Emacs 解剖学

Lecture 3

Gosling登場

井田昌之

Escape キーの印字表現

第1回と第2回とそれぞれ，Escape（以下，Esc）キーが登場しました．Esc キーの二つの役割が紹介されました．一つは，TECO に関連するもの，もう一つは Meta キーの話に関連するものです．

どちらも，Esc キーは，それ単独では印字されないキーなので，話が多少わかりにくくなる面がありました．

たとえば，私が個人的に知っている方からも次のような電子メールをもらいました．

『*Hackers Dictionary* の TECO のプログラム例は，**bit** 6月号の7ページの ^P の解説でやっとなぜ動くのかわかりました（本来の TECO には組込みのソートなどないもの）．

このプログラム例の最後のほうに“$は escape”と書いてあるのは何かの間違いでしょ．

esc は 00011011 だから 16 進なら 1B，8 進なら 033 ですが...』

（また，**bit**には「エディタとテキスト処理」の連載があり（1982年4月～1983年6月），その6回目が TECO になっているとの情報をもらいました．）

これに対して出した返事は以下のようなものです．

『$ は，altmode キー（IBM キーボードの ALT にあらず）の入力の印字表現です．これを，純粋な ‘$’ の文字の入力と間違える人が多いので，‘0’ と ‘$’ の二重打ちのような字が作られたけれど，多くのプリンタでは単に $ と出ます．altmode キーはその後の ASCII キーボードになって，同じ8進033を出す esc キーに吸収されました．』

このことは第1回の本文にも書きましたが，意外に多くの人が $ と表示されるものが，本当に $ をタイプするのだと思っているらしいことに気がつきました．

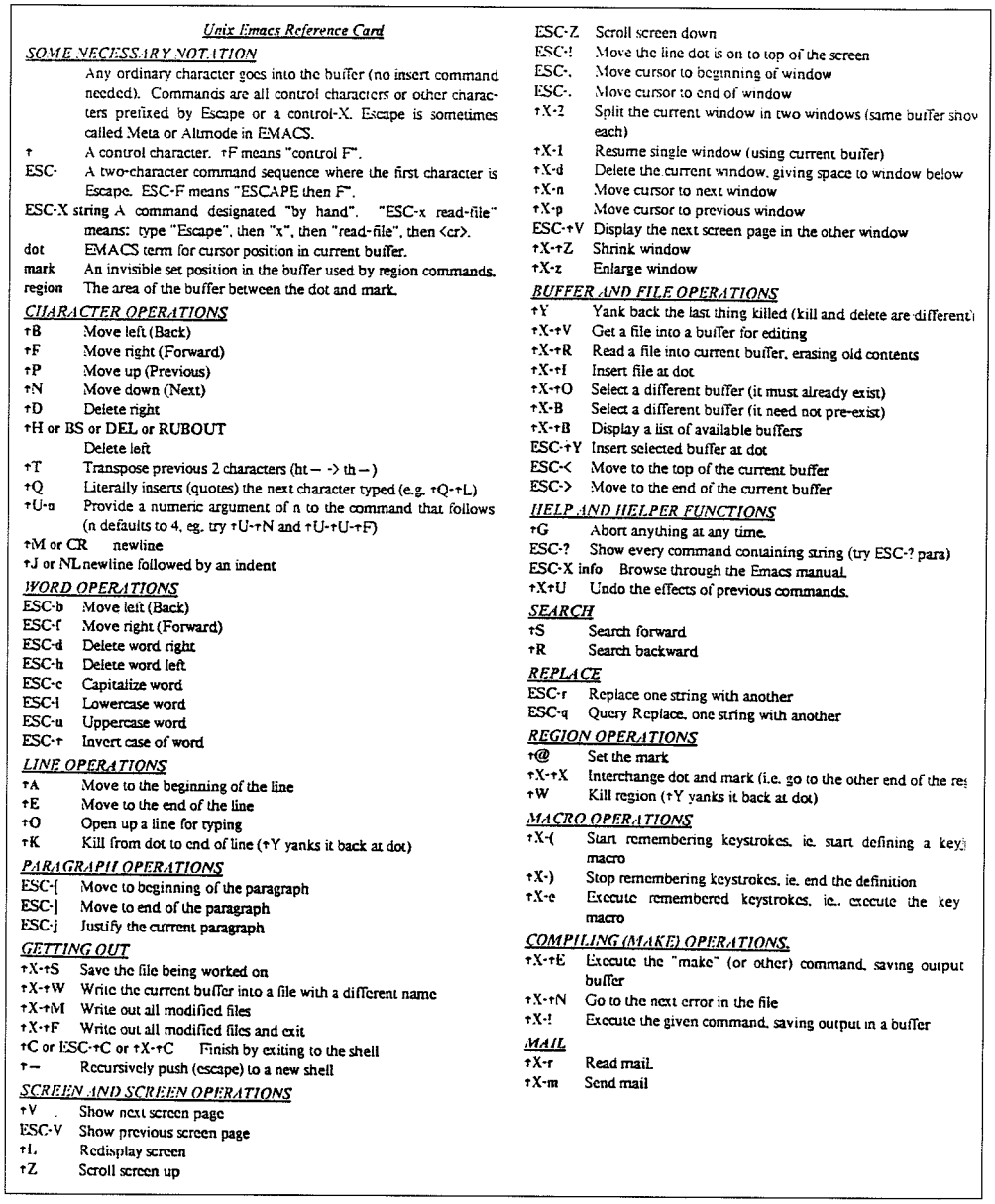
これは，ちゃんと説明をしないといけない，と思って，少し予定を変えて Esc の話を強化し，今回の話とします．

Meta キー：A Funny Key from Keyboards at Stanford and MIT

「Meta-X を ESC-X と教えるシステムもありました」

「Meta-X を，『ESC-X，Esc キーを押してから X を押す』と教えるシステムもありました」と前回書きました．これは，今をときめく Java を作った James Gosling の作った，いわゆる Gosling Emacs がそれです．これを，彼は UNIX Emacs と呼びました．

Gosling Emacs のコマンドサマリを図に示します．Meta-X などというべきところがすべて，ESC-X などとしてあります．また，「control を押しながら」ということを「↑」という表現で表わしています．多くのコマンドは今日の GNU Emacs に共通するものがありますが若干の違いも同時に読み取れます．これらについて順に解剖していきましょう．



Unix Emacs Reference Card

SOME NECESSARY NOTATION

Any ordinary character goes into the buffer (no insert command needed). Commands are all control characters or other characters prefixed by Escape or a control-X. Escape is sometimes called Meta or Altmode in EMACS.

↑ A control character. ↑F means "control F".

ESC- A two-character command sequence where the first character is Escape. ESC-F means "ESCAPE then F".

ESC-X string A command designated "by hand". "ESC-x read-file" means: type "Escape", then "x", then "read-file", then <cr>.

dot EMACS term for cursor position in current buffer.

mark An invisible set position in the buffer used by region commands.

region The area of the buffer between the dot and mark.

CHARACTER OPERATIONS

↑B Move left (Back)
↑F Move right (Forward)
↑P Move up (Previous)
↑N Move down (Next)
↑D Delete right
↑H or BS or DEL or RUBOUT
Delete left
↑T Transpose previous 2 characters (ht – -> th –)
↑Q Literally inserts (quotes) the next character typed (e.g. ↑Q-↑L)
↑U-n Provide a numeric argument of n to the command that follows (n defaults to 4, eg. try ↑U-↑N and ↑U-↑U-↑F)
↑M or CR newline
↑J or NL newline followed by an indent

WORD OPERATIONS

ESC-b Move left (Back)
ESC-f Move right (Forward)
ESC-d Delete word right
ESC-h Delete word left
ESC-c Capitalize word
ESC-l Lowercase word
ESC-u Uppercase word
ESC-↑ Invert case of word

LINE OPERATIONS

↑A Move to the beginning of the line
↑E Move to the end of the line
↑O Open up a line for typing
↑K Kill from dot to end of line (↑Y yanks it back at dot)

PARAGRAPH OPERATIONS

ESC-[Move to beginning of the paragraph
ESC-] Move to end of the paragraph
ESC-j Justify the current paragraph

GETTING OUT

↑X-↑S Save the file being worked on
↑X-↑W Write the current buffer into a file with a different name
↑X-↑M Write out all modified files
↑X-↑F Write out all modified files and exit
↑C or ESC-↑C or ↑X-↑C Finish by exiting to the shell
↑– Recursively push (escape) to a new shell

SCREEN AND SCREEN OPERATIONS

↑V Show next screen page
ESC-V Show previous screen page
↑L Redisplay screen
↑Z Scroll screen up
ESC-Z Scroll screen down
ESC-! Move the line dot is on to top of the screen
ESC-, Move cursor to beginning of window
ESC-. Move cursor to end of window
↑X-2 Split the current window in two windows (same buffer show each)
↑X-1 Resume single window (using current buffer)
↑X-d Delete the current window, giving space to window below
↑X-n Move cursor to next window
↑X-p Move cursor to previous window
ESC-↑V Display the next screen page in the other window
↑X-↑Z Shrink window
↑X-z Enlarge window

BUFFER AND FILE OPERATIONS

↑Y Yank back the last thing killed (kill and delete are different)
↑X-↑V Get a file into a buffer for editing
↑X-↑R Read a file into current buffer, erasing old contents
↑X-↑I Insert file at dot
↑X-↑O Select a different buffer (it must already exist)
↑X-B Select a different buffer (it need not pre-exist)
↑X-↑B Display a list of available buffers
ESC-↑Y Insert selected buffer at dot
ESC-< Move to the top of the current buffer
ESC-> Move to the end of the current buffer

HELP AND HELPER FUNCTIONS

↑G Abort anything at any time.
ESC-? Show every command containing string (try ESC-? para)
ESC-X info Browse through the Emacs manual.
↑X↑U Undo the effects of previous commands.

SEARCH

↑S Search forward
↑R Search backward

REPLACE

ESC-r Replace one string with another
ESC-q Query Replace, one string with another

REGION OPERATIONS

↑@ Set the mark
↑X-↑X Interchange dot and mark (i.e. go to the other end of the re
↑W Kill region (↑Y yanks it back at dot)

MACRO OPERATIONS

↑X-(Start remembering keystrokes, ie. start defining a key macro
↑X-) Stop remembering keystrokes, ie. end the definition
↑X-e Execute remembered keystrokes, ie., execute the key macro

COMPILING (MAKE) OPERATIONS

↑X-↑E Execute the "make" (or other) command, saving output buffer
↑X-↑N Go to the next error in the file
↑X-! Execute the given command, saving output in a buffer

MAIL

↑X-r Read mail.
↑X-m Send mail

図 Gosling Emacs のコマンドサマリ

James Gosling の強い方針

彼の書いた文章を引用してみます（文献1)より)．

『The name meta for the ESC character comes from funny keyboards at Stanford and MIT that have a Meta-shift key which is used to extend the ASCII character set. Lacking a Meta key, we make do with prefixing with an ESC character. You may see (and hear) commands like ESC-V referred to as Meta-V. Sometimes the ESC key is confusingly written as $, so ESC-V would be written as $V. ESC is also occasionally referred to as Altmode, from the labeling of a key on those old favorites, model 33 teletypes.』(ESC 文字をメタなんて呼ぶのはね，ASCII 文字セットを拡張するために，スタンフォードや MIT の連中が，彼らのおかしなキーボードに Meta シフトキーというのをつけていたからなんだ．うちらのキーボードにはそんなものは

ないさ．ESC 文字を前につけることで同じことをさせるんだよ．だからさ，ESC-V を Meta-V なんていうのを聞いたことがあるだろ．それから，ESC キーのことを，まぎらわしく$って書かれる場合があるんだ．そうすると，ESC-V は $V って書かれることになる．もう一つ言うと，ESC は Altmode って言われることもあるよ．これは，むかしの好きものが使ってたモデル 33 テレタイプにあるキーに，そういうラベルがついていたからなんだ．）

また，

『Unix Emacs was called Emacs in the hope that the cries of outrage would be enough to goad the author and others to bring it up to the standards of what has come before.』（私の UNIX Emacs を Emacs と呼んだのは，どんどん強引にやりこめられる風潮に泣いている人たちがいて，それが私を，いろいろあった Emacs の標準を作ろうということに突き動かしたんだ．）

と述べています．

Gosling Emacs は，ある程度は，当時起こり始めていた Emacs をめぐる大きな流れ，つまり Richard Stallman による大旋風と，MIT の AI 研を中心とするもう一つの大旋風に対して，「どっこい，カーネギーメロンでもちゃんとやるよ」というような彼の心意気が随所に感じられる作品なのです．

そして，それは，UNIX 時代を読んで，そのための Emacs ということに先鞭をつけたものでした．1981，82 年頃のことです．またそれは，GNU is Not UNIX という反応も呼びます．

GNU Emacs ばかりが Emacs ではない

Emacs は，もともと TECO のリアルタイム画面モード機能から出たということをお話してきました．そして，Lisp マシンなどでの役割にも多少触れました．

しかし，Emacs は，Richard Stallman が作った GNU Emacs がすべてではありません．ほかの Emacs もあります．この辺の整理からいきます．

まず，TECO のリアルタイム画面モードに端を発し，GNU Emacs へと流れていった Emacs の系統があります．現在ではこの Emacs だけが Emacs だと言われている感があります．もともとのシステムが ITS というものだったので，これを ITS Emacs と呼ぶこともあります．

前回に紹介した Lisp マシンの系統の Emacs は，ZWEI あるいは Zmacs などと呼ばれる（商用）Lisp マシンでの Emacs へと進みます．（これらすべてをこの人がやったわけではありませんが，Dan Weinreb などという人も大きな役割をしていました．彼は，Symbolics を経て，Object Design 社に行きます．）

一方，1978 年頃から，同じ MIT で作られ，80 年代に使われていったけれども，まったく ITS Emacs とは独立して開発されたものに，Multics 用の Multics Emacs があります．これはすべて MacLisp で書かれた Emacs でした．Multics Emacs はその OS である Multics のライフサイクルと軌を一にします．これについては次回に話をします．作者は Bernard Greenberg という人です．

したがって，MIT の人工知能研究所（AI Lab）を母体にするものを考えても，いくつかの異なった Emacs があったことになります．そしてこの近辺には，CCA Emacs など，いくつかのケンブリッジ生まれの Emacs がその後できます．

Gosling Emacs の仕様から

dot とカーソル位置

Gosling Emacs では，dot という概念があります．TECO（リアルタイムモード）あるいはその頃の先駆的画面エディタと同様に，dot は文字と文字の「間」を指しました．文字やテキストの挿入は dot の位置で行なわれます．dot はエディタが処理をする対象の位置を決めるのに重要な役割を果たしていました．

その頃普及し始めたディスプレイ端末で，カーソルは文字の上に載るようになっていました．文字の間ではありません（言い換えると，実際の画面上で，文字の間に何かのマークを表示するような方法は次第にすたれてきたということです）．

そこで，カーソルの位置と dot の位置との関係を定義する必要が出てきます．

- 「カーソル位置は，dot のすぐ右の文字位置を指す」

ということが決められます．

これで，実際の操作感覚ともかなり合った仕様ができます．一つだけ，考慮すべきことが残ります．それは，削除の定義です．dot は文字と文字の間を指しているので，文字を削除しようとするときにどの文字になるのかをちゃんと定義しておかなければなりません．

それで，「dot の右を削除する」という操作と「dot の左を削除する」という操作を区別して定義することになります．この結果，

- control-D (^D) は，dot の右の 1 字を削除する
- control-H (^H)，BS，DEL，RUBOUT などと書かれたキーは，dot の左の 1 字を削除する

というように定められます．

この仕様では，というか，この頃の操作概念では，^D と DEL は関係付けられていないこと，^H や BS と DEL/RUBOUT などは同じ機能のように考えられていたことがわかります（最近では，DEL と ^D が似たものとしてくっつけられるケースが多いでしょう）．

また，ここまでの議論からも察していただけるかもしれませんが，Gosling は，現在標準的になった ASCII キーボードをベースに考えていました．この点では，凝ったキーボードや表示を考えていた当時の MIT 的なアプローチより先を見ていたということが言えるかもしれません．もっともこうした比較はすべて結果論ですが．

UNIX 用だということ

Gosling Emacs は，基本的に今日の GNU Emacs に流れている性質をもっています．同様のスクリーン構成，Lisp 型言語による機能拡張/カスタマイズ，キー操作の基本部分などについてです．Gosling Emacs は，ITS でも Multics でも Tenex でもなく，あるいは Lisp マシンでもなく，UNIX 用の Emacs として設計され，作られました．ここに特徴があります．

具体的にはどんなことでしょうか？ 次のようなキーワードにまとめてみました．

（1） shell との会話機能
（2） 端末属性の取得（/etc/termcap）
（3） make のサポート

シェルとの会話機能をもたせるという概念，端末属性の取得の二つは，ごく自然なアイディアであったように思います．ベースになるシステムの機能を利用しようということはごく自然だからです．Emacs の中にいるままで，シェルを使えるようになります．また，画面のサイズだけでなく，カーソルの制御方法その他の細かな端末属性を扱えるようになります．

三つ目の make のサポートに注目したいと思います．彼の書いた文献 1)には，彼の思い入れがこの機能の紹介とともに書かれています．UNIX 的なプログラム開発スタイル，特に今日でもよく使われる使い方であるエディタでソースを修正しながらテストを進めていく考え方が，Emacs に与えた機能の説明とともに書かれているのです．

この Emacs の成立はおよそ 1982 年初頭なので，これもなかなか先見性のある仕組みだと考えられます．

具体的にどうなっているかを書いておきます．

まず，makefile を用意しておきます．そして，開発したいプログラムのソースモジュールをそれぞれバッファに広げます．これらをデバッグしていきます．コンパイルしてテストするにはちょこちょこと直したそれぞれのソースをファイルに書き戻したうえで，make をかけ，実行させるわけですが，Gosling Emacs では，^x^e をたたくと，すべての作業中のバッファに対して，ファイルへの書き戻しが行なわれ，その後，make が起動されます．make にエラーがあれば，それが順に，あるバッファに入って表示されていきます．したがって，開発中はソースを直して，^x^e をたたくということになるのです．この機能を compile-it と呼びます．

用意されている機能

Gosling Emacs に入っている機能は次のようなものがあります．

- c-mode，dired，ITS 流 incremental search，process，rmail，scribe 対応，spell，tags など

また，キーマップによって各キーが実際に対応する機能を柔軟に変更することができます．

マクロ機能があります．さらに，言語が用意されていて，それによって，機能拡張をすることができます．

MockLisp

Gosling が用意した機能拡張用言語は，MockLisp という名前です．MockLisp のソースプログラムは，Lisp らしい表現形式，つまり，カッコで囲むプログラムスタイルをしています．けれども，リスト処理機能はなく，cons といった関数も用意されていませんでした．あくまで，エディタに特化していて，文字列の変換や，バッファの処理などに関する機能に重点が置かれていました．また，関数を定義するためには，defun が用意されていましたが，その文法は，同じ名前で多くの Lisp システムにあるものとは，引数の受渡しの記述などがまったく異なっていました．

たとえば，次のように書きます．

```
(defun
    (foo x (setq x 5)(bar x)))
```

これは，foo という名前の関数を定義しています．それには，ローカル変数として x がとられ，実行されると x に 5 が入れられ，そして，別の関数 bar が呼び出されます．ローカル変数が複数ある場合には，カッコに入れずに，ただ単に並べていきます．

引数を考える場合，arg ないしは argv という関数を利用します．

arg は，その実行により引数をもってきます．引数は複数与えることができ，何番目かを 1 から始まる番号で指定します．また，その関数がコマンドとして実行された場合，引数を束縛するのではなく，プロンプトを表示して，そこで入力を求め，タイプされた値を返します．

したがって，arg の書き方の一般形は次のようになります．

```
(arg i [prompt-string])
```

これを利用して以下のような定義があったとします．

```
(defun
    (bar(+(arg 1 "Number?")1)))
```

そうすると，この関数 bar が他の MockLisp 関数から呼び出されたときには，その引数の値がとられ，直ちに 1 との加算がおき，答が返されます．もし，bar が会話的に呼び出されると，

```
Number?
```

と表示して，数値の入力を待ち，その値に 1 が加えられて返されます．なかなか便利な機構だと思いませんか？

これは，定義された機能がエディタコマンドとして起動されるメカニズムから，引数の束縛ということをすべて取り去り，シンプルにした結果だと考えています．(なお，GNU Emacs では，伝統的なちゃんとした Lisp のスタイルをとり，それに，interactive 宣言をすることで，関数呼出しとして実行された場合と会話的に呼び出された場合を切り分けながら，一つの定義で両者を記述するようになっています．)

もう一つ，argv という関数があります．これは，Emacs が UNIX のコマンドとして起動されたときのコマンド行に書かれた引数をとってきます．

```
shell $ emacs hello
```

として呼び出したとすると，(argv 1) により “hello” という文字列が得られます．

MockLisp に用意された機能のうち，一部を紹介しましょう．ランダムに選ぶという意味で，最初に 1 字が ‘c’ で始まるものを並べてみます．

- c-mode，c=（文字の等値），case-region-capitalize，case-region-invert，case-region-lower，case-region-upper，case-word-capital-

ize, case-word-invert, case-word-lower, case-word-upper, change-current-process, change-directory, char-to-string, checkpoint, compile-it, concat, continue-process, copy-region-to-buffer, current-buffer-name, current-column, current-file-name, current-indent, current-process, current-time

これらの多くは，文字列ないしは文字の処理に関連する機能です．この比率は全体を見てもそれほど変わりません．

このようなこと全体から，MockLisp が Lisp に似ているのは，カッコで囲んで記述するその全体的なイメージだけだ，という人がいるかもしれないと Gosling 自身が述べています．

MockLisp に存在するもの

まず，MockLisp は，インタプリタベースで，MockLisp で書かれたプログラムをコンパイルするというような概念はどこにもないようです．(compile-it という make の起動は別のものです．) そして，「ソースプログラムは，リストとして内部表現され，…」というような，一般的な Lisp にあるような記述はありません．むしろ，MockLisp プログラムは，文字列として存在し，文字列のままで実行されます．したがって，(foo x) という式があるとしましょう．ふつうの Lisp では，「(foo x) が読まれると，リストに変換され，これが実行されようとするとき，car 部の foo というシンボルに与えられている機能が，cadr 部の x のそのときの値を引数として実行される」というようになるのですが，MockLisp では，「手続きの呼出しは (foo x) のような形をしています．つまり，文字 '(' があり，そのつぎに 'foo' が置かれ，一つ以上の空白文字があって，その後に 'x' のような引数の名前の文字列，最後に ')' が置かれます．この部分が実行されようとすると，foo の実体，そして x の実体が調べられ，それらで解釈実行が行なわれます．」ということになります．

この MockLisp には四つの存在しかありません．

整定数，文字列定数，名前，手続き呼出しの四つです．なお，MockLisp では，関数呼出し (function call) という言葉は極力避けられていて，手続き呼出し (procedure call) という言葉が代わりに用いられています．

整定数は，10 進数です (これに対して，当時よく使われた Lisp ではデフォルトは 8 進数が多かったのです)．

文字列は，引用符 (") で囲んだ文字の並びです．文字列と整定数は可換です．

名前は，変数や手続き名などに使われます．変数には，スコーピングルールがあって，グローバル，ローカル，そしてバッファローカル (正確に Gosling の用語を使えば，buffer specific) の 3 種類に分かれます．ローカル変数の実体は，それが宣言されたブロックの中だけで有効で，そのブロックの実行から出ると消滅します．

手続き呼出しは，再帰的呼出しも許されています．

また，mark と dot という関数だけが返せる値として marker という存在があります．これは，バッファ名の文字列とそのバッファ内の dot 位置からなる特殊な構造です．もし，数がほしい文脈で使われると dot 位置が，文字列がほしい文脈で使われるとバッファ名が返されます．

Emacs から Java へ

James Gosling の名前は，現在では Java の開発者として有名です．その前は Sun の NeWS というウィンドウシステムの設計で有名でした．これは，PostScript をディスプレイに適用しようとした最初の試みの一つだと言えます．さらにその前にはいろいろなことをしていたようですが，この Emacs が大きな仕事だったと思います．

これらに共通するものは何だろうか，ということに興味が出てきます．細かなことで，彼のクセややり方について共通点を見いだすことも可能ですが，それらは別の機会に譲るとして，私は言語による仕組みを用意しておいて，それによってダイナミックなユーザ環境を志向しているというのが見るべきことではないかと思っています．

ところで，Java ですが，その言語仕様書はまだ刊行されていません．その理由に関係するエピソードがあります．それは，James Gosling と Bill Joy の意見の違いがあって，仕様を一つに決められないで困っていたが，Guy Steele が Sun に入ってさっそく両者の

調停によばれ，一日できれいに解決して，それで，言語仕様の策定作業はGuy Steeleに任されて現在に至っているというものです．これは，大筋で真実のようです．彼は5月現在で，なおも必死に仕様書の原稿と格闘しています．また，筆者はそれに対応してできうる限りのコメントを送っています．それらを反映して文献2)を書きました．

特にアメリカで，新しい仕組みを考え出し，それを実現しようとする人たちの，個性というか，頑固さというか，その執念は，時としてall or nothingの選択になり，捨て去られた技術もたくさんあるわけです．逆に実った場合には，芸術作品ともいうべきスタイルできれいにまとまっているように思います．細部をおのおの見ていくと，結構良くない部分も見えてくるのですが，それはそれで，一人の人の作品と考えると，いちいちあげつらわずに，そんなものだろうと受け止めるべきだなとも思います．

Emacsの世界では，たくさんのEmacs的処理系がありました．それらの中で，Richard StallmanのGNU Emacsが勝者となっていきます．だんだん，彼のEmacsの話に入っていきます．

第1回への反響から

このLectureの第1回「TECO——Emacsのみなもと」に対して，いくつかの連絡をいただきました．

それから一つを紹介します．

8月号12ページの ^V について．

「^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．」としておいたところ，takagi@jpitc.japan.eds.comさんがメールで，説明を送ってくれました．

takagiさんは，DEC-20のTECO/Emacsを使っていたそうです．MIDASアセンブラでインプリメントされていて，その上にTECOのライブラリが載っていて，それからダンプされたEmacsだそうです．

以下に引用します．

『この説明ですが，英文を素直に読むと以下のような解釈だと思います．

まず，前提ですが，

- FS IBASE$ には，数字引数の基数(base)が入っている
- すなわちn進数のnが入っている
- 通常はもちろん10か8です（この辺がITS/Tenex/Twenex）

^Vは，

^V-123Xのように正または負の数字引数と数字でない最初の文字をコマンド引数として取り，数字引数をFS IBASE$を基数とした数に変換してXで指定されるコマンドに，ここで計算された数を引数としてわたす．

^Qがその次の文字をliterallyすなわちコマンドではなく，文字そのものとするquoteの機能であるのに対して，universal argumentの機能を実現するものです．

具体的な使い方としては，^V10^Nと打ち込むと，10行下へ行くというような感じになります．

現在のGNU Emacsでは^Uがこの機能になっているものだと考えられます．GNU Emacsで^Uとやると，画面の下のエコーバック領域に

```
C-u-  と出て，以下数字を打つと
C-u 1-
C-u 1 2-
```

のようになり，コマンドを打った途端に（通常は）その回数分コマンドが繰り返し実行されるのはご存知と思います（^U10^Nとやると10行先に行きます）．

元の文章をもう少し忠実にに訳せばこんな感じでしょうか．

「^Vは次のコマンドに対する基本引数を設定する．引数は数字と（必要ならば）マイナス符号から成り，スクリーンの一番下にエコーバックされる．この引数は現在の基数（FS IBASE$）を用いて数に変換される．最初の非数字文字が引数を終了させ，その（訳注：今入力された非数字の）文字がコマンドとして扱われる．^Gは引数を入れなかったことにする．」

実際にGNU Emacsで^Uで引数を入れている最中に^Gを打つと引数がキャンセルされますが，GNU Emacsでのキャンセルは^Gが引数を無視してキャンセルを行なうコマンドだからなの

で，本当はちょっと違うのです．でも，動作としてはまさにこの記述にあっていると思います．

お役に立てましたでしょうか？

高木 拝』

この辺は使ったことがなく，ゆっくりと時間をかけて調べるのをはしょっていました．それを高木さんが補ってくださいました．紙面をお借りしてお礼申し上げます．

Emacs についての情報をください

Emacs には，いろいろなファクタがあります．また，いろいろな処理系がありました．GNU Emacs とそれに対する Richard Stallman 氏の業績がその大部分であることは言うまでもありません．けれども，Emacs にはいろいろな人の貢献と，いろいろな見方が練り込まれていることを感じます．そういう点でも読者の方々からのいろいろな意見/資料を歓迎いたします．

参考文献

1) James Gosling : "*Unix Emacs*", CMU, May, 1982.
2) 井田昌之：『はやわかり Java』，共立出版，1996.

（いだ まさゆき　青山学院大学 国際政治経済学部）
[ida@sipeb.aoyama.ac.jp]

bit 悪魔の辞典

ポスト（post）

「後(あと)の」という意味の接頭辞．計算機に関するいろいろな言葉を造るのに使われる．「遅れてくる」という意味にも拡大解釈され，単独で「郵便物」を意味することがある．

◆**ポストモーテム・ダンプ**（post-mortem dump）：バグで計算機が停止した後に，計算機のメモリの内容を打ち出すこと．UNIX 文化ではコアダンプともいうが，誰も読む人はいない．ふつうは「後の祭」と訳される．

◆**ポストモダン**（post-modern）：意味はよくわからないけれど，気取りたいときに使う言葉．一部の計算機エリートたちはポストモデムとなまった発音をし，やっとモデムが使える程度で喜んでいる一般大衆を馬鹿にしている．

◆**ポストイット**（post IT）：IT（Information Technology）の次に来るもの．現在は IT が全盛であるが，いつかはブームが去る．住友3M（Money, Mortgage and Market）はそれが紙であると読み，Post It™ という商品名で売り出した．好調である．実際，最近の統計によれば，パソコンブームにより，製紙業界は空前の売れ行き増になっている．ペーパレス社会は嘘であった．

◆**ポスト問題**（Post-problem）：性差別用語撤廃を進める米国では，ポストポストマン（post-postman，郵便配達夫の後）はポストパーソン（postperson，郵便配達人）である．すなわち，post を演算と考えたとき，$post^2(man) = post(person)$ である．このような単語の対応に関する一般的な問題をポスト問題という．論理学者の Post が提案したので余計ややこしい．

◆**ポストスクリプト**（Postscript）：計算機のディスクの余白を埋めるための「後書き」言語．1枚の絵だけで十分にディスクを埋めることができるので，効率がよい．

◆**ポストオフィス**（post-office）：分離・分割，さらには人事をめぐって郵政省と NTT は激しい闘いを繰り広げている．NTT では，郵政省を時代遅れの官僚的役所（オフィス）といい，郵政省はポストオフィスへ脱皮しなければならないと主張している．しかし，庶民はすでに郵政省の出先機関のことをポストオフィスと呼んでいるので，いまいち迫力がない．郵政省のほうがはるかにしたたかで，電気通信局長室には〒〒という社名の入ったポスターが貼ってある．