新 Lecture 1

Emacs 解剖学

TECO—Emacsのみなもと

井田昌之

はじめに：TECO?

今回から連載で Emacs の話をすることになりました．Emacs は，広く使われている画面エディタです．日本では，mule あるいは nemacs のほうが有名かもしれませんが，とりあえず，皆同じものだと思ってください．

Emacs は，一般には UNIX 上のフリーソフトとして位置付けている人が多いかもしれません．けれども，DOS，Windows NT やその他の多くの OS の上で動くバージョンがあります．また，一世を風靡した Lisp マシンのエディタでもありました．さらに，Emacs 上でしかモノを作らないという人たちが，ソフトウェアのエキスパートの中にはたくさんいます．

「Emacs は，画面エディタです」という紹介に不服の人たちがたくさんいると思います．実は私自身もそうです．Emacs にはたくさんのサブシステムがあって，Emacs を立ち上げればその中でほとんどすべての作業ができると言ってもいいくらいです．統合環境だと考えてもいいと思います．テキストの編集だけでなく，電子メールの処理，コンパイルとデバッグの開発環境，WWW のブラウザ機能などなど，多方面の機能があります．

ところが，この第1回では，Emacs ではなく，いきなり「TECO について」という直接 Emacs には関係なさそうなよくわからないもの（?），をテーマにしています．

それには理由があります．TECO と Emacs の間には深い関係があるのです．その話を通して Emacs の仕様の背景に触れます．Emacs の考え方について本質的な理解を得る助けになると思っています．

TECO はもともと，(Paper) Tape Editor and COrrector の頭文字をとったものでした．TECO は20年も前に MIT で広く使われていたエディタです．その名前の由来にあるように TECO はもともと紙テープエディタでした．もう少し具体的に言えば，一口に TTY 用のエディタでした．TTY というのは，紙テープリーダ/パンチの付いたタイプライタ型端末のことです．そのあとで TECO は，Text Editor and COrrector の意味だと，意味づけが変えられました．なお，「TECO」の読み方ですが，日本では「てこ」と読んでしまうケースが多いようですが，アメリカでは「てぃーこー」(tee'koh) と発音します．

TECO と Emacs の関係は，主に次のようなことです．

1) TECO は，いわば MIT AI 研での「Emacs の先代にあたる道具」でその歴史的な位置がある．
2) Emacs は TECO に対する機能拡張マクロから始まった．
3) TECO に対して機能拡張する考え方は，Emacs のそれに通ずるものがある．
4) TECO の画面編集モードのコマンドは Emacs のコマンドの先祖である．

これらについて順に触れていきます．

TECOの基本的な性格

TECOはエディタです．TECOは大変強力なエディタで，プログラミング言語のような機能も持っていました．それを使って，いろいろな機能拡張がされました．ソースプログラムの編集の際に，その言語のシンタクスに合わせてプリティプリントする機能だとか，そういった組込みがありました．最初の処理系はPDP-1用のようです．PDP-10で広く使われて有名になりました．一番最初に誰が作ったのか何人かの人に聞きましたが，確とした返事はもらえませんでした．おそらく何かのプロジェクトでできた小さなエディタが，いろいろな人の手でだんだん大きく育っていったのではないかと推理しています．

TECOは，特に最初は紙テープエディタでした．紙テープ上のプログラムやテキストに対して，削除をしたり，追加をしたり，変更したりして編集します．1本の連続した紙テープの上に穴を開けて記録していくものです．空いた穴をふさぐというのは，(緊急時や職人わざがいるときにはやりますが) ふつうはしません．というか，原則的にはできないことだと考えます．したがって，そのようなテープの編集には，次の基本的な性質が生じます．

1) 編集はコピーをしていきながら，その過程で行なう．

この考え方が根本の概念にありました．紙テープリーダに修正元のテープをセットし，それを読んで，内容を紙テープパンチにセットしてあるブランクの（何も打たれていない）紙テープにコピーしながら処理をするのです．

たとえば，「字を削除する」という編集は「コピーをしないで読み飛ばす」ということで，「字を変更する」という編集は「コピー元をスキップし，そのかわりにキーボードから字を入力して，それを入れる」ということで，「追加をする」には「キーボードから入れたものをそのままパンチする」，というふうにするのがそもそものアイディアでした．それを，修正対象をできるだけメモリに先に読んで蓄えておいて，無駄な紙テープ出力をしないで便利にしようというようになってきます．そして，紙テープ編集ではなく，ディスクなどにあるファイルを編集するように変わってくると，ファイルをすべて読み込んで，それを編集して，結果をファイルに書き込むというようになっていきます．

キーボードからの入力は，編集指示（コマンド）と挿入されるべき文字そのものということになりますが，いろいろな試行錯誤から，コマンドとして使う字をテキストに入力される文字とできるだけ異なるものにすれば手間なく入力と指示の切替えができる，というように考えられてきました．そこからコマンドは，基本的にはctrlキー（コントロールキー）を押しながら文字をたたき，ふつうの文字をそのまま入れると，それはそのままそこに挿入されていくと考えるのが便利だと思われるようになってきます．

コントロールキーと1字の入力だけでは，すべての処理を規定できるとは言えません．それで，キー入力の組合せで意味を増やすようになります．キーコンビネーションといいます．

2) 紙テープというシリアルなメディアで，かつ，1字1字読みながら順に処理をしていくので，処理の単位としては（「行」ではなく），「字」を単位としている．

たとえば，磁気テープや磁気ディスクなら，1回の読込み操作の単位はレコード（複数文字）で，1字のみだということはありません．そうすると，自然に「行」という単位がなじむ面が出てきます．3行目は3レコード目で，3回リード操作をした部分だ，というふうに．あるレコードのなかのある文字は，「何行目の何文字目」という数え方のほうがやりやすく思える場合もでてきます．

また，紙テープとならんで当時よく使われていたカードは，文字どおり，1枚に80文字までの情報を打ち込んでいたので，

カード1枚＝1レコード＝1行

という考え方がなじんでいました．

しかし，TECOでは，「行」という単位には（行エディタと呼ばれるたぐいのエディタに比べて）それほど固執していませんでした．ソースプログラムの場合には，「行」が大きな意味をもつようなものもあります．たとえばアセンブラはだいたい1行に1命令を書きます．けれども，たとえばLispの場合，行というものは機能に関してほとんど意味がありません．1行に詰めて書いても，複数行にまたがって書いても同じ

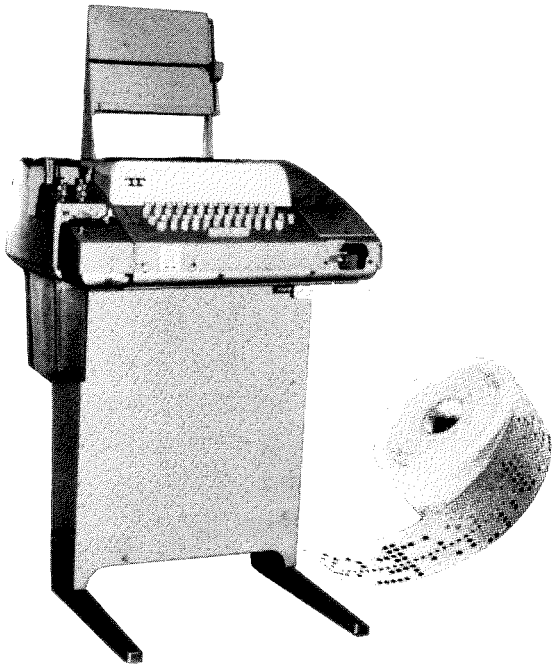


写真　テレタイプと紙テープ

機能を表わします．
たとえば，

```
(defun add2 (x) (+ x 2))
```

と1行に書いても，

```
(defun add2 (x)
  (+ x 2))
```

と2行に分けて書いても同じことです（もっとも，徐々にプリティプリントとか，見やすさのための段付けなどということが始まってきて，エディタでは行の役割が増してきますが）．

こうしたいろいろなことから，TECO はいわゆる「字エディタ」（キャラクタエディタ）なのだということが言われる必然性があったことを理解してほしいと思います．

当然，紙テープの時代というのは，70年代後半には急激に終りとなります．そして，磁気テープや磁気ディスク，フロッピディスクの時代になります．

この間に TECO は，進化していって，紙テープベースからファイルベースに変わります．同時に，さまざまな機能をもつようになり，古くさい仕様は捨てられていきます．TECO はかなり長い間生き残るのですが，結局画面でリアルタイムに直接編集できるいろいろな画面エディタの登場によって博物館行きとなります．

図1に TECO のコマンドの説明書の一部を示します．75年の日付でバージョンが477などとなっています．

TECO の機能拡張

TECO のプログラム機能について次に考えてみましょう．それをプログラミング言語として見る場合，その構文則は，かなり「それもの」(hairly) で，普通の人にはわかりやすいとは言えない代物です．もっとも，そもそもエディタコマンドの組合せをプログラムとするので，1字1字に機能が凝縮されているので仕方のないことだとは言えます．少し前は APL，最近では Perl などに対する筆者のイメージに共通するものがあります．まぁ，言語としてみたらプロ用のよく切れる小道具という感じでしょうか．

少し具体的に紹介してみましょう．TECO のコマンドには編集機能や普通のテキストエディタのような文字サーチなどなどの機能に加えて，ソートをする機能が入っています．これを使って，ランダムに入れたレコードをその中の項目を使って並べ換えたりすることができます．こんな話について友人の Guy Steele に聞いたら，彼の書いた *Hackers Dictionary* に例があるよ，と教えてくれました．

次のようなものです．

『今，Loser, J. Random
　　Quux, The Great
　　Dick, Moby
という3行の人名リストがあるとしよう．これを姓でソートし，かつカンマを取って，姓を後ろにもっていきたいとしよう．
こんなふうに．
　　Moby Dick
　　J. Random Loser
　　The Great Quux
これをするには，次の TECO プログラムを走らせればいい．

```
[1 J^P$L$$
J <.-Z; .,(S,$ -D .)FX1 @F^B $K :
L I $ G1 L>$$
```

（$は，altmode キーあるいは escape（ASCII で

```
        LIST OF TECO COMMANDS, TECO VERSION 477
        LAST UPDATED 14 SEP 1975.  Z=135177

(NOTE THAT AN UPARROW FOLLOWED IMMEDIATELY BY ANOTHER CHARACTER
SIGINIES A CONTROL CHARACTER. A COMMAND WITH THE UPARROW
MODIFIER IS REPRESENTED WITH A SPACE BETWEEN THE UPARROW AND THE
COMMAND, EXCEPT INSIDE FS FLAG NAMES, WHERE THAT IS NOT NECESSARY
BECAUSE CONTROL CHARACTERS ARE NOT ALLOWED ANYWAY.
ALSO, ALTMODE IS ALWAYS REPRESENTED BY A DOLLARSIGN.)

<N>^E   FOR NONNEGATIVE <N>, IS THE SAME AS ".,.+<N>".
        FOR NEGATIVE <N>, IS THE SAME AS ".+<N>,.".
        "10^EXA" PUTS THE 10 CHARACTERS AFTER THE POINTER
        IN A STRING IN QREG A.
<M>,<N>^E
        RETURNS THE VALUE <N>-<M>.

^A      EXCLUSIVE OR (AN ARITHMETIC OPERATOR).

^B      TURNS ON LINE PRINTER OUTPUT (ACTUALLY TO TPL).
        TPL OUTPUT MEANS THAT EVERYTHING WRITTEN OUT BY THE
        COMMANDS "P" AND "PW" GOES INTO THE TPL FILE.
        USE "^E" TO END TPL OUTPUT, AND CLOSE THE TPL FILE.
:^B     TURNS ON TPL OUTPUT AND RETURNS -1.
        NOTE: ^B INSIDE SEARCH STRING IS A
        SPECIAL CHAR WHICH IS MATCHED BY ANY
        DELIMITER CHARACTER. THE SET OF DELIMITER CHARS IS
        SPECIFIED BY THE CONTENTS OF QREG ..D; INITIALLY, THE
        DELIMITER CHARACTERS ARE PRECISELY THE NON-SQUOZE
        CHARACTERS (THAT IS, ALL EXCEPT LETTERS, DIGITS, ".",
        "%" AND "$").
        ^B MAY BE USED IN A FILE SPECIFICATION AS THE SECOND
        FILE NAME TO MEAN THAT THE DEFAULT SECOND FILE NAME
        SHOULD BE USED.

^C      WHEN TYPED IN FROM CONSOLE, TERMINATES THE
        COMMAND STRING, AND STARTS EXECUTION. IF
        THE COMMAND EXECUTES WITHOUT ERROR, TECO
        RETURNS TO ITS SUPERIOR WITHOUT FLUSHING THE TYPE-IN
        BUFFER. WHEN PROCEEDED, IT WILL AUTOMATICALLY
        REDISPLAY THE BUFFER ON DISPLAY CONSOLES.
        WHEN TECO RETURNS, AC 2 WILL CONTAIN
        THE ADDRESS OF THE 7-WORD "BUFFER BLOCK" DESCRIBING THE CURRENT
        BUFFER - SEE THE SECTION "BUFFER BLOCK" AT THE END.
        TO TYPE IN A ^C, USE ^]^Q^C, WHICH IS SPECIALLY
        ARRANGED TO INHIBIT THE NORMAL ACTION OF ^C
        AT COMMAND STRING READ-IN TIME.
        A ^C ENCOUNTERED AS A COMMAND IS AN ERROR.

^D      IF TECO DOESN'T HAVE THE 340, GRABS IT IF FREE.
        IF TECO DOES HAVE IT, IT IS RELEASED.
        WHEN TECO IS STARTED ON A TTY NEXT TO A 340 SLAVE (THAT IS,

                        ⋮
                      〈中略〉
                        ⋮

^P      ALPHABETIC (ASCII)SORT COMMAND.
        THE ENTIRE BUFFER, OR THE PART WITHIN THE VIRTUAL BOUNDARIES,
        IS SORTED, AFTER BEING DIVIDED INTO
        SORT RECORDS (I.E., THINGS TO BE
        SORTED) ON THE BASIS OF THE ARGUMENTS
        GIVEN TO THE COMMAND IN THE FORM OF
        THREE TECO COMMAND STRINGS FOLLOWING
        THE ^P, SEPARATED BY ALTMODES.
        (NOTES: (1) TWO SUCCESSIVE NULL ARGS
        WILL RESULT IN A PREMATURE END OF
        COMMAND INPUT, SO USE SPACES WHERE
        NEEDED.  (2) A DOLLAR SIGN IN ANY
        ARG WILL BE REPLACED BY AN ALTMODE.
        (3) THE THREE ARGS WILL BE LEFT IN Q-REGS ..0, ..1, ..2)
        THE THREE COMMAND STRINGS ARE USED
        TO DIVIDE THE BUFFER INTO SORT
        RECORDS, EACH OF WHICH HAS A SORT KEY
        (WHICH MAY BE ANY PART OF THE RECORD,
        OR OUTSIDE THE RECORD).  THIS IS DONE
        AS FOLLOWS:
         1. THE POINTER IS MOVED TO THE
        BEGINNING OF THE BUFFER, WHICH IS THE
        BEGINNING OF THE FIRST SORT RECORD.
         2. THE FIRST COMMAND STRING IS
        EXECUTED. THIS SHOULD MOVE THE
        POINTER FROM THE BEGINNING OF ANY
        RECORD TO THE BEGINNING OF ITS KEY.
         3. THE SECOND COMMAND STRING IS
        EXECUTED.  THIS SHOULD MOVE THE
        POINTER FROM THE BEGINNING OF ANY KEY
        TO THE END OF THAT KEY.
         4. THE LAST COMMAND STRING IS
        EXECUTED.  THIS SHOULD MOVE THE
        POINTER FROM THE END OF ANY SORT
        KEY TO THE END OF THE RECORD, I.E.,
        THE BEGINNING OF THE NEXT RECORD.
         5. IF STEP 3 OR 4 LEAVES THE POINTER
        AT THE END OF THE BUFFER, OR
        EXECUTES A SEARCH WHICH FAILS (THIS
        WILL NOT CAUSE AN ERROR; THOSE
        STEPS ARE DONE AS IF INSIDE AN
        ITERATION), THE CREATION OF SORT
        RECORDS IS COMPLETE, AND THE SORT
        TAKES PLACE.  OTHERWISE, GO BACK
        TO STEP 2.
        SORT RECORDS AND KEYS MAY BE

                        ⋮
                      〈中略〉
                        ⋮

^R      REAL TIME EDIT FEATURE. INTENDED MAINLY FOR DISPLAY TERMINALS.
        THE POSITION OF THE POINTER IS REPRESENTED BY THE TERMINAL'S
        HARDWARE CURSOR, RATHER THAN BY ANY PRINTED CHARACTERS
        (^R MAKES THAT HAPPEN BY DOING "[..A @U..A").
        ALL NON-CONTROL-NON-RUBOUT CHARACTERS
        ARE NORMALLY SELF INSERTING; THE OTHERS ARE NORMALLY
        EDITING COMMANDS. THE USER MAY REDEFINE ANY CHARACTER
        BY MEANS OF THE FS ^RCMAC$ FLAG.
        IN ^R MODE ECHOING IS TURNED OFF, SO TYPED-IN CHARACTERS
        MANIFEST THEMSELVES ONLY BY THEIR EFFECT ON THE DISPLAYED
        BUFFER CONTENTS (BUT SEE FS ^R ECHO$).

        WHY THE SCREEN GETS SHORTER IN ^R MODE:

        WITHIN ^R MODE, THE SCREEN WIDTH IS ROUNDED DOWN
        TO A MULTIPLE OF 8. THIS IS DONE BECAUSE OTHERWISE
        IT WOULD BE EXTREMELY DIFFICULT TO HANDLE TABS IN
        CONTINUATION LINES PROPERLY. ONE MAY PREFER TO HAVE
        THE SCREEN MADE PERMANENTLY SHORTER INSTEAD OF HAVING
        IT CHANGE WHEN ^R IS ENTERED OR EXITED; THE FOLLOWING
        COMMANDS WILL DO THE TRICK: FSWIDTH$/8*8FSWIDTH$W

        ANY COMMAND MAY BE GIVEN A NUMERIC ARGUMENT, WHICH MOST
        COMMANDS (INCLUDING ALL CHARACTERS THAT INSERT THEMSELVES)
        TREAT AS A REPETITION COUNT. NOT GIVING AN ARGUMENT
        EXPLICITLY IS EQUIVALENT TO GIVING 1 AS AN ARGUMENT.
        THE ARGUMENT IS COMPUTED AS FOLLOWS:
        <ARG> = <BASIC ARG> * (4 ** <EXPONENT-OF-4>)
        WHERE <BASIC ARG> IS INITIALLY 1 AND SETTABLE BY ^V,
        AND <EXPT-OF-4> IS INITIALLY 0 AND INCREMENTED BY ^U.
        ALL COMMANDS EXCEPT ^U AND ^V, EVEN THOSE THAT IGNORE
        THE ARGUMENT, REINITIALIZE THOSE 2 VARIABLES.

        ANY CHARACTER MAY HAVE A
        PROGRAM ASSOCIATED WITH IT, USING THE FS^RCMACRO$ COMMAND.
        IF THAT IS DONE, WHEN THAT CHARACTER IS TYPED, TECO
        WILL EXECUTE THE PROGRAM INSTEAD OF INSERTING THE CHAR
        OR USING IT AS A BUILT-IN COMMAND. THE DEFINITION OF A
        CHARACTER MAY ALSO BE TREATED AS A Q-REGISTER IN THE
        "Q", "U", "X", "G", "[", "]", "M" AND "FQ" COMMANDS;
        SEE "Q" FOR DIRECTIONS. WHEN THE PROGRAM IS EXECUTED,
        QREG ..0 WILL CONTAIN THE CHARACTER BEING HANDLED.

        ONE MAY WISH TO HAVE
        A MODE IN WHICH MOST EDITING COMMANDS ARE DISABLED, AND
        MOST CHARACTERS THAT ARE NORMALLY EDITING COMMANDS ARE
        SELF-INSERTING INSTEAD. THE FS ^RSUPPRESS$ FLAG, WHEN
        NONZERO, SUPPRESSES ALL BUILT-IN COMMANDS EXCEPT RUBOUT
        AND ALL USER DEFINED COMMANDS WHOSE DEFINITIONS DO NOT
        BEGIN WITH "W" (SINCE "W" AT THE BEGINNING OF A MACRO
        IS A NO-OP, THE ONLY REASON TO HAVE ONE THERE IS TO
        PREVENT SUPPRESSION). WHEN A CHARACTER IS SUPPRESSED
        AS A COMMAND, IT BECOMES SELF-INSERTING.

        THE ^R-MODE INPUT DISPATCH TABLE IS ACTUALLY INDEXED BY
        9-BIT TV CHARACTER CODE. EACH 9-BIT CODE CAN BE REDEFINED.
        THE LIST OF ^R-MODE INITIAL DEFINITIONS THAT FOLLOWS
        REFERS TO THE CHARACTERS OBTAINABLE ON NON-TV'S - IN OTHER
        WORDS, THE 9-BIT CHARACTERS WHICH ARE THE RESULTS OF
        READING IN THE 14-BIT CODES 0000 THROUGH 0177, WHICH ARE
        PRECISELY THE 9-BIT CHARACTERS WHICH ARE EQUIVALENT TO
        SOME 7-BIT ASCII CHARACTER.
        A SUBSYSTEM WHICH IS NOT TV ORIENTED NEED NOT WORRY ABOUT
        THE 9-BIT CHARACTER SET; BY USING "FI", AND "FS^RCMACRO$"
        ALWAYS WITHOUT THE UPARROW MODIFIER, IT CAN HANDLE ASCII
        CHARACTERS THROUGHOUT. TECO WILL AUTMOATICALLY DO THE
        CONVERSION TO AND FROM 9-BIT CHARACTERS ON TV'S.
        FOR THOSE WHO WISH TO HANDLE THE 9-BIT CHARACTER SET,
```

図1　TECO コマンド説明書（1975 年）より

```
!^Filename^:! !This file contains ?'s normally used macros.!
?Macs
[

!List Commands:! !U List all user commands.
Lists all macros of class "U", intended to be called directly
by the user with MM.  Other macros include subroutines (class S)
and ^R command definitions (class "^R").  Use List Subroutines
and List ^R Commands to see their names.!
    m(m.m&_List_Some_Macros$)$U $ K
[

!List Subroutines:! !U List all subroutines
(Macros intended mainly to be called by other macros).
Subroutines are identified by having class "S ".
Most subroutines have names starting with "& ".!
    m(m.m&_List_Some_Macros$)$S $ K
[

!List ^R Commands:! !U List macros intended to be ^R-commands
(whether or not they are actually put on any character now).
They are macros of class "^R " (says their documentation).!
    m(m.m&_List_Some_Macros$)^R$^R $ K
[

!& List Some Macros:! !S List all loaded macros of a given class.
Takes two string arguments:  a name prefix, and a documentation
prefix (macro class).  Only macros whose name and documentation
start with the specified prefixes are listed.
The classes normally used are "U " for user commands,
"S " for subroutines, and "^R " for macros intended to be
made the definitions of ^R-mode commands.!
    [7 :i7⎋⎋$                       !* Get Name prefix in q7!
    [8 [9 :i9⎋⎋$                    !* Get "class" (doc prefix) in q9!
    fs:ejpage$*5120+4000000000000.u8  !* 1st file in q8!
    < -fq8;                         !* Stop if no more files.!
      q8m( m.m &_List_One_File$)⎋7⎋9$  !* List macros of this class in one file!
      q8+fq8+4u8>                   !* Advance to next file.!
    K
[

!& List One File:! !S List some of the macros in a specific file.
A pointer to the file should be the first argument.
There should be two string arguments too, specifying the
name prefix and the documenation prefix.!
    [9 [8 :i8⎋⎋$ :i9⎋⎋$          !* Q9 gets doc prefix, Q8 gets name prefix!
    [0 [7
    f[b bind$                       !* Make a buffer for temporary use!
    f[s string$
    m.m&_Get_From_File$ u7          !* Use ^XM7 instead of M.M!
    g(⎋m7^Filename^$)              !* get this files name!
    z"e i0(( anonymous ))$'         !* Some files may not have a ~filename~!
    j <:s
$: -2d>                             !* Flush any CRLFs from the filename.!
    ftIn_file_⎋⎋..O:
```

図2　List Redefinition プログラムの原形

16進02))』

ちなみに，The Great Quux は，Guy Steele 自身のしゃれの別名です．Quux は，ちょうど foo とか bar などのように何かを指す名札として彼が学生の頃に考え出した言葉で，同時に，それを自分自身を指すためにも使っていました．そのときにふざけて，Quux だけでは完全でないから（姓と名があるべきなので，）Quux The Great とか The Great Quux という遊びのサインをしたのが続いているとのことです．

話を元に戻します．

いろいろな人がちょっとした機能をもつコマンド列の集まりを，「こんなのどうだい？　便利だろ？　使ってみてよ」などと互いに言い合うようになります．ちょうどそういうことのできる仕組みが用意されて，人の作ったいろいろな機能を利用するようになりました．あらかじめ機能の組合せを作っておいて，それをあとで利用します．TECO では，これらをマクロといっています．次の節で話をするリアルタイム編集機能もマクロです．

たくさんのマクロができて，それを登録しておいて呼び出す仕組みができました．図2に例を示します．Guy Steele と Richard Stallman が最初に共同作業をしたプログラムだとのことです．

マクロの多くには，「なんとか MAC」あるいは「なんとか MACS」という名前が付けられました．どんなマクロがあるかというディレクトリもできまし

た．その中で，「E ではじまるものがなかったので，Emacs とつけたんだ」と Richard Stallman は，Emacs の名前の由来について語ってくれています．

TECO リアルタイム編集機能

現在の多くのエディタで見られるような画面編集機能をエディタがもつということは，最近の人には当然のことのようでも，当時としては画期的な出来ごとだったと言えます．このリアルタイム画面編集機能を呼び出すには，TECO で，^R を入力します．

文字単位の編集をする TECO エディタが，当時使われだしたディスプレイ端末で，その画面全体を利用して編集機能を提供するようにしようと，最初に誰が言い出したのか，あるいは，そのほうがいいと最初に誰が気がついたのか，この原稿を書いている段階では，はっきりとつきとめることはできませんでした．少なくとも 1975 年にはその機能が入っているので，それ以前のことには違いありません．

Richard Stallman にこの辺のいきさつを聞くと，彼の返事は，

「Carl Mikkelson という人がいて，彼が最初に ^R モードを作ったんだ．でも，出来が悪くてよく落っこちる，おまけに，画面の 5 行分しか使わない．それで，そいつを私が作り直して，使いものになるようにしたんだ．」（“Carl Mikkelson implemented the first ^R mode. It was inefficient and unreliable and used only 5 lines of the screen. I reimplemented it and made it practical to use.”）

ということなので，Carl Mikkelson という人が最初に作ったようにも考えられます．

TECO version 495 の資料（76 年 10 月 17 日付け）を見ると，さまざまな改良点の報告があり，その中で，画面編集モードはこの 1 年で 2 倍速くしたこと，ウィンドウサイズを制御できるようにしたこと，全画面でなく縦方向の高さを制御できるようにしたこと，幅も以前の 8 の倍数の大きさだけでなく，画面の幅全部に表示できるようにしたこと，^R の中でエラーがあったり quit したりしても影響のある一番内側だけから出るように変更したこと，そうしたことが Richard Stallman によって記述されています．したが

表1　TECO リアルタイム編集モードでのコマンド

キー	機　能
^A	現在行の先頭に移動する
^B	1文字戻る（back）
^C	コメントモードスイッチの反転（本文中に解説あり）
^D	ポインタ位置の次の文字を削除する
^E	行末（End of Line）に移動する
^F	1文字先へ進める（forward）
^G	数値引数や case-shift をフラッシュし，マークがあればそれをはずし，case-lock をリセットする （^R モードのマクロを書く人は入力のところで，^G が適切な quit action をするように insure しなければならない． そのマクロでの処理を中断・アンドゥし，^R モードへ戻るようにする．）
^H	(Backspace) inserts itself.
^J	(Linefeed) inserts itself.
^K	行末までキルする．削除されたテキストは QREG "..K" に入れられる．
^L	画面の再表示
^M	CRLF の挿入
^N	次行へ
^O	CRLF を挿入し，その前に移動する． （^OFOO と入れるのは，FOO^M と同じ効果だが，しばしば，^O を使うほうが再表示がいらなくなるので便利である．カーソルは元の行にいるので．）
^P	前行に戻る
^Q	続く文字をそのまま入れる（コマンドとしての意味を無視して）数値引数が与えられていると続く文字がその個数だけ繰り返して入れられる．
^R	ポインタがある位置をコメントカラムとする．
^S	1文字読み，それを探す．（^R モードでの ^SA は，TECO での SA$ に等しい）
^T	現在のポインタ位置に^R モードマークをセットする．
^U	次のコマンドを4倍する．これはふつうそれを4回繰り返して実行することに等しい．
^V	次のコマンドに対して，basic 引数をセットする．
^W	現在のポインタ位置とマークの間のすべてをキルする．削除されたテキストは QREG "..K" に置く．何も削除されなければベルが鳴る．
^X	現在のポインタ位置にマークをセットし，その後，それまでマークがあったところへ飛ぶ．別の言い方をすれば，マークとポインタを交換する．マークがなければ何もしない．^W をする前に，このコマンドを何回か実行して削除する範囲の両端を確認するとよい．
^[	(Altmode) 編集の終了
^]	QREG 名を読み，その QREG をマクロとして実行する．
^?	手前の1字を削除する．(RUBOUT. delete backwards)

って，彼自身の説明からしても75年あたりからはほとんど彼の手によって維持されていたように推理しています．

それでは，このモードで使える機能について見ていきましょう．リアルタイム編集モードに入ると，**表1**のようなコマンドが使えるようになります．もし，Emacs をすでに使っている人がこれを読んだら，かなり現在の Emacs コマンドに似ているので驚くことと思います．なんと，多くのコマンドは現在の Emacs にそのまま受け継がれています．だから，これらを順に見ていってそれで，Emacs コマンドの基本的な部分を解釈していってみようと思います．

その前に，まず，表記の仕方ですが，コントロールキーを押しながら A という文字を叩くことを

　^A

と表記しています．これは現在にも受け継がれています．なお，一時は，図2の中にあるように，漢字の「がんだれ」のように上と左側を修飾する特殊な文字表現が考案され，AI 研にあった Xerox Graphic Printer（XGP）にフォントが作られて使っていました．その方式では，^A は，

　⌈A

のように表現されます．

また，O と$を重ね打ちしたような字も作られ，これで，altmode キーの入力を表わしていました．こういう工夫まであったのを発見するのは驚きです．

それぞれのコマンドの意味

ちょっと脱線しましたが，ABC 順に表1を追いかけてみましょう．

^A は，現在行の先頭に移動します．ようするに右端に行きます．^B は，1文字戻ります．ただ端に位置が1字手前に移動するだけで，テキストが削除されたりすることはありません．これらは，現在の Emacs にも受け継がれています．

^C の場合，現在の Emacs とは異なる動きをします．コメントを書く際の制御をするようになっています．コメント入力をする場合には C を，テキスト入力をする場合には T をスクリーンの最下行でたたきます．そして，コメントモードになると，^N と ^P コマンドの役割が増えます．それらは行末にカーソルを移動させ，もし最後の文字がセミコロンであればそ

れと，もしその前にタブ文字があればそれも削除し，次行または前行に移動して，その行にセミコロンがあればそのセミコロンの後にカーソルを移動します．もしセミコロンがなければ行末に移動し，十分なタブを入れてコメントカラム（コメントが入るとすればそこに揃えられるという指定位置）に移動し，セミコロンを置き，その後にポインタの位置を置きます．Emacs では，こうなってはいません．

^D は，「ポインタ位置」の次の文字を削除するという機能をもっています．昔のディスプレイでは，カーソル位置が直接文字位置と一致しているのではなく，文字位置と文字位置の間に小さなマークが入るようになっていて，文字と文字の間にポインタが置かれる形式もありました．キャレットなどという用語があったのも思い出します（だいぶ前のことなので，正確なことは忘れました．今回，この記事を書くために資料を探してみましたが，私の記憶がどこまで厳密に正しいのか断定できません．年はとりたくないですね．どなたかよく覚えておられる方がいたら，この辺のことを記事にしてくださってもおもしろいと思っています）．とにかく，ポインタと言うのは，文字と文字との間の位置ですが，「ポインタ位置の次の文字を削除する」というのは，現代の感覚で言えば，「今カーソルのあるところの文字を削除する」ということと等しいという理解でかまいません．そうすると，現在の Emacs と同じ機能が同じ ^D に割り当てられています．

^E は行末に，^F は次の文字に，それぞれ位置を進めます．これらの設定のされかたも Emacs と同じです．

^G は，少し機能の進化の中で役割が違いますが，基本的に，今やっている機能を中止する，quit するという役割をもっていて，現在の Emacs と同じ機能だといえます．また，大本の TECO でも，^G はこのような感覚の使われ方をしています．

^H および ^J はそれぞれ，backspace 文字，Linefeed 文字を入れる，と記されています．したがって，コマンドと言うよりその文字がそのまま入ると考えていいと思います．

^K ですが，これは Emacs と同じように，キルを行なうコマンドになっています．現在のポインタ位置から，行末までを削除します．Emacs では，キルしたものはキルリングという特殊なデータ型に入っていきますが，TECO のリアルタイム編集モードでは，..K という名前の Q レジスタ（QREG）に入ります．

ここで三つの話の確認をする必要が出てきます．第一に，キルの動作そのもの，第二に，Q レジスタということ，第三に，..K というレジスタについてです．

キルということ

キルをすると何が起きるか，ということは，Emacs の機能の習得においても大変重要なテーマです．キルをする場合には，テキストの範囲を指定します．その指定された範囲のテキストを別のところに移し，元あったところからなくすというのが，共通した基本概念です．^K では，範囲の指定は自動的に行なわれます．つまり，^K を実行したときのポインタの位置から行末までという範囲が対象になります．しかし，当然，好きな範囲，^k の場合のように 1 行の中だけでなく，何行も一度に対象範囲にしたい，ということが出てきます．それをするには，^W というコマンドがあります．^W はその時設定された範囲のテキストが対象になります．それは何行でも構いません．行の途中から別の行の途中までででも構いません．^W では，「マークする」という別の操作を前提とします．マークは ^T コマンドで行ないます．マークというのは，テキスト中のある場所に見えない目印をつけることです．そうしておいて，「マークされた場所から現在ポインタがある位置まで」という形である対象範囲というのを決めます．だから，^W を使ってキルする場合には，

1) ^T でマークする（Emacs では別のキーです）
2) 対象範囲の反対の端までポインタを移動する
3) 移動した先で ^W を実行する

という手順が一般的なものになります．

これで，キルの概念がわかってもらえたと思います．

□ レジスタ

次に Q レジスタ（Qreg）というものの話をします．

Qreg は，情報/テキストを保存しておくレジスタです．各レジスタには名前がついていますが，二つのピリオドではじまる 3 字の名前のレジスタはシステム固有の役割をもっています．75 年の仕様書では..A か

ら..P までの16個があります．たとえば，..D という QREG には，詳細は省略しますが，デリミタディスパッチテーブルというものが入っていて，7ビット，128字の ASCII 文字それぞれのディスパッチの指示が入ります．

76年10月17日付けの TECO version 495の資料（Richard Stallman による）を見ると，このときにQレジスタは，レジスタのベクタではなく，（GC 対象となる）普通のバッファに等しい扱いになるように変えられたとあります．そして Emacs になるとQレジスタは姿を消します．

Q レジスタ..K には，キルされたテキストが入ってきます．この..K というQレジスタの内容を読み出せば，キルして切り取ったテキストをもう一度戻すことができます．だから，..K レジスタを，誤って切り取ったときに復元するために使えます．また，よく使われるような「テキストの移動」という機能のために使うこともできます．「テキストの移動」は，^K や ^W で，ある範囲をキルし，それから，それを別のところで読み出せば実現できます．Emacs でも基本的にはそうやってテキストの移動をします．

これで，脱線して説明したキル，マーク，Qreg の話は終わります．それでは，表のコマンドの説明に戻りましょう．

それぞれのコマンドの意味（つづき）

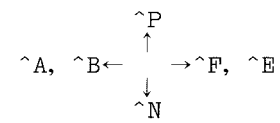
^K の次は ^L の話になります．

^L は画面の再表示をします．TECO のリアルタイムモードを使っていないので実感はわかりませんが，書かれていることを読むと，現在の Emacs と同じような機能のようです．現在の Emacs では ^L をたたくと，画面の再表示がされ，かつカーソルのいる行が画面の中央にくるようになります．

^M はようするに改行です．行を変えるということは「行末文字」を入れるということが相当しますが，これは，具体的には CR と LF の2文字の連続のことでした．行末文字は具体的には何かということは現在まで続く大きなテーマなのですが，ここでの Emacs の話に直接関係がないので，その話は省略します．

^N は次の行（画面でいえば下の行）へポインタを進めます．ついでにまとめて説明すると，^P は手前の行（画面でいえば上の行）へポインタを進めます．

この^P の話までで，基本的なポインタの移動のためのコマンドはすべて出てきました．図にすると次のようになります．



そして，この移動のためのコマンドは Emacs にも受け継がれていて，同じなのです．

次の ^O は，表の中で説明されているように ^M に準じた働きをします．「そこから後は次の行へ追いやって新しいテキストを入れる」という感じだといえます．これも Emacs に受け継がれています．

^Q は，表で説明されているとおりです．Emacs では，続けて8進数3桁を入れるように使われます．

^S は，サーチのためのコマンドです．このレベルでは1文字のサーチだけです．しかし，Emacs では同じ ^S がインクリメンタルサーチという大変強力で使いやすいものになっています．これについてはいずれ紹介します．

^T はマークをセットします（Emacs ではそうではありません．注意してください）．^T でセットされるマークは，TECO のコマンドを直接使った場合，FS ^RMARK$ で同じ値を見ることができます．FS は F と S を2字続けて打ちます．この FS ではじまる TECO コマンドはたくさんあり，それらは，その後で指定された「もの」の値を取り出したり，関連するいろいろなアクションをしたりします．この FS の概念を自然に収束させたのが，Lisp の採用であり，そこでの変数の概念でしょう．これについてはいずれ紹介します．

それで，このマークは^R モードコマンドの ^X と ^W で利用されます．もし，FS ^RMARK$ が−1であれば，そこにはマークがないことを表わします．マークがないのにそれを利用しようとするとベルが鳴ります．

^V の正確な機能はよくわかりません．賢明な読者が説明できるかもしれないので，以下に説明文をつけておきます．

> "^V sets the basic arg for the next command. The argument is composed of digits optionally preceded by a minus sign, echoed at the bottom

of the screen and turned into a number in the current radix (FS IBASE$). The first non-digit terminates the arg and is treated as a command. ^G will flush the argument."

^W はすでに説明しました．また，^X は表中に細かく説明しているので，省略します．

^] は，QREG 名を読み，その QREG をマクロとして実行します．その QREG には，ふつうの TECO コマンドが入っていなければなりません（^R モードコマンドではいけない）．数値引数を与えると，それはそのマクロに渡され，^Y の値としてそれを見ることができます．そのマクロは，値を返すことができ，その値はバッファのどの領域を画面に表示するかを指定します．たとえば，TECO コマンド ".U0 G,.K Q0,." は，QREG ..K を得て，変更が生じたバッファの範囲を示す二つの値を返します．

これで一通り説明しました．

TECO の起動とカスタマイズ

TECO が動き出すとき，いろいろなデータを初期化し，バージョン番号をプリントします．通常の起動では，それでそのまま TECO に入ります．実行開始が正常な開始番地+1からだと，ある場所からファイル名などをとり，TECO ダンプファイルをロードします．ようするにこれでシステム全体の機能を変更できます．その後，TECO は自分自身をリスタートします．..L QREG にマクロがあるとその後それが実行されます．"TECO INIT" ファイルという概念があって，TECO が起動されるときに ".TECO." というファイルの内容が読まれ，実行されるようになっています．その初期化ファイルの置かれている位置，ネーミングなども整備されていくようになります．

そうすると，TECO そしてその後の画面編集機能をみんなで維持管理して育てていくというスタイルができてきます．

そのうち，TECO プログラミングスタンダードなどというものでてきました．良いプログラミングの仕方というわけです．

もっとも，手元にある資料では，五つしか項目がありません．それには，

コメントはタグを書く！を使って
!< comment >!
のようにせよ，

とか，

よいプログラム例は RMAIL だから，それを見よ，

といった記述があります．

みんなでマクロを共有し，育てていく

いかがですか？ TECO の話を通して，Emacs の本質的な部分の背景を説明したつもりです．また，Emacs の基本的な操作の核に触れたつもりです．

TECO を使うときには，初期化ファイルが読まれて，それでカスタマイズすることができます．また，マクロといってプログラム機能を使って一連の処理を登録しておいてそれを後で引用できることがあります．さらに，ディスプレイ端末を生かせるように，現在では普通になった画面編集機能もマクロとして入ったということもお話しました．そして，それが Emacs に発展していったのです．その中心人物に Richard Stallman がいて，彼はその当時から現在までも同じ世界でこつこつと作業を進め，改良を続けているのです．

また，このプロセスの文化として，みんなでマクロを作ってそれを共有し，育てていくという考え方があることもわかってもらえたでしょうか？ それは，Richard Stallman が現在も進めている GNU ソフトウェアおよびフリーソフトウェア財団に流れる考え方のきわめて自然な流れの源流になっているのです．

次回はキーボードとの関係，なぜ Lisp が登場したか，そうしたことへ進んでいきます．

（いだ まさゆき　青山学院大学 国際政治経済学部）