWWWwWWWvwWWwWwvwWWwwwwwWwwvwwWWw
WWWwvWwvwwwWWWwwWwwWWWWwvWwwwWWWWwWWWWwwwWwwWWWWWwWWWWWWvwwWWwWWWWvWwwWWwwwwWW
Wwwwwwww WWWWWWWWWWwwwwwwwwwWwwWWwwwwWWWwwwwwwvwwwWWWwwWwwWWWWvWwwwwwwWWwwwwWWWww
```

```
wwWWWWwWWWWWWWWWWWWWWWWWWWwwwwwWwwWWwWWWwWWWWwvwWWWWWWWWWWWWWWWWWWWWWWWWWWWWWww
wwwwwwwWwwwwwwwwwWwwwwwwWwwwwwwwWwwwwwwwWwwwwwwwwwwwwwwwwwwwWwwwwwwwwwwwwwwwwwww
wwwwvwwvWWwWwwWwwwwwwWwwwwwwWwwwwwwwwwWwwwwwwwwwwwwwwwwwwWwwwwwwwwwwwwwwwwwvw
```

Malbolge による Hello World!プログラム

```
(=<`:9876Z4321UT.-Q+*)M'&%$H"!~}|Bzy?=|{z]KwZY44Eq0/{mlk**
hKs_dG5[m_BA{?-Y;;Vb'rR5431M}/.zHGwEDCBA@98\6543W10/.R,+O<
```

エゾテリスム (秘教、秘技的な思想) とは、ピュタゴラスやケプラー以来、宇宙の調和と響き合う音によって神の叡智と接しようとした音楽のように、目で見たり、触れて感じたりすることのできるフィジカルな現象の向こう側に、それらを超越した真理や本質を求める考え方である。それは、言い換えれば、世界を分析的にではなく、全体的、総合的に捉えることで、さまざまな学問分野を縦横に交錯させ、さまざまなパラダイムや文化を結合していこうとする試みであり、のちのロマン主義、象徴主義、シュルレアリスムの源流とも呼べるものであった [1]。

難解プログラミング言語は、知識や技術を難解かつクローズドにすることで、逆にその知識に到達する過程で内的な変成 (transmutation) を導こうとするエゾテリスムの思想を、今日のプログラミングの文化と繋ぐものである。ここでは、この難解プログラミング言語のひとつである、Befunge[10]を取りあげる。Befunge は通常のテキスト型のプログラミング言語と異なり、個々の命令が一次元の列ではなく二次元の格子上に並んでいて、実行する向きを上下左右に変更できる。 そのため Befunge の任意の 1 命令は、ソースコード中で 1 文字で表現され、ソースコード中の行と桁がプログラムの実行順序を規定する格子に対応している。

Befung の基本仕様

- プログラムは、格子の左上から右向きにの実行を開始する。
- 格子の上端が 0 行目、左端が 0 桁目である。
- プログラムの実行が端に達したら、反対側の端へジャンプする[11]。
- 整数を要素とする 1 つのスタックがある。このスタックを使って、命令に引数を渡したり、命令の結果を受け取ったりすることができる。
- 93 年に Chris Pressey によって作られた Befunge の最初のバージョンの Befunge-93 では、プログラム空間は 80 桁 ×25 行=2000 文字の 2 次元テキスト空間に限られている。

[10] http://en.wikipedia.org/wiki/Befunge

[11] つまり Befunge のプログラム空間は平面ではなくトーラスである

Befung の命令

制御

文字	意味
`<`	実行の向きを左にする
`>`	実行の向きを右にする
`^`	実行の向きを上にする
`v`	実行の向きを下にする
`_`	スタックをポップして、その値が 0 ならば実行の向きを右に、そうでなければ左にする
`|`	スタックをポップして、その値が 0 ならば実行の向きを下に、そうでなければ上にする
`?`	実行の向きを上下左右のいずれかにランダムで変更する
(空白)	何もしない
`#`	次の文字が表す命令を実行しない
`@`	プログラムの実行を停止する

リテラル

文字	意味
`0-9`	数値をスタックにプッシュする
`"`	次に"が出現するまで、文字の ASCII コードをスタックにプッシュする

入出力

文字	意味
`&`	ユーザに数値を入力させ、その値をスタックにプッシュする
`~`	ユーザに 1 文字入力させ、その文字の ASCII コードをスタックにプッシュする
`.`	スタックをポップして、その値を十進表示し、続けて半角スペースを出力する
`,`	スタックをポップして、その値を ASCII コードに持つ文字を表示する

演算

文字	意味
`+`	スタックから y, x をポップして、$x + y$ の値をプッシュする
`-`	スタックから y, x をポップして、$x - y$ の値をプッシュする
`*`	スタックから y, x をポップして、$x \times y$ の値をプッシュする
`/`	スタックから y, x をポップして、x/y の値をプッシュする
`%`	スタックから y, x をポップして、x/y の余りをプッシュする
`` ` ``	スタックから y, x をポップして、$x > y$ ならば 1 を、そうでなければ 0 をプッシュする
`!`	スタックをポップして、その値が 0 ならば 1 を、そうでなければ 0 をプッシュする

スタック操作

文字	意味
:	スタックをポップして、その値を 2 回プッシュする
\	スタックから y, x をポップして y をプッシュし、その後 x をプッシュする
$	スタックをポップして、その値を使用しない

メモリ操作

文字	意味
g	y, x をポップして、y 行 x 桁目の文字の ASCII コードをスタックにプッシュする
p	y, x, v をポップして、y 行 x 桁目を v の ASCII コードに持つ文字に書き換える

Befunge で Hello World のプログラムを書くと、以下のようにいろいろな書き方ができる。

プログラム 1

```
v @_       v
>0"!dlroW"v
v  :#     <
>" ,olleH" v
   ^       <
```

プログラム 2

```
"!dlrow olleH">v
               ,:
               ^_@
```

プログラム 3

```
<v"Hello world"0
 <,_@#:
```

プログラム 4

```
                  v
>v"Hello world!"0<
,:
^_25*,@
```

プログラム 5

```
"!dlroW olleH">,#$:_@
```

プログラム 6

```
v Hello World in Befunge
>"dlroW olleH",,,,,,,,,,,@
```

プログラム 7

```
v v"Hello, world!!"<
>                  ^
> >:#v_@
^   .<
```

プログラム 1 の場合、プログラムの出発点が左上なので、まずそこから下に 1 行下って右に曲る。スタックに「0」を積んでから、「World!」を逆順に積む。右端からまた 1 つ下の行に移り、「#」によって「:」を飛び越えてから、そのまた 1 つ下の行で、スタックに「Hello, 」を積む。今度は下から上に向って「,」によってスタックの文字を表示。次の文字を「:」でコピー (複製) してから、「!」によって、コピーしたスタックの値が「0」ならば「1」を、そうでなければ「0」をスタックに積む。一番上の行の「_」によって、その値が「0」であれば右に曲り、そこからぐるっと大きく回って再び一番下の行から「,」へとループする。スタックに最初に積んだ「0」が現われるまでこのループを繰り返し、0 が現われたら 1 行目の「_」で左折して「@」でプログラムが終了する。

一見シンプルなプログラムだが、「#」命令によって、プログラムの経路を立体交差させていると同時に、表示テキスト用の「,」と「!」を命令としても用いることで、1 つの文字に複数の意味 (機能) を与えている。こうした直感的な理解を拒むトリッキーな実行に潜む奥義性や内奥性こそが、難解プログラミング言語が難解 (エソテリック) たる、まさにその所以である。

2.2 算法詩 (Algorithmic Poetry)

具体詩 (Concrete Poetry)、あるいはより広く視覚詩 (Visual Poetry) と呼ばれる表現がある。通常の詩が、文字の意味や音を基本としてつくられているのに対して、具体詩では、文字や言葉の意味よりもむしろ、文字の音やかたち、配置といった言葉の「物質性」に根差した視覚的表現としての側面が重視される。

具体詩はフランスの詩人ステファヌ・マラルメ最晩年の『骰子一擲 (さいいってき)』(1897) をその起源とし、ギヨーム・アポリネールの『雨が降る』(1916) などのカリグラムや、ことばのかたちと意味内容を切り離した未来派やダダの活動を経て、現代的な具体詩は、スイスの詩人、オイゲン・ゴムリンガーの 1950 年代の作品 (『アヴェニダス』や『シレンチオ』) や、アロルド・デ・カンポス、デシオ・ビニャタリといたブラジルの作家グループ「ノイガンドレス派」の活動に始まるとされている。日本においては、北園克衛と新国誠一が、具体詩の創始者として名高い。

今日では、話し言葉がもつ身ぶり手ぶりや声の抑揚、手書き文字がもつ字形のゆれや乱れが込められない、電子メイルなどのデジタル・テキストによるコミュニケーションで、フェイスマーク (スマイリー) と呼ばれる文字の組み合わせによる絵記号を、コミュニケーションの補助に使ったり、同様の手法でさらに大きなイメージを表現した、アスキーアートが広く用いられている[12]。

そこでこの具体詩〜アスキーアートに倣いながら、文字の配列が意味を持つプログラミング言語 Befunge を使って、アポリネールや新国誠一が取り上げた「雨」をテーマに、実行することができる詩としての「算法詩 (Algorithmic Poetry)」の実験を行なってみる。

算法詩 1

```
v
```

```
                                        >>>>>>>>>>v<
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv"Rain."<vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
????????????????????????????????????????<>,#$:_@ >????????????????????????????????????
```

この算法詩 1 を実行すると、プログラムの左上からまず下に向ってプログラムの実行場所 (プログラムカウンタ) が、雨のように落下していく。プログラムカウンタが一番下の行に達すると「?」命令によってそれが左右にランダムにシフトし、再び一番上の行から落下を繰り返す。何度かプログラムカウンタが落下を移動を繰り返す内に、それが中央の「Rain.」と書かれた周辺に落ちると、「Rain.」という文字を出力して、プログラムが終了する。

[12]アスキー・アートには、フェイスマークのように数文字しか使わない小さなものから、千文字を越える大規模なものまで、さまざまなものがある。

この算法詩 1 の中にいくつか「?」を散布すると、雨 (プログラムカウンタ) がこの命令に当たると、任意の向きに雨の方向が変化する (算法詩 2)。

算法詩 2

```
v
                                ?
                                                   ?
                                                              ?
                                                                       ?
      ?
                                     ?
                                                  ?
                                                                          ?
  ?
                                               ?
                    ?
     ?
                                                                        ?
              ?
                                                                             ?
                                        ?
                                                                            ?
?
   ?
                                                                  ?
                                                                   ?
                                    >>>>>>>>>>v<
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv"Rain."<vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
?????????????????????????????????????<>,#$:_@ >?????????????????????????????????
```

さらに「?」命令を稠密に敷きつめると、雨はもはや下に落下するものではなく、ブラウン運動のように四方に不規則に動き回る (算法詩 3)。

算法詩 3

```
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
```

```
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????^^^^^^^^^???????????????????????????????????
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^v"Rain."0<^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                                    <,_@#:
```

算法詩 4

```
 v    <                              <
>?" ",^
>?".",^
>?"R",25*,v
 ^
          v    <
         >?" ",^
         >?".",^
         >?"a",25*,v
          ^
                   v    <
                  >?" ",^
                  >?".",^
                  >?"i",25*,v
                   ^
                            v    <
                           >?" ",^
                           >?".",^
                           >?"n",25*,^
                            ^
```

この算法詩 4 を実行すると、以下のように、空白とピリオド、そして「Rain」の各文字を順に生成し続ける。この文字 (テキスト) を、生成詩 (Generative Poetry) と呼ぶことも可能だろう。

算法詩 4 から生み出される生成詩 (の例)

```
.       .  .  . . ..  ..  .  .  .   .R
 a
   ....i
   .     .  .n
  R
 .  .  .  . ..  a
.  . .. i
.  n
.. ...        .   . R
  . ..  .     .    a
 ... .        .. i
        .     .   n
        .   .R
  .  ...    .   ... .   a
i
 . n
  .     R
  a
....   . .. i
  . .     ..  n
```

また、以下の算法詩 5 は、実行すると詩 (プログラム) のテキスト空間内に、「Rain.」の文字の断片を書き換え続ける。

算法詩 5(実行前)

```
vp                    <
 v          v      v
>?48*      >?38*  >?2^
>?85*6+    >?38*1+>?3^
>?99*1+    >?38*2+>?4^
>?99*44*+  >?38*3+>?5^
>?99*46*+  >?38*4+>?6^
>?99*46*+5+>?38*5+>?7^
 ^          ^      ^
```

算法詩 5(実行中)

```
vp                    <
 v          v      v
```

```
>?48*      >?38*  >?2^   R
>?85*6+    >?38*1+>?3^      R
>?99*1+    >?38*2+>?4^  .  Ra
>?99*44*+  >?38*3+>?5^    Ri.
>?99*46*+  >?38*4+>?6^    RR.a
>?99*46*+5+>?38*5+>?7^   .R.in
 ^          ^      ^
```

同様に、この算法詩 6 は、実行することによって詩の中の「Rain...」というテキストを確率的に自己解体していく。

算法詩 6(実行前)

```
>        vp          <
         v  v   v  v
RRRRRRRR>?6>?8g>?2>?2>?<
aaaaaaaa>?4>?2g>?0>?5>?<
iiiiiiii>?1>?5g>?6>?6>?<
nnnnnnnn>?7>?3g>?5>?9>?<
........>?0>?9g>?3>?8>?<
........>?3>?4g>?1>?4>?<
........>?5>?7g>?7>?3>?<
        >?2>?6g>?4>?7>?<
         ^  ^   ^  ^
         ^p          <
```

算法詩 6(実行中)

```
>        vp          <
         v  v   v  v
nRnnRR.R>?6>?8g>?2>?2>?<
ana...n.>?4>?2g>?0>?5>?<
.a...Rn.>?1>?5g>?6>?6>?<
..nnna.n>?7>?3g>?5>?9>?<
..n..nn.>?0>?9g>?3>?8>?<
.......i>?3>?4g>?1>?4>?<
.nn..nn.>?5>?7g>?7>?3>?<
R.....n.>?2>?6g>?4>?7>?<
         ^  ^   ^  ^
         ^p          <
```

13

2.3 実行可能な絵画 (Executable Picture)

アスキーアートが、文字 (キャラクター) をピクセルとしたビットマップであると見なすことができることから、算法詩と同様のアプローチで、プログラムとしてのビットマップ—すなわち実行できる絵画を考えることも可能である。

最もシンプルな方法としては、Befunge の各命令を、例えば<、>、^、v のそれぞれを、赤、青、緑、黄の各色に対応させるなど、単純に色と一対一対応させることでプログラムを画像に変換することができる。すると、例えば算法詩 1 は以下のような画像として表現することができる。

図 2.1: 算法詩 1 に相当するビットマップ画像

文字の色への単なる置き換えではなく、より本格的な、実行可能な絵画としてのビットマップ・プログラミング言語のひとつに Piet がある[14]。Piet という名前は、20 世紀初頭の抽象画家ピエト・モンドリアンにちなんで名づけられた。

この言語も Befunge と同じように、実行する方向を上下左右に変化させることができる、スタックを計算領域として用いる 2 次元プログラミング言語である。コードの実行は Befunge 同様左上から右向きに開始され、ビットマップの各ピクセルが命令に相当する。そのため、Piet においては、このコードとしてのピクセル (pixel) をコーデル (codel) と呼ぶ。同じ色のコーデルの数が数値に、実行するコーデル間の色相と明度の相対的な変化によって、以下の 17 の命令が表わされる。

push 値をスタックに積む

13

[14]PIet は、前述の Ook!や Chef などいくつかの難解プログラミング言語を設計した David Morgan-Mar によってつくられた。`http://www.dangermouse.net/esoteric/piet.html`

pop 値をスタックから取り出す

add 値を 2 つスタックから取り出し、加算した結果をスタックに積む

subtract 値を 2 つスタックから取り出し、2 つ目の値から 1 つ目の値を減算した結果をスタックに積む

multiply 値を 2 つスタックから取り出し、乗算した結果をスタックに積む

divide 値を 2 つスタックから取り出し、2 つ目の値を 1 つ目の値で除算した結果をスタックに積む

mod 値を 2 つスタックから取り出し、2 つ目の値を 1 つ目の値で除算した際の余りをスタックに積む

not スタックの一番上の値が 0 でなければ 0、0 であれば 1 に変更する。

greater 値を 2 つスタックから取り出し、2 つ目の値が 1 つ目の値より大きければ 1、そうでなければ 0 をスタックに積む

switch 値をスタックから取り出し、その数だけプログラムの進行方向 (DP：ディレクション・ポインタ) を時計回りに 90 度回転する

pointer 値をスタックから取り出し、その数だけプログラムのコーデルからの出口の方向 (CC:：コーデル・チューザー) を変化させる。

duplicate スタックの 1 番上の値を複製する

roll 値を 2 つスタックから取り出し、スタックの 2 つ目の値の深さに 1 つ目の値の数を埋め込む

in 標準入力から値 (数値あるいは文字) を取得する

out 標準出力へ値 (数値あるいは文字) を出力する

さて、さっそくこの Piet を用いて、先ほどと同じように「Rain. 」という文字を繰り返し表示し続ける算法絵画詩を書いてみる。左上のブルー (#0000FF) から実行が開始され、ブルー、シアン (#00FFFF)、グリーン (#00FF00)、イエロー (#FFFF00)、レッド (#FF0000)、マゼンタ (#FF00FF) の各色のコーデル数が、R(82) a(97) i(105) n(110) .(46) space(32) の各アスキーコードに対応している。

自然言語と同じように、プログラミング言語においても、同じ表現を生成する複数のバージョンが可能である。Piet においては、ブラック (#000000) と画像端は、CC と DP を変化させる命令であり、ホワイト (#FFFFFF) は何も行わない (nop) 命令なので、これらを用いて名前通りモンドリアン風の算法絵画詩を制作することも可能である。

これらの算法絵画は実寸だと、それぞれ 40×40、36×36 ピクセルの画像ファイルになる。そこで、これらを実行可能な算法アイコンと呼ぶこともできるだろう。それは 1 次元のバーコードや 2 次元マトリックス型の QR コードのような単なるデジタル・データとは異なる。同様に、例えば以下のように「A」というフォントを (繰り返し) 出力するアスキーアート=メタ・ビットマップ・フォントとしての Befunge プログラムを考えることもできる。

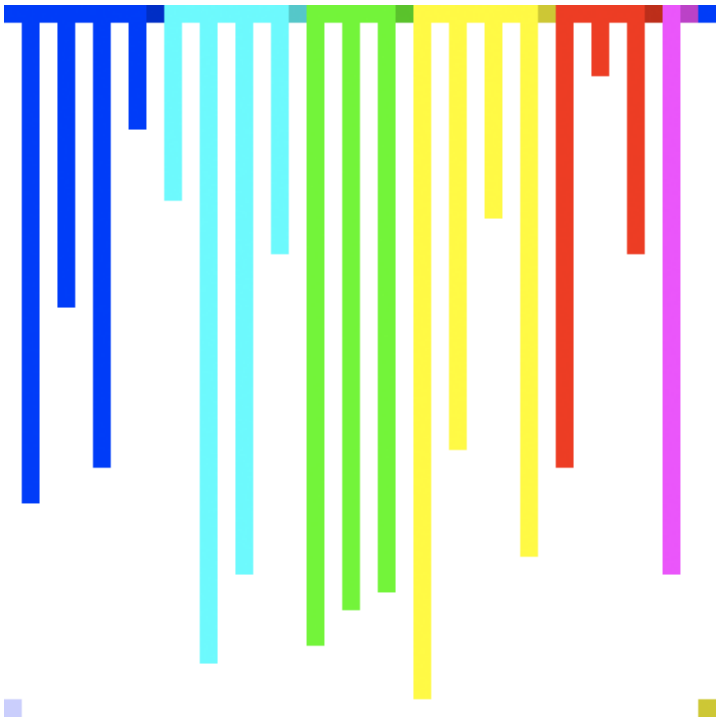

図 2.2: Rain. を繰り返し生成する算法絵画詩

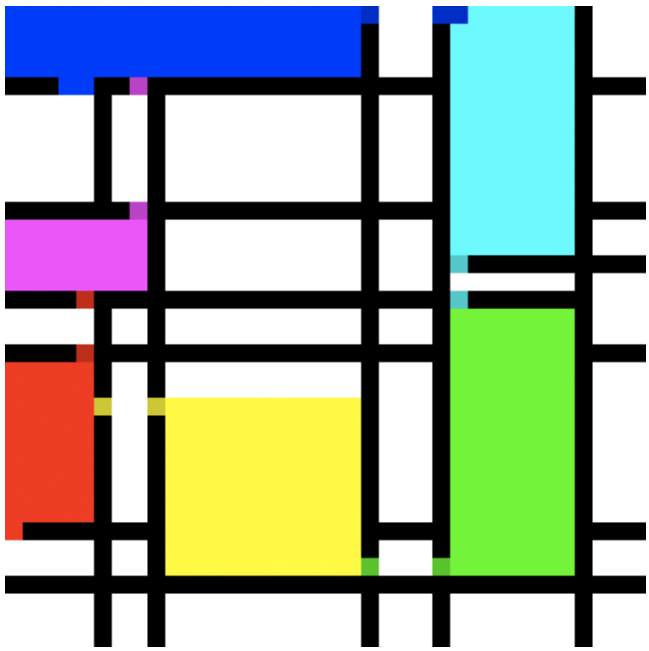

図 2.3: Rain. を繰り返し生成する算法絵画詩 (モンドリアン・バージョン)

A を出力し続ける A (Befunge による 5x7 算法フォント)

```
  > > v
2       *
*       7
^       <
> ^ A > v
,       3
+       3
```

実行結果

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAA…(繰り返す)
```

2.4 自己複製芸術

クワイン[15]とは、実行すると自分自身 (ソースコード) を出力するプログラム、すなわち自己複製を行うプログラムのことである。前述の Hello World プログラム同様に、プログラミングの世界では良く知られている (良く書かれる) プログラムのひとつである[16]。

例えば、以下のプログラムは、Befunge で書かれた、わずか 14 文字のクワインプログラムである。スタックの一番底に積まれている数値がインデックスとなって、g でそれを読み出し、, で書き出す。`:93+‘#@_1+`の部分は繰り返しとインデックスのインクリメントの処理部分である。

自分自身を出力する Befunge プログラム

```
:0g,:93+‘#@_1+
```

実行結果

```
:0g,:93+‘#@_1+
```

同様に、出力先をプログラムと同じ空間にすることもできる。

自分自身をメモリ上にコピーする Befunge プログラム

```
::0g\1p:96+‘#@_1+
```

[15] クワインという呼び名は、自己参照についての研究を行なったアメリカの哲学者・論理学者のウィラード・ヴァン・オーマン・クワインにちなんでいる。

[16] 前述の「A を出力する A」はクワインではないが「自分自身と同じ意味を持つものを出力する」という観点から、このクワイン的な側面を持っているといえる。

実行結果

```
::0g\1p:96+‘#@_1+
::0g\1p:96+‘#@_1+
```

さらにもう少し工夫すれば、コピーした部分を次に実行することで、次々と自己を増殖させていくプログラムを書くこともできる。

自己増殖するプログラム

```
>011p013p>11g13gg:84*‘#v_84*>11g13g4+p$v
#     13p^p11          <>    ^         >
          v                              <
#     13pv>11g1+:85*-#^_011p13g1+:4%#^_
```

実行結果

```
>011p013p>11g13gg:84*‘#v_84*>11g13g4+p$v
#     13p^p11          <>    ^         >
          v                              <
#     13pv>11g1+:85*-#^_011p13g1+:4%#^_
>011p013p>11g13gg:84*‘#v_84*>11g13g4+p$v
#     13p^p11          <>    ^         >
          v                              <
#     13pv>11g1+:85*-#^_011p13g1+:4%#^_
>011p013p>11g13gg:84*‘#v_84*>11g13g4+p$v
#     13p^p11          <>    ^         >
          v                              <
#     13pv>11g1+:85*    …(繰り返す)
```

1950 年代にイギリスで生まれ、その後アメリカのアンディー・ウォーホールやロイ・リキテンシュタインによって発展した「誰でもが知っているイメージを、誰でもが可能な技法で複製する」ポップ・アートや、オリジナルとコピーの違いを消失させるアプロプリエーション (盗用) によるシミュレーショニズムのように、20 世紀後半の美術にとってメディアの複製や編集は、いわば作品を成立させるための前提であり、そこから多くの「複製芸術」が生れてきた。

一般に、任意のプログラミング言語で、自己複製プログラムとしてのクワインを書くには、プログラムを

1. プログラムコードをデータとして記述する部分
2. データとしてのプログラムコードの出力を行う部分

の 2 つに分けて考える。しかしながら、先ほどの Befunge のように、プログラムとデータが同じ方法で表現されている場合、プログラムがそのプログラム自体を、つまり自分自身を参照したり変更したりすることができる。Befunge の場合、g や p という命令が、それに相当する。すると、何もわざわざ上記の「プログラムコードをデータとして記述する部分」を別枠で用意する必要がなくなる。なぜなら、プログラムがプログラム自身を参照すれば良いからだ[17]。前出の「自分自身を出力する Befunge プログラム」もそのように書かれている。何かがそれ自身を参照することを「自己参照」あるいは「自己言及」と呼ぶ。

複製することは、本来 2 つのレベル (階層) を必要とする。複製するものは、複製されるものよりも上位の階層にいる。コピーするものは、コピーされるものよりも上位の階層にいて、コピーマシンが自分をコピーすることができないように、コピーするものがコピーされることはない。

しかし、自己言及によって、コピーするもの (プログラム) とコピーされるもの (データ) が同じレベルにある場合、コピーするものと、コピーされるものが同一になる。つまり、自己複製によって生まれたプログラムを、今度はそれを実行することで、自己複製を反復的、連鎖的にフィードバックさせることができる。すると、有限な自己複製プログラムが、無限の複製プロセスとしての「自己増殖」になる。

芸術は、それが常に作家の自画像的な性質を持つ、という意味で自己言及的である。自己言及は絵画でいえば、「自画像」を描くことに相当するといえるだろう。自己増殖するプログラムは、無限に自画像を描き続けるプログラムである。それは、ハウリングやディレイループ、ビデオ・フィードバックのように、自己相似の構造をつくり出したり、あるいはその反復過程にゆらぎやノイズ、あるいはエフェクトを挿入することで、さまざまなバリエーションを生成することができる。

17