複数の計算機システムにおける共通コマンド体系の画面エディタの開発

本田道夫
中村邦彦

I　はじめに

エディタは，FORTRAN, C, COBOL, LISP, PROLOG, アセンブラ等の各種プログラム言語によるプログラムの作成・修正を行う場合には必須のものであり，またそれらのプログラムのためのデータの作成・修正を行う場合にも用いられる。このようにエディタは，計算機を使用するにあたって最もよく使われる必須のソフトウエアの１つであり，そのためエディタは，各計算機ごとに，あるいはそれらの上でのオペレーティング・システム (OS) ごとに，それぞれ独自のものが用意されており，通常は購入時に付属してくる基本ソフトウエアに含まれている。しかし，このようにシステム独自のものが用意されているために，計算機システムが異なれば，使用できるエディタが異なることになり，操作に戸惑うことがしばしばである。例えば，本学において我々が使用できる計算機は，パーソナル・コンピュータ（パソコン）から計算センタの計算機まで合計８機種のシステムがあり，さらにそのうちの何台かは複数の OS が使用可

能であり，結局，計10種類以上のエディタを使い分けなければならないことになる。

また，従来，エディタは，タイプライタ方式の端末装置を想定した行エディタ (line editor)，あるいは文字列エディタ (string editor) と呼ばれるものであったが，最近のほとんどの計算機システムでは，端末装置はCRT端末であり，上記のエディタよりも強力な画面エディタ (screen editor) が備わってきている。実際，画面エディタと行エディタあるいは文字列エディタを用いたプログラムの作成・修正ではその効率において何倍もの差がある。しかし，本学において使用できるシステムに付属しているエディタは，最初は，ごく一部のものを除いては，すべて行エディタであるか文字列エディタであった。

このような状況から，我々は，「種々の計算機システムおよびOS上で同じ命令体系の画面エディタを使用できること」を目標に「マイクロEMACS/JMACS」という画面エディタを，徐々に各計算機システムおよびOS上に実現してきた。このエディタは，アメリカのマサチューセッツ工科大学 (MIT) で開発された「EMACS」[1]という画面エディタを参考に，その操作命令のうち比較的よく用いられるものをパソコン上に実現したものである。マイクロEMACSと同じ命令体系を有し，全角の日本文字も扱えるように開発したものがマイクロJMACSである。(ただし，以後特に断わらない限り，どの場合であるかは，前後の関係から明かであることが多いので，「マイクロEMACS」，あるいは単に「EMACS」とは，マイクロEMACSのみ，あるいはマイクロEMACSとマイクロJAMCSを指すとする。MITのEMACSを指すときは，MIT-EMACSと呼ぶことにする。)

なお，我々の作成した「マイクロEMACS」は，1980年当時京都大学理学部数学科の学生であった萩野達也氏が，8ビットのパソコンのCP/MというOS上に実現したもの，およびその後，同氏が16ビットのパソコンのCP/M−86というOS上にアセンブラ言語で記述・実現したものを出発点としたものである。ただし，現在のアセンブラ言語で記述されたものは，他のOS上への移植，処理速度の向上，画面表示速度の向上，日本語の処理機能等の種々の改良・機能拡

張のため，処理方式はかなり大幅に変更されている。また，同様に大幅な処理方式の変更を加え，他システムへの移植の容易な言語Cで記述したものの作成も行った。[(1)] これらの経験から，我々なりの画面エディタの処理方式が確定してきたので，「マイクロEMACS」の基本的な概念について簡単に説明し，その実現のための処理方式の要点，およびそのような方式を採用した理由について述べる。なお，命令体系の一覧は付録としてつける。

II　マイクロEMACSの基本的な概念

一般に，プログラムやデータあるいは文書のように，エディタの対象となるものを「テキスト」と呼ぶ。現在EMACSが対象とするテキスト中の文字は，ASCIIコード表により定義される半角文字の英数字，特殊記号，カタカナおよび制御文字であり，JMACSでは上記の文字以外に，さらに全角文字の平仮名，カタカナ，および漢字等の日本語に用いられる文字が含まれる。全角文字の縦の大きさは半角文字と同じであるが，横の大きさは半角文字の2個分の大きさで表示される。また制御文字は「^」と英大文字の2個の半角文字を用いて表示される。このため，制御文字と半角文字2個とは見ただけでは判別できないが，カーソルを移動することにより判別できる。なお，以下では画面には半角文字で縦MAXROW行，横MAXCOL列の表示ができるものとして説明する。(通常は，縦24行，横80列である。)

テキストは，計算機の補助記憶であるディスク上にファイルとして保存されており，エディタによる編集は，まず編集したい部分（普通は，可能ならばテ

(1)　8ビットのパソコンへの改良・移植は，クロメンコ製CP/Mシステムへは本田が，富士通製FM-8のCP/Mシステム，ザイログ製MCZシステムには中村が行った。16ビットのパソコンへの改良・移植は，日本電気製PC-9800, N5200, 富士通製FM-11, 三菱電機製MULTI-16のCP/M-86システム上，および日本電気製PC-9800, 富士通製FM-11, R-50/60のMS-DOSシステム上へは本田が行った。言語Cでの記述は，最初に中村が，ザイログ製SYSTEM-8000のUnixシステム上に行なった。それはさらに本田が機能拡張し，SYSTEM-8000, 東芝製UX-300, DCL製U-Station, 立石電気製Super-Mate, SX-9010, およびサンマイクロシステムズ製SUN-3のUnixシステム上，PC-9800のMS-DOSシステム，およびPC-9800, FM-11のCP/M-68Kシステム上へ移植した。日本語の処理への拡張は，アセンブラおよびC双方とも本田が行った。

キスト全部）を計算機の主記憶上に読み込み，それを編集し，編集後はファイルに書き出すというように行なわれる。つまり，編集がおこなわれるのは主記憶上に読み込まれたものであり，直接ディスク上のファイルの修正が行なわれているのではない。（ただし，ユーザはこのことを意識する必要はない。）新しくテキストを作成する場合には，最初の読み込みが行われないだけで以後は同じである。以下では，主記憶上のテキストを読み込むところを「テキスト・バッファ（あるいは単にバッファ）」と呼び，テキスト・バッファに読み込まれ，編集されつつあるものもテキストと呼ぶことにする。EMACSでは常にバッファ上のテキストの一部が画面に表示されており，その部分の各文字と表示されている画面上の各文字とは1対1に対応している。

EMACSでは，表示画面の連続性を保つために，現画面からテキストの先頭方向へと画面が移るときは，現画面の先頭の数行（通常は4行に設定している）は，次画面の最後の数行として表示され，逆にテキストの最後の方向へと画面が移るときは，現画面の最後の数行が次画面の最初の数行として表示される。以下では，このように重複して表示される行の行数をOVLL (OVerLap Lines)として説明する。

EMACSは編集のための命令としては，カーソルの移動，画面の制御，画面分割，文字／文字列の削除，文字列の検索，文字列の一括置換等の，通常の画面エディタに見られるような命令以外にも，文字列の選択置換，マクロ命令の定義等の非常に強力な命令，および大文字／小文字変換，ファイルの表示・操作，エディット対象のファイルの変更，計数表示，命令の取り消し，繰り返し実行等の命令が用意されている。これらの命令の一覧は付録として与える。以後このようなエディタに対する命令を「コマンド」ともいう。

EMACSでは，キーボードから入力した文字は，画面に表示されているカーソルの位置する文字とその前の文字の間に挿入される。また，テキストの先頭方向への1文字削除のコマンドにより，カーソル位置の前の文字が削除され，テキストの最後の方向への削除は，カーソル位置の文字から削除される。従って，画面上のカーソルの対応するテキスト上の位置は，カーソルの位置する文

字の位置とその前の文字位置の間であると考えればよい。なお，以下では画面上のカーソルの位置を「画面カーソル位置」あるいは単に「カーソル位置」といい，画面カーソル位置に対応するテキスト上の位置を「テキスト・カーソル位置」ともいう。

なお以下の説明中で，「Ctrl-α」とは，キーボード上のコントロール・キー（キーの上にCTRLと記されているキー）を押したままで，あるキーαを押す操作を示し，「Esc-α」とは，エスケープ・キー（ESCと記されているキー）を1度押して離した後に，あるキーαを押す操作を示すこととする。

III　テキストの管理と基本的命令に対する処理方式

上記のコマンドを有するエディタの実現方法について，テキスト・バッファの管理方式，画面表示とテキスト・バッファの管理方式，基本的なコマンドの処理方式の観点から説明する。これらを決めれば，プログラムでの処理方式をかなり定めたことになる。なお，以下では単にテキストとは，テキスト・バッファ中のテキストを示すものとする。

1.　テキスト・バッファ中の文字コード

IIで述べたように，EMACSでは，半角文字の英数字，特殊記号，カタカナ，制御文字，全角文字の平仮名，カタカナ，および漢字等の日本語に用いられる文字をエディトの対象としている。現在EMACSを移植しているシステムでは，半角文字および制御文字は1バイト，全角文字は2バイトのコードであらわされる。制御文字には「改行」，「タブ」，「改頁」等の意味が定められたコードがあり，ディスクのファイル中では，これらのコードはその意味で用いられてテキストが管理されている。そこで，EMACSのテキスト・バッファ中でも，これら制御文字についてはファイル中と同じ意味で扱うことにしている。なお，半角文字，制御文字と全角文字の最初の1バイトは，当然異なるコードとなっているが，全角文字の2バイト目については，制御文字と重複することはないが，半角文字とも全角文字の1バイト目とも重複することがある。

画面エディタは，画面カーソル位置およびテキスト・カーソル位置を常に把握しておかなくてはならない。そして，たとえばカーソルを左側に1文字分移動させるときには，カーソルの左側の文字が半角文字であれば，画面上では半角文字分左にカーソルを移し，それに対応する位置はテキスト上では先頭方向に1バイト分移ったところとなる。全角文字の場合は，画面上ではカーソルを全角文字分，それに対応して位置はテキスト上では2バイト分先頭方向に移さなければならない。このように，カーソルを左側に移す場合には一般的にはカーソルの左側の文字が全角か半角かを判断しなければならない。

扱う各文字が1バイトの半角文字だけであるEMACSでは，この判断の必要が無く，従ってテキスト・バッファ中には半角文字の1バイトのコードを順に配置しても特に処理方式が複雑にはならない。タブ等の制御文字の場合には，カーソル位置の計算は，半角文字の場合とは異なるが，それほど複雑とはならないし，1バイトのコードであるのでテキスト中では半角文字と同じ処理をすればよい。

一方JMACSの場合は，カーソルの左側の文字が全角であるか半角であるかの判断が必要である。もし，テキスト・バッファに半角文字と全角文字をそのままそれぞれ1バイトおよび2バイトで配置した場合，カーソルの前の1バイトだけでは，上に述べた理由からそれが半角文字であるか全角文字の2バイト目であるか判断できないことがある。カーソルの左側を2バイト以上調べても判断できない場合もある。従ってこのような配置では何らかの情報が必要であり，処理方式は半角文字だけの場合に較べて複雑になる。

ゼロは全角文字の1バイト目のコードにはなりえないので，半角文字に対してゼロを1バイト目のコードとして付け加え，全ての文字を2バイトとしてテキスト・バッファに配置すれば全角／半角の判断は簡単にでき，処理方式も複雑にはならない。しかしこの方式では半角文字は2バイトになるので，利用効率が悪く，テキスト・バッファとして利用できる領域がそれほど大きくない場合には扱えるテキストの大きさがかなり小さく制限される。

このことから，JMACSでは，16ビットのパーソナル・コンピュータのよう

に，特別な工夫をしない限り一度に扱える主記憶の大きさが制限されている場合には，テキスト・バッファがあまり大きくとれないので前者の配置を，32ビットのシステムのようにかなり大きなバッファが利用できる場合には後者の配置をとっている。

2. テキスト・バッファ中の行と画面上の行

1.で述べたように，テキスト・バッファ中で行の区切りは1バイト（あるいは1バイト目がゼロである2バイト）の「改行」コードを用いている。つまりテキスト中では改行コードと改行コードの間を1行とみなし，特に行の長さを制限することはしない。一方画面上に表示できる行長は当然物理的に制限される。従ってテキスト中での1行は，画面上の1行の長さを超える場合も有り得る。このような場合EMACSでは，テキスト上の長い行は画面上では何行にも渡って表示する。ただし各行の最後に「!」を表示し，テキスト中では連続した1行であることを示している。

上記の方法以外に，IBMのフロッピィディスクの管理の1方式，および初期のいくつかのワードプロセッサに見られるような，1行の最大の長さを固定し，その長さに満たない行は，行末を空白で埋める方式も考えられる。しかし，エディタが対象とするテキストにおいて行長を制限することは一般には好ましいことではない。また長さの極端に異なる行が混在する場合には，固定行長の方式ではバッファを有効に利用できないことにもなる。以上の点から，このような方法は採用しなかった。

3. バッファ中でのテキストの管理

IIで述べたように挿入や削除は，カーソル位置に対応したテキスト上の位置に対して行われるので，バッファ内でのテキストは図1の参照部分のように，先頭からカーソル位置の前の文字までの部分と，カーソル位置の文字からテキストの最後までの部分の2つに分けて管理し，前半の部分はバッファの最初から配置し，可能な限りの間を空けて，後半の部分はバッファの後方に配置する。

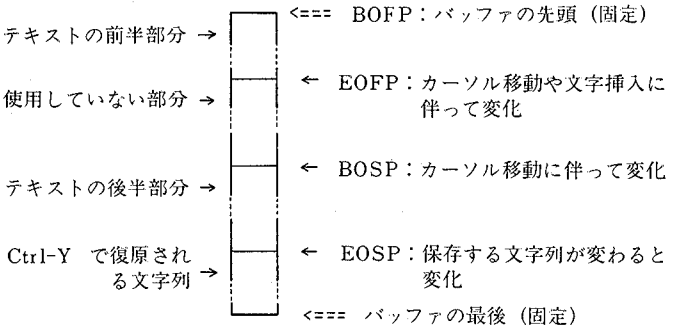


図1　テキスト・バッファの管理

このように配置すれば，文字を挿入する場合には，前半部の最後に挿入された文字を追加すればよく，迅速に挿入が行われる。文字の削除についても，同様に迅速に処理を行うことが可能である。もし前半部分と後半部分の間を空けずに連続して配置したときは，文字の挿入／削除のたびに，後半部分を移動させなくてはならず，後半部分が大きい場合には迅速に応答できないことは明白であろう。

このようなテキストの管理は，テキストの前半部分の先頭と最後＋1，後半部分の先頭と後半部分の最後＋1のバッファ上の位置（メモリ上の位置）を指す変数を用いることにより行える。以下の説明のため，これらの変数をそれぞれ BOFP(Begin Of First Part), EOFP(End Of First Part), BOSP(Begin Of Second Part), EOSP(End Of Second Part) と呼ぶことにする。ただし，BOFP はメモリ上でバッファとして確保された領域の先頭を示し，その値はエディット中固定したものとなる。（図1の「Ctrl-Y で復現される文字列 →•••」の部分は，4. で説明する。）

4. Ctrl-Y により復元される文字列の管理

コマンド Ctrl-K, Ctrl-W あるいは ESC-D により削除され，Ctrl-Y により復元される文字列は，主記憶のどこかに保存しておかなくてはならない。この保

存する場所を「キル・バッファ」と呼ぶことにする。必要とされるキル・バッファの大きさは，1文字程度の小さい場合もあれば非常に長い場合もある。従って，キル・バッファとして，最も長い場合でも可能な固定した大きさの領域を主記憶上に割り当てるのは，主記憶の利用効率からみて好ましくない。特に，一度に扱える主記憶の大きさが制限されている場合には，有効に主記憶を利用しないと，エディット対象となるテキストの大きさが制限されることになる。

EMACSでは，図1のように，テキスト・バッファの最後の部分を必要なだけ，動的に割り当てる方式を採用している。文字列を保存する時には必要とされる大きさは決まるので，そのたびにテキストの後半部分を，テキスト・バッファ中で移動することにより必要な領域を確保している。もちろんテキストの移動にはある程度の時間必要ではあるが，テキストを保存しなければならないCtrl-Kのようなコマンドの使用は，文字入力程頻繁ではないので，あまり移動の時間は気にならない。

5. 画面表示と操作性

エディタには「操作性が良い」ことが要求されるが，我々の経験から「操作性が良い」とは，まず

(1) コマンドの体系がうまく設計されている

(2) コマンドに対する応答が迅速である

を満足するであると思われる。

(1)のコマンドの体系がうまく設計されているということは，たとえばテキストをエディットするために必要，あるいは望ましい多くのコマンドが用意された強力なコマンド体系であり，しかも頻繁に用いるコマンドはキー入力が少なく簡単に行なわれるように割り当てられている等が考えられる。MIT-EMACSの場合は，ほとんどの人が，強力でありしかもコマンドのキー入力への対応もかなり良く設計されていると評価しているようである。それに，最近では，unixというOS上の計算機には，ほとんどMIT-EMACSと同一とみなせる「何々EMACS」なるものが登載されてきている。従って，MIT-EMACSの命令体系

のうち通常よく用いるコマンドを，現在我々が使用する計算機上で同一の命令体系のエディタを開発していくということは意義あるものと思っている。

(2)のコマンドに対して迅速に応答するためには，コマンドの処理，さらに画面エディタの場合には，その結果の表示を迅速にしなければならないが，我々の経験によればEMACSのほとんどのコマンドの処理は，テキストの管理方式の工夫により，かなり迅速に処理でき，画面表示の高速化を図ることの方が操作性向上につながる場合が多かった。もっとも，計算機の本体と表示画面のあるターミナルをRS−232Cのような通信回線で結んでいる場合には，その通信回線の速さを超えて高速に画面表示を行うことは不可能ではある。(通信回線の速さが1200ボーでは多少遅いと感じ，4800ボー以上が望ましいようである。)EMACSでは，次の点に注意し画面表示に関する操作性の向上を図っている。

(a) 文字または文字列がプログラムから出力されてから表示されるまでの間ができるだけ短時間に行える出力命令を用いる。プログラムから出力するのにOSの出力機能のうちの1つを用いる場合にはできるだけ高速なものを用いることにし，たとえば1字ごとに出力命令を呼び出すのではなく1行分をまとめて出力する命令を用い，しかもOSでバッファリングしないものを用いている。またOSの出力命令を用いずに直接に画面(VRAM)に書き出せる場合にはその方法をとる。VRAMに書き出せるのは今のところいくつかのパソコンの場合に限られているが，画面表示は瞬時に行え，OSの出力命令を用いる場合よりかなり高速となる。

(b) コマンドの処理結果を表示する場合に必ずしも画面全体を再表示する必要はない場合もあるので，プログラムからの出力そのものを必要最小限にする。たとえば，

(b1) 画面内でカーソルを移動する場合には画面の再表示の必要はなく，カーソルを移動させる命令だけを出力すればよい。このため画面表示の必要性の有無を管理する変数(DISPLAY)を用意し，各コマンドの処理部分でこの変数に有無の情報(ONまたはOFF)を代入するようにする。

(b2) 画面内の位置で文字が挿入された場合には，再表示は挿入位置より下

の画面部分のみでよい。また，文字が挿入された行が短い場合には，挿入がそれ以後の行に影響をおよぼさないので，その行より下側の部分は再表示の必要が無い。

このためEMACSでは，次に表示を開始すべき画面カーソル位置とテキスト・カーソル位置に関する情報を管理している。また，画面に表示されている文字の情報を主記憶上にも持っており，各行を表示するときにその行と主記憶上の情報を比較し，異なる列位置から表示を行なっている。（違いがなければその行の表示は行なわない。）一般に主記憶での比較はOSの出力命令に比べて短時間で実行できるのでこのようにして，できるだけ出力を少なくした方が画面表示の処理が迅速になる。

(c) 表示途中であっても，次の文字あるいはコマンドが入力されれば，表示を中断して入力の処理を先に行う。たとえば，

(c1) 文字が入力され，その文字以降の部分を再表示している途中で，さらに次の文字が入力されれば画面の下の方を表示するよりは，入力された文字の表示を優先すべきである。

(c2) 現在の画面よりテキストの最後の方向に3画面分移動したいときには，通常，画面移動のコマンドCtrl-Vを3回連続して入力するが，画面の表示は3画面目のものだけでよく，もし1回目のCtrl-Vにより1画面目を完全に表示し，さらに，次のCtrl-Vにより2画面目を完全に表示し，というような処理をすると，全く無駄な画面表示を行うことになり，操作性に満足できないことになる。

6. 画面表示と基本的なカーソルの上下左右への移動のための管理情報

画面エディタでは，画面カーソル位置とそれに対応するテキスト・カーソル位置等の画面表示に関する情報は文字入力やほとんどのコマンドの処理の場合に必要となるので，それらを常に把握していなければならない。もちろん把握するということは，必要となるたびに，たとえばテキストの先頭から計算して求めてもよいが，先頭からの計算はテキストが大きくカーソルがテキストの最

後の方にある場合にはかなりの時間を要する。それに対して，テキストと画面に関する情報の管理を多少工夫すると，ほとんどの場合には，現在の位置と，コマンドあるいは入力される文字から，次の位置が比較的簡単に求まるので，EMACSでは，コマンドの処理ごとに位置を求め値を記憶している。(カーソルの位置は変数に記憶し，それに対応するテキスト上の位置は，2.で述べたようにバッファ上でのテキスト配置の管理情報として記憶されている。)これらの情報は，多すぎると管理に時間がかかるし，また記憶しておくために占める主記憶の量も問題となるので，有効に利用できるものを厳選しなければならない。

EMACSで画面表示および基本的なカーソルの上下左右への移動に関しては，以下の情報を管理している（図2，3，4の参照部分)。

(1) 現在の画面の左上端（以後画面の先頭とも呼ぶ）に対応するテキスト上の位置：変数SOSAD (Start Of Screen ADress) で管理。

(2) 現在カーソルのある画面上の位置：変数CURXY (CURsor XY) で管理。

(3) 画面表示を開始すべき行の先頭の画面上の座標，およびそれに対応するテキスト上の位置：変数DYAD (Display Y ADress) およびDSAD (Display Start ADress) で管理。なお，DSADは，5.の(b-2)のように，不要な再表

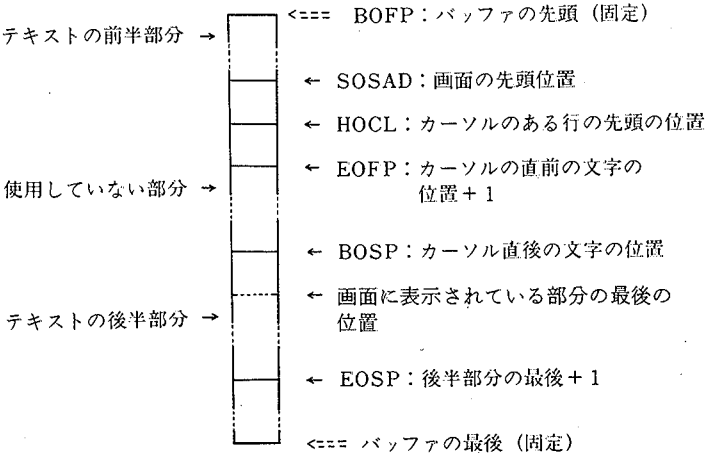
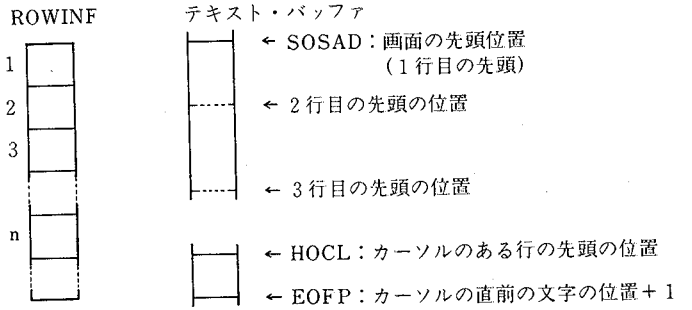
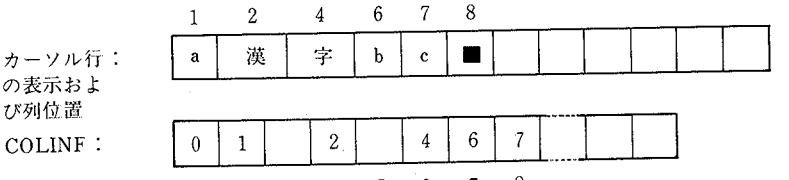


図2　画面とテキスト・バッファの対応の管理



カーソルのある行を第n行とする。

図3　ROWINF とテキスト・バッファの対応の管理



「a，b，c」は半角文字であり，「漢，字」は全角文字である。
■ はカーソルを示す。
COLINF 中の？は意味のある数値は入っていないことを示す。

図4　COLINF と表示画面のカーソル行の管理の例

示を行なわないために用いる。その値は(1)の SOSAD と一致することもある。

(4)　現在カーソルのある行の画面上の先頭位置に対応するテキスト上の位置：変数 HOCL (Head Of Cursor Line) で管理。

(5)　現在画面に表示されている行のうちカーソルのある行とその上側の各行の先頭に対応するテキスト上の各位置：画面の最大行数 (MAXROW) × 2 の大きさの配列 ROWINF で管理。

(6)　カーソルのある行について，先頭からカーソル位置までの各文字に対して，その1つ前の文字の表示列位置：画面の最大列数 (MAXCOL) に等しい大きさの配列 COLINF で管理。

画面全体を表示をするにはまず(1)の位置から始めるので，その必要性は明かであろうし，1度計算すれば，コマンドあるいは文字入力によりカーソルが現在の画面から外に移るときまでは変わらないので，その値が他のコマンドの処理でよく用いられることを考えれば，変数に記憶しておくのは妥当である。

(3)の位置は，5.の(c)で述べた画面表示を中断して次のコマンドあるいは文字入力を処理した場合に，次に表示を再開するために必要である。処理としては，1行表示するごとにキー入力の有無を調べ，もし入力があれば，次に表示すべき画面上の位置とそれに対応するテキスト上の位置を保存し表示を終了する。そして入力されたコマンドを処理した後，保存しておいた値の場所から再び表示を開始する。ただし，コマンド処理中に，テキストあるいは画面の状態によっては保存した値を修正しなければならない場合もありうる。たとえば表示している途中で，次の画面に移るコマンドが入力された場合には，そのコマンドの処理の中で，表示開始の画面位置は1行1列に，それに対応するテキスト位置は新しいSOSADと同じ値とされる。

(4)の位置は，現在の画面上の行の先頭の位置へカーソルを移動するコマンドの処理に有用であることは明かである。この情報は，(5)の情報とカーソル位置から計算することもできるが，その利用頻度を考慮して管理情報としている。

1.で述べたようにテキスト中で半角文字は1バイト全角文字は2バイトとして管理している場合，カーソルの位置からテキストの最後の方向にカーソルを進める場合にはテキスト・カーソル位置から順に1バイトあるいは2バイトを調べることにより計算できる。しかし，テキストの先頭方向にカーソルを戻す場合には，逆方向に数バイトのコードを調べるだけでは半角／全角文字の判別ができず次のカーソル位置が求まらないこともある。また，左側のコードがタブコードの場合にも，次のカーソル位置が現在のカーソル位置からは求まらない。そのために(5)，(6)の情報を保持しているのであるが，その情報の有用性，管理方法はカーソルの左右への移動と密接に関連するので，その処理方法と共に説明することにする。ただし，(1)，(2)，(3)，(4)についての管理方法についても副次的に説明することにもなる。(なお以下は，テキスト・バッファ中で半角

文字は1バイト，全角文字は2バイトで管理している場合の説明である。半角文字のを2バイトで管理する場合にもほぼ同様のことを行なう。)以下の説明での処理の流れは,次に進むところが明示されていなければ下に進むものとする。

(A)　カーソルを右方向（テキストの最後の方向）へ移動する場合（Ctrl-F）

(A-1) 現在のカーソルの行位置をr，列位置をcとして保存しておく。

(A-2) 図1で示されるテキストの後半部分の先頭から1バイト分のコードを前半部分の最後に移し，列位置を1増やす。

(A-3) 移された1バイトのコードについて，

- もしそれが改行コードであれば(A-7)へ進む。
- もしそれがタブコードであれば次のカーソルの列位置を計算・更新し(A-4)へ進む。
- もしそれが改行コードとタブコード以外の制御文字であれば，列位置をさらに1増やし，(A-4)へ進む。(EMACSでは，たとえば制御文字0x01 コントロールAは Â と半角文字2個で表現される)。
- もし，それが全角文字の1バイト目であれば，さらに次の1バイトを前半部分の最後に移し，列位置を1増やし(A-4)へ進む。

(A-4) 新しい列位置が画面の最大列数(MAXCOL)－1以上であれば，連続行を示す「!」を行末に表示し(A-6)へ進む。さもなければ(A-5)へ進む。

(A-5) COLINF [c] に新しく更新されたカーソルの列位置を代入し，終了する。

(A-6) ROWINF [r] に現在のテキストの前半部分の最後＋1の位置を代入し，カーソルの行位置を1増やし，カーソルの列位置を1とする。もし更新されたカーソルの行位置が画面の最大行数（MAXROW）を超えていれば(A-7)へ進む。さもなければ終了する。

(A-7) ROWINF [MAXROW－OVLL＋1] からROWINF [MAXROW] の値がそれぞれROWINF [1] からROWINF [OVLL] の値となるように，ROWINFの内容を上方にずらす。

(A-8) 画面の先頭に対応するテキストの位置(SOSAD)として，ROWINF[1]

の値を代入し，カーソルのある画面上の行の先頭位置に対応するテキスト上の位置(HOCL)として，ROWINF [OVLL+1] を代入する。さらに画面表示を開始すべき行の先頭の画面上の座標(DYAD)を1行1列とし，ROWINF[1]の値をそれに対応するテキスト上の位置(DSAD)として代入する。そして画面表示の必要性を管理する変数を ON とし終了する。

(B) カーソルを左方向へ移動する場合 (Ctrl-B)

(B-1) 現在のカーソルの行位置をr，列位置をcとして保存しておく。

(B-2) 図1で示されるテキストの前半部分の最後から1バイト分のコードを後半部分の先頭に移し，列位置を1減ずる。もし更新された列位置が0であれば(B-4)へ進む。

(B-3) 移された1バイトのコードについて，

- もしそれがタブコードであれば，COLINF [c] の値を次のカーソル列位置として終了する。
- もしそれがタブコード以外の制御文字であれば，列位置をさらに1減じて終了する。
- もし，それが制御文字でなくて，更新された列位置がCOLINF [c] と等しくなければ，全角文字の2バイト目であるので，さらにテキストの前半部分の最後から1バイト分のコードを後半部分の先頭に移し列位置を1減じ，その値が0であれば，(B-4)へ進む。さもなければ終了する。(テキストが正しく管理されていれば，改行の制御文字が現われる場合には，(B-2)で列位置が0となるので，ここで考慮する必要はない。)

(B-4) カーソルの行位置を1減じる。もし，更新された行位置が0となれば(B-7)へ進む。

(B-5) (2)のカーソルのある画面上の行の先頭位置に対応するテキスト上の位置として，ROWINF [r−1] を代入する。

(B-6) テキスト上の ROWINF [r−1] の位置から，テキストの前半部分の最

後までの各文字について順にその列位置を計算し，COLINF の適当な位置に代入し，終了する。（テキストが正しく管理されていれば，これらの文字は画面上の１行におさまるはずである。）

(B-7) テキストの前半部分の最後が MAXROW−OVRLL 行目に表示されるためには，テキストのどの位置から画面左上端に表示し始めればよいかを計算し，その位置を上記の画面表示のための情報の(1)，(3)の値とする。（この位置の計算については次の (X) で述べる。）なお，この値を計算するときに，副次的に画面表示のための情報の(2)，(4)，(5)も求めておき終了する。

(X)　指定されたテキスト上の位置 (pos とする）が，指定された行位置（r とする）に表示されるためには，テキストのどの位置から画面左上端に表示し始めれば良いかの計算は次のように行なう。

(X-1)

- pos が現画面の先頭に対応する位置 (SOSAD) よりもテキストの先頭方向にあるときには，dr = r とし (X-2) へ進む。
- pos が SOSAD とカーソルに対応する位置 (EOFP) の間にあるときは (X-5) へ進む。（SOAD，ROWINF，EOFP を調べれば判断できる。）
- pos が EOFP よりもテキストの最後の方向にあるときには，(X-6) へ進む。

(X-2) pos からテキストの先頭方向に１バイトずつ，改行コードの数，文字数，テキストの先頭にきたか否かを調べていく。その途中，

(ア)　もし，テキストの先頭になれば，その位置を「仮の画面左上端に対応する位置（仮の SOSAD)」とする。

(イ)　もし改行コードの個数が dr になれば，改行コードの１つ最後の方向の位置を「仮の SOSAD」とする。

(ウ)　文字数が「dr ×画面の行の長さ (MAXCOL)」になれば，その位置を「仮の SOSAD」とする。

(X-3) 「仮の SOSAD」が１行１列目に対応するとして，pos が何行目となる

か（r3とする）を計算する。この計算中に副次的にROWINFとCOLINFの値も求めておく。

(X-4)

・もしr3≦drであれば，「仮のSOSAD」を「SOSAD」とし，終了する。

・もしr3＞drであれば，r3−dr+1行目の先頭に対応する位置を「SOSAD」とし，終了する。

(X-5) posのある行をr4とする。

・もしr4≧rであれば，r4−r+1行目の先頭に対応する位置を「SOSAD」とし，終了する。

・もしr4＜rであれば，dr=r−r4とし，(X-2)へ進む。

(X-6) カーソルに対応する位置(EOFP)をカーソル座標に対応するとして，posが何行目となるか（r5とする）を計算する。この計算中に副次的にCOLINFの値も求めておく。

(X-7)

・もしr5≧rであれば，r5−r+1行目の先頭に対応する位置を「SOSAD」とし，終了する。

・もしr5＜rであれば，dr=r−r5とし，(X-2)へ進む。

（注） (X-3)での行数の計算はテキストの最後の方向への計算であるので簡単に行える。ただし，(X-2)の(ウ)で「仮のSOSAD」が求められたときには，その「仮のSOSADが全角文字の2バイト目である可能性もある。しかし(ウ)の場合には，かなり大きめにテキストの先頭方向に「仮のSOSAD」を設定していること，JMACSで扱うテキストの場合一般には適当なところに改行コードが入れられていることから，(X-5)で求められる「SOSAD」までの間に1バイト／2バイト文字の判断が正しく行なわれるので，実際上は問題は生じていない。

IV　その他の命令の処理方式

その他のほとんどのコマンドは，III．の管理情報およびIII．6．で説明したカーソルのテキストの前後への移動，指定した位置を指定した行に表示するための計算を行なうサブルーチンを用いれば容易に実現できるので，ここでは，カーソル・画面の移動，文字列の検索・置換に関するいくつかのコマンドについてのみ説明する。なお，説明の都合でIII．6．の(A)，(B)に続くコマンドとして，(C)，(D)，…と順番を与える。

1．カーソル・画面移動のコマンドの処理方式

(A)，(B)　以外のカーソル・画面移動のコマンドとしては，次のものがある。

(C)　カーソルを下方向（テキストの最後の方向）へ移動する場合 (Ctrl-N)

(D)　カーソルを上方向（テキストの先頭方向）へ移動する場合 (Ctrl-P)

(E)　カーソルを次の画面（テキストの最後の方向）に移す場合 (Ctrl-V)

(F)　カーソルを前の画面（テキストの先頭方向）に移す場合 (Ctrl-Z)

(G)　カーソルのある行を画面中央の行として表示する場合 (Ctrl-L)

(H)　カーソルをテキストの先頭に移動する。

(I)　カーソルをテキストの最後に移動する。

例として，このうち(C)，(G)についてその処理方式を説明する。

(C)　カーソルを下方向（テキストの最後の方向）へ移動する場合

(C-1) 現在のカーソル位置を (r,c) とする。

(C-2)

- もし，r+1 が画面の最大行数 (MAXROW) 以下であれば，(C-3) へ進む。
- もし，r が MAXROW に等ければ，(C-4) へ進む。

(C-3) カーソル位置が (r+1,c) となるまで，上記III．6．の (A) のカーソルを右方向へ1文字進めるサブルーチンを用いてカーソルを進め終了する。ただし，

(ア) もしr行目の途中でテキストの最後に到達したときは，その次の位置を新しいテキスト・カーソル位置とし移動を終了する。

(イ) もしr+1行目の途中でテキストの最後に到達したときは，その次の位置を新しいテキスト・カーソル位置とし移動を終了する。

(ウ) もしr+1行目の途中で改行コードに出会ったときは，その改行コードの直前の次の位置を新しいテキスト・カーソル位置とし移動を終了する。

(C-4) カーソル位置が(OVLL+1,c)となるまで，上記(A)のカーソルを右方向へ1文字進めるサブルーチンを用いてカーソルを進める。ただし，その途中でテキストの最後あるいは改行コードに到達したときは，(C-3)と同様に処理する。

(G) カーソルのある行を画面中央の行として表示する場合 (Ctrl-L)

(G-1) テキスト・カーソル位置を「指定された位置」，画面中央の行位置(MAXROW/2)を「指定された行位置」として，III. 6.の(X)をサブルーチンとして呼び出し，画面先頭に対応する位置(SOSAD)等を求める。

2. 文字列の検索・置換の処理方式

(I) 文字列の検索の場合 (Ctrl-S あるいは Ctrl-R)

テキストの最後の方向への検索について説明する。(先頭方向への検索についても同様である。)

(I-1) カーソルに対応する位置(EOFP)から指定された文字列を検索する。

・文字列が見つからなければ，その旨表示し終了する。

・見つかったときは，見つかった文字列の位置をposとする。

(I-2) テキストの後半部分の最初(BOSP)からposまでの文字列をテキストの前半部分の最後に移し，その最後＋1を新しくEOFPの値とし，posをBOSPの値とする。

(I-3) 前のカーソル位置から行数を計算し，新しいEOFPに対応する位置が画面内にあるか否かを調べる。

・画面内にあれば，新しいEOFPをカーソル位置とする。

・画面内になければ，新しいEOFPを「指定された位置」，画面中央の行位置(MAXROW/2)を「指定された行位置」として，Ⅲ．6．の(X)をサブルーチンとして呼び出し，画面先頭に対応する位置(SOSAD)等を求める。

(J)　文字列の置換の場合(ESC-RあるいはESC-Q)

置き換えられるテキスト中の文字列を「文字列1」，置き換わる文字列を「文字列2」とする。

(J-1) (I)の文字列検索のサブルーチンを用いて「文字列1」を見つける。(SOSAD, EOFP, BOSP等の更新は行なわれている。)

・見つからなければ，その旨表示し終了する。

・見つかったときは，(J-2)へ進む。

(J-2)

・ESC-Rによる置換のときは(J-4)へ進む。

・ESC-Qによる置換のときは，画面の表示を行なった後，置き換えるが否かあるいは置き換えを中止するかを問い合わせる。

(J-3)

・置き換えを中止する指定がなされれば終了する。

・置き換えの指定がなされれば(J-4)へ進む。

・今見つかった「文字列1」は置き換えないが，以降のテキストについては置き換えを継続する指定がなされれば(J-1)へ進む。

(J-4) EOFPの値を「文字列1」のバイト数分減じ(「文字列1」を削除)，さらに「文字列2」を挿入し，(J-1)へ進む。

Ⅴ　おわりに

我々が萩野氏のEMACSを最初に8ビットのパソコンに移植した1980年当時は，本学で使用していた計算機システムは，パソコンと計算センタのFACOM-230/45であり，前者は本体とCRTターミナルからなるシステムではあったが，OS付属のエディタは画面エディタではなかったし，後者はパンチ

カードによるプログラム作成が主であった。その後FACOM-230/45にもフロッピィ入力装置が増設され，パソコン上で開発したプログラムをFACOM-230/45へ入力できるようになりプログラム開発の効率がかなり向上した。それ以後現在に至るまで約7年間，プログラムの開発環境の整備の一環として，同一のコマンド体系を有した画面エディタを我々の周辺の計算機システム上に次々に開発してきた。また，EMACSを有効に生かせるようなソフトウエアの開発も行なってきた［7］［8］。

一方現在では，16ビットのパソコンによってはいくつかの画面エディタが開発・販売されており，それらの中には我々の開発したEMACSに較べて，より強力なものもあるようである［2］［3］。（ただし，我々の開発したものはMIT-EMACSのごく一部の機能を実現したものであり，我々はMIT-EMACSはそれらよりも機能も豊富で強力であると感じている。）また，計算機関連の書籍・雑誌に画面エディタ開発の記事やプログラムも記載されているようである［4］［5］［6］。

このように計算機によってはいくつかの良いエディタが利用できる環境となっているが，「周辺の計算機で同一のコマンドでエディタを使用したい」という気持ちもあり，我々は今後しばらくは主としてEMACSを使用し，また必要があれば新機種導入ごとにEMACSの開発を続けるつもりである。また，EMACSの機能強化，有効に生かせるようなソフトウエアの開発についても必要となるごとに行なうつもりである。

参考文献

［1］ Richard M. Stallman『EMACS Manual for TWENEX Users』MIT AI-Memo 556, 1980

［2］ メガソフト社『MIFES—98ユーザーズマニュアル』

［3］ エー・エス・ピー社『FINALユーザーズマニュアル』

［4］ 島内剛一『エディタを作ろう』ｂｉｔ, Vol. 16, No. 11 (1984)-Vol. 17, No. 3

［5］ 千葉宗昭『C言語の本格的応用－Cでつくるスクリーン・エディタとプログラミング・ツール』CQ出版社，1986年

［6］ 石田裕久『MS－DOSの汎用エディター PSE』PC-World Vol. 5, No. 12, 1976

[7]　本田道夫『計算機システムの環境整備へのパーソナルコンピュータの応用』香川大学経済論叢 第59巻第1号，1986年6月

[8]　本田道夫『通信機能を備えたLISPシステム』文部省特定研究・香川大学経済学部「地域経済高度情報化のための管理科学的手法の開発」昭和61年3月

付録　EMACS命令一覧

以下の一覧の説明中で，「Ctrl-α」とは，キーボード上のコントロール・キー（キーの上にCTRLと記されているキー）を押したままで，あるキーαを押す操作を示し，「Esc-α」とは，エスケープ・キー（ESCと記されているキー）を1度押して離した後に，あるキーαを押す操作を示す。

═══ カーソルの移動 ═══

Ctrl-F ……カーソルを右方向に移動 (Forward) →（カーソル移動キー）
Ctrl-B ……カーソルを左方向に移動 (Backward) ←
Ctrl-N ……カーソルを次行に移動 (Next line)　↓
Ctrl-P ……カーソルを前行に移動 (Previous line)　↑
Ctrl-A ……カーソルのある行の先頭に移動 (Ahead of line)
Ctrl-E ……カーソルのある行の最後尾に移動 (End of line)
Esc-F ……カーソルを一単語ごとに前に移動 (Forward)
Esc-B ……カーソルを一単語ごとに後に移動 (Backward)
Esc-＜ ……カーソルをファイルの先頭に移動
Esc-＞ ……カーソルをファイルの最後尾に移動
Ctrl-X Ctrl-A ……カーソルのある行の画面上の先頭に移動
Ctrl-X Ctrl-E ……カーソルのある行の画面上の末尾に移動

═══ 画面制御 ═══

Ctrl-V ……次の画面を表示 (View)
Esc-V ……前の画面を表示 (Ctrl-Z と同じ)
Ctrl-L ……現カーソル行を中心に表示
Esc-X redraw ESC ……画面の再表示 (Unix & CP/M-68K)

挿　入

Ctrl-O ……カーソルのある行に空白行を挿入 (Open)

Ctrl-Q ……コントロール文字の挿入

削　除

Ctrl-D ……カーソル位置の一文字を削除 (Delete)

Ctrl-H ……カーソルの前の一文字を削除 (BSキーと同じ)

Esc-D ……カーソル位置の一単語を削除

Ctrl-K ……カーソル位置からその行の終りまでを削除 (Kill line)

Ctrl-W ……カーソルとマークの間の全文字を削除

Ctrl-Y ……Esc-D,Ctrl-K,Ctrl-Wで削除した部分を復元 (Yank)

Esc-BS, Esc-Ctrl-H……カーソルの前の一単語を削除

MS-DOS, CP/M-86 でのみ使用可能

マークの設定

Esc-space……現カーソル位置をマークする

Esc-Esc ……マークした場所にカーソルを移動させる

Ctrl-W ……カーソルとマークの間（リジョン）の全文字を削除

サーチ（検索）

Ctrl-S string Esc ……指定の文字列 (string) をカーソル位置からファイルの最後の方向にサーチ (Search)

Ctrl-S Ctrl-S ……前回指定文字列をファイルの最後の方向にサーチ

Ctrl-R string Esc ……指定の文字列(string)をカーソル位置からファイルの最初の方向にサーチ (Reverse search)

Ctrl-R Ctrl-R ……前回指定文字列をファイルの最初の方向にサーチ

置　換

Esc-R string1 Esc string2 Esc……一括置換 (Replace)

Esc-Q string1 Esc string2 Esc……選択置換 (Query replace)

（置換：space,　スキップ：BSキー,　終了：他のキー）

大文字／小文字変換

Esc-U……大文字化 (Upper case letter)

Esc-L……小文字化 (Lower case letter)

Esc-C……先頭大文字化，以降小文字化

マクロ・コマンド

Esc-M Esc-（コマンド列 Esc-）……マクロ・コマンドの定義・登録 (Macro)

Esc-X M R　……リジョン内テキストをマクロとして登録

Esc-X M G　……登録マクロをカーソル位置に取り出す

Esc-E　……マクロ・コマンドの実行 (Execute)

ファイルの表示・操作

ファイル名のあとには <CR> が必要！

Ctrl-XD ファイル名……ディレクトリの表示 (Directry)

……MS-DOS, CP/M-86

Ctrl-XT ファイル名……指定ファイルの内容を画面表示 (Type)

……MS-DOS, CP/M-86

Ctrl-XR ファイル名……カーソル位置に指定ファイルの内容を挿入する

Ctrl-XS ファイル名……リジョンの内容を指定ファイル名で書き出す

エディット対象のファイルの変更

Ctrl-X Ctrl-V ファイル名……エディット対象ファイルの変更 (Visit)

バッファ制御

Ctrl-XA……64K バイトになるまでディスクからテキストを読む.
Esc-XA, Esc-XN (CP/M-86 & MS-DOS),
Esc-X append ESC, Esc-X next ESC (Unix & CP/M-68K) も同機能

Ctrl-XW……最初からカーソル位置までのテキストをディスクに書く.
Esc-XW (CP/M-86 & MS-DOS), ESC-X write ESC (Unix & CP/M-68K)
も同機能

複数のバッファ及び画面分割

(Unix & CP/M-68K EMACS でのみ使用可能)
バッファ名のあとには <CR> が必要！

Ctrl-XB バッファ名……メモリバッファの切り換え
Ctrl-XK バッファ名……メモリバッファの削除
Ctrl-XA バッファ名……リジョンの他メモリバッファへの挿入
Ctrl-X2 …… 2 画面表示
Ctrl-X1 …… 1 画面表示
Ctrl-XO …… 2 画面表示中の画面間カーソル移動
Ctrl-X Ctrl-B ……ウィンドウ, バッファ等に関する情報の表示

計数表示

Ctrl-X L……バッファ上の現在の位置を表示 (Line)

（現在位置の行　＋　残りの行数　＝　バッファ全行数）

Ctrl-XC ……文字数の合計を表示 (Character)

（使用済み文字数　，削除文字数　，未使用文字数）

命令の取り消し

Ctrl-G……命令を取り消す

コマンドの繰り返し実行

Ctrl-U 回数 コマンド　　同じ命令の繰り返し実行

その他の機能

Esc-XT <tab_size>　　……タブサイズの変更
(CP/M-86 & MS-DOS)

Esc-X tab ESC <tab_size>　……タブサイズの変更
(UNIX & CP/M-68K)

Esc-XC <column_size>　　……カラム数の変更

EMACS の終了

Ctrl-XE ……エディットした部分をファイルに書き出して終了 (End)

Ctrl-XQ ……元のファイルのままで終了 (Quite)