Jmacs 入門

本 田 道 夫

目　　次

はじめに

プログラムの作成・編集等のためのエディタとしては，MIT（マサチュウセッツ工科大学）の R. Stallman によって開発された画面エディタ Emacs があります。Emacs は，その編集コマンド体系の機能の豊富さや使い勝手のよさなどからみて非常に有用かつ強力な面画エディタです。さらに大きな特徴として，Emacs では，ユーザがより複雑な編集操作のためのコマンドを開発・登録し機能を拡張することができ，ユーザ自身によるカスタマイズが可能であるということがあります。また Emacs はプログラムの作成だけでなく，論文や文書の作成にも有用な編集コマンドも用意されています。もっとも論文や文書の作成には D. Knuth による TEX が有名です。しかし，その TEX に対する入力ファイルは通常はエディタを用いて作成しますが，その場合にも Emacs は非常に有用・強力なエディタとして利用できます。

このように Emacs はプログラム開発以外にも広く用いられており，現在では，Stallman 以外の人によって開発されたものも含めて，各種 Unix マシン，あるいは各種パーソナル・コンピュータなどの数多くのコンピュータ・システム上でも利用できるようになっています。これらの中には，Emacs の編集コマンド体系の一部分だけが実現されているものもありますが，逆に，Emacs の中から，各種言語のコンパイラを呼び出す，あるいはコンピュータの OS（オペレーティング・システム）の機能を利用できるように拡張され，単なるエディタというよりも，総合的なプログラム開発環境そのものを与えるものへと発展したものもあります。ただし，これらの Emacs は，基本的な編集操作以外で，若干ではありますが，お互いに編集コマンドが異なる場合があります。

一方，我々は従来から，DEC (Digital Equipment Coorporation) の OS TOPS-20 の Emacs の編集コマンドのサブセットを有したエディタを，初期には，8 ビット・マイクロ・コンピュータの CP/M-86 の上で，その後，16 ビット・マイクロ・コンピュータの CP/M-86 と MS-DOS の上で作成・使用してきました。初期にはメモリ上で扱えるテキストの大きさが 64 K バイトであり，半角英数字のみからなるテキストしか扱えませんでしたが，その後，全角文字の扱い，利用可能なメモリの大きさまでのテキ

ストが扱え，しかもマルチバッファ，マルチウインドウの機能等が実現されました。[1] また，日本語文字，英数字だけでなく，ロシア語等の編集の希望があり，同じ編集能力を有した多ケ国語に対応できるエディタへと発達してきました。ただし，現在のバージョンでは，日本語，英数字以外の文字については，別に提供するプログラムを用いて作成した文字フォントを，本田が組み込むようになっており，ユーザが自分で組み込むことはできません。また，日本語入力にはATOK, VJEなどのフロントエンド・プロセッサを利用する必要があります。

以下では，そのエディタ——Jmacsと命名——について，その概念と基本的なコマンドおよび操作方法を中心に説明します。現バージョンの編集コマンド体系は，Unixマシン上のPDS (Public Domain Software) であるGNU Emacsコマンド体系のサブセットに若干の独自のコマンドを加えたものです。したがって，Jmacsの使用は，GNU Emacsなどの使用に際しても参考となるでしょう。

本書の第1章では，Jmacsを使用するために必要な最小限の用語・概念などを簡単に説明します。第2章では，Jmacsのごく基本的な編集コマンドについて，操作例を示しながら説明します。第3章では，Jmacsをより有効に用いるためのいくつかの編集コマンドについて説明します。第4章では，キーボードから入力された一連の編集コマンドをマクロとして定義し，簡単な操作で，その一連のコマンドを何度でも実行させることができるキーボード・マクロについて説明します。第5章では，互いに参照しながら2つ以上のファイルの編集をおこなうマルチ・バッファと，CRT画面を分割して2つ以上のテキストを表示するマルチ・ウインドウについて，その概念とコマンドについて説明します。

---

(1) 8ビット版と16ビット版の初期のバージョンは，萩野達也が作成し，その後の機能拡張や日本語対応版は本田が発展させました。現在のバージョンは，それらの処理方式を参考にして，本田が改めて作成したものです。

1. Jmacs の概念と用語

エディタについて習得しようとする場合，基本的な編集コマンドについて，とにかく使ってみるということも１つの方法でしょう。しかしその場合でも，エディタの基本的な概念を知っていて，その概念と照らし合わせながら各コマンドを試みる方が，そのコマンドについての理解も容易となるでしょう。そこで，Jmacs について，まず知っておくべき基本概念と用語について説明します。

1.1 ファイル，バッファ，テキスト

ディスク上のファイルを編集するとき，Jmacs では，図１のように，コンピュータの主記憶上にそのファイルの編集のための領域が確保され，そこに指定されたファイルの内容を読み込みます。

この主記憶上の領域のことを**バッファ**と呼び，領域を確保することを，バッファを開くという言い方をするときもあります。また，バッファに読み込まれたファイルの内容をバッファ上のテキスト，あるいは単にテキストと呼びます。ユーザはこのバッファ上のテキストに対して，Jmacs のコマンドを用いて編集作業を行います。必要な編集を行ったのち，ディスクに書き込むためのコマンドによって，バッファ上のテキストをディスク上のファイルに保存します。この書き出しのためのコマンドを実行させない限り，編集されたバッファ上のテキストはディスク上のファイルには保存され

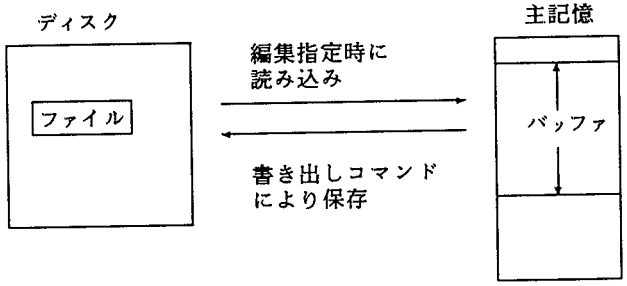


図１　ディスク上のファイルと主記憶上のバッファ

ませんので注意してください。

なお，「バッファ上のテキストをディスク上のファイルに保存する」ことを，「バッファをファイルに保存する」，あるいは「テキストを保存する」等の言い方をする場合もあります。また，バッファとその上のテキストは，入れ物とその内容という関係であり，たとえば，バッファの先頭とテキストの先頭は同じ位置を指します。したがって，以後，位置を示すような場合にはバッファとテキストを，あまり区別せずに用いることもあります。

ユーザが編集を指定したファイルについては，パス名を除いたファイル名と同じ名前のバッファが開かれます。たとえば，ファイル¥HONDA¥TEST.TXTが指定された場合には，バッファ名はTEST.TXTとなります。

なお，Jmacsでは，複数のファイルを指定して，ファイル毎に対応した複数のバッファを開くこともできるマルチバッファの機能もあります。この機能では，たとえばあるバッファの一部分を，別のバッファに挿入するなどの操作ができます。ただし，最初は，1つのファイルの編集を行うこととし，マルチバッファについては第5章で説明することにします。

Jmacsでは，ユーザが指定したファイルに対応して開かれるバッファ以外にも，自動的に開かれる名前の決まった2つのバッファ——現在開かれているすべてのバッファの情報を表示するとき用いられるバッファ(#BUF_LST)，マルチバッファ時に現在編集しているバッファが消されたときに使用されるバッファ(#DEF_BUF)——があります（第5章参照)。

1.2 Jmacsの画面

CRT画面に表示できる行数には限度がありますので，図2のように，CRT画面にはバッファ上のテキストの一部分が表示されます。

Jmacsを起動したときには，画面は図3のように表示されます。画面の最も上の行から，反転表示されている行の上までの領域には編集対象となっているファイルに対応したバッファ上のテキストの一部分が表示され，カーソルはこの領域の左上に位置します。ただし，指定したファイルが新しく作成しようとしているファイルであれば，

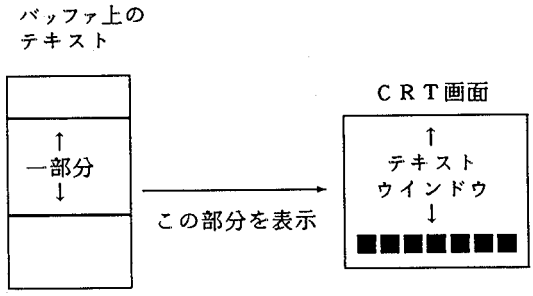


図2　バッファ上のテキストとCRT上の表示

テキストはありませんので，この部分には何も表示されません。

その下の反転表示されている行は**モード行**と呼ばれ，編集中のバッファに関する情報が表示されます。上記の2つの部分を合わせて**テキストウインドウ**あるいは単に**ウインドウ**と呼びます。つまり，テキストウインドウは，バッファ上のテキストの一部分を覗き見るモード行という枠のついた移動可能な窓と考えられます。なお，テキストウインドウのことをCRT画面全体と混乱が生じない場合には，画面ということもあります。

モード行は（実際にはリバース文字で）

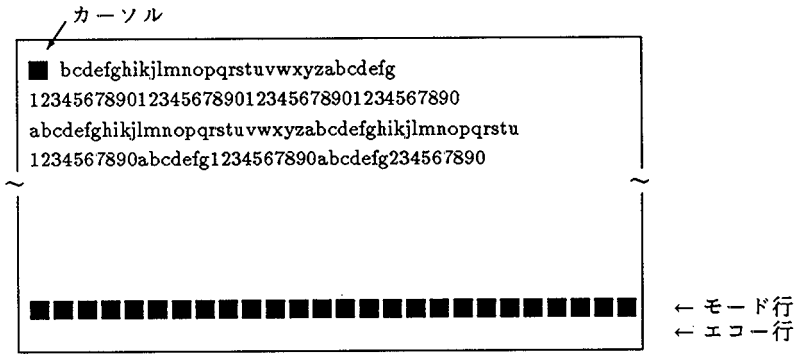


図3　Jmacsの画面表示

(line, colomn) EMACS *(n. nn)* : *buf mmcp*

のように表示されます。ここで，*line* は，カーソルの位置している行のテキストの先頭から数えた行数，colomn は，カーソルの位置している桁位置です。*(n. nn)* は Jmacs のバージョン番号です。buf はウインドウに表示され編集対象となっているバッファ名で，パスを除いたファイル名が採用されます。次の mm は%%，——，＊＊のいずれかであり，バッファの修正に関する情報を示します。

%% バッファの修正ができないことを示します。編集を指定したファイルの属性が読み取り専用の場合にこの状態になります。

—— バッファは修正可能ですが，まだ修正は行われていないことを示します。

＊＊ バッファに対して修正が行われ，バッファの内容とそれに対応したファイルの内容が異なっていることを示します。

mm の次の *c* は，Jmacs の文字列検索コマンドの場合の英大文字・小文字のとり扱いのモードを示します。Jmacs を起動したときは，大文字の C が表示されていますが，その場合には，英大文字と小文字を区別しないで検索が行われます。小文字の c の場合には，区別した検索が行われます。このモードの変更方法については，第3章で説明します。

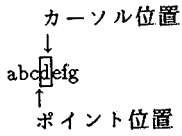
最後の *p* は，括弧の対応モードを示すもので，大文字の P が表示されている場合には),], } の右括弧を入力したときに，それぞれ対応する (, [, { の左括弧の位置にしばらくカーソルを戻して表示します。小文字の p が表示されている場合にはそのようなことを行いません。LISP あるいは C のような括弧の多いプログラム言語を用いる場合には非常に有効な機能です。Jmacs を起動したときには，大文字の P が表示されています。このモードの変更方法についても第3章で説明します。

モード行の下の CRT 画面の最下行はエコー行と呼ばれる領域です。この行には，たとえば C-x C-c のような2個以上のキー入力が必要なコマンドに対して，それまでの入力が順にエコー表示されます。

なお，エコー行には，エコー表示以外にも，入力したコマンドに対する Jmacs からのメッセージの表示も行われます。このようにエコー行には２つの異なったものが表示されますが，その違いはあきらかに識別できますので，あまり気にしなくてもよいでしょう。

1.3 ポイントとカーソル

Jmacs では，文字の挿入や消去・削除などの編集コマンドによる操作の対象となるバッファ上の位置を示すものとして**ポイント**という概念を用います。ポイントの位置をテキストウインドウ上で示すためにカーソルが用いられます。ただし，正確には，ポイントはカーソルの位置している文字ではなく，図４のように，カーソルの位置している文字とその前の文字の間にあると考えます。なお，ポイントがテキストの最後にある場合，すなわちバッファの最後にある場合には，ポイントの後に表示される文字はないので，カーソルは，当然のことながら何も表示されていない部分に位置します。



・カーソルは反転表示された文字ｄの位置にある。
・ポイントは文字ｄとその前の文字ｃの間にある。

図４　ポイントとカーソル

ほとんどの場合ポイントとカーソルは１対１に対応するため，以後では，「ポイントの次の文字」の代わりに「カーソル位置の文字」，「ポイントの移動」の代わりに「カーソルの移動」などのように，ポイントという言葉に代えてカーソルという言葉を用いることもあります。

1.4 Jmacsの編集コマンドの入力とその表現

Jmacsではすべての編集コマンドに名前がつけられています。たとえば1文字分カーソルを進めるコマンドの名前はforward-charです。しかし，編集コマンドを指定するたびにそのようなコマンド名を入力していたのでは効率のよい編集は行えません。そこで，比較的よく用いる編集コマンドには，コントロール文字かエスケープで始まる簡単な指定方法が用意されています。ただし，編集コマンド名でしか指定できないコマンドもあります。

1.4.1 コントロール文字で始まるコマンド

コントロール文字の入力は，つぎのように行います。

> キートップにCtrlあるいはCTRLなどと記されたコントロールキーを押したまま，英文字などを入力します。

このようにコントロールキーと英文字○を用いて入力されるコントロール文字を，以後C-○のように表現することにします。たとえば，コントロールキーを押したまま，英文字aを入力する場合には，C-aと表現します。コントロール文字の入力では，英大文字でも英小文字でも同じコントロール文字の入力となります。したがって，以後C-aのように英小文字を用いて表現することにします。なお，コントロール文字以外の文字，すなわち空白(スペースバーにより入力)，英数字，記号文字などを，通常の文字ということにします。

C-f：コントロールキーを押したままfを入力します。
C-@：コントロールキーを押したまま@を入力します。(キーボードによっては，@の入力はシフトキーを用いる場合もあります。)

ほとんどのコントロール文字は，それ1文字で編集コマンドとなりますが，C-xの場合は，つづいて入力される文字（通常の文字あるいはコントロール文字）とあわせ

て編集コマンドとなります。

C-xa：コントロールキーを押したまま x を入力し，コントロールキーと x のキーから指を離したのち，さらに a を入力します。

C-x C-c：コントロールキーを押したまま x を入力し，いったん x のキーから指を離したのち，コントロールキーを押したまま c を入力します。(この場合 C-x と C-c の入力の間で，コントロールキーから指を離さなくてもかまいません。)

ほとんどのコントロール文字は Jmacs の編集コマンドとなりますが，中にはテキストへの入力となるコントロール文字もあります。たとえば C-m と C-i は，一般にテキスト中での改行とタブを表わすものとして用いられます。Jmacs でも，そのように扱いますので，そのままテキストへ C-m と C-i が入力されます。なお，これらはプログラムあるいは文書の編集において，よく用いられるため，コントロールキーを用いずに入力できるように，RET, TAB などとキートップに表示されたキーが割り当てられています。このような別にキーが与えられているものとしては，空白の入力のための，キーボードの手前の長いスペースバー，キートップに ESC と記されているエスケープキーがあります。以後の説明やコマンドの表現では，このようなキーを入力すること，あるいは入力されるものを，それぞれ RET, TAB, SPC, ESC で表わす場合もあります。

1.4.2 エスケープで始まるコマンド

エスケープは C-[でも入力できるコントロール文字ですが，計算機操作のために比較的よく用いられるので，キートップに ESC などと表示されたキーが割り当てられています。ESC で始まる編集コマンドの場合も C-x と同様に，つづいて入力される文字との 2 文字で編集コマンドとなります。ESC で始まるコマンドの入力は、つぎのようにおこないます。

ESC を入力し，つづいて次の文字を入力します。

ESC で始まるコマンドの場合には，コントロールキーの場合と異なり，ESC を入力し，指を離したのち次の文字を入力します。なお，以後の編集コマンドの説明では，ESC で始まる編集コマンドの表現には，慣例に従って，次に入力される文字が通常の文字の場合には M-○，コントロール文字の場合には C-M-○ の表現を用いることにします。

M-f： ESC につづいて f を入力します。
M-%： ESC につづいて%を入力します。
C-M-f：ESC につづいて，コントロールキーを押したまま f を入力します。

編集コマンドによっては，つづいて文字列あるいはファイル名などの入力が必要なものもあります。たとえば，問い合わせながら，文字列 *oldstring* を *newstring* で置き換えるコマンド M-% の場合には，それぞれ**改行**で終わる 2 つの文字列の入力が必要です。このようなコマンドの場合には，改行を RET で記し，

M-% *oldstrig* RET *newstring* RET

とそのコマンド形式を表現することにします。改行の入力には，キートップに RET のように表示されているキーを用います。以後，コマンド形式だけでなく，説明文中でも，改行のことを RET と記すこともあります。

編集コマンドによっては，文字列の検索のコマンド (C-s) のようにエスケープで終わる文字列の入力が必要なものがあります。このような場合には，エスケープを ESC で表現することにします。

C-s *string* ESC

つまり，編集コマンドの始まりとしてのエスケープは慣例に従って M- と表現し，それ以外でのエスケープは ESC と表現することにします。

ここで，エスケープとそれに続く文字による編集コマンドが M-○ と表現されることについて簡単に説明しておきましょう。Emacs が最初に開発されたアメリカの計算機のキーボードにはメタキーと呼ばれるキーが用意されていたようです。そして，コントロールキーのように，メタキーを押したまま他のキーを押すことによりメタ文字と呼ばれる文字の入力が行えました。Emacs ではこのメタ文字を編集コマンドとし，それをコントロール文字の表現 C-○ にならって M-○ のように表現したようです。日本でのコンピュータシステムの場合には，メタ文字のコードが半角カタカナのコードなどに相当するため，編集コマンドとして用いることができませんし，またメタキーもキーボードに用意されていません。アメリカのコンピュータシステムの中にも，メタキーの装備されていないシステムもあるようです。そこで，そのような場合も考慮して，Emacs では，ESC と，続く文字で編集コマンドを指定できるようにも設計されています。

1.4.3 編集コマンド名での指定

編集コマンド名での指定は，M-x とコマンド名，最後に RET を入力します。たとえばタブのサイズを変更するコマンド tab-set を実行するには，

M-x tab-set RET

と入力します。

しかし，すべての編集コマンドがこのような入力方法では，実用的ではないので，よく用いるコマンドは，コントロール文字あるいはメタ文字として簡便に指定できるようになっているわけです。ただし，コントロール文字，C-x あるいは ESC で始まる編集コマンドの数は限られますので，あまり頻繁には使用されないコマンドは，M-x を用いてだけ指定することができるようになっています。なお，編集コマンド名を知っていれば，コントロール文字やメタ文字のように簡便に指定できる編集コマンドを覚

え易いこともありますので，以後，いくつかのコマンドについては，たとえば，

M-f 単語単位の右へカーソルを移動します (forward-word).

のように，各コマンドの要約の部分に括弧で囲ってコマンド名を記す場合もあります。なお，GNU Emacs などでは，コントロール文字で実行可能なコマンドでも M-x とコマンド名でも実行できますが，Jmacs ではプログラムのサイズを小さくするためにコントロール文字，あるいはエスケープ文字で実行可能なコマンドでは，名前での指定はできなくしています。

1.4.4 入力されたコマンドのエコー行への表示

C-a のような1文字のコマンドは，入力するとただちに実行されますが，C-x や ESC で始まるような2文字以上の入力が必要なコマンドに対しては，それまでに入力したものが，エコー行に表示されます。ただし，エスケープなどは通常は表示できないので，Jmacs では ESC は，$, RET は c_n, これら以外のコントロール文字 C-○ は，^○ のように表示します。

2. 基本的な編集コマンド

Jmacs を使用するために必要な基本的な概念の準備ができましたので，とにかく実際に操作してみましょう。

2.1 Jmacs の起動

Jmacs は，OS のコマンドとして，

A > JMACS *file-spec*

と入力（下線部分）することにより起動されます。ここでA > は MS-DOS のプロンプトです。また *file-spec* とは，一般に OS のコマンドレベルで処理対象とするファイ

ルを指定するときの方法と同じであり，ディスク・ドライブ名とコロン(:)，パス名，ファイル名，ピリオド，拡張子からなり，ディスク上のファイルを指定するものを意味します。たとえば，Bドライブのディレクトリ ¥HONDA のファイル TEST. TXT を編集する場合には，

A > JMACS B: ¥HONDA¥TEST. TXT

と入力します。ただし，ディスク・ドライブ名，パス名，ファイル名などは大文字でも小文字でもかまいません。なお，ディスク・ドライブ名とコロン，パス名は必要な場合にだけ指定します。たとえば，現在ディレクトリがBドライブの ¥HONDA であれば，単に

B > JMACS TEST. TXT

と入力してもかまいません。

なお，以後のコマンドの説明における *file-spec* とは，ここでと同様にパス名を含む場合もあるものを意味するとします。

2.2 Jmacs の基本的な編集コマンド

まず，存在しない新規のファイル名(たとえば aaa)で Jmacs を起動してください。すると，上で説明したような構成の画面表示がなされ，Jmacs はユーザからの編集コマンドの入力を待ちます。この状態を Jmacs の**コマンド受付状態**と呼ぶことにします。Jmacs はこの状態で，入力されたコマンドを実行・処理し，再度このコマンド受付状態に入ります。このことを終了コマンドが入力されるまで繰り返します。

コンピュータでプログラムを起動したとき，その終了方法が分からずに困ることがよくありますので，まず Jmacs を終了するコマンドから説明します。

2.2.1 Jmacs の終了

Jmacsを終了するには,Jmacsのコマンド受付状態でC-x C-cと入力します。もし,Jmacsを起動してから,あるいはバッファ上のテキストをファイルに保存するコマンドによって保存されて以後,そのバッファが変更されていなければ,ただちにJmacsを終了し,OSのコマンドレベルに戻ります。もし変更されていれば,エコー行に

Save file: *file-name (y or n)* ?

と尋ねてきます。ここで,yを入力すると,変更した結果をファイルに保存したのち,Jmacsを終了し,OSのコマンドレベルに戻ります。nを入力すると,

1 modified buffers exist, do you really want to exit ? (yes or no)

と尋ねてきます。いったんJmacsを終了すると,変更した結果は復元できませんので,保存せずに終了する場合には,本当に終了してよいかを再度尋ねてくるわけです。

ここで,'yes RET'と入力した場合には,変更結果をファイルに保存せずに終了し,OSに戻ります。'no RET'と入力した場合には,Jmacsのコマンド受付状態に戻ります。ただし,'と',およびyesとRETの間の空白は,入力しないでください。見やすくするために入力する文字列の間に空白をあけて表示するときには,このように'と'で入力を囲んで表現することにします。空白の入力が必要な場合には,SPCを用いて表現しますので,説明中の入力中の文字列間の空白は入力しないでください。

なお,'yes RET'でも'no RET'でもない入力に対しては,また,

1 modified buffers exist, do you really want to exit ? (yes or no)

と尋ねてきますので,'yes RET'または'no RET'を入力してください。

2.2.2 新規ファイルへの文字の入力

まず,文字の入力・挿入について説明します。新規のファイル名を指定してJmacs

を起動した場合は，新しいファイルであるため，テキストウインドウには何も表示されず，カーソルは1行目の最初に位置しています。以後，この状態から始まるものとして説明します。なお，◇で始まる箇所では指示されたように文字あるいはコマンドを実際に入力し，その結果を確認してください。

◇ Jmacs のコマンド受付状態で，キーボードから，通常の文字 a を入力してください。

するとカーソルがあった位置に a の文字が表示され，カーソルはその右に移動します。このように，Jmacs ではコマンド受付状態で，通常の文字をキー入力した場合には，その文字がバッファのポイントの位置に挿入（入力）され，ポイント（およびカーソル）はその挿入された文字の右に移動します。別の見方をすれば，通常の文字はそれ自身をバッファに入力するコマンドと考えられます。

◇つづいて 'bcdefg SPC hijk SPC' と入力してください。

カーソルが移動しながら，順に 'bcdefg SPC hijk SPC' がバッファに入力・表示され，最後にカーソルは文字 k の右に 1 文字分空白を空けた位置に移動します。空白に対してはテキストウインドウには何も表示されませんので，文字と文字の間の場合はともかく，この場合のような行末の空白にはカーソルを移動しなければ，その存在が分かりません。したがって，プログラムや文書の作成時に，しばしば行末やバッファの最後に無駄な空白を残しやすいので注意してください。もっとも，残していたとしても，主記憶やディスクの容量を多少余計に占有するだけで，ほとんどの場合には実害はないでしょう。

◇さらに lmnopqr と入力し，つづいて RET を入力してください。

カーソルは，次の行の先頭に位置します。RET を入力した場合，RET は通常の文

字と同様にバッファのポイントの位置に入力されます。RET は画面上では，改行しているように表示されますが，表示以外の点では，通常の文字と同様に1文字として扱われます。このことは，以下で説明するカーソル移動や文字の消去・削除において除々に明らかになると思いますが，意識しておいてください。

◇ここで順に 'stuv TAB wxyz RET stu TAB vwxyz RET st TAB uvwxyz RET' および 'abcde SPC fghij SPC klmno SPC pqrst SPC uvwxy SPC z RET' を入力してください。

これまでの入力で，CRT 画面の表示は図5の状態になっています。TAB を入力したところは，ある特定の位置までいくつかの空白があるように表示されます。ただし，バッファ中の TAB は，その表示は通常の文字と異なりますが，表示以外では，通常の文字と同様に1文字として扱われます。

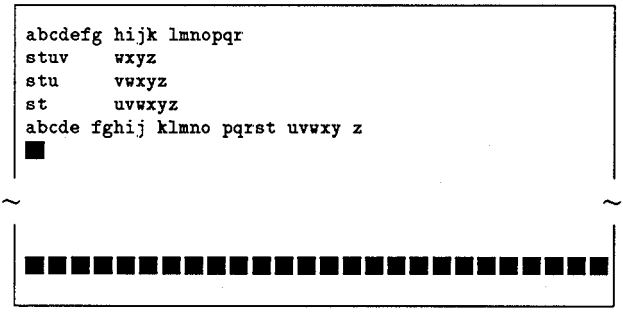


図5 文字入力後の画面表示

2.2.3 上下左右への基本的なカーソル移動

カーソル移動のコマンドとしては，つぎに示すようなコマンドがあります。なお，順方向とはバッファの最後の方向，逆方向とはバッファの先頭の方向を意味します。

C-p 1行上（逆方向）へカーソルを移動します (previous-line)

C-n 1行下（順方向）へカーソルを移動します (next-line)

C-f　1文字右（順方向）へカーソルを移動します (forward-char).

C-b　1文字左（逆方向）へカーソルを移動します (backward-char).

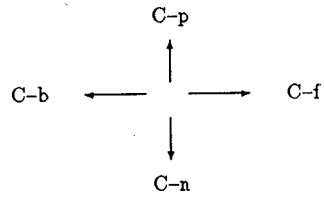


図6　カーソル移動コマンド

◇C-p, C-n, C-f, C-b を適当に入力して，カーソルの移動を試みてください。行の先頭での C-b, あるいは行の最後での C-f を試みて，テキストの存在しない部分へのカーソル移動はできないことを確認してください。

原則的には，C-p, C-n, C-f, C-b は，それぞれ，上，下，右，左への1行あるいは1文字の移動となりますが，TAB や RET をカーソルが通過する上下左右への移動の場合には，その原則に従わないように見える移動をすることがあります。そのような移動について，以下 (A) と (B) で説明しますが，そのことがあまり気にならなければ，2.2.4 へ進んでください。

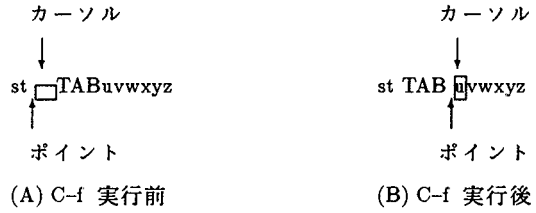
(A)　TAB, RET に関連した C-f, C-b によるカーソルの移動

通常の文字の前後での C-f, C-b によるカーソルの移動は，正確に1文字分の移動となりますが，TAB, RET の前後では1文字の移動とはならない場合があります。

◇図5のテキストの状態で，4行目の文字 t の右の TAB を通過する左右へのカーソルの移動を試みてください。

C-f あるいは C-b により，カーソルが TAB を通過するときは，その TAB に応じた

いくつかの空白を一度に移動することに注意してください。このことは，ポイントとカーソルの関係を考えれば分かりやすいでしょう。さきに述べたようにJmacsでは，バッファ上の文字と文字の間に位置するポイントが操作対象となる位置を示しますが，画面上で正確にポイントの位置を表示することができないため，ポイントの右の文字の位置にカーソルを表示しています。したがって，たとえば図7のように，TABの左にポイントがあるときに，C-fが入力されれば，ポイントは1文字分順方向に進み，TABの右，uの左に位置します。その結果として，カーソルは文字uの上に位置することになります。



TABの左右の空白は見やすくするためのもので，実際のテキスト中にはないと考えてください．

図7 TABの前後のポイントとカーソル

行末あるいは行頭でのC-fやC-bによるカーソル移動についても，同様にテキスト中のポイントを考えれば納得できるでしょう。つまり，ポイントがRETを通過することになり，結果として次の行の先頭あるいは最後にカーソルが移動することになります。

(B) TAB, RETに関連したC-p, C-nによるカーソルの移動

通常の文字の部分でのC-p, C-nによるカーソルの移動は，正確に真上あるいは真下への移動となりますが，TAB, RETがあるときは，真上あるいは真下への正確な移動とはならない場合があります。

◇まず，図5のテキストの状態で，カーソルを1行目の最初に移動したのち，C-f により文字fの上に移動してください。そして，C-n を入力してください。カーソルは，2行目の文字wの上に移動します。

C-n では，ポイントはつぎの行の同じカラム位置を目標として移動します。しかし，上の場合のように，次行の目標カラム位置がTABに応じた空白の途中の部分(最初の空白ではない)であれば，ポイントはTABと次の文字の間に位置することになり，したがって，カーソルはTABのつぎの文字の上に移動します。ただし，目標カラム位置がTABの直前となるばあいには，カーソルはそのTABに応じた最初の空白の位置に移動します。このことは，つぎのようにして確かめられます。

◇図5のテキストの状態で，カーソルを1行目最初に移動したのち，C-f により文字eの上に移動してください。そして，C-n を入力してください。カーソルは，2行目の文字のvの右，すなわちTABに応じた最初の空白の位置に移動します。

C-n が連続して入力された場合のカーソルの移動について説明します。まず，次のように入力してください。

◇上の操作につづけて，さらにC-n を入力してください。カーソルは，3行目の文字vの上に移動します。このことを確認したあと，さらにC-n を入力してください。カーソルは4行目の文字uの上に移動します。つづいて，もう一度C-n を入力してください。こんどは，カーソルは5行目の文字cの上に移動します。

C-n が連続して入力された場合，2回目以降のC-n に対しても，ポイントは最初のC-n が入力されたときのカラム位置を目標カラム位置として，次行へと進みます。もしその目標カラム位置が，TABに応じた空白部分の途中の場合には，TABと次の文字の間に移動することになります。したがって，結果としてカーソルは，TABの次の文字の上に位置します。上の最後のC-n の場合のように，目標カラム位置に通常の文

字があれば，ポイントはその位置に移動します。したがって，カーソルは最初のC-nのときと同じカラム位置の文字の上に位置することになります。

一方，もし次の行が目標カラムよりも短い場合には，ポイントはつぎの行のRETの前に移動し，結果として，カーソルはつぎの行の最後に位置します。

◇図5のテキストの状態で，カーソルを1行目の最初に移動したのち，C-fにより1行目の最後，文字rの右にカーソルを移動してください。そのあと，C-nを入力してください。カーソルは，2行目の最後，文字zの右に移動します。

◇つづけてC-nを入力すると，4行目までは目標カラムよりも短いので，カーソルは行の最後に位置しますが，5行目では目標カラム位置に通常の文字pがあるので，カーソルはその文字の上に位置します。

なお，上方への移動のC-pについても，C-nと同様なカーソル移動となります。C-nの場合と同様にして確認してください。

2.2.4 バッファへの文字の挿入

2.2.2での通常の文字の入力は，すべてポイントがバッファの最後にあったときの入力なので，バッファの最後への追加入力となりました。Jmacsではコマンド受付状態で，通常の文字をキー入力した場合には，その文字がバッファ中のポイントの位置に入力され，カーソルはその入力された文字の次に移動するのが原則です。したがって，もしバッファの途中にポイントがあれば，キー入力された文字はそのポイント位置への入力，つまりテキスト途中への挿入となります。

行の途中で，RETがキー入力された場合，すでに述べたように，RETも通常の文字と同じように扱われ，バッファ中に1文字として入力され，ポイントはRETの次の位置に移動します。RETをCRT画面に表示すると，そこで改行が行われたように見え，結果的には，カーソルの位置で行が2つに分割されたことになります。

◇上下左右へのカーソル移動コマンドを用いて，テキストの途中の文字の位置にカーソルを移動してください。そのあと，通常の文字をキー入力してください。また，行の途中で RET をキー入力してその結果を確認してください。

2.2.5 バッファ中の文字の消去

バッファ中の文字を１文字単位で消去するのには，ポイントの位置からバッファの先頭方向（逆方向）への消去と，バッファの最後の方向（順方向）への２通りの消去があります。逆方向への消去には，キートップに BS のように表示されたバックスペースキーを用います。また，順方向への消去には，C-d を用います。なお，これらのコマンドには，右側に記した名前がつけられています。

BS　逆方向へ一文字消去します (delete-backward-char)
C-d　順方向へ１文字消去します (delete-char)

◇上下左右へのカーソル移動コマンドを用いて，行の途中の位置にカーソルを移動したあと，まず，BS を１度入力してください。

BS ではポイントの前の文字，すなわちカーソルの位置している直前の文字が消去されます。そして，カーソル以降の文字は一文字分先頭方向に詰められます。カーソルのカラム位置は１文字分前に移りますが，同じ文字の上に位置していることを確認してください。

◇さらに，連続して BS を入力して，文字が逆方向に連続して消去されるようすを確認してください。

◇今度は，C-d を１回入力してください。

C-d ではポイントの後の文字，すなわち，カーソルが位置している文字が消去され

ます。そして，カーソル以降の文字は1文字分先頭方向に詰められます。カーソルのカラム位置は同じですが，消去前のカーソル位置の次の文字の上に位置していることを確認してください。

◇さらに，連続してC-dを入力して，文字が順方向に連続して消去されるようすを確認してください。

◇2行目以降の先頭でBSを入力して，その行と前の行がつながることを確認してください。また，行の最後でC-dをキー入力しても，2つの行がつながることを確認してください。

何度も述べましたが，バッファ中のRETは，表示は特殊ですが，他の点においては通常の文字と同様に扱われます。したがって，もしRETの後にポイントがあり，BSが入力された場合には，RETが消去され，結果としては2行がつながり1行となります。行末でのC-dも同様な理由で2行をつなげる結果となります。

このことはバッファ中のTABについても同様で，BSあるいはC-dによりTABを消去した場合には，TABに対応したいくつかの空白が消去されたように見えることも確認してください。

2.2.6 編集コマンドの取り消し

Jmacsでは，2回以上のキー入力が必要な編集コマンドの場合，まだ最後まで入力し終わっておらず，したがって実行されていないときには，コマンドC-gで，その編集コマンドに対するそれまでの入力を取り消して，コマンド受付状態に戻ります。

C-g 現在入力途中のコマンドの入力を取り消し，コマンド受付状態に戻ります。

◇C-xを入力して，エコー行に表示されたのち，C-gを入力して，コマンド受付状態に戻ることを確認してください。

現在のバージョンでは，実行されてしまったコマンドの結果を取り消して，以前の状態に戻す Undo の機能は実現されていません。

3. Jmacs の編集コマンド

第 2 章では，とにかく Jmacs を使うための基本的な編集コマンドを説明しました。以下では，それ以外の編集を有効に進めるためのコマンドについて，必要な概念・用語とともに説明します。ただし，キーボード・マクロについては第 4 章で，マルチバッファとマルチウインドウについては第 5 章で説明します。

なお，以下では，編集コマンドによっては，◇で始まる使用例の指示をしない場合もありますので，説明の切れ目等の適当な時点で，いろいろなテキストとカーソルの状態で編集コマンドを試みてください。

3.1 テキストウインドウの枠を越えるカーソル移動

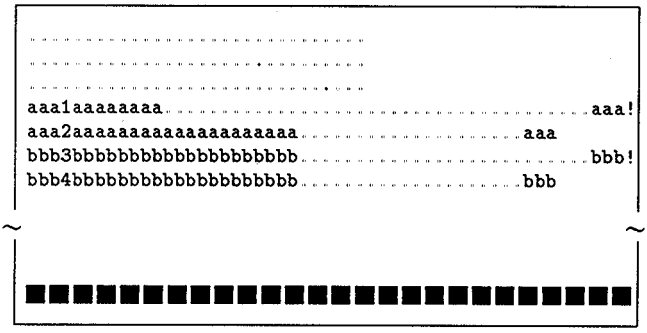
テキストウインドウの横幅を越えるような長い行の入力と，上下の枠を越える挿入・削除およびカーソルの移動などで，画面の表示がどうなるかを説明します。

3.1.1 長い行の表示とカーソル移動

Jmacs のコマンド受付状態で通常の文字を入力すると，その文字がバッファのポイントの位置に挿入されますが，もし，挿入によりポイントのある行がテキストウインドウの横幅よりも長くなる場合には，画面上では折り返して 2 行以上に渡って表示されます。ただし，それらの行が連続した行であることを示すために折り返す行の最後に！が表示されます。なお「行」という言葉を，「バッファ上で RET と RET の間の文字の並びとしての行」と，「画面上での行」との 2 通りの意味で用いていることに注意してください。以後の説明では，ほとんどの場合は，どちらを示しているかは前後関係から明らかな場合が多いと思いますが，もし両者を区別する必要のある場合には，前者を「バッファ上の行」，後者を「画面上の行」あるいは「ウインドウ上の行」という呼び方をします。

◇2行に渡るように文字を入力し，その表示がどうなるか，とくに画面上で折り返される部分の！を確認してください。また，その！を通過するようなカーソル移動のコマンド C-f, C-b を試みてください。

カーソルの上下への移動のコマンド C-n，C-p でいう行とは，画面上の行を指します。したがって，画面上で2行以上に渡って表示されている長い行に対して，C-n あるいは C-p は，画面上では次の行への移動であっても，バッファ上では移動先は同じ行である場合もあります。



移動前カーソル位置	コマンド	移動後カーソル位置
行Aの1の上	C-n	行Aの2の上
行Aの2の上	C-n	行Bの3の上
行Bの3の上	C-p	行Aの2の上
行Aの2の上	C-p	行Aの1の上

図8　長い行に対するC-p, C-n によるカーソル移動

◇図8のように2つの長い行を入力し，C-n と C-p によるカーソルの移動を確認してください。

なお，GNU Emacs などでは，C-n と C-p は，画面上の行に対するカーソル移動で

はなく，バッファ上の行に対するカーソル移動となります。したがって，図8の行Aの1の上にカーソルがあったとき，C-n では，行Bの3の上にカーソルが移動します。また，行Aの2の上にカーソルがあれば，コマンド C-n では，行Bの4の上に移動します。GNU Emacs などを使用する場合には注意してください。

3.1.2 画面表示のオーバーラップ行

Jmacs では，テキストウインドウの最下行で RET を入力した場合，次のカーソル位置がウインドウ内になるように，ウインドウ内でテキストがスクロールされます。そのとき表示されるテキストのうち，上の何行かは，前にウインドウの下の何行かとして表示されていたものです。つまり，両表示で重複する部分（オーバーラップ行）があります。これは，ウインドウの境界付近のテキストのつながり具合いがよくわかるようにするための配慮です。この重複して表示される行数は，ウインドウの高さにより異なりますが通常は1行から4行程度です。

◇テキストウインドウの最下行まで，文書あるいはプログラムを入力してください。さらに，最下行で RET を入力すれば，テキストの表示がどのようになるか注意してください。入力には，適当な文書あるいはプログラムを参考にすればよいでしょう。ただし，各行は，ある程度の長さがあれば充分で，画面の右端まで入力する必要はありません。

このようなウインドウ単位でのテキストのスクロールは，文字入力のときだけでなく既に説明した C-f, C-b, C-p, C-n などのコマンドの実行により，ウインドウに表示されている最上行あるいは最下行でのカーソル移動に際してもおこなわれます。

(a) ウインドウの最上行の左端での C-b の入力
(b) ウインドウの最下行の行末での C-f の入力
(c) ウインドウの最上行の行途中での C-p の入力
(d) ウインドウの最下行の行途中での C-n の入力

画面ではテキストがスクロールされて表示されますが，テキストとカーソルの位置関係は，同一画面内でカーソルが移動する場合と同じです。

◇再表示されたあとの画面に対して，もう数行テキストに入力してください。そして，上記のようなウインドウの枠外にカーソルが移動する場合を試みて，画面の表示を見てください。ただし，いずれもその方向にテキストがある場合に試みてください。

3.1.3 コントロール文字のテキストへの挿入

ファイルを字体，字間あるいは行間隔を替えながらプリンタに印刷する場合，そのファイルのテキストに直接プリンタの制御コマンドを挿入する方法があります。これらの制御コマンドは，一般にはESCなどのコントロール文字で始まりますが，それらのほとんどがJmacsの編集コマンドに該当しますので，単にそのコントロール文字を入力したのでは，編集コマンドとして解釈されテキストへの挿入とはなりません。そこで，そのようなコントロール文字をテキストに挿入する場合のために，Jmacsには編集コマンドC-qがあります。

C-qはつぎに入力される文字が何であれ，つまりコントロール文字であれ，通常の文字であれ，その文字を編集コマンドではなく通常の文字と同様に扱いテキストに挿入するコマンドです。

一般に，キーボードからのRET (改行)あるいはC-mのキー入力では，コントロールM (16進数でD) のコードがプログラムに渡されます。JmacsではMS-DOSのテキストの管理に併せて，通常は，入力されたコントロールMのコードをコントロールMとコントロールJ (16進数でA) の2バイトのコードに変換します。ただし，C-qのあとのRET (C-m) については，そのままのコードか，2バイトのコードに変換するかを選択できます。選択の方法については，3.12.4で説明します。

◇テキストの適当な位置でC-q C-aと入力してください。

挿入されたコントロール文字は，ウインドウでは，記号文字^を用いて２文字分で表示されます。たとえばC-q C-a と入力したときにはC-a がテキストに挿入され，^A と表示されます。C-a を入力したときのａが英小文字であっても，ウインドウには英大文字で表示されます。

ところで，テキスト中に連続して文字^と文字Ａがある場合にも，その表示は^A となります。つまり１文字のコントロール文字C-a と２文字の^とＡは画面上で見ただけでは区別することはできません。しかし，編集コマンドC-f またはC-b を用いて，その文字を通過するようにカーソルを移動することにより識別できます。もし，１回のC-f またはC-b でカーソルが２文字分移動すればコントロールＡであり，１文字分しか移動しなければ^とＡの２文字です。

◇先に入力したコントロール文字を通過するように，C-f あるいはC-b を試みてカーソルの移動を確認してください。

3.2　カーソルの移動

カーソル移動のコマンドとしては，既に述べたC-p, C-n, C-f, C-b などの１文字あるいは１行単位の移動コマンド以外に，より大きな単位での移動コマンドとして，単語単位の移動，行頭と行末へのバッファの先頭と最後への移動などのコマンドがあります。これら以外にも，たとえば，指定した文字列を検索するコマンドのように，実行結果としてカーソルが移動するようなコマンドもいくつかあります。また，ウインドウをテキストバッファに対して上下に移動して画面に表示されるテキストの部分を移動させる画面制御のコマンドなどもあります。ただし，文字列検索と画面制御などでのカーソル移動は，それぞれのところで説明します。

3.2.1　単語単位でのカーソル移動

M-f　単語単位で右（順方向）へカーソルを移動します (forward-word)。
M-b　単語単位で左（逆方向）へカーソルを移動します (backward-word)。

単語とは英字と数字からなる文字の並びを意味します。SPC (空白), %, $, TAB, RET などは，単語に対する句切り文字とみなします。

カーソルが単語の先頭あるいは途中の文字の上にあれば，M-f により，その単語の直後の句切り文字の上に移動します。もし，句切り文字の上にあれば，その位置から順方向に最初に出会う単語の直後の句切り文字の上に移動します。

一方，M-b では，カーソルが単語の途中あるいは直後にあれば，その単語の最初の文字の上に移動します。もし，句切り文字の上にあれば，その位置から逆方向に出会う単語の最初の文字の上に，カーソルを移動します。

◇カーソルが単語の最初，単語途中，単語と単語を区切る空白や特殊文字の上にある場合に，M-f と M-b による，カーソルの移動を確認してください。

3.2.2 行の先頭および最後へのカーソル移動

C-a 行の先頭へカーソルを移動します (beginning-of-line)

C-e 行の最後へカーソルを移動します (end-of-line)

ここでいう行とは，バッファ上の行，すなわちテキストバッファ上の RET で区切られた行です。したがって，画面上で2行以上に渡って表示されている場合，C-a により，カーソルは画面上では何行か上の行の先頭に移動することもあります。C-e についても同様な場合には，カーソルが画面上では何行か下の行の最後に位置することもあります。

なお，Jmacs 固有のコマンドですが，画面上の行の先頭あるいは最後への移動などに関して，次のコマンドもあります。

C-^ C-a 画面上の行の先頭へカーソルを移動します。

C-^ C-e 画面上の行の最後へカーソルを移動します。

C-^ < 画面上の最上行の左端へカーソルを移動します。

C-^ > 画面上の最下行の左端へカーソルを移動します。

コマンド C-^ C-a は，画面で 2 行以上に渡って表示されている長い行であっても，その画面上の行の先頭にカーソルを移動します。逆に C-^ C-e は画面で 2 行以上に渡って表示されている長い行であっても，その画面上の行の最後にカーソルを移動します。またコマンド C-^ < と C-^ > は，それぞれ現在の画面の左上端，左下端に移動します。たとえ，最上行が長い行の 2 行目以降であっても，現画面の左上端に移動します。

3.2.3 バッファの先頭あるいは最後への移動

M-< バッファの先頭へカーソルを移動します (beginning-of-buffer)。
M-> バッファの最後へカーソルを移動します (end-of-bufier)。

ポイントがどこに位置していようと M-< では，ポイントはバッファの先頭に移動します。したがって，画面にはテキストの最初の部分が表示され，カーソルは先頭に移動します。逆に，M-> では，ポイントはバッファの最後に移動します。

3.3 画面制御

C-v 順方向へ 1 画面（ウインドウ）スクロールします (scroll-up)。
M-v 逆方向へ 1 画面（ウインドウ）スクロールします (scroll-down)。
C-1 現在カーソルのある行が画面（ウインドウ）上の中央の行となるように，何行か上下にスクロールして再表示します (recenter)。

C-v では，順方向にほぼ 1 画面の行数分テキストをスクロールします。このとき，実行前に表示されていた下の何行かがオーバーラップして，つぎの上部に表示されます。カーソルは，そのオーバーラップ行の次の行の先頭に位置します。

また，M-v では，逆方向にほぼ 1 画面分の行数分テキストをスクロールします。C

-v と同様に，実行前の上の何行かがオーバーラップして表示され，カーソルはオーバーラップ行の前の行の先頭に位置します。

C-l では，カーソルが現在位置している行を画面の中央の行となるように，テキストを上下にスクロールし，再表示します。ただし，バッファの最初から画面の半分以内にカーソルが位置している場合のように，画面中央にその行を位置させるのが不可能な場合には，そのようなスクロールは行われません。

◇2画面以上に渡るテキストに対して，C-v, M-v を試みて，画面表示を確認してください。また，C-l も試みてください。

画面単位の移動で，C-v は比較的入力し易いのですが，逆方向へのスクロールは比較的頻繁に使用するわりには ESC と v のキーを2回押さなければならず，あまり使用しやすくありません。そこで，GNU Emacs などとは異なりますが，Jmacs では，M-v と同じ働きをするものとして C-z を使用することにしていますので，そちらを使用した方が便利でしょう。ただし，GNU Emacs などを使用する場合には注意してください。なお，著者は，GNU Emacs でも scroll-down のコマンドを C-z にキー割当を変更して使用しています。

C-z 逆方向へ1画面（ウインドウ）スクロールします (scroll-down)。

3.4 編集コマンドの繰り返し実行——数引数

いくつかの Jmacs の編集コマンドは，実行の繰り返し回数を数引数として指定することのできるものがあります。数引数の指定は，その繰り返したいコマンドの直前にコマン C-u と回数を入力します。

たとえば，C-u 20 C-f と入力することにより，C-f を 20 回繰り返し実行させることができます。C-u と回数の直後に入力できるコマンドは1文字のコントロール文字のコマンドだけでなく，ESC で始まるような2文字以上の入力が必要なコマンドでもかまいません。C-u 3 M-f と入力すれば，単語単位のカーソルの移動コマンド M-f を 3

回入力したのと同じことになります。また，C-u と回数につづいて，コマンドではなくて通常の文字が入力された場合には，指定された回数その文字を入力したと同じこと，つまり，その個数のその文字がテキストに挿入されることとなります。

回数の指定がなくて C-u につづいて編集コマンドが入力されたときは，回数として 4 が指定されたものとなります。たとえば C-u C-f は C-u 4 C-f と同じことです。また C-u と数引数が 2 回以上入力された場合には，繰り返しの回数は，各 C-u で指定された回数の積となります。たとえば，C-u 5 C-u 6 C-f は C-f を 30 回入力したのと同じであり，C-u C-u C-f は 16 回の C-f の入力と同じことです。

コマンドによっては，負の数引数をとることができるものもあります。その場合，GNU Emacs などでは，原則的には指定されたコマンドと反対のことを行います。ただし，Jmacs では現在のところ，負の引数が次の C-l のように特別の意味のある場合についてだけ処理が行われ，それ以外の場合には，無視されます。

なお，正負の数引数について以上で述べたことはあくまで原則であり，編集コマンドによっては，数引数がそのコマンドの繰り返しとはならない場合もあります。たとえば，現在カーソルのある行を画面の中央に位置させる C-l に対して数引数が与えられた場合は，つぎのようになります。

C-l　数引数 n が正の場合には，カーソルのある行が画面の上から n+1 行目になるようにスクロールして再表示します。n が負の場合には，カーソルのある行が画面の下から −n 行目になるように再表示します。

数引数を与えて行う操作には，単純な繰り返しの場合や，上記の C-l の場合のように他のコマンドを用いても比較的簡単にできる操作もありますが，なかには C-u を用いなければできない操作もあります。また，C-u が入力されていることだけが意味をもち，たとえ C-u のあと回数が与えられても，その値は意味を持たない場合もあります。このような場合は，C-u で始まる別のコマンドと考えた方がよいかもしれません。以後の各コマンドの説明に際しては，主として，数引数が単純な繰り返しとならない場合について，数引数の役割について説明することにします。

なお，C-1以外にも，これまで説明した編集コマンドのうた，C-v, M-v, M-<, M-> に対しては，数引数はそのコマンドの単純な繰り返しの指定とはなりませんので注意してください。

3.5 マークとリージョン（領域）

C-@ 現在のカーソル位置をマーク位置として設定します
(set-mark-command)

Jmacsには，テキスト上の位置を記憶しておくための**マーク**という概念があります。コマンドC-@により，現在カーソルのあるテキスト上の位置がマークに設定されます。マークはいったん設定されると，次に設定されるまでは変更されません。したがって，C-f, C-b, C-p, C-n等でカーソルを移動してもマークに設定された位置は変わりません。なお，「以後，マークに設定されているテキスト上の位置」のことを，単に「マーク位置」ということもあります。

マーク位置とカーソル位置の間のテキストを**リージョン（領域）**といいます。ただし，マーク位置とカーソル位置のテキスト上での前後の位置関係の制限はありません。どちらが前方であっても同じリージョンとなります。マークはカーソルの迅速な移動に用いられ，リージョンはテキストの削除，バッファやファイルへの保存に用いられます。

C-x C-x マーク位置とカーソル位置を入れ換えます(exchange-point-and-mark)。
C-w リージョンを削除します(kill-region)。
C-^ C-w リージョンを指定されたファイルへ保存します。

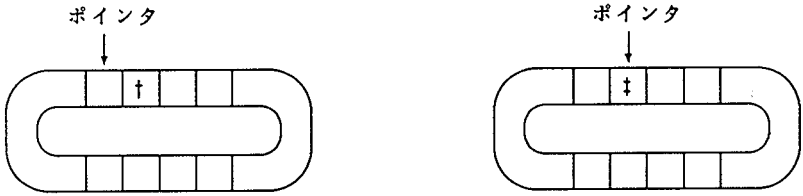
コマンドC-wとC-^ C-wについては，それぞれ3.6と3.11.4で説明します。
コマンドC-x C-xは，マーク位置とカーソル位置を入れ換えます。マーク位置が現

画面内にあれば，カーソルがマーク位置に移動するだけですが，マーク位置が現画面外の場合には，マーク位置を含むテキストの部分を画面に表示することになります。なお，C-x C-x はマーク位置とカーソル位置を入れ換えるので，その実行直後に再度 C-x C-x を入力すればマーク位置とカーソル位置は元に戻ります。

マークはテキスト中の位置を記憶させておき，いったんカーソルを別のところに移動したあと再度，その位置にカーソルを戻すときに利用できます。なお，このようなテキストの位置を記憶するものとして，**レジスタ**というものもありますが，これについては 3.7 で説明します。

◇テキストの適当な位置をマークに設定し，C-f, C-b, C-p, C-n, C-v, M-v 等でカーソルを移動したあと，C-x, C-x を入力してください。また，その直後に C-x C-x を再度入力してください。

マークは，マークリングと呼ばれるいくつか（16 個程度）のマーク位置が記録できるリング状の入れ物と，現在のマークの位置を指すポインタで管理されています。マークに設定されている位置とは，マークリングの中のポインタで指されているところに記録されている位置のことです。マークが設定されるときは，リング内でポインタを（図 9 では右回りに）1 つ進めたあと，そのポインタの指すところにテキストの位置が記録されます。ただし，図 9 はそのようなマークリングの状態をイメージ的に表わしたもので Jmacs での実現方法と一致したものではありません。



(a) 設定前　　(b) 設定後

以前に設定されていた位置 † は，‡ に変更される。

図 9　マークリング

このように以前に設定されたいくつかのマークは，設定された順にマークリングに保存されていますので，以前のマーク位置にカーソルを戻すことができます。そのためにはC-@ に対して数引数を与えます。

C-u C-@ マーク位置とカーソル位置を入れ換えたあと，マークリングのポインタを1つ戻します。なお，この場合C-u のあとに数値が与えられてもその値はなんら意味をもちません。

C-u C-@ はマークリングのポインタを1つ戻しますので，繰り返して入力すると，つぎつぎと遡ってリングに記録されたマーク位置にカーソルを移していくことになります。

◇テキスト abcde に対して，a から順に e までの位置をマークに設定したあと，C-x C-x, C-u C-@, ···, C-u C-@ と入力し，カーソルの移動を確認してください。また，同様にマークを設定したあと，C-u C-@, C-u C-@, ···, C-u C-@ と入力し，カーソルの移動を確認し，その違いについて確認してください。

なお，マークの設定はC-@ 以外の編集コマンドでも副作用的に行われる場合があります。たとえば，いままで述べたコマンドのうち M-< と M-> は現在のカーソル位置をマーク位置として設定したあと，カーソルをバッファの先頭あるいは最後に移動します。したがって M-< あるいは M-> の直後に C-x C-x を入力すると，カーソルは元の位置にもどります。

3.6 テキストの消去・削除と複写・移動

Jmacs には，テキスト中の文字の消し方として，復元できる消し方と，復元できない消し方があります。前者の消し方を**削除 (kill)**，後者の消し方を**消去 (delete)** という呼び方をします。消去のコマンドとしては BS と C-d がありますが，すでに説明していますので，ここでは削除と復元のコマンドについて説明します。

3.6.1 テキストの削除

M-d　順方向へ単語を削除します (kill-word)
M-BS　逆方向へ単語を削除します (backward-kill-word)。
C-k　カーソル位置から行末までを削除します (kill-line)。
C-w　リージョンの削除 (kill-region)。

M-d で削除される範囲は，カーソル位置から，コマンド M-f を実行したときカーソルの移動する位置までです。同様に，M-BS では，カーソル位置から，M-b で逆方向にカーソルの移動する位置までのテキストが削除されます。いずれも，数引数があれば，その指定した回数の繰り返しとなります。

C-k はカーソルの位置が行末であるか否か，数引数が与えられているか否か，によって処理が異なります。

(a) カーソル直後の文字が RET でない（カーソルが行の途中にある）場合：
そのカーソル位置から行の最後，RET の直前までのテキストが削除されます。
(b) カーソルの直後が RET の場合：その RET だけを削除します。

したがって，ある行の先頭で連続して C-k を入力すると，まず最初の C-k で，その行が空行——何も表示されない RET だけの行——となり，2 つ目の C-k で RET が削除されて次の行が繰り上がります。さらに，その行でも同様に，まず空行となり，つぎに削除されて次の行が繰り上がり，…となります。つまり何行かが削除されることとなります。

C-k に対し数引数が与えられた場合には以下のようになります。

(a) 0 の場合：
カーソル位置からバッファ上の行の先頭まで，つまり逆方向に最初に出会う RET の直前までのテキストが削除されます。

(b) 正の数引数の場合：
与えられた場合は，カーソル位置から順方向にその個数分のRETまでのテキストが削除されます。

(c) 負の数引数の場合：
カーソル位置から逆方向に，その個数＋1分のRETの直後までのテキストが削除されます。

C-wでは，リージョン，すなわちマーク位置とカーソル位置の間のテキストが削除されます。なお，リージョンの設定は行にとらわれることなく，テキストの任意の位置に設定できますので，たとえば1行中のある部分をリージョンとして設定して削除できます。またある行の途中から別の行の途中までをリージョンとして設定して削除することもできます。

3.6.2 キル・バッファとテキストの複写・移動

M-BS, M-d, C-k, C-wなどで削除されたテキストは，キル・バッファと呼ばれるところに保存されます。連続して削除コマンドが入力された場合，削除されたテキストはひとまとまりのものとしてキル・バッファに保存されます。保存されているテキストをカーソル位置に復元・挿入するコマンドがC-yです。

C-y キル・バッファに保存されているテキストをカーソル位置に復元・挿入します(yank)。このとき，復元・挿入されたテキストの先頭をマークに設定します。

◇C-kにより何行かのテキストを削除したのち，C-yで復元してください。またM-BS, M-dあるいはC-kでいくつかの単語を削除したのちC-yで復元してください。これらの削除コマンドを取り混ぜてテキストを削除しても，連続して削除される限りC-yで元のように復元されます。

なお，キル・バッファに保存されているテキストは削除するコマンド以外では変更

されません。したがって，C-y でテキストを復元・挿入してもキル・バッファのテキストは依然としてそのまま保存されており，さらに C-y を用いることにより何度でも復元・挿入できます。

先にも述べたように，C-w では，ある行の途中から別の行の途中までをリージョンとして削除できますので，このコマンド C-y と組み合わせて用いれば，テキストの任意の部分を複写・移動できます。たとえば削除を行ったのち，その位置で C-y を入力し，ついでカーソルを別の位置に移動し C-y を入力すれば，**テキストの複写**ができますし，削除した位置以外で復元・挿入を行えば，**テキストの移動**となります。

このようにキル・バッファへテキストを保存することにより，テキストの移動や複写ができますが，複写のためには，リージョンのテキストを削除することなくキル・バッファに保存するコマンド M-w があります。

M-w　リージョンのテキストを削除することなくキル・バッファに保存します (copy-region-as-kill)。

キル・バッファもマークと同様に，リングのイメージでつながったいくつか（最大 10 個程度）のキル・バッファと，現在のキル・バッファを指すポインタで管理されています（図 10)。ただし，図 10 はそのようなキル・バッファの状態をイメージ的に表現したもので Jmacs での実現方法と一致したものではありません。

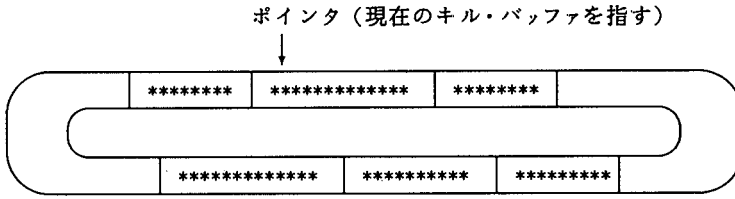


図 10　キル・バッファのリング

削除コマンドが連続して入力されている限りポインタを進めずに同じキル・バッ

ファに追加して保存されますが，削除以外のコマンドを実行したあと，さらに削除コマンドを入力したときには，ポインタを1つ（図10では右方向に）進めて，削除されたテキストをつぎのキル・バッファに保存します。

テキストを編集しているときには，テキストのあちらこちらに散らばっている部分を，ある位置にまとめたい場合もあります。このようなとき，テキストのある部分を削除したのち別の位置にカーソルを移動し，その位置でさらに削除コマンドを用いても，削除コマンドの間に削除以外のコマンドが実行されますので，削除されたものは，それぞれ別のキル・バッファに保存されます。しかし，削除以外のコマンドを実行したあとでも，各削除コマンドの直前でコマンド C-M-w を入力しておけばポインタを進めずに同一のキル・バッファにまとめて保存することができます。

C-M-w　この直後のコマンドで削除するテキストを，キル・バッファを進ませずに，まえに削除したテキストと合わせて保存します (append-next-kill)。

以前に削除されたテキストのうち最後のいくつか（10個程度）はリング状に管理されているキル・バッファに保存されていますが，それらはコマンド M-y により復元できます。

M-y　C-y あるいは M-y の直後だけで使用できます。直前のコマンドで復元されたテキストに代わり，リングで1つ前のキル・バッファに保存されているテキストを復元します (yank-pop)。一連の M-y に先だって最初には C-y が入力されていなければならないことに注意してください。

なお，M-y では復元対象となるキル・バッファはリング内で前に遡りますが，このとき現在のキル・バッファを指すポインタは戻されません。したがって，つぎに削除されたテキストは，最初の C-y で復元対象となったキル・バッファの次のキル・バッファに保存されることになります。

3.6.3 長方形のリージョンのテキストの削除と複写・移動

現在の Jmacs では，長方形コマンドについてはまだ実現していません。

3.7 レジスタ

Jmacs には英字１文字の名前の A, B, C, D, E の５つの**レジスタ**と呼ばれるものがあります。レジスタは，マークのような**テキスト上の位置の記憶**と，キル・バッファのような**テキストの復元・挿入のための保存**の２通りの目的で用いられます。位置の記憶とテキストの保存に用いられるレジスタは，それぞれ５つずつあります。

3.7.1 レジスタによる位置の記憶

テキストを編集する場合，ある特定の位置の内容を参照しながら，他の部分への入力・修正をおこなうことがしばしばあります。しかし，その特定の位置の記憶にマークを用いると，ある位置を設定しても，それ以後実行されるマーク設定のコマンド C-@, あるいは副作用的にマークを設定する C-y のようなコマンドにより，その位置はマークリングの中では古いものとなります。そのため，その位置にカーソルを戻すためには，Cx C-x ではなく，C-u C-@ を何度か用いなくてはなりません。

それに対して，レジスタ保存された位置は，たとえコマンド C-@ あるいは，C-y などが実行された場合でも変更されません。したがって，上記のような何度もある特定の位置にカーソルを戻し参照する場合には，位置を記憶するレジスタが有用です。

レジスタに位置を記憶するには，記憶させたい位置にカーソルを移して，コマンド C-x / につづいてレジスタ名を入力します。また，記憶してある位置にカーソルを戻すには，コマンド C-xj につづいてレジスタ名を入力します。

C-x/*register*

レジスタにカーソル位置を記録します (point-to-register)。レジスタの指定は，/の直後に１文字のレジスタ名を入力します。

C-x j *register*

レジスタに記録されている位置にカーソルを移動します (register-to-point)。

3.7.2 レジスタによるテキストの保存

テキストのある特定の部分を何カ所かに複写したいとします。ただし，複写先ではテキストを削除することもあるとします。このようなとき，復元するテキストをキル・バッファに保存していても，テキストを削除するにしたがって，キル・バッファのリングのポインタが進められますので，目的のテキストを復元するためには，C-y だけではなく，M-y を何度か用いなければなりません。

それに対して，レジスタ保存されたテキストは，たとえコマンド C-k などの削除コマンドが実行された場合でも変更されません。したがって，上記のような何度もある特定のテキストを復元・挿入する場合には，テキストを保存するレジスタが有用です。

レジスタにテキストを保存するには，コマンド C-xx につづいてレジスタ名を入力します。ただし，これ以前に，保存したいテキストをリージョンに設定しておかなくてはなりません。また，保存してあるテキストをカーソル位置に復元・挿入するには，コマンド C-xg につづいてレジスタ名を入力します。

C-xx *register*

レジスタにリージョンを保存します (copy-register)。レジスタの指定は，x の直後に1文字のレジスタ名を入力します。なお，数引数が与えられた場合には，レジスタに保存したあとそのリージョンを消去（削除ではない！）します。

C-x g *register*

レジスタに保存されているテキストをカーソルの位置に復元・挿入します (insert-register)。このとき数引数がなければ，復元されたテキストの前にカーソルを，後ろにマークを設定します。数引数が与えられていれば，テキストの前にマークを設定し，後ろにカーソルを位置します。

もしレジスタの指定を間違えた場合には，復元されたテキストはリージョンに設定されていますので，コマンド C-w で削除したのち，正しいレジスタ名で再度復元・挿入すればよいでしょう。

3.7.3 レジスタに関する情報の表示

テキストの保存に用いられるレジスタに関する情報を表示するにはコマンド M-x view-register を用います。

M-x view-register RET

A, B, C, D, E のレジスタに保存されているテキストの大きさを表示します。

3.8 文字列の探索

文字列の探索とは，テキスト中でその文字列を見つけて，その位置までカーソルを移動することをいいます。Jmacs には，一括探索とインクリメンタル探索の 2 通りの探索があります。一括探索とは，探索したい文字列を入力したあと，ESC により探索を開始する探索で，他のエディタで一般に行われている探索です。一方，インクリメンタル探索では，探索したい文字列の最初の 1 文字を入力すると直ちに探索を開始して，その文字の位置までカーソルを移動します。さらに文字を入力するたびに，それまで入力した文字列を探索してカーソルを移動します。探索したい文字列が見つかって探索を終了する場合には ESC を入力します。どちらの探索であっても，成功し探索を終了したときには，探索を開始した位置がマークに記録されます。

一括探索に慣れた方には，文字を入力する度にカーソルが移動する——移動先によっては画面が変更される場合もあります——インクリメンタル探索には，最初は戸惑うかもしれませんが，探索したい文字列の入力を間違った場合にもすぐ気がつくなど，慣れると有用な探索方法です。

インクリメンタル探索と一括探索の双方とも，カーソル位置からバッファの最後の方向（順方向）への検索と，カーソル位置からバッファの先頭方向（逆方向）への検索のコマンドがあります。

順方向の探索は C-s，逆方向の探索は C-r の編集コマンドで指定しますが，つぎに入力される文字により，その探索方法がインクリメンタル探索か，一括型探索か，あるいは前回に探索した文字列と同じものを探索するか，が定まります。

(a) 通常の文字の場合には，その文字で始まるインクリメンタル探索の開始となります。

(b) ESC の場合には，以後入力される文字列の一括型探索の開始となります。文字列の終了は RET で指示します。

(c) C-s の場合には，これより以前の最後の C-s あるいは C-r により探索されたのと同じ文字列が順方向に探索されます。

テキスト中にコントロール文字を挿入するのと同様に，C-q を用いて，探索対象となる文字列中に，コントロール文字を含めることができます。探索文字列の入力中に C-q につづいて文字が入力された場合，C-q は探索文字列に追加されませんが，つぎに入力された文字は，たとえコントロール文字であっても，探索文字列に追加されます。なお，RET は C-q を用いなくても入力できますが，その場合はコントロール M とコントロール J の 2 バイトのコードに変換されます。探索文字列にコントロール M だけを含める方法については，3.12.4 で説明します。

3.8.1 インクリメンタル探索

C-s *string* ESC

順方向へのインクリメンタル探索を行います (isearch-forward)。

C-r *string* ESC

逆方向へのインクリメンタル探索を行います (isearch-backward)。

C-s を入力すると，エコー行に順方向へのインクリメンタル探索であることを示す

^S-I-Search:

が表示されます。C-r の場合には，

^R-I-Search:

が表示されます。

つづいて探索する文字列を入力すると，1文字ごとにその文字をエコー行に表示するとともに，カーソル位置から順方向(C-r の場合は逆方向)に探索します。探索が成功した場合には，探索された文字列の後にカーソルを移動します。探索が失敗すれば，警告音とともにエコー行には

Fail search: それまで入力した文字列

が表示されて探索失敗の状態となります。探索が成功でも失敗でも，つぎの入力を待ちますが，その入力によって，以下のような処理がなされます。

(a) 通常の文字あるいは RET, TAB の場合

・探索が成功している状態のとき

その文字をエコー行に表示するとともに，これまでに入力された探索文字列に追加して，順方向にその文字列を探索します。この探索が失敗することもあります。

・探索が失敗している状態のとき

その文字を探索文字列に追加しますが，カーソルは移動しません。探索は行いませんが，たとえ行っても失敗となることは明白ですので，探索失敗の状態のままです。

(b) C-s の場合

・探索が成功している状態のとき

これまでに入力された文字列のつぎの出現を探索します。ただし，その探索が失敗することもあります。最初に探索された文字列の位置が目的とする位置でない場合には，C-s を入力することにより，同じ文字列のつぎの出現を探索して，そこがまだ目的とする位置でなければ，再度 C-s を入力することに

より，そのつぎの位置にカーソルを移動することができます。

・探索が失敗している状態のとき

何もしません。(ただし，GNU Emacs などでは，バッファの最初から探索文字列を探索します。)

(c) C-r の場合

・探索が成功している状態のとき

カーソル位置から逆方向にこれまでに入力された文字列を探索します。ただし，カーソルは探索した文字列の前に位置します。もし，C-s を入力しすぎて目的の位置を通りすぎた場合には，C-r を入力することにより，逆方向に探索し目的の位置にカーソルを戻すことができます。そのときに C-r を入力しすぎた場合には C-s を用いて戻せばよいでしょう。

・探索が失敗している状態のとき

何もしません。(ただし，GNU Emacs などでは，バッファの最後から探索文字列を探索します。)

(d) C-q の場合

・探索が成功している状態のとき

この直後に入力する文字が，たとえコントロール文字であっても，その文字を探索文字に追加し，探索を行います。コントロール文字を含む文字列の探索を行う場合に用います。ESC も C-q の直後に入力すれば，探索の終了ではなく，探索文字列に含めることができます。

・探索が失敗している状態のとき

直後に入力される文字列を探索文字列につけ加えます。

(e) C-w の場合

・探索が成功している状態のとき

カーソル位置以降の単語（句切り文字までの文字列）を探索文字列に追加し，カーソルをその単語のあとに移動します。探索したい単語がカーソル位置以降にある場合には，いくつかの文字の入力が省力できます。

・探索が失敗している状態のとき

何もしません。

(f) C-y の場合

・探索が成功している状態のとき

カーソル位置からその行の行末までの文字列を探索文字列に追加し，カーソルを行末に移動します。

・探索が失敗している状態のとき

何もしません。

(g) DEL の場合

・探索が成功／失敗している状態のとき

DEL (キートップに DEL と記されているキー) を入力するたびに，これまで入力した文字が逆順に 1 つずつ取り消され，カーソルも順に以前の位置に戻ります。

取り消された文字が探索文字の場合にはその文字が探索文字から削除され，カーソルはその文字が入力される以前の位置に移動します。取り消された文字が C-s または C-r であれば，それらを入力する以前の位置にカーソルが移動します。なお，BS は探索文字の取り消しには使えません(以下の(j)参照)。

(h) C-g の場合

・探索が成功している状態のとき

探索を中止し，カーソルを探索を開始した位置に戻します。

・探索が失敗している状態のとき

これまで入力された文字列のうち探索が成功した部分だけを残し，それ以降の文字を探索文字列から消去します。探索が成功する部分までを残すので，探索成功の状態となります。途中から間違った探索文字列を入力し過ぎたときに用いてください。

(i) ESC の場合

探索を終了し，コマンド受付状態に戻します。このときカーソルはその位置にとどまります。目的の文字列の位置までカーソルを移動できて終了するときなどに使用します。

(j) 上記以外のコントロール文字の場合

探索は終了し，そのコントロール文字で指定される編集コマンドが実行されます。たとえば探索途中に C-a を入力した場合には，ESC を入力して探索を終了したのち，C-a を入力することと同じことになります。BS を入力すると，探索を中止したあと，逆方向へ1文字削除することになりますので注意してください。

ここで，つぎのように入力して，実際に探索を試みてください。

◇(1) 文字列 abccef SPC をテキストのいろいろな位置に3個程度入力したのち，コマンド M-< でカーソルをバッファの最初の最初に移動してください。ただし，SPC は空白の入力を示します。

◇(2) つぎに C-s を入力し，さらに1文字ずつカーソルの移動を確認しながら abcd と入力してください。カーソルは最初の abcd の直後に移動します。

◇(3) つづいて C-s を入力してください。カーソルは2番目の abcd の直後に移動します。

◇(4) つづいて e を入力してください。カーソルは1文字分進みます。

◇(5) ここで DEL を入力してください。カーソルは1文字分戻ります。エコー行の探索文字列から最後に入力した e が取り消されていることを確認してください。

◇(6) さらに DEL を入力してください。(3)で入力した C-s が取り消されますので，カーソルは最初の abcd の直後に戻ります。

◇(7) C-w を入力してください。現在のカーソル位置から単語の最後の文字までが探索文字列につけ加えられるとともに，カーソルも単語の直後まで移動します。

◇(8) C-s を入力して，2番目の abcdef の直後にカーソルが移動することを確認してください。

◇(9) さらに xyz と入力してください。x を入力した時点で探索失敗となりますが，気にしないで入力してください。エコー行に探索失敗のメッセージにつづ

いて abcdefxyz が表示されていることを確認してください。

◇(10) ここで C-g を入力してください。カーソルは移動しませんが，エコー行の探索文字列が，探索が成功している文字列，すなわち abcdef まで取り消されます。また，エコー行も探索が成功しているときと同じ表示になっていることに注意してください。

◇(11) ESC を入力して，探索を終了してください。

3.8.2 一括探索

文字列探索のコマンド C-s の直後に ESC を入力した場合には，インクリメンタル探索ではなく，一括探索となります。ESC につづいて文字列を入力しても，エコー行に表示されるだけで，1文字毎の探索は行われません。探索を開始するには，必要な文字列を入力したあと，再度 ESC を入力します。なお，探索文字列中に ESC を含めるためには，ESC の直前で C-q を入力してください。(GNU Emacs などでは，探索開始は RET で行うようになっています。インクリメンタル探索では，探索文字列の入力中に RET を入力しても，それは文字列に追加されますが，一括探索では探索の開始となるので，C-q を用いなければならないことになります。ESC を探索文字列に含める場合には逆になります。このため Jmacs では双方を揃える意味で，一括探索の終了を ESC としました。)

以下の◇で指示するように入力し，文字列 abc の一括探索を試みましょう。なお，ここでは，先のインクリメンタル探索を行ったのと同じ内容のテキストがバッファにあるものとします。

◇M-< を入力し，バッファの先頭にカーソルを移動してください。

◇C-s ESC を入力し，そのあと文字列 abc を入力してください。まだ探索は行われず，カーソルは移動していないことを確認してください。

◇さらに続いて ESC を入力してください。探索が開始され，カーソルが最初の abc の次に移動することを確認してください。

3.8.3 前回の探索文字列と同じ文字列の探索

テキストの編集中には，ある文字列を探索し，そこで削除や挿入等の編集操作をおこなったのち，再度同じ文字列の探索・編集操作を繰り返すような場合がしばしばあります。Jmacsでは，前回の探索と同じ文字列を探索するときには，再度その文字列を入力する必要はありません。C-s C-sと入力することにより，前回のC-sあるいはC-rで探索したのと同じ文字列を探索することができます。なお，C-s C-sで文字列を探索したあとは，つぎの入力を待ちますが，その入力に対する処理は，インクリメンタル探索の場合と同様です。

C-s C-s　前回の探索と同じ文字列をカーソル位置から順方向に探索します。

C-r C-r　前回の探索と同じ文字列をカーソル位置から逆方向に探索します。

C-s C-sは，このコマンド入力以前の最後に行われたインクリメンタル探索あるいは一括探索で探索したのと同じ文字列を順方向に探索します。逆方向に同様な検索をおこなう場合には，C-s C-sの代わりにC-r C-rを用います。同じ文字列を探索する場合に，文字列の入力が省略できますので非常に有用なコマンドです。

3.9 文字列の置換

テキスト中のある文字列を別の文字列で置き換えるには，置き換えられる文字列が見つかるたびに，置き換えるか否かを問い合わせる方法と，問い合わせを行わず，見つかれば無条件に置き換える方法の2通りがあります。

3.9.1 問い合わせながらの置換

M-% *oldstring* RET *newstring* RET

順方向への問い合わせ置換を行います(query-replace)。

M-%を入力すると，エコー行には，

^Q-REplace:

と表示されます。つづいて現在テキスト中にあり，置き換えられる文字列（*oldstring* とする）と RET を入力します。文字列 *oldstring* は入力にしたがってエコー行に表示され，RET を入力したときには，エコー行には，

^Q-Replace *oldstring* $ with $:

と表示されています。つぎに新しく置き換わる文字列（*newstring* とする）を入力し，最後に RET を入力します。

最後の RET が入力されると，現在のカーソル位置から順方向(バッファの最後の方向)に，*oldstring* を探索します。探索が成功した場合には，見つかった位置までカーソルを移動。画面表示し，入力を待ちます。ここでの入力はそれぞれ次のような意味となります。

(a)SPC　この文字列を置換したのち，次の探索に進みます。

(b)BS　この文字列は置換せずに次の探索に進みます。

(c)ESC　これ以上置換せずに終了します。

(d)・　現在探索されている文字列を置換したのち終了します。

(e)!　現在探索されている文字列を含めて，カーソル以降の該当する文字列すべてを問い合わせなしに置換します。

(f)^　1つ前に探索した位置に戻ります。その位置では，既に SPC あるいは DEL を入力し，置き換えの実行／非実行を指定しているはずですが，この入力により戻っても，既に行っている指定は取り消せません。この指定は，直前の指定の確認か，誤って置換した直後に，その位置に戻るために使用します。もし正しければ，SPC を入力して，つぎの位置に進めることができます。誤って置換していれば，ESC で，置換を中止して文字列を元に戻せばよいでしょう。なお，この^は，連続して用いることはできません。

カーソル位置以降に *oldstring* が見つからなければ，エコー行には

Done

と表示され，置き換えは終了します。このとき，カーソルは最後に置き換えられた文字列の直後の位置に留まります。また，置換を開始したときのカーソル位置は，マークに設定されます。

なお，文字列 *oldstring* あるいは *newstring* の入力中の打ち間違いは，BS あるいは DEL キーでとり消すことができます。また，*oldstring* あるいは *newstring* の入力中に，この置後コマンドを取りやめる場合には C-g を入力します。するとエコー行に

Quit

と表示され，コマンド受付状態に戻ります。

コマンド M-% では，RET の入力は，文字列 *oldstring* あるいは *newstring* の最後を示すものとして使用されます。そのため、RET (改行) を含む文字列を *oldstring* あるいは *newstring* として指定するには，文字列中の RET の直前で C-q を入力することが必要です。なお，文字列にコントロール M だけを含める場合については，3.12.4 で説明します。

3.9.2 無条件置換

コマンド M-% のように，置き換えられる文字列 (*oldstring* とする) が見つかるたびに，どうするかを問い合わせるのではなく，カーソル以降の全ての *oldstring* を *newstring* で置き換えるには，コマンド replace-string を用います。

強制置換の実行と同じことは，M-% と ! を用いても，同程度の手間でできますので，この無条件置換はあまり使用しないようです。そのためか，GNU Emacs などでは，

このコマンドに対しては，標準ではコントロール文字あるいはメタ文字が対応づけられておらず，つぎのようにM-xとコマンド名により指定するようになっています。Jmacsでは，旧バージョンとの関係もあり，コマンドC-^rを対応づけています。

M-x replace-string RET *oldstring* RET *newstring* RET

順方向への無条件置換を行います(replace-string)。

C-^r *oldstring* RET *newstring* RET

順方向への無条件置換を行います(replace-string)。

M-xにつづいてreplace-stringと入力するか，あるいはC-^rと入力すると，エコー行には，

^R-Replace:

と表示されます。つづいて，M-%による置換と同様に，テキスト中の置き換えられる文字列とRET，置き換わる文字列とRETを入力します。最後のRETを入力すると，現在のカーソル位置からバッファの最後までを範囲として文字列の置き換えが行われます。このコマンドの実行が終了したときには，エコー行に

Done

と表示され，Jmacsのコマンド受付状態に戻ります。このとき，カーソルは最後に置き換えられた文字列の直後の位置に留まります。また，置換を開始したときのカーソル位置は，マークに設定されます。M-%の場合と同様に，文字列の入力中に，このコマンドをやめる場合には，C-gを入力します。

また，文字列中にRETを含ませる場合には，問い合わせ置換のM-%の場合と同様に，それらのRETの直前でC-qを入力することが必要です。

3.10 ファイル関連のコマンド

3.10.1 バッファのファイルへの保存

現在編集中のバッファ上のテキストをファイルに保存するには，C-x C-s を用います。

C-x C-s 現在編集中のバッファ上のテキストをディスク上のファイルに保存します (save-buffer)。

もし，バッファが修正されていない場合には，エコー行に

No changes need to be written

のメッセージが表示され，保存は行われません。保存が行われた場合には，バッファのテキストと，バッファに対応するファイルの内容は同じですので，モード行の＊＊は，——に変更されます。

このコマンドは，編集の途中で万一の事故に備えて，それまでの編集結果をファイルに保存するために，ときどき使用すればよいでしょう。また，編集を終わるときに，このコマンドを用いてテキストを保存したあと，C-x C-c で Jmacs を終了するようにしてもよいでしょう。

3.10.2 編集対象のファイルの切り替え

現在編集しているファイルとは別のファイルの編集を行うには，コマンド C-x C-v を用います。

C-x C-v *file-name* RET

編集対象のファイルを切り換えます (find_alternative_file)。

fild-name は新しく編集しようとするファイルの名前です。編集中のファイルに対応したバッファを閉じたのち，指定されたファイルに対応するバッファを新しく開き，

ファイルからテキストを読み込みます。なお，編集中のバッファが修正されていれば，内容をファイルに保存するか否かを問い合わせてきますので，必要があれば保存を指定します。

3.10.3 バッファの別のファイルへの書き出しと編集対象ファイルの切り替え

バッファをある程度編集した後に，元のファイルはそのままで保存する必要があったことに気がついたとします。一方，今までの編集された結果も必要で，さらに編集を続けたいとします。このような場合に，コマンド C-x C-w を用いて，既に変更されたバッファを元のファイルではなく，指定された別のファイルに保存するとともに，編集対象ファイルも指定されたファイルに切り替えて，編集をつづけることができます。なお，編集対象となるファイルが切り換えられるので，当然バッファ名も変更されます。

C-x C-w *file-name* RET

現在のバッファのテキストを対応しているファイルには保存せずに，*file-name* で指定されるファイルに書き出し，編集対象のファイルを，そのファイルに切り換えます。(write-file)。

3.10.4 リージョンのファイルへの書き出し

C-^ C-w *file-name* RET

リージョン（マークとカーソルの間）を指定されたファイルに書き出します。

コマンド C-^ C-w を入力すると，ファイル名を尋ねてきますので，保存したいファイル名と RET を入力してください。指定されたファイルが存在しない場合には，新しく作成されます。既に存在していた場合には，その内容は上書きされますので注意してください。なお，このコマンドは Jmacs 独自のものです。

3.10.5 カーソル位置への別のファイルの読み込み

C-xi *file-name* RET

カーソル位置へ指定されたファイルを読み込みます。

C-xi を入力すると，ファイル名を尋ねてきます。読み込みたいファイル名と RET を入力してください。ポイントの位置（カーソル位置の文字の前）にそのファイルを読み込みます。このコマンドの実行後，カーソルは読み込まれたテキストの最初に位置します。

3.11. その他のコマンド

3.11.1 一時的に MS-DOS のコマンドレベルに戻るコマンド

C-xC-z 現在の Jmacs の状態を保存したままで，一時的に MS-DOS のコマンドレベルに戻ります。

Jmacs でファイルを編集中に，たとえば，コマンド C-xi で読み込むファイルの名前およびその内容を確認したいというようなことがしばしば生じます。このようなときはコマンド C-x C-z により，MS-DOS のコマンドレベルに戻ることができますので，MS-DOS のコマンド DIR あるいは TYPE 等を実行して確認すればよいでしょう。MS-DOS のコマンド EXIT により，Jmacsを中断した状態に戻し，編集をつづけることができます。このコマンドで MS-DOS に戻ったとき，原理的にはコンパイラなどを起動することもできるはずですが，実際にはメモリの余裕がない場合もありますので注意してください。

3.11.2 バッファを変更していないとするコマンド

M-″ バッファを変更されていないものとします。

コマンド M-″ は，たとえ変更されたバッファであっても，変更していないものとします。したがって，バッファを変更したとき表示されるモード行の＊＊は，変更していないことを意味する--に換えられます。ただし，このコマンド以後にバッファを変更した場合には，当然バッファは変更されたものとなります。

誤って文字を入力したあとその文字を消しても結局は変更しないことになっていますが，このように，わざわざファイルに保存する必要のない場合，あるいは，バッファを変更したけれどもその変更総果を保存したくない場合などに，このコマンドを用いておくと，不用意に保存することが避けられます。

3.11.3 バッファの使用状態の表示

C-x = バッファの使用状態を文字数に関して表示します。

C-xc バッファの使用状態を文字数に関して表示します。上記の C-x = と，表示方法が異なります。

C-xl バッファの使用状態を行数に関して表示します。

コマンド C-x = は，現在編集対象としているバッファの使用状態を文字数(バイト数）に関して，エコー行に次のように表示します。ただし，ページについては，バッファ内のコントロールL（16進数でC）をページの句切りとみなします。なお，バイト数ですので，アルファベットなどの半角数字は1バイトですが，全角の漢字などは2バイトとして計算します。

Chars: カーソルより前の文字数（現ページ内のカーソルより前の文字数）
＋カーソルより後の文字数（現ページ内のカーソルより後の文字数）＝
テキストの文字数（現ページ内の文字数）

Rest: まだ利用できる文字領域の大きさ

Kill: キル・バッファに保存されている全文字数
（最後に保存された文字数ｉｅC-yで復元されるテキストの文字数）

コマンドC-xcは，現在編集対象としているバッファの使用状況を，文字数に関して，C-x =よりも簡単に，エコー行に次のように表示します。

Chars: テキストの文字数，まだ利用できる文字領域の大きさ，最後に保存された文字数

コマンドC-xlは，現在編集対象としているバッファの使用状況を，行数に関して，エコー行に（1行で）次のように表示します。

Lines: カーソルより前の行数（現ページ内のカーソルより前の行数）＋
カーソルより後の行数（現ページ内のカーソルより後の行数）＝
テキストの行数（現ページ内の行数）

3.11.4 編集モードの設定

コマンドmode-setは，文字列の探索のコマンドの実行の場合に，大文字と小文字を区別するか否か，あるいは右括弧を入力した場合に，対応する左括弧のところまでカーソルを一時的に戻すか否かなど，いくつかのコマンドの実行に関するモードを対話的に設定します。

M-x mode-set RET
コマンドの実行に関するモードを設定します。

M-x mode-set RETと入力すると，エコー行に，御以下の順で，モードの設定を尋ねてきますので，スペース(SPCAE)あるいはバックスペース(BS)を入力することに

より，適切なモードに設定してください。

Case fold search? (yes: SPACE/no: BS):
Parenthesis match? (yes: SPACE/no: BS):
Macro-regist-with-non-exec? (yes: SPACE/no: BS):
RET-key after C-q becomes C-m & C-j? (yes: SPACE/no: BS):

(a) Case fold search? (yes: SPACE/no: BS)
コマンド C-s あるいは C-r による文字列探索のとき，あるいは，文字列の置き換えのコマンド M-探索のときに，大文字と小文字を区別するか否かの設定です。区別しないときはスペース (SPACE) を，区別するときはバックスペース (BS) を入力してください。なお，Jmacs が起動されたときの初期設定は，区別しないモードです。

(b) Parenthesis match? (yes: SPACE/no: BS)
半角の右括弧を入力したとき，対応する左括弧の位置までカーソルを戻して表示するか否かを設定します。対応関係を表示するときはスペース (SPACE) を，表示しないときはバックスペース (BS) を入力してください。なお，Jmacs が起動されたときの初期設定は，対応関係を表示するモードです。対応を表示する場合，その対応位置でカーソルがしばらく留まりますが，カーソルが元の位置に戻るまで次の入力を待つ必要はありません。対応位置が確認できれば，次の入力をしてもかまいません。

(c) Macro-regist-with-non-exec? (yes: SPACE/no: BS)
次の第 4.13 で説明するキーボードマクロを定義するとき，キーボードからコマンドが入力されたとき，定義に追加すると同時に実行もするか否かを設定します。実行せずに定義だけをする場合には，スペース (SPACE) を，定義と同時に実行するときはバックスペース (BS) を入力してください。なお，Jmacs が起動されたときの初期設定は，同時実行をしないモードです。

(d) RET-key after C-q becomes C-m & C-j? (yes: SPACE/no: BS)

通常，MS-DOS のテキスト中では，改行はコントロールMとコントロールJの2文字が使われていますので，Jmacs でも，起動されたときの初期計定は，RET あるいは C-m のキー入力を，その2文字に変換するモードになっています。しかし，まれに，バッファへの挿入や，探索文字列中や置換コマンドでの文字列中に，コントロールMだけの入力が必要な場合があります。

この問い合わせに応えることにより，C-q につづいて入力される RET あるいは C-m を，初期設定と同じく2文字に変換するか，変換せずにコントロールMだけとするかのモードを設定することができます。SPACE を入力すると変換するモード，BS を入力すると変換しないモードとになります。なお，いったん設定されると，つぎに設定されるまではモードは変わりません。

3.11.5 タブサイズの設定

M-x tab-set RET *tab-size*

タブサイズの設定を行います。

Jamcs では，ファイルのクエステョンにより，タブのサイズを自動的に設定します。たとえば，エクステンションが c であれば 4，for，f または lsp であれば 6，これ以外の場合には 8 に設定されます。しかし，コマンド M-x tab-size RET に続いて，タブサイズを入力することにより，ユーザがタブサイズを設定することもできます。なお，設定したタブサイズの情報は，ファイルには保存されませんので，別の機会にそのファイルを編集する場合には，再度タブサイズを設定する必要があります。

3.11.6 目的行へのカーソルの移動

C-^ g *line-number*

line-number で指定された行の先頭にカーソルを移動します。

コンパイラでのエラーメッセージでは行番号が表示されます。このような場合に，コマンドC-^g につづいて行番号を入力すると，その番号の行にカーソルを移動させることができます。

3.11.7 行の折り返し位置の変更

C-u *column-size* C-x f

行を表示するときのカラム位置を，*column-size* で指定します。

Jmacs を起動したときは，行の長さが半角文字で 78 字を越えると，79 文字目に！が表示され，残りは画面上の次の行に表示されます。文章の編集のときには，あまり困りませんが，横方向が 78 文字以上の表などを扱うときには，折り返されると上下の関係が分かりにくく非常に不便です。このようなとき，C-u につづいて，行の長さ，C-xf を入力することにより，折り返しの始まる長さを変更することができます。行の長さとしては，折り返しの始まる位置＋1 を与えてください。たとえば 129 文字目で折り返すためには，

C-u 130 C-xf

と入力します。

CRT 画面には横方向に最大 80 文字しか表示できませんので，このコマンドにより 80 字以上に変更した場合にも，画面に表示される文字数は変わりませんが，この場合，その設定された値で折り返し表示されたテキストを横幅 80 文字分の窓で覗いていると考えてください。

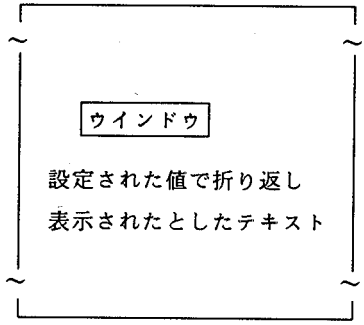


図11 バッファとウインドウ

このコマンドで，折返しの位置を80カラム以上としているときに，カーソルを現在表示されているウインドウの左右の端を越えるように移動すると，カーソル位置が表示されるようにウインドウも左右に移動します。また，既に述べたコマンドC-lは，縦方向だけでなく，横方向にも，可能な限りカーソル位置が画面の中央になるように再表示されます。

なお，この行の折り返し位置の設定は80カラム以下でもかまいませんが，設定できる値には，50以上250以下という制限がありますので注意してください。

4. キーボード・マクロ

Jmacsには，キーボードから入力した一連の連続したコマンドをキーボード・マクロとして定義し，簡単な操作で，その定義した一連のコマンドを実行する機能があります。一連のコマンドを何度も実行するような場合には，非常に有用です。キーボード・マクロの定義は，コマンドC-x (で始まり，C-x) で終了します。この間の一連のコマンドが定義内容となります。なお，Jmacs独自のコマンドですが，リージョン内のテキストをキーボード・マクロとして定義する機能と，既に定義されている内容を，バッファのカーソル位置に取り込む機能もあります。

C-x(キーボード・マクロの定義を開始します。

C-x) キーボード・マクロの定義を終了します。

M-x macro-def バッファの領域をキーボード・マクロとして登録します。

M-x macro-get 定義されているキーボード・マクロをカーソル位置に取り込みます。

C-xe 定義されているキーボード・マクロを，現在のカーソル位置から実行します。

4.1 C-x(と C-x) による定義

C-x(につづいて入力される一連のコマンドあるいは通常の文字がキーボード・マクロとして定義されます。定義の終了は C-x) で指定します。当然のことですが，新しい内容で定義されれば以前の定義はなくなります。

定義には，コマンドを入力する度にそのコマンドを実行しながら定義するモードと，定義中にはコマンドを実行しないモードとがあります。この2つのモードを切り換える方法については 3.11.4 を参照してください。

(a) 定義的に実行されるモードの場合

C-x(が入力されると，エコー行に

Defining KBD-macro

と表示されます。以後，コマンドあるいは通常の文字を入力するたびに，コマンドであれば実行されますし，通常の文字であればカーソル位置への入力となります。ただし，コマンドの入力が誤っている——存在しないコマンドあるいは，その引数等が間違っている——場合や，存在するコマンドであっても，実行がエラーとなった場合には，定義は中止されます。途中でエラーとなることなく正常に C-x) で終了した場合には，エコー行に

KBD-macro defined

と表示され，新しくキーボード・マクロが定義されたことを示して終了します。

(b) 定義時に実行されないモードの場合

C-x(が入力されると，エコー行に ^X(が表示されます。以後，コマンドあるいは通常の文字を入力するたびに，エコー行にその入力が順に表示されます。ただし，コマンドの入力が誤っている場合には，定義は中止されます。なお，文字列探索コマンド C-s を入力すると，つぎに ESC を入力しなくても，このモードの場合には，一括探索として処理されます。したがって，*string* を探索する場合には，つぎのように入力してください。

C-s string ESC

キーボード・マクロの定義を終了するには，このモードの場合も C-x) を入力します。エコー行に現在までの表示に代わり，

KBD-macro defined

と表示され，新しくキーボード・マクロが定義されたことを示して終了します。

どちらのモードでも，キーボード・マクロの定義中に，C-g を入力すれば，定義を中止することができます。実行しないモードでの定義中に C-g により中止された場合は，単に定義を中止するだけですが，実行するモードに C-g により中止された場合には，それまで入力したコマンドによって実行された状態は，そのままで，定義開始の時の状態に戻るわけではありません。Jmacs の現在のバージョンでは，「既に実行したコマンドの取り消し」を行うコマンドが備わっていませんので，実行しないモードで定義する方が安全です。

定義中に実行しないモードの場合には，定義中に DEL キーを入力することにより，既に入力したコマンドを最後から順に取り消し，定義の修正を行うことができます。

DEL でなはなく BS を入力した場合には，逆方向への一文字消去のコマンドキーがキーボード・マクロの定義に含まれることになります。

ここで，キーボード・マクロの定義例と使用例を示します。次のテキストは MS-DOS のコマンド DIR の出力をリダイレクションによりファイルに取り込み，上下の何行かを消去したものです。ただし，各行の先頭は，最初の J の文字の位置とします。

```
JMXABC EXT 191 88-09-09 10:05
JMDTAB EXT 4998 89-06-21 0:30
JMDSA EXT 353 88-09-27 22:04
JMCT EXT 1283 89-04-03 18:25
JMMNABC EXT 457 89-01-26 21: 52
JMMTAB EXT 413 88-09-10 18:58
JMCXA EXT 726 88-09-11 0:27
JMTB EXT 931 88-10-03 23: 52
JMCCABC EXT 113 88-09-04 20: 06
```

このファイルを Jmacs で編集し，

```
PRINT JMMXABC. EXT
PRINT JMDTAB. EXT
PRINT JMDSA. EXT
PRINT JMCT. EXT
PRINT JMMNABC. EXT
PRINT JMMTAB. EXT
PRINT JMCXA. EXT
PRINT JMTA. EXT
PRINT JMCCABC. EXT
```

のように変更することを考えます。キーボード・マクロの定義内容としては，いくつか考えられますが，ここでは，キーボード・マクロの実行の前にカーソルをファイルの先頭に位置づけているものとして，次のように定義します。なお，空白の入力はSPCで表わしています。

C-x(PRINT SPC M-f C-@ M-f C-k M-b C-w C-n C-a C-x)

最初にPRINTと空白を入力し，次に1単語分カーソルを進めてピリオド(.)を入力し，そこをマークし，さらに1単語分カーソルを進めます。ここでカーソルはEXTの直後に位置しています。そこでC-kにより，カーソル以降の文字を削除します。そしてM-bによりEXTの直前にカーソルを戻し，C-wで先ほどマークした位置（ピリオドの直後）とカーソルの間の空白を削除した後，C-nとC-aによりカーソルを次の行の先頭に移します。

このようにマクロを定義したあと，最初の行の先頭にカーソルを移した後，繰り返しコマンドC-uを用いて，

C-u 15 C-x e

と，15回のキーボード・マクロの実行を指定します。繰り返し回数15は十分であろうと思われる適当な値です。少なければ，また実行すればよいでしょうし，多すぎてもファイルの最後までカーソルが移動した直後のC-nでエラーが生じ，マクロの繰り返し実行は中止され，Jmacsのコマンドレベルに戻るので，あまり神経質に正確な回数を指定する必要はないでしょう。

4.2 領域をマクロとして定義する方法

コマンドM-x macro-registにより，マークされた位置とカーソル位置の間のリージョン内のテキストをキーボード・マクロとして定義することができます。ただし，バッファ中のテキスト中にC-fのようなコマンドを入力するには，コントロールQ(C

-q) を用いる必要がありますので，注意してください。

長いマクロを定義した後で，しばしば間違いに気がつくことがあります。そのような場合，再度マクロを入力するよりも，(3)で説明するコマンド M-x macro-get により，カーソル位置に取り出して，修正した後，このコマンドで定義する法が簡単でしょう。

また，一連のコマンドをいくつかのファイルに定義しておき，必要なときにそのファイルを，コマンド C-xi などで読み込み，適切な部分をリージョンとして，このコマンドでキーボード・マクロとして定義することもできます。

4.3 定義されているマクロをカーソル位置に取り込む方法

コマンド M-x macro-get により，現在のキーボード・マクロの定義内容をカーソル位置に取り込むことができます。(2)で述べたように，このコマンドで取り込んだ定義内容を修正した後，コマンド M-x macro-regist を用いて定義することにより長いキーボード・マクロの定義の修正などが簡単に行える場合があります。

5. マルチバッファとマルチウインドウ

プログラムあるいは原稿などをいくつかのファイルに分けて作成するときなどは，それらのファイルは互いに関連しており，あるファイルを参照しながら別のファイルを編集する必要がしばしば生じます。このようなとき，エディタで 2 つのファイルが扱え，しかも画面を 2 つに分割して，一方には参照されるファイルを表示し，他方は編集するファイルを表示することができれば，編集の作業は非常に楽になります。

Jmacs では，複数のバッファを開き，編集対象のバッファを切り替えながら，それぞれのバッファを編集することができます。また，CRT 画面を分割して同時にいくつかのバッファの内容を表示することができます。このような機能は，それぞれ，**マルチバッファ**，**マルチウインドウ**と呼ばれます。ただし，マルチバッファ，マルチウインドウといっても，各時点で編集の対象となるのは，いずれかのウインドウに表示されている 1 つのバッファだけであることに注意してください。現在編集対象となっているバッファを**カレントバッファ**，そのバッファを表示し，カーソルが存在している

ウインドウを**カレントウインドウ**と呼びます。

ところで，Jmacsではバッファとウインドウは必ずしも1対1に対応する必要はなく，1つのバッファの同じあるいは別の部分を複数のウインドウに表示することもできます。また，バッファを必ずしもウインドウに対応させる必要もありませんので，表示されていないバッファも存在します。

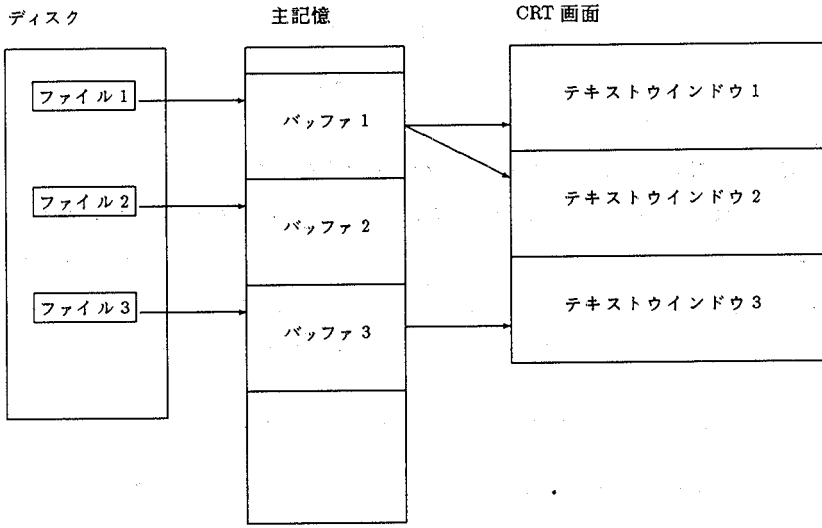


図12 ファイル，バッファ，テキストウインドウの関係

5.1 マルチバッファに関するコマンド

C-x C-f *file-name*

カレントウインドウに指定されたファイルを探して表示します。

C-x b[*buffer-name*]

カレントウインドウに指定されたバッファを表示します。

C-x C-b

開かれているバッファに関する情報を表示します。

C-x k *buffer-name*

指定されたバッファを閉じます。

C-x a[*buffer-name*]

リージョンを指定されたバッファのカーソル位置に挿入します。

C-x C-s

カレントバッファを保存します。

C-x s

変更された全てのバッファを対話的に保存します。

コマンド C-x C-f を入力するとファイル名を尋ねてきますのでファイル名と RET を入力してください。指定されたファイルが編集のために既にバッファに読み込まれていれば，そのバッファが編集対象となりウインドウに表示されます。まだ編集されていなければ，新たにバッファを開いたのち，ファイルを読み込み，ウインドウにはそのバッファを表示します。コマンド入力前に表示されていたバッファは，このウィンドウからは切り離されますが，依然として開かれています。この点でコマンド C-x C-v と異なります。

このコマンドにより複数のバッファが開かれていますが，いずれのバッファでも今まで説明したコマンドすべて利用できます。ただし，マークリングは各バッファごとに管理されていますが，キル・バッファとレジスタは全バッファに共通して管理されています。したがって，どのバッファで削除されたテキストでも共通のキル・バッファに蓄積され，その蓄積されたテキストはそのバッファで復元できることは当然ですが，他のバッファでも復元することができます。

カレントウインドウに別のバッファを表示させるには，コマンド C-x b を用います。このコマンドを入力すると，エコー行にバッファ名を尋ねてきます。このとき，[ ]中にバッファ名が表示されますが，バッファ名を入力せずに RET だけを入力すると，このバッファがウインドウに表示されます。このように単に RET を入力したとき，対象とにるバッファをデフォルトバッファと呼びます。

コマンド C-x C-b は，現在開かれているバッファに関して，対応するファイル名，バッファ中のテキストの大きさ，バッファのモードの情報を表示します。コマンドの

入力時に開かれているウインドウが1つの場合には,CRT画面が2つのウインドウに分割され，上側のウインドウには，今までのバッファが表示されます。ただし，カーソルの位置が表示されるように，上側のウインドウで表示部分が変更されることもあります。下側のウインドウには，バッファに関する情報が表示されます。開かれているウインドウを1つに戻す方法については，次の5.2で説明します。もし，既に2つ以上のウインドウが開かれていた場合には，カレントウインドウでないどれかのウインドウに表示されます。以前に表示されていたバッファはそのウィンドウからは切り離されますが，バッファは開かれたままです。

情報が表示されるバッファは，#BUF__LSTというJmacsが自動的に開くバッファです。このバッファは他のバッファと同じようには修正はできますが，バッファ名と同じ名前のファイルに保存することはできません。保存のためにはコマンドC-x C-wなどを利用してください。

不必要になったバッファを閉じるためには，C-x kを用います。閉じるバッファ名を尋ねてきますが，単にRETを入力すると，カレントバッファが閉じられます。このときカレントウインドウには，Jmacsが自動的に作成する#DEF__BUFが対応づけられて表示されます。カレントウインドウに他のバッファを対応させるためには，コマンドC-x bを用いてください。なお，閉じられるバッファが修正されていれば，保存するか否かを尋ねてきます。

コマンドC-x aは，カレントバッファのリージョンを，指定されたバッファのカーソル位置に挿入します。なお，キル・バッファを通して，あるバッファの一部を別のバッファに挿入することも可能ですが，このコマンドC-xaでは，その方法とはキル・バッファにテキストを蓄積しないことに違いがあります。

5.2 マルチウインドウに関するコマンド

C-x 0　カレントウインドウを閉じます。

C-x 1　カレントウインドウ以外をすべて閉じます。

C-x 2　カレントウインドウを2つに分割します。

C-x o　別のウインドウをカレントウインドウにします。

C-x^　カレントウインドウの縦方向の大きさを変更します。

C-M-v　別のウインドウを順方向にスクロールします。

C-M-z　別のウインドウを逆方向にスクロールします。

コマンド C-x 2 は，カレントウインドウを上下 2 つに分割し，上側のウインドウを新しくカレントウインドウとします。それぞれのウインドウをさらに 2 つに分割することもできます。ただし，開けるウインドウは全部で 6 個に限られています。CRT 画面に表示できる行数からみてこの程度のウインドウ数で充分でしょう。

分割された 2 つのウインドウは分割前と同じバッファに対応づけられており，同じ部分を表示しています。この状態で，一方のウインドウをスクロールして，同じバッファの別の部分を表示することもできます。このように同じバッファを 2 つのウインドウで表示しているとき，一方を変更した場合，その部分を表示しているウインドウはすべてその結果を表示します。逆に，同じバッファに対応していても，変更されたバッファを表示していないウインドウの表示は変わりません。

このように，分割した画面に別のファイルに対応しているバッファを表示させるには，コマンド C-x C-f を用います。コマンド C-x C-v では，カレントバッファを閉じて，指定されたファイルに対応したバッファを開きますので，前のバッファを表示している全てのウインドウは新しく開かれたバッファを表示することになります。これに対して，コマンド C-x C-f は指定されたファイルに対応するバッファをカレントウインドウに表示しますが，前のバッファを閉じるわけではありませんので，カレントウインドウ以外の前のバッファの表示は変更されません。したがって，コマント C-x 2 で画面を 2 つに分割し，一方にコマンド C-x C-f で別のファイルを選択し，その対応するバッファを表示することによって，2 つのバッファをそれぞれ別のウインドウに表示することができます。

ウインドウが 2 つ以上開かれているとき，別のウインドウをカレントウインドウにして編集するためには，コマンド C-x o を用います。o は，アルファベットの o(オー)です。このコマンドにより，1 つ下側のウインドウがカレントウインドウとなります。

ただし，最も下側のウインドウの1つ下は，最も上側のウインドウと考えてください（以後同様な言葉遣いをします)。このコマンドを何回か用いて目的ウインドウをカレントウインドウとしてください。

他のウインドウをすべて閉じ，カレントウインドウだけをCRT画面に表示する場合には，コマンドC-x 1を用います。1は数字の（イチ）です。

カレントウインドウを閉じるには，コマンドC-x 0を用います。0は数字の0（ゼロ）です。このときには，その下側のウインドウが閉じられるウインドウの領域を取り込んで，カレントウインドウとなります。

カレントウインドウの行数が少ない場合には，コマンドC-x ^ で1行分大きくすることができます。ただし，下側あるいは上側のウインドウがその行数分縮小されます。C-uにより数引数を与えることができますが，負の数引数を与えた場合には，カレントウインドウはその分縮小します。ただし，縮小するウインドウがあまりにも小さいときは，縮小せずに警告音とともにメッセージを表示します。

コマンドC-M-vとC-M-zによって，カレントウインドウの下側のウインドウを，それぞれ順方向，あるいは逆方向にスクロールすることができます。