Emacs 解剖学

Lecture 2

Emacsの成立とキーボード

井田昌之

70年代中頃まで

第1回では，Emacs のそもそものはじまりについて，そのみなもとである Teco を紹介しました．文字単位で編集をするエディタである Teco にリアルタイム編集機能として，今日広く見られるような画面編集モードが組み込まれるようになり，そのためのマクロが Emacs に発展したことを見てきました．これらはおよそ1960年代から1970年代前半での流れと言うことができます．そして，資料としては1976年の Teco のドキュメントまでを触れました．

第2回では，70年代後半での流れを追っていきます．まず，前回を受けて77年，78年くらいの Emacs の成立について触れます．そして，時計を少し戻して，1974年頃の Lisp マシンの発端とそこでの考え方からはじめ，80年代前半のキーボード談義へと進みます．

Teco の時代から受け継がれた思想

第1回の表1に示した Teco リアルタイム編集機能でのコマンドは，その後の Emacs に多くが引き継がれていることは説明しました．引き継がれているのは具体的なコマンドの意味と役割だけではありません．その本質的な設計思想も引き継がれています．

いろいろなことがあるのですが，簡単にポイントとなることをもう一度まとめると，次のようなことだと思います．

1) コマンドはできるだけ簡単なキーの入力でそのまま実行されること：たとえば，テキストの入力中に control＋a を押せば，直ちにその行の先頭にカーソルが移動します．コマンド指示のモードに変更してからでないと指示が出せないなどということはなく，テキストの入力の途中で直ちにコマンドを実行させて，また直ちにテキストの入力作業を継続できます．
2) 機能拡張は，それぞれに対応してコマンドキーを増やしていくのではなく，プログラミング機能を用意して，それによって，動的に追加/拡張ができるようにしたこと．

この1)の性質のためには，入力する文字とコマンド指示をうまく切り替えられるキーボードが使えることが鍵になってきます．たとえば，a と押せばそのまま，a の文字がテキスト中に入り，control を押しながら a を押すと，コマンドとして起動されるようになっている，といったことは大変重要な意味をもってきます．そして，キーボードの設計に大きな影響がありました．

文字キーに対してはシフト，コントロールだけでなく，メタキーが導入されます．こうしたモディファイヤキーを同時に，あるいは順に押す操作概念，さらに，ファンクションキーを多数用意し，少ないキータッチで機能選択をする仕組みができ上がっていきます．

2)の性質のためには，プログラミング言語としての

良さということが鍵になってきます．Teco ではやりにくかったことが Emacs では大変きれいにできるようになりました．そして，いろいろなプログラムが共有され，互いにプログラムを直しあっていく文化の根源をつくり出します．Lisp の採用とそれによるダイナミックなソフトウェアの開発改良というスタイルができてきます．

メタキーと meta-x

文字を修飾するモディファイヤキーとしてよく知られているものはシフトキーそして，コントロールキーです．コントロールキーについては説明の必要はないと思います．この Emacs 解剖学の第1回でも頻繁にでてきました．

メタキーは，文字キーと組み合わせて別の機能を実行するために使われます．たとえば，x と書いてあるキーを押すとその文字がテキストに入りますが，コントロールを押しながら x を押してある機能を実行させる，あるいは，メタを押しながら x を押して別の機能を実行させるといった具合です．

今まではコントロールキーを押しながら x を押すことを，^x と書いてきました．これから先，コントロールキーと言うことをはっきりさせたい場合には，c-x などと表記をします．

メタキーを押しながら文字キーを押すことを表わすには，M-x あるいは m-x あるいは meta-x などと書きます（本稿では，主に m-x を使用します）．

また，コントロールキーとメタキーを同時に押しながら文字キーを押すという拡張もされるようになります．たとえば，c-m-a というと，コントロールキーとメタキーを押しながら a を押すということになります．これは，今回後述する Lisp マシンキーボードなどでは多用される考え方になります．しかし，標準の ASCII キーボードなどではメタキーがなく，代用キーを使うことが多く，コントロールキーとメタキーを同時に押す方法はそれほどはやらなくなりました．

メタキーの特殊な使い方として，m-x をエディタの中でたたき，次に機能の名称を入れるという使い方が出現してきます．古くは Teco のマクロ名でした．これが，Emacs では Emacs Lisp で定義したコマンド名になっていきます．名前を文字列的に入力して，その名前が表わす機能が呼び出されるような仕組みにします．そうすると，拡張性が高くなるわけです．

キーコンビネーションと control-x

アルファベットは26文字と限られていて，数字や特殊文字を入れても，コントロールキーとの組合せでそれほど多くのコマンドを設定することはできません．それで，m-x で拡張することもできますが，第1回でもちょっと触れたキーコンビネーションという考え方が出てきます．複数のキーの組合せで指示を出すということです．あるキーを押すとそれ自身では仕事をせず，その次に押したキーが仕事を決めるというようにします．Emacs では，c-x などがこのために使われるようになります．

たとえば，c-x c-c は Emacs からの脱出を，c-x c-w はファイルの書き出しを指示する，というようになります．

初期の Emacs コマンドセット

図1に初期の Emacs コマンドセットの表を示します．手元には77年11月01日版と78年12月01日版がありますが，78年版のほうがコピーがきれいなので，それをつけました．

見出しに ^R コマンドと書かれているのは，Emacs が ^R で呼び出される Teco のリアルタイムモード機能であることをひきずっています．

図1では，コントロールを同時に押して実行させるコマンド，メタを押しながら実行させるコマンド，コントロールとメタを同時に押しながら実行させるコマンド，c-x を前置してキーコンビネーションで実行させるコマンドが並べられています．現在の Emacs コマンドを知っている人は順に比べていくとおもしろいと思います．

77年版と78年版の二つの違い，つまり77年からの1年間の進歩を**表1**に示します．

キー入力という観点では ^h が77年では存在せず，78年になって加えられたというのは，筆者個人としては驚きでした．

こうした流れの中で，TECO では，画面編集モードで実行できないコマンドは，そこから出て TECO

```
Control Characters:

α     ..   ^R Complement SAIL Mode
Backspace moves back one character (or several).
Tab        inserts itself.
Linefeed  ^R Indent New Line
Return    ^R CRLF
Altmode   exits from ^R mode.
Space     ^R Set/Pop Mark
%     ..   ^R Replace String
-     ..   is part of the next command's argument.
0 thru 9   is part of the next command's argument.
;     ..   ^R Indent for Comment
<     ..   ^R Mark Beginning
=     ..   ^R Print Hpos
>     ..   ^R Mark End
?     ..   ^R Describe
@     ..   ^R Set/Pop Mark
A     ..   ^R Beginning of Real Line
B     ..   moves back one character (or several).
C     ..   ^R Prefix Control-Meta
D     ..   deletes one character (or several) forward.
E     ..   ^R End of Real Line
F     ..   moves forward one character (or several).
G     ..   quits execution of any command.
H     ..   moves back one character (or several).
I     ..   inserts itself.
J     ..   ^R Indent New Line
K     ..   ^R Kill Line
L     ..   ^R New Window
M     ..   is bare TECO's Control-M
N     ..   ^R Down Real Line
O     ..   inserts one CRLF (or several) after point.
P     ..   ^R Up Real Line
Q     ..   ^R Quoted Insert
R     ..   ^R Reverse Search
S     ..   ^R Incremental Search
U     ..   ^R Universal Argument
V     ..   ^R Next Screen
W     ..   ^R Kill Region
X     ..   is a prefix character.  See below.
Y     ..   ^R Un-kill
Z     ..   ^R Return to Superior
\     ..   ^R Prefix Meta
]     ..   is bare TECO's Control-]
^     ..   ^R Prefix Control
Rubout     deletes backwards, turning tabs into spaces.
```

```
Meta Characters:

Linefeed  ^R Indent New Comment Line
Return    ^R Back to Indentation
Altmode   ^R Execute Minibuffer
#     ..   ^R Change Font Word
%     ..   ^R Query Replace
(     ..   ^R Make ()
)     ..   ^R Move Over )
-     ..   is part of the next command's argument.
0 thru 9   is part of the next command's argument.
;     ..   ^R Indent for Comment
<     ..   ^R Goto Beginning
=     ..   ^R Count Lines Region
>     ..   ^R Goto End
?     ..   ^R Describe
@     ..   ^R Mark Word
A     ..   ^R Backward Sentence
B     ..   ^R Backward Word
C     ..   ^R Uppercase Initial
D     ..   ^R Kill Word
E     ..   ^R Forward Sentence
F     ..   ^R Forward Word
G     ..   ^R Fill Region
H     ..   ^R Mark Paragraph
I     ..   ^R Tab to Tab Stop
J     ..   ^R Indent New Comment Line
K     ..   ^R Kill Word
L     ..   ^R Lowercase Word
M     ..   ^R Back to Indentation
N     ..   ^R Down Comment Line
P     ..   ^R Up Comment Line
Q     ..   ^R Fill Paragraph
R     ..   ^R Move to Screen Edge
S     ..   ^R Center Line
T     ..   ^R Mark Word
U     ..   ^R Uppercase Word
V     ..   ^R Previous Screen
W     ..   ^R Copy Region
X     ..   ^R Execute MM Command
Y     ..   ^R Un-kill Pop
[     ..   ^R Backward Paragraph
\     ..   ^R Delete Horizontal Space
]     ..   ^R Forward Paragraph
^     ..   ^R Delete Indentation
_     ..   ^R Underline Word
Rubout     ^R Backward Kill Word
```

```
Control-X is an escape prefix command with these subcommands:

^X ^B      List Buffers
^X ^C      ^R Return to Superior
^X ^D      ^R Directory Display
^X ^E      ^R Edit File
^X ^F      ^R Find File
^X Tab     ^R Indent Rigidly
^X ^L      ^R Lowercase Region
^X ^N      ^R Set Goal Column
^X ^O      ^R Delete Blank Lines
^X ^P      ^R Mark Page
^X ^Q      ^R Do Not Write File
^X ^R      ^R Read File
^X ^S      ^R Save File
^X ^T      ^R Mini Visited File
^X ^U      ^R Uppercase Region
^X ^V      ^R Visit File
^X ^W      ^R Write File
^X ^X      ^R Exchange Point and Mark
^X Altmode ^R Re-execute Minibuffer
^X #       ^R Change Font Region
^X .       ^R Set Fill Prefix
^X 1       ^R One Window
^X 2       ^R Two Windows
^X 3       ^R View Two Windows
^X :       ^R Set Column
^X ;       ^R Set Comment Column
^X =       ^R Print Hpos
^X A       ^R Append to Buffer
^X B       ^R Select Buffer
^X D       ^R Dired
^X F       ^R Set Fill Column
^X G       ^R Open Get Q-reg
^X I       ^R Info
^X K       ^R Kill Buffer
^X L       ^R Count Lines Page
^X M       Mail
^X N       ^R Set Bounds Region
^X O       ^R Other Window
^X P       ^R Set Bounds Page
^X R       ^R RMAIL
^X T       ^R Exchange Characters
^X W       ^R Set Bounds Full
^X X       ^R Put Q-reg
^X ^       ^R Grow Window
^X _       ^R Underline Region
```

```
Control-Meta Characters:

Backspace ^R Mark Defun
Tab       ^R Indent for LISP
Linefeed  ^R Indent New Comment Line
Return    ^R Back to Indentation
(     ..  ^R Backward Up List
)     ..  ^R Forward Up List
-     ..  is part of the next command's argument.
0 thru 9  is part of the next command's argument.
;     ..  ^R Kill Comment
?     ..  List Redefinitions
@     ..  ^R Mark Sexp
A     ..  ^R Beginning of DEFUN
B     ..  ^R Backward Sexp
C     ..  exits from ^R mode.
D     ..  ^R Down List
E     ..  ^R End of DEFUN
F     ..  ^R Forward Sexp
G     ..  ^R Format Code
H     ..  ^R Mark Defun
I     ..  ^R Indent for LISP
J     ..  ^R Indent New Comment Line
K     ..  ^R Kill Sexp
M     ..  ^R Back to Indentation
N     ..  ^R Forward List
O     ..  ^R Split Line
P     ..  ^R Backward List
Q     ..  ^R Indent SEXP
R     ..  ^R Move to Screen Edge
T     ..  ^R Mark Sexp
U     ..  ^R Backward Up List
W     ..  ^R Append Next Kill
X     ..  ^R MM Via Minibuffer
[     ..  ^R Beginning of DEFUN
\     ..  ^R Indent Region
]     ..  ^R End of DEFUN
^     ..  ^R Delete Indentation
Rubout    ^R Backward Kill Sexp
```

```
Non-Control Non-Meta Characters:

Backspace moves back one character (or several).
Tab       inserts itself.
Linefeed  ^R Indent New Line
Return    ^R CRLF
Altmode   ^R Prefix Meta
Rubout    deletes characters backwards.
```

図1　1978年12月1日付けのEmacsコマンド表

表1　Emacs コマンドセットの77年版から78年版で変更された部分

項目	変更キー	その機能
1	^h が加えられた	1字戻る（^b と同じ）
2	^i が加えられた	挿入
3	^j が加えられた	indent new line（インデントして次行へ）
4	meta #が変更された	change font region から change font word へ
5	meta $がなくなった	underline region（^x_がその機能をするようになる）
6	meta _が変更された	underline region から underline word へ
7	c-m-Linefeed が加えられた	indent new comment line
8	c-m-j が加えられた	indent new comment line
9	^x^p が加えられた	mark page
10	^x#が加えられた	change font region
11	^x 3 が加えられた	view two windows
12	^x_が加えられた	underline region

コマンドとして実行します．ところが，Emacs では，もともと TECO のコマンドとして存在した機能は画面編集機能上に拡張してもつようになります．

つまり，TECO の場合と逆の関係になってくるのです．これは，言ってみれば，DOS の上に Win 3 があるのを，ひっくりかえして，Win NT ではその中に DOS がある，そんな進歩の歴史を思い起こさせてくれます．

1974年のLispマシン登場

MIT の人工知能研究所（AI Lab.）は，Lisp の発祥の地であり，また，Lisp 自身について，あるいは Lisp を道具として使って，いろいろな研究や開発が進められてきました．

また，兄弟研究所であるコンピュータサイエンス研究所（LCS）と一緒にコンピュータについての先端的な開発をしてきました．その道具として TSS システムをふんだんに使い，また PDP-1，PDP-10 などがフルに活用されます．

Lisp マシンは，こうした中でさまざまな意味合いをもってプランされます．もちろん，Lisp 向きのアーキテクチャの探求とそれに関連した研究が主なのですが，同時にスタンドアロンの個人用のコンピュータ，今でいうパソコン，としての期待も大きいものがありました．

Richard Greenblatt は彼の書いた文献1)の中で，

「午後3時でも午前3時と同じパフォーマンスで動く!!」

ということを強調しています．当時のコンピュータは，すでに多数の利用という状況により昼使うのでは遅くて仕方がないという様子があったようです．

しかし，まだそこでは，開発環境，特にエディタとの一体性ということは特にうたわれていません（なお，Lisp 言語の仕様としては，MUDDLE - CONNIVER との親和性，MacLisp との互換性を述べています）．

統合環境的な意味づけは次の Lisp マシンを待つ必要があります．

CADR マシン〔1973-1978〕と Lamoda マシン〔-1983〕

最初の Lisp マシンは CONS マシンという名前でした．これに続いて，Thomas F. Knight, David Moon, Jack Holloway, Guy Steele らにより，CADR マシン[2]が作られます．CONS マシンの後継の CADR マシンは，Unibus を使ってコンソール，キーボードを付けた専用マシンでした．筆者が AI 研を訪ね出した84年ころには，（つぶれた？）CADR マシンはまだ，tech square の9階に置いてありました．

この CADR は1978年に完成の域に達したとされます．そして，1979年の夏には Lisp Machine Incorporated（LMI）社により Series III という名で商品化の準備がされ，1980年に同社が創立され，それを販売します．

なお，そのパンフレットには，Lisp のメリットとして次のようなことをあげています．

1) 会話性：プログラムやデータを簡単に端末に表示したり，内容を修正したりできる．
2) コンピューティング環境：プログラム編集やデ

バッグのツールとして最も良いものを探しているコミュニティではLispを選んできた．

3) 機能：Lispは高機能である．

4) Uniformity：プログラムとデータは同一の形式をしている．

そして，このCADRマシンでは，EmacsがLisp環境と密接に連係させ，機能発揮させることが述べられています．LMIでのEmacsはZmacsと呼ばれました．

LMI Lambdaの開発は1981にはじまり，1983年に出荷となります．Lambdaでは，CADRの上位互換のLispマシンと68000ベースのUNIX環境が並存されます．

これらのマシンのキーボードは100キーで，ASCIIのスーパーセットとされています．

このLambdaマシンのキーボードについては，文献3)にMoonらが説明を書いています．このキーボードは，Emacsとそしてそれが中心となっている哲学に強い関係があります．次にそれを説明しましょう．

Lispマシンキーボード登場

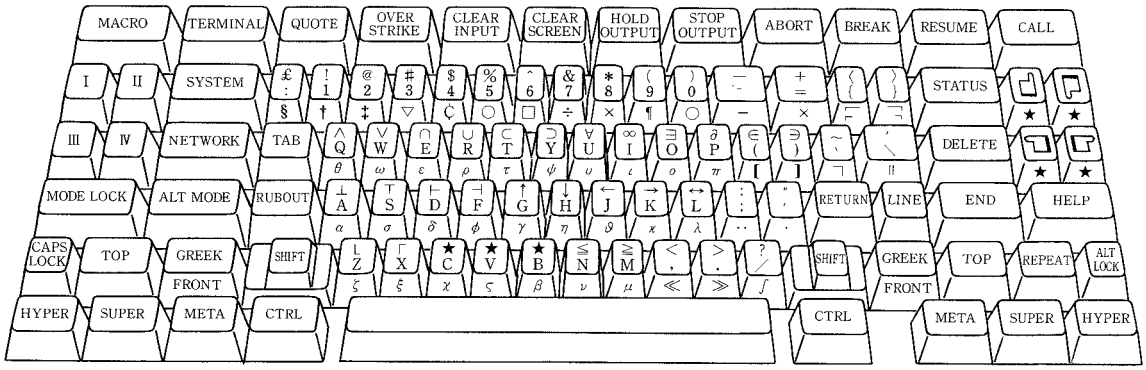
David Moonらは，81年春に出た“Operating the Lisp Machine”というペーパ[3]の中で，「the old Knight keyboardから新しいものに変えた．」と記しています（AI研にいた93年にはTom Nightとも何度も話したことがあったのですが，この「旧型になったナイトキーボード」のことは聞きそびれました．今度会ったときの楽しみにしようと思っています）．

また，「古いキーボードの文字をすべて新しいキーボードでも発生できるが，移行期には数か月トラブルがあるかもしれない．」とも書いています．

文献3)の内容の骨子は，そのままLMI社の説明書に移されています．

この頃のAI研の様子については，Steven Levyの“*Hackers*”[4]などに詳しいですが，商品として研究成果を世に出すことと，コミュニティの共同財産として自分たちの自由度を保とうとする動きの対比，Richard Stallmanがなげいて，独自の世界を構築しようとする過程などがわかります．

このキーボードに関して少し細かな説明をします



★は不明

図2 Lisp マシンキーボード（このキーボードを完全に解明できませんでした．いくつかのキーについて不明，不正確な部分が残っています．ご存じの方は編集部［mtadano@po. iijnet. or. jp］までご連絡ください．）

(図2参照)．n-key rollover 機能とそれに続く説明文は LMI 社の文と同じものであり，100 キーのキーボードだと述べられています（したがって，LMI 社の Lambda マシンは，ここで言われている new keyboard だということがわかります)．

この新型キーボードのキーは，三つのグループに分けられます．ファンクションキー，文字キー，モディファイヤキーです．文字キーは基本的にグレーで，ほとんどのファンクションキーは大きくてブルーの色をしています．

- ファンクションキー：abort, break, call, clear-input, clear-screen, delete, end, help, hold-output, line, macro, network, over-strike, quote, resume, return, rubout, status, stop-output, system, tab, terminal
- 文字キー：英字，数字，特殊文字に alt-mode キーとスペースバー
- モディファイヤキー：mode-lock, caps-lock, repeat, alt-lock, そして，左右に存在する top, greek, shift, hyper, super, meta, ctrl

mode-lock, caps-lock, alt-lock はトグルキーです．shift, top, greek キーは印字文字を生成するモディファイヤです．greek は front とも呼ばれ，各キーの全面に表示された文字を生成します．hyper, super, meta, ctrl キーは主に Emacs で使われ，1字の入力でコマンドする場合に使います．

Lisp マシンでは ctrl, meta に加えて，super, hyper というものがあったのです．たとえば，a という文字に対して，ctrl-a, meta-a, super-a, hyper-a などということができるのです．

alt-lock, mode-lock キーというものがあり，これらの役割はこの文書の書かれた時点では未定とされています．

rubout キー，return キーなどはファンクションキーの仲間に入っています．

Symbolics キーボード

LMI 社に少し遅れて作られ，LMI 社よりも成功を収めた Symbolics 社の Lisp マシンでは，88 キーのキーボードとなりました．図3にそれを示します．

元の Lisp マシンのキーボードから，control, meta, super, hyper のモディファイヤキーは受け継がれています．本稿の文脈に関係するものから紹介すると，alt-mode キーが消えました．かわりに機能的には近い escape キーが加わります．また，delete キーが消えて，backspace キーが入ります．これで，

「rubout=delete で，backspace は別のもの」

という概念の明確化ができたことになります．

greek キーや top キーがなくなって，symbol キーが入ったこと，いくつかのファンクションキーが整理され，funcitonキー，local キー，select キーなどが新設されたことなどがあります．

これらのキーボードは，Lisp とは別の，むしろ Emacs マシンとでも呼ぶべき点もあるものなのです

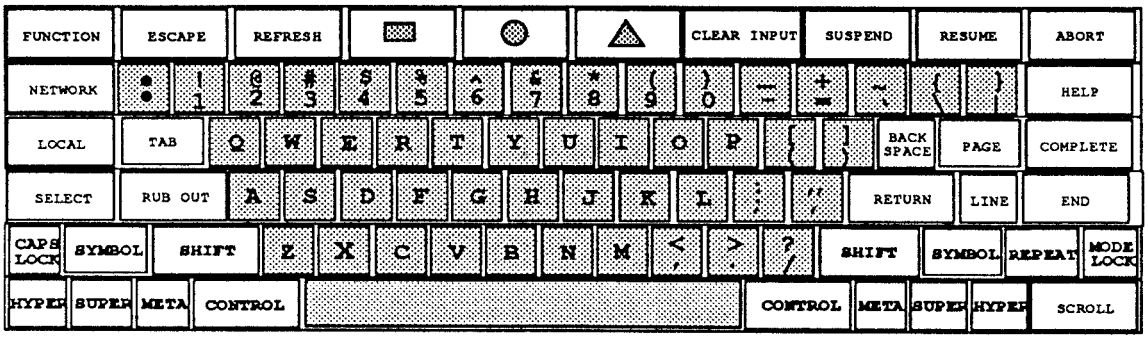


図3　Symbolics キーボード

が，Lisp マシンの盛衰と軌を一にすることになります．いわば，職人用のキーボードとでも言うべきこれらのキーボードは工業製品として標準的ないわゆるASCII キーボードにとってかわられます．

良いキーボード？

コントロールキーとメタキー
——誰にとって良いキーボードか

ここまでで見て来たように，Emacs のコマンドは，コントロールキーおよびメタキーとよぶキーを用いて指示を出します．コントロールキーはほとんどすべてのコンピュータのキーボードに付いています．メタ（meta）キーは現在よく使われているキーボードにはほとんど存在しません．

だとすると，現在の普通のキーボードを使っているEmacs の使い手はどうやっているのでしょうか？答えは簡単で，どうにか工夫をして使っているわけです．パソコンのキーボードでは，IBM に端を発する alt キーにメタを割り付けるハックを利用することがよくあります．そして，それをメタキーとして使うのです．Sun のキーボードとか他のキーボードではスペースバーのそばにある適当なモディファイヤキーをメタキーの役割にさせます．

メタキーとして割り付けるものが存在しないとEmacs は本当は使えないのですが，それでは悲しいので，エスケープキー（esc などと書かれているキー）を押してから対応するキーを押すという2段の操作で同じ意味をもたせるように Emacs の処理系は作られています．

筆者のところでは98などで使う場合は，仕方なくesc を押してから対応する文字キーを押しています．悲し．

そこで，esc をメタキーの代わりにして入力するコンピュータでは，実際的には esc x と書いても，m-x と書いても同じことになります．だから，

「m-x というのは抽象的なキー操作で，エスケープキーを押してから離し，それから x キーを押すということが実際の操作です．」

という説明が見受けられるようになっています．これ

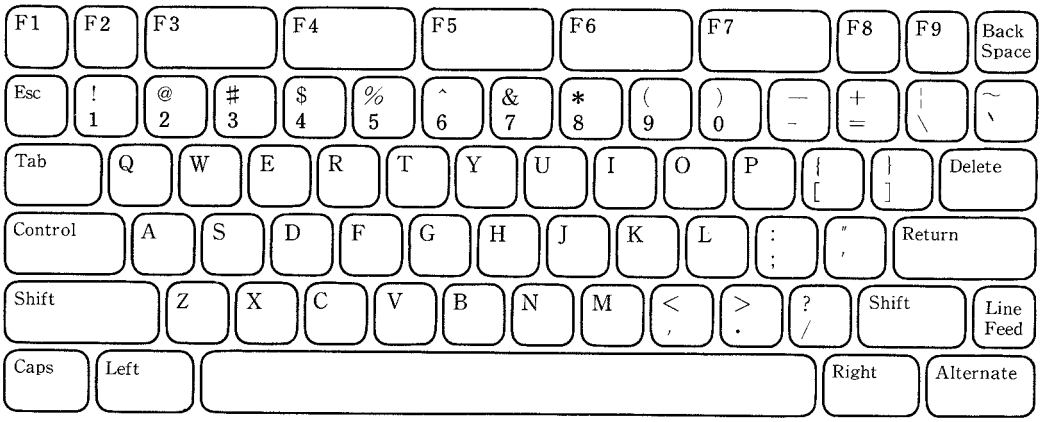


図4 Sun キーボード

も時代の流れかもしれません.

c-h か del か?

Emacs の標準のキーバインドでは,1字抹消は del もしくは rubout, あるいは delete などと書かれたキーになっています. ここまでに触れてきた歴史的経緯をみれば, そして rubout キーが大変押しやすい場所, 左手の小指のホームポジションの隣の位置にあることを知れば Emacs の標準キーバインドの意味もうなずけます. 1文字抹消が rubout であるという設定は大変便利で使いやすいものだと言えます. 入力を間違えたら, ただちに, 左手の小指をちょっと横へ伸ばして, たたけばいいわけです. そうすると, 今打った誤字が消えるわけです. 図2, 図3を見てください.

けれども, 最近の普通の ASCII キーボードあるいは JIS キーボードなどでは delete キーが rubout と同じコードを出しますが, それらは必らずしも使いやすい位置にはありません. たとえば, 図4の Sun キーボードを見てください. これでは, delete キーは右手の端のほうにあります. このキーボードはたいへん筆者にとっては使いやすく離れ難いものなのですが, 右手の小指をぴーんと立てて delete キーをたたくというのは願いさげです. backspace キーならまだ1字抹消にふさわしく使える位置にあるキーだといえます.

backspace キーもよく1字抹消に用いられますが, Emacs の元々のキーの割り付けではヘルプ機能の起動になっています.

さらに, delete キーと backspace キーの位置は, キーボードによって千差万別まちまちです. delete キーあるいは del キーなどと書かれたキーはまったくさまざまな位置に置かれています. みなさんのキーボードを眺めてみてください. こうしたキーに, しかも右手小指が守備範囲のキーに, 重要な働きをさせておくのはいらいらのもとです.

また, 同時によく使う return キーも右手の小指の守備範囲です. しかも, 右手側には1列多く文字キーが並べられています. return も, delete も, backspace も, 左と比べて小指の標準の位置からは遠いのです. できるだけ右手小指のすることを減らしたいと考えていくのは当然のことでしょう.

それで, 筆者は, 1字抹消は backspace 文字でその機能をさせ, それをしかも, (内部コードは同じなので) c-h キーで実行させるということをしています.

c-h は,「もしコントロールキーが使いやすいところにあれば」どんなキーボードでも非常に打ちやすい位置にあります. コントロールキーの使いやすい位置としては一般の最近のキーボードでは a の隣で, Lisp マシンキーボードでは rubout があったところです.

その場合, 左小指と右人指し指をちょっと左めにして同時に押せばいいのです. これのほうが右手の小指をピンと立てて delete キーを押すよりはるかに楽です. このための条件は, コントロールキーが使いやすい場所にあれば, ということです. 図5にあるノートパソコンのキーボードは, ctrl キーが a の横

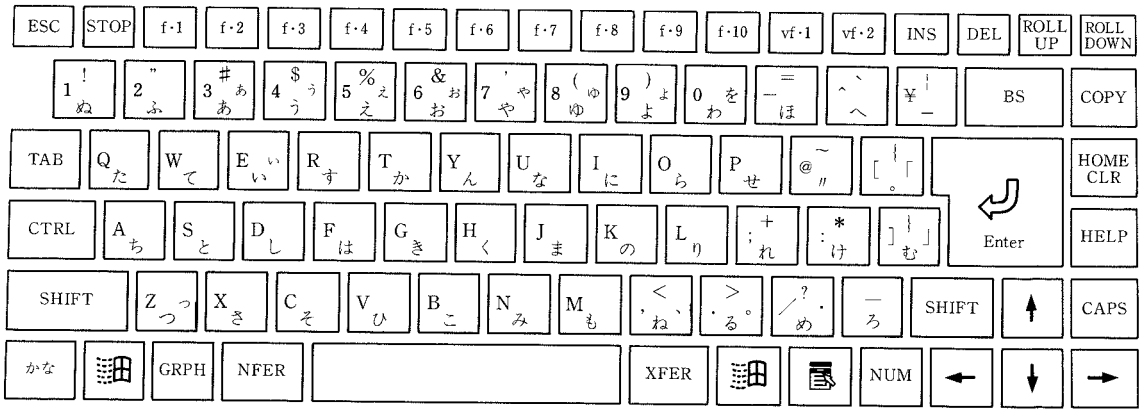


図5　ノートパソコン（98 NOTE）キーボード

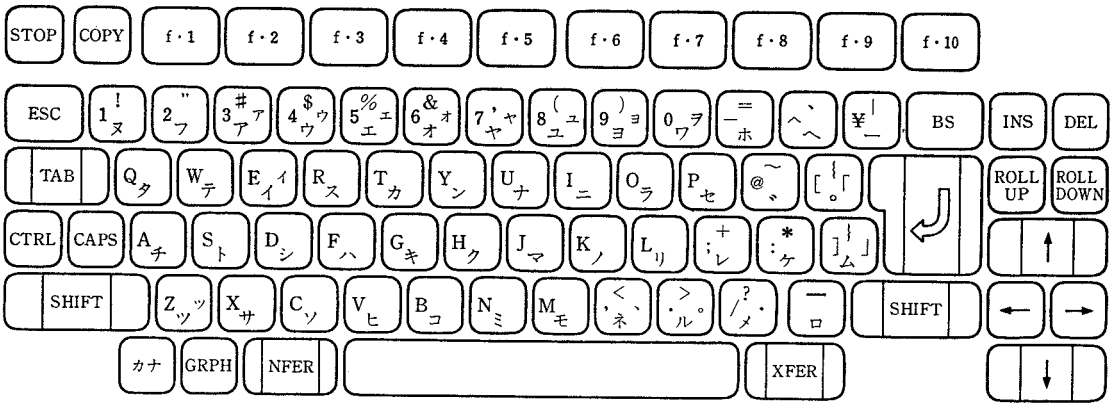


図6　98標準キーボード

にあります．この一点だけで，このノートパソコンは同じ会社の標準キーボード（図6）より格段に親和性がよく使いよい感じがしてしまうのです．

Emacs 標準のキーバインドでは，c-h はヘルプを起動し，delete が1字抹消になっています．これらの機能を交換することを指定できればそれですむことになります．Emacs では簡単にそれができます（たとえば図7）．図7に示すプログラムを Emacs の初期化ファイルに入れておけば，その二つのキーの機能を入れ換えてしまうことができます．

この Lecture では，これ以後も，筆者の習慣と以上に述べた考え方から，c-h を1字抹消とし，del キーをヘルプの起動として割り付けていることを前提としていきます．

```
(let ((the-table (make-string 128 0)))
  (let((i 0))
    (while(< i 128)
      (aset the-table i i)
      (setq i(1+ i))))
  ;; Swap ^HH and DEL
  (aset the-table ?"177 ?"^Hh)
  (aset the-table ?"^Hh ?"177)
  (setq keyboard-translate-table the-table))
```

図7　c-h と del の機能の入換え（bobcat.el より）
（c-h と del の意味だけを入れ換えた表を作って，それをキーの読み換えの表として使うように指定）

リターンキー

もう一つ，ついでに筆者の好みを紹介します．return キーはめったに使わないのです．return キーと同じコードを出す c-m をかわりに使います．そうすると，気持ちとしては

「return をしたければ，左手小指をちょっと左にして，右人指し指はちょっと左下にして同時にたたく」

という感じになります．

これで，右手小指の負荷は著しく軽減されます（c-m が return キーと同じコードを出さない場合には，それなりの工夫をしておきます）．

こうなってくると，左手のコントロールキーの位置は大変重要だということになります．「a」のキーと ctrl キーの間に「caps」などというほとんど役に立たず使わないキーが割り込んで置かれているキーボード，たとえば図6，を使う場合にはこの caps キーを抜いてしまうとかいろいろ工夫をすることになります．こういう工夫のできないキーボードのついているコンピュータは，筆者にとってはまったく利用価値のないただの箱だということになります．

参考文献

1) Richard Greenblatt : The Lisp Machine, Working Paper 79, AI Lab., MIT, Nov. 1974(draft).
（Lisp 専用マシンのアイディアをまとめている．）
2) Thomas F. Knight, David Moon, Jack Holloway, Guy Steele : CADR, AI Memo 528, AI Lab., MIT, May 1979.
3) David Moon, Allan Wechsler : Operating the Lisp Machine, Working Paper 209, AI Lab., MIT, April 1981.
4) Steven Levy : *Hackers*, Dell Book, 1984.

（いだ まさゆき　青山学院大学 国際政治経済学部）

bit悪魔の辞典

インターとイントラ（inter and intra）

進化論の研究によると，生態系において一つの種が巨大なヘゲモニーを獲得すると，必ずその亜種が出現し，種の生き残りを賭けた生存競争が起こるという．インターネットに対するイントラネットもそれではないかと学界では議論されている．ただし，最初に繁栄していたのはイントラネットであり，インターネットに侵蝕されたために，いまは復讐を仕掛けているのだという説もある．

最近『日米の語源事情』（研究社，1996年10月刊行予定）を脱稿し，ますます元気な活動を続けている語源学の大家，吉原源三郎氏の説によると，inter は「相互，間」，intra は「内」という意味だという．（余談だが，氏の卓抜なる研究は JAVA にも言及している．英語はすべからく日本語を語源としていると主張する氏によれば，JAVA は J リーグなどすべからく J をつけたがる最近の日本の風潮と，A∨A という無意味な OR が連接されてできたものであるという．そしてこの無意味な OR とはまさに無意味な「組織（or）」，すなわち ICOT，RWCP，Sun Microsystems などを指すのだという．さらに，日本のプログラマがすべからく JAVA でプログラムするようになると，ダイクストラがセマフォの研究で証明した定理「P のあとにはすべからく V が来る」により，Japanese は Javanese になるとも予言（氏の言葉によれば予源）している．このような鋭い指摘は本辞典の領域をはるかに越えた素晴らしいものである．刊行が待たれる．）

たしかに世の中では，あちこちでイントラへの移行が進行している．その代表例がイントラネットに先立つイントラネック（IntraNEC，「うちくび」と訳されているが語呂が悪いので NEC は公式使用を避けている）であろう．インターネット時代，市場に蔓延するかに見えた PC-98 は，NEC 内部にしか通用しない論理に走ったイントラネックのおかげで，20世紀の黒死病といわれずに済みそうな気配となった．イントラネット時代は，これに類した悪病への感染を防ぐためにファイアウォールという装着器具を使うとともに，安全なインターネット計画ができるように配慮している．出遅れていた NTT もイントラネッティ（IntraNTT の愛称）をスタートし，廃絶の危機にあった「電電語」を復活させようとしている．

インターネットが落ち目なのは，フランスで“Internet porno”（監獄とポルノ）と呼ばれているように，インターネットに手を出すと逮捕される可能性が高くなったからである（これも吉原源三郎氏の指摘である）．フランス語で internat は寄宿舎という意味で，たしかにポルノと病気が蔓延しやすい環境である．