好々爺の部屋 (4) — TECO

上田 和紀*

竹内郁雄先生の巧みな執筆勧誘電話にのって，好々爺（じじい）の仲間入りをすることになった．与えられたお題は TECO というテキストエディタである．

このお題が私に与えられたのは，今を去ること 20 年前，大学院生だったころに TECO を実装して公開していたからに違いない（しかし，そのことを記憶していらした竹内先生の老人力には感服である）．当時は，テキストエディタの設計が，プログラム好きの間での大変ホットな話題であった．プログラミング・シンポジウムなどで集まるたびにエディタ談義に花を咲かせたものである．

今も昔も，本物のプログラマは，端末に向かっている時間の大半をエディタとともにすごす（今なら大半は WWW ブラウザだ，という人は，おそらく本物のプログラマではない）．だがそのころの典型的なプログラム開発環境は，メインフレーム上の（むろん非 Unix の）TSS，通信速度 300bps，タイプライタ端末，というものであった．だから，与えられたハードウェア環境でいかに快適に仕事をするかを考えるとき，エディタの機能やユーザインタフェースは，プログラマにとって大変な重大事であった．

さて，TECO とは，Text Editor and COrrector の略である．テキストエディタだという説と，テキストエディタ記述言語だという説とがあるが，TECO をそのまま利用する人もいたし，Emacs の初期の版は Guy Steele の発案がきっかけで，Richard Stallman が TECO で書いたものだという事実もあるから，両方正しいと言える．

現在では，TECO が Emacs の初期の記述言語として使われたことは十分に記録されているし，よく知られている．しかし，TECO 自身の技術的側面については

- 文字ベース（⟷ 行ベース）のテキストエディタである（そんな小さな単位を相手にして使い易いのか？）
- やたら多くのコマンドを擁する（覚えきれるのか？）
- 強力なマクロ機能による拡張可能性が売りだが，人の書いたマクロはほとんど読めない（それじゃあスクリプト言語設計の参考にもならないのでは？）

といった，要するに「複雑怪奇」だという評判しか伝わってこない．これでは，1970 年代前期の主要なテキストエディタであり，その名前が「TECO で編集する」という意味の他動詞にまでなっていた [5] TECO の機能，面白さ，良さが忘れ去られてしまう．また，TECO 以上に複雑なソフトウェアツールは，その後いくらでも生まれてきているから，複雑怪奇という評価さえも，もはや撤回されなければならなくなってきているような気がする．そこで好々爺の部屋に登場とあいなるわけである．

TECO はマクロによる拡張可能性を売りにしていたが，それだけではなくて TECO 自身の仕様もさまざまな人がさまざまに拡張していった．初期の Multics TECO [11] は簡潔だったが，「最後の仕様書」と言われる 1985 年版の Standard TECO の中年太りはかなりのものである．この Standard TECO を本家とすると，著者が実装した TECO は，Multics TECO をベースにした分家である．その説明書 [8] は，エラーメッセージ一覧や索引まで入れて 35 ページしかない．もちろん，スクリーン端末のためのリアルタイム編集モード [3] もない．本稿は [8] に基づいて紹介するが，TECO の基本概念や基本コマンドは，どの TECO でも共通である．

*早稲田大学理工学部情報学科．最終版は **bit**, Vol. 33, No. 2 (2001 年 2 月号), pp. 48–54 所収.

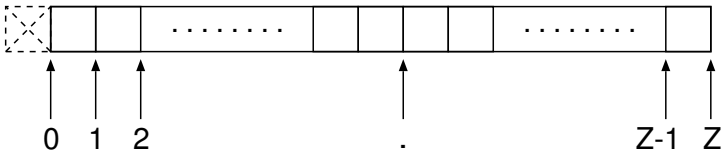

図 1: テキストと文字間隔．“.” は現在の注目点．先頭には，そこが行頭であることを示すために，仮想的な改行文字（破線）が常に置かれている．

TECO の基本

TECO の基本概念は，実はとても簡単である[1]．データ構造としては

- 編集中のテキストを格納するテキストバッファが 1 個
- 文字列や数値を格納できる Q レジスタ（それぞれ半角 1 文字によって識別される）が，図形文字の種類の数だけ
- Q レジスタのスタックが 1 本

これだけである．テキストバッファで特徴的なのは,「現在の注目位置」を示すポインタが，文字ではなくて文字と文字の間（またはバッファの両端）を指すことで（図 1），これにはいたく感心した覚えがある．特定の 1 文字を指定するときは「ポインタの右の字」とか「左の字」とか言わなければならないが，文字列の挿入位置や，バッファ上の範囲—つまりは文字の列—を指定するには，ポインタが文字間を指している方がはるかに自然なのである．改行文字がバッファ上で単なる文字として扱われ，自由に挿入削除できることも，メインフレーム文化で育った人には新鮮であった．

多い多いと言われるコマンドも，枝葉を切れば

- ファイル入出力
- バッファの内容の表示
- バッファ上のポインタの移動
- テキストの消去と挿入
- テキストの探索と置換
- Q レジスタへの情報の格納と取出し
- 制御構造 (反復，条件判断，goto)
- Q レジスタ内のコマンドの実行（マクロ実行）

に分類できる．Emacs のコマンドを覚えるのとは違って，十分覚えられる分量である．何しろすべてのコマンドは，半角[2]1 文字か，少数のプレフィクス文字を 1 文字目とする 2 文字構成であったから，たかが知れているのである．

コマンドは，前に 0〜2 個の整数引数を，後ろには Q レジスタ名と文字列引数を配することができる．つまり [] を省略可，[] ... を 0 回以上の繰返しとすると，

$$\boxed{[m\,[,n]\,]\,X\,[q]\,[s]\ldots}$$

という形をしている．m と n は整数式でもよくて，下のものもその構成要素として使える．

Z	バッファ内の文字数
.	ポインタの現在値 (0 ≤ . ≤ Z)
Qq	q は Q レジスタ名，Q レジスタ q の整数値または Q レジスタ q の中の文字列の長さ
%q	q は Q レジスタ名，Q レジスタ q の整数値を 1 増やし，その値を返す
^c	文字 c のコード

文字列引数 s は，文字列リテラルまたは Qq'（Q レジスタ q' 内の文字列を指定）である．文字列リテラルは，筆者のシステムでは同一文字で囲まれた文字列を採用していた．ESC で終わる文字列を採用していた TECO も多い．

[1] 言語やシステムの「単純さ」に関する世の中の基準は，この十数年で大幅に動いたようだ．だから，今となってはこう言い切ることができる．

[2] ASCII と書こうとしたのだが，筆者の TECO が稼働していた日立 Hitac シリーズの OS のコード系は EBCDIK であった．小文字が使える拡張 EBCDIK になったのは 1980 年ごろである．

たとえば,「現在の行の第 6 カラムを $ にする」には,

```
0L6CDI/$/
```

というおまじないを打って，最後に空白＋リターン（本家の TECO は，ESC 2 個）を打つのであった．

TECO のコマンドはマイクロ命令のようなものなので，タイプライタ端末環境では，いくつかのコマンドをまとめて送信して実行させるのが TECO 流である．さらに文字列引数には改行文字も入れることができるので，リターンだけではコマンド実行が始まらない．本家の TECO は ESC 2 個をコマンド列の終りとしていたが，半二重通信の環境ではこれを採用するわけにもいかない．そこでいろいろ考えたあげく，空白＋リターンという「ふだんあまり使わない」シーケンスをコマンド列の終りとして起用したのである．これはものすごく打ちやすいシーケンスで，本家の TECO よりずっと使いやすかった．

さて，上のおまじないはこう読む．

0L	(Line) . を行の先頭へ
6C	(Character) . を 6 字右へ
D	(Delete) . の右の 1 字を消去
I/$/	(Insert) . の左に $ を挿入

Emacs で同じことをする場合と比べて，打鍵数も大差ない．

つぎに「現在の行を 5 行後ろに移動する」おまじないは,

```
0LX0K5LG0
```

である．

0L	. を行の先頭へ
X0	(eXtract) . から行末までを Q レジスタ 0 にコピー
K	(Kill) . から行末までを消去
5L	. を，5 個めの改行文字の右へ
G0	(Get) . の左に Q レジスタ 0 の内容を挿入

大事なことは，これらのおまじないはプログラムでもあるということである．プログラムの特徴と醍醐味[3]は，単純な命令を組み合わせて，いくらでも複雑な機能を実現できるところにある．TECO は，この楽しさを，テキスト編集という日常的な場面でたっぷり味わわせてくれる．

もっと大事なことは，もうちょっとの拡張で，ふつうのテキストエディタでは簡単にはできないことが可能になる点である．たとえば,「. のいる行から始まる 10 行の第 6 カラムを $ にする」は,

```
0L10<6CDI/$/L>
```

である．10< ... > は，... を 10 回繰り返す．Unix の grep に対応する「文字列 PROG の存在する行をすべて表示する」は

```
J<:S/PROG/;V>
```

となる．

J	(Jump) . をバッファの先頭へ
:S/PROG/	(Search) . から右向きに文字列 PROG を探し，. をその右側に置く．探索に成功したら −1 を，失敗したら 0 を返す
;	(exit) 値引数が非負ならば最内ループから脱出
V	(View) . のある行を表示

これを「手続き抽象」のために一般化すると

```
&0J<:SQ0;V>
```

となる．&q は，マクロの文字列引数を Q レジスタ q に格納するコマンドである．この文字列を :IG/&0J<:SQ0;V>/ などとして Q レジスタ G に格納し（:Iq は文字列引数を Q レジスタ q に入れる），

```
MG/PROCEDURE/
```

[3]数年前の早朝，“醍醐味” が “粗大ゴミ” の cdr であることを布団の中で発見して，ガバッと起きてしまった．その話を某メーリングリストに流したときに「すばらしき発見」とほめてくれたのは竹内先生である．NTT 研究所とさきがけ 21 で，怪しくも楽しいプログラミング言語の研究に没頭している原田康徳氏は,「その醍醐という食品，京都の醍醐寺で売っていましたよ」と教えてくれた．これを手がかりに調べたところ，現在では醍醐寺の売店では扱っておらず，製造元の「きょうらく」から通信販売で入手できる．牛乳の水分を蒸発させて乳脂肪分だけの固形としたもので，インドのギー（ghee）に近い．全国の高級料亭で需要があるそうだが，残念ながら著者は，まだ醍醐を味わったことがない．

を実行すれば，“PROCEDURE” の探索ができる．

「行末の改行を除いて 72 文字を越える行を，. のある行の次から探しはじめ，最初にみつかった行を表示する」は，Emacs ではなかなか容易でない．TECO だと

```
L<.U.L.-Q.-74;>-LT
```

となる．

L	次行の先頭へ
.U.	(Update) 現在の . の値を Q レジスタ . に格納
.-Q.-74;	. と Q. の差が 74 以上ならばループ脱出
-L	前行の先頭へ
T	(Type) . から行末までを表示

マクロは，再帰呼出しも可能である．たとえば

```
<C 0A,^)"E 1; ' 0A,^("E MP '>
```

を Q レジスタ P に入れて MP を実行すると，. が ‘(’ の右にあるときに，対応する ‘)’ の右まで . を移動させることができる．ここで

0A	(Ascii) . の左の文字のコード
0A,^)"E	(Equal) . の左の文字が ‘)’ ならば何もしない（つまり次のコマンドへ行く）．そうでなければ対応する ’ (fi) までスキップ

である．つまり，右括弧が出てきてループを脱出する（1;）まで . を 1 字ずつ右へ移動してゆく（C）が，途中で左括弧が出てきたら，再帰によって対応する右括弧の右まで移動するわけである．なお，コマンドとコマンドの間には空白列があってもよい．

TECO は，テキスト編集における大多数の作業について,「その作業を実現するために必要な入力文字数が少ない」という意味で dense なツールである．プログラミング言語として見ればいろいろ批判ができるとしても，dense であるということは，会話的なコマンド言語としての本質をついているものと思う．

実装の動機と使用経験

私が TECO に興味をもったころ，メインフレーム上のテキストエディタでプログラムを作成するには，コマンド体系の問題もさることながら，複雑なファイルシステムとも向き合わなければならなかった．ファイルを作るにはレコード形式，レコード長，ブロックサイズ，初期サイズ，増分などに気を配らなければならず，ソースプログラムファイルとデータファイルの形式が全く異なるものであった，という話を爺さんがして，どれだけの人に信じてもらえるだろうか．

これではいかんと，1970 年代終りには，東大大型計算機センターでも「当時最新鋭の行エディタ」であった QED（Unix の ed）のメインフレーム版が公開された．しかしこれを使っていると不思議なほど CPU 課金がかさんだ（そう，当時は課金制であった）．それでは，ということで，効率がよく，しかも QED よりはマニアックな TECO を作ることにしたのである．

記述言語は Pascal とアセンブラ．C 言語の出てくる前であるから，動的にファイルを作成したり，割込みハンドラを定義したりするにはアセンブラルーチンを書くほかなかった．積極的にアセンブラをつかった部分も一つある．それは，大量の文字列コピーを，ハードウェアが提供する「大型命令」を利用して行なうためである．TECO の標準の実装法は，. までの文字列をテキストバッファ用の大きな配列の左側に，. 以降の文字列を右側に詰め，真ん中をあけておくというものだった．こうするとテキストの挿入，削除，探索は安いが，. の移動は高い．だがそれは大型命令で高速化できる．本稿の執筆中に詳細な実装メモが出てきたのだが[4]，それに「(大型命令で）30 倍速くなった」と走り書きがある．

文字列探索は，前向きと後ろ向きそれぞれに対して長い（の基準は 6 文字以上）文字列用と短い文字列用を用意し，長い文字列の探索には,「当時最新鋭」の Boyer-Moore のアルゴリズムを bit で読んで採用した．Q レジスタ用領域は，参照カウント法とフリーリストを用いた可変長ブロック管理で，

[4] 20 年前のメモが出てくるべきときに出てくるところが爺くさいが，この機会に古文書保存の観点から PDF にした [9].

連続領域が足りなくなったら sliding compaction をしていた．全部で 2000 行 (Pascal 部分) ほどのシステムだが，このように勘所は押さえてあったから，あまり最適化を行なわない Pascal コンパイラでも，十分に高速であった．

TECO には思想はあっても標準仕様はないと思っていたから，例外処理ハンドラなども勝手に設計して入れた．バイトコードへのコンパイラを作ったらどうかというお勧めもあったが，結局作らなかった．コマンド列を解釈実行するたびに字句解析器が走りまわっていたわけだが，それで困るということはなかった．

TECO を作成公開して得た教訓は二つある．

1. 拡張可能言語を公開すると，それで大規模なものを書く人がきっと現れる．あるとき「日本語エディタ（TECO で数百行）を作ったのですが，落ちることがあるのです．診てもらえませんか」と，わざわざ研究室まで相談に来られた人がいたのだが，ユーザになってくれて嬉しいと思う一方で，困ったなぁと思わざるをえなかった．Emacs が TECO で書かれたのを知る前のことである．

   TECO らしいものとしては，戸村哲氏によるミニエディタの実装 [9] があった．この簡潔な行エディタは，TECO で 50 行ちょっとである．

2. 自分自身は，当然ほとんどの編集作業を TECO で行なうようになったが,「宵越しのマクロ」はごく少数しか持たなかった．必要なときにその場でさっとマクロを書き，使い終わったら未練を残さずに捨ててしまうのが粋な使い方だと心得ていた，というわけでもないが，実際のところ裸の TECO コマンドと「即席マクロ」で十分だったのである．

ちなみに，前述の実装メモを見ると，テキスト編集のほかに，

1. ファイルの中身を見る（見るだけ！）
2. プログラムの手続き一覧表やキーワードの統計等を作成する

ためにしばしば使った，とある．前者からは，当時の TSS の想像を絶するコマンドインタフェースをうかがうことができる．後者は，今ならば Unix のフィルタ，パイプ，リダイレクト機能か，もしくはスクリプト言語の領分だろうが，その役を当時は TECO が担っていた．

TECO の現在

TECO は重要な遺産であるから，Eric S. Raymond と John Kowan の主宰する Retrocomputing Museum [6] にもちゃんと所蔵されている．Pete Siemsen の TECO collection [7] は，さまざまな実装や情報の入手に有用である．Standard TECO の仕様書もここから手に入った．処理系は，Unix 上では Pete Siemsen による TECOC が代表的で，それを Tom Almy が Windows に移植した版 [1] も WWW から入手できる．Matt Fichtenbaum も C による実装を行なっていて，それをベースに Dale R. Worley が書いた Emacs Lisp による実装 teco.el もある．

著者自身のシステムは，最新版が 1/2 インチオープンリールテープに保存してあるのだが，簡単には読み出せない状況である．好々爺にとっては，古い媒体とそれを読むための周辺機器が，頭の痛い問題である．1/2 インチテープドライブはまだ製造されているのだろうか？ 使えるのだろうか？

ついでだが，Retrocomputing Museum には，いかにも好々爺の話の種になりそうなプログラム言語やハードウェアが集められている．また,「古き佳き」を通り越して「怪しげな」プログラミング言語に興味のある人には，たとえば Brian Connors の The Turing Tarpit というページ [2] がある．高等無形文化財．

マクロ二題

図 2 は，1981 年当時の東大のシステムで動かしたハノイの塔である．まずは `ER` (External Read) でマクロをバッファに読み込み，`HXH` で Q レジスタ `H` に格納し（最初の `H` は，`0,Z` つまりバッファ全体（wHole）を表す），`:TQH` でその内容を表示している．

```
>>TECO
TECO READY, V-02 L-33 *** SEE #Z0096.TECO.VDATA(NEWS) ***
@ER/#Z0096.TECO.VDATA(HANOI)/
@HXH :TQH
U9 &1&2&3 [N[A[B[C Q9UN :IAQ1 :IBQ2 :ICQ3
QN"G QN-1MHQAQCQB
     :T/MOVE / QN= :T/FROM / :TQA :T/ TO / :TQB=
     QN-1MHQCQBQA
  ' ]C]B]A]N
@3MH/SAPPORO//FUKUOKA//TOKYO/
MOVE 1 FROM SAPPORO TO FUKUOKA
MOVE 2 FROM SAPPORO TO TOKYO
MOVE 1 FROM FUKUOKA TO TOKYO
MOVE 3 FROM SAPPORO TO FUKUOKA
MOVE 1 FROM TOKYO TO SAPPORO
MOVE 2 FROM TOKYO TO FUKUOKA
MOVE 1 FROM SAPPORO TO FUKUOKA
@EQ
>>
```

図 2: ハノイの塔とその実行例（文献 [8] より）．たった 7 手なのに，出力結果の実施にはものすごい金と時間がかかる．

U9	マクロへの整数引数を Q レジスタ 9 に格納
&q	文字列引数を Q レジスタ q に格納
[q	Q レジスタ q の内容をスタックにプッシュ
:IqQq'	Q レジスタ q' の文字列を Q レジスタ q にコピー
Qq"G	Q レジスタ q の内容が正でなければ ’ までスキップ
:T/MOVE /	“MOVE ” と表示
]q	スタックの先頭の内容をポップして Q レジスタ q へ

図 3 は，円周率の計算マクロである．Spigot Algorithm [4] と呼ばれるもので，1 本の 1 次元配列を用意するだけで，多倍長計算ルーチンなしに動き，しかも上位の桁からインクリメンタルに表示してくれることが特徴である．Martin Ambuhl がネットニュース comp.lang.c に流したものがいくつかの WWW ページで引用されていたが，本稿ではそれを Windows 2000 上の TECOC [1] でデバッグしたものを掲載する．TECO には演算子の優先度の概念はなく，整数引数は左から右に評価されることに注意して解読してみてほしい．読破したら TECO 道中級を自称してよかろう．以下のコマンドの中で * 印は，[8] でなくて Standard TECO の仕様によるものである．

GZ	文字列引数をテキストバッファに挿入*
\	．の右側の数字列に対応する整数値として返す*
n\	数値 n の文字列表現を ．の左に挿入*
10@I//	改行文字を挿入*
n"E ... | ... ’	if $n = 0$ then ... else ...*
m,nT	区間 m, n の文字列を表示
m,nK	区間 m, n の文字列を削除

「TECO には Q レジスタが数十個あるだけなのに，配列はどうするの？」と言って頭をかかえてしまうのはもやしっ子．爺さんたちは,「テキストバッファがあるじゃないか」と言いながら，それを改行文字で区切って配列に仕立ててしまったのである．

表示桁数を n とするとこのアルゴリズムの空間計算量は $O(n)$，時間計算量は $O(n^2)$ である．100 桁ならばあっという間に計算が終わるが，図 3 のように 1 万桁ともなると，400MHz の K6-2 CPU を搭載したパソコンで 1 秒 2 桁くらいのペースとなる．1 桁出力するたびに 3 万行あまりのテキストバッファを全面的に「編集」していることを考えると，あながち遅いとも言えない．マクロを改造し，

```
GZOJ\UNQN"E 4OUN ’
0UH 1UV HK
QN< J BUQ QN*10/3UI
    QI< \+2*10+(QQ*QI)UA 0L K QI*2-1UJ QA/QJUQ
        QA-(QQ*QJ)-2\ 10@I// QI-1UI >
    QQ/10UT QH+QTUW
    QW-9"E QV*10+QWUV |
      QW-10"E 0UW QV+1UV ’
      .UP QV\ QP+1,.T QP,.K QW+10UV ’
    QQ-(QT*10)UH >
EX
```

(a) マクロ

```
C:\Ueda\teco>tecoc mung pi2c.tec 10000
31415926535897932384626433832795028841971693993751058209749445923078164062862089
98628034825342117067982148086513282306647093844609550582231725359408128481117450
...（途中大幅略）...
59240190274216248439140359989535394590944070469120914093870012645600162374288021
09276457931065792295524988727584610126483699989225695968815920560010165525637 56
```

(b) 実行結果

図 3: 円周率を計算するマクロ (Standard TECO) とその実行結果．繰返し回数 10000 を指定すると，頭の 3 を含めて全部で 9999 桁出る（最後の 1 桁は，少なくともあと 1 回ループを回さないと確定しないので出ない）．

基数を 10 ではなくて 100, 1000, 10000 と大きくすれば，多少は速くなろう．

むすび

おっと，爺さんの話はついつい長くなる．ところで，知名度と迫力で TECO に対抗できるもう一つの拡張可能言語は plain TEX である．TECO のマクロが読みにくいと言っても，plain TEX と比べればどっこいどっこいではないか．私も国際会議の査読結果集計プログラムを plain TEX で書いたことがある．投稿論文を，評点の良いもの，悪いもの，ボーダーライン，割れているものに自動分類する機能までついていた．Andrew M. Greene は，plain TEX で Basic インタプリタを書いた．拡張可能言語としての plain TEX の欠点も挙げればきりがないが，利点もある．それは，仕様も実装も枯れているので，ものすごく移植性がよくて，普通のフォントだけを使っている限り，どこの Unix マシンに持っていっても動くことである．

スクリプト言語とミドルウェア全盛の時代となって，TECO や plain TEX のような独特の体臭を放つ言語は肩身が狭くなり，人間世界だけでなく計算機言語の世界でも，個性が薄らいだ感がある．だが，進化には多様性の維持が大切であることを忘れてはならない．テキスト処理や図形処理など特定分野向きの，香り豊かな拡張可能言語の新作に出会ってみたい．

参考文献

[1] Tom Almy, Text Editor and COrrector, July 1999.
`http://www.aracnet.com/~tomalmy/teco.html`

[2] Brian Connors, The Turing Tarpit, July 2000.
`http://www.geocities.com/ResearchTriangle/Station/2266/tarpit/`

[3] 井田昌之：Emacs 解剖学 (1) TECO — Emacs のみなもと．**bit**，Vol. 28, No. 6，pp. 4–13, 1996.

[4] Stanley Rabinowitz and Stan Wagon, A Spigot Algorithm for the Digits of Pi, American Mathematical Monthly, Vol. 102, pp. 195–203, 1995.

[5] Eric S. Raymond, The Jargon File Resources.
`http://www.tuxedo.org/~esr/jargon/`

[6] Eric S. Raymond, The Retrocomputing Museum.
`http://www.tuxedo.org/~esr/retro/`

[7] Pete Siemsen's TECO collection.
`http://metalab.unc.edu/pub/academic/computer-science/history/pdp-11/teco`

[8] 上田和紀：テキスト・エディタ TECO，東京大学大型計算機センター，1981 年 7 月．

[9] 上田和紀：TECO (Text Editor and COrrector) メモ（第 2 版），1980 年 3 月．
`http://www.ueda.info.waseda.ac.jp/~ueda/readings/teco/`

[10] 和田英一：連載 エディタとテキスト処理 — (6) Teco，**bit**, Vol. 14, No. 10, pp. 76–82, 1982.

[11] TECO command, Multics Programmer's Manual, MIT, 1972.