# コンピュータリテラシ＃9 – テキストファイル/Emacsの詳細

久野 靖 (電気通信大学)

2017.5.19

## 1　今回の目標

今回の目標は次の通りです。

- コンピュータ内部でのテキストの表現について学ぶ — 実験データなどもコンピュータ上ではテキストファイルなのでその原理を知っておくと有益です。
- 日本語の複数の文字コードとその問題について学ぶ — 日本語の文字化けトラブルは多いので文字コードの知識は必須です。
- テキストエディタ Emacs の多様な機能について学ぶ — 実際にプログラムやレポートを書くのに使う道具なので有効に使えるようになっておくことは価値があります。

## 2　テキストファイル

### 2.1　テキストファイルと文字コード exam

**テキストファイル** (text file) とは、人間に読める文字を内容とするファイルのことでした。しかしコンピュータ上の情報はすべてビット列で表現されているわけですから、「文字を内容とする」というのはどういうことか、もう少し考えて見ましょう。

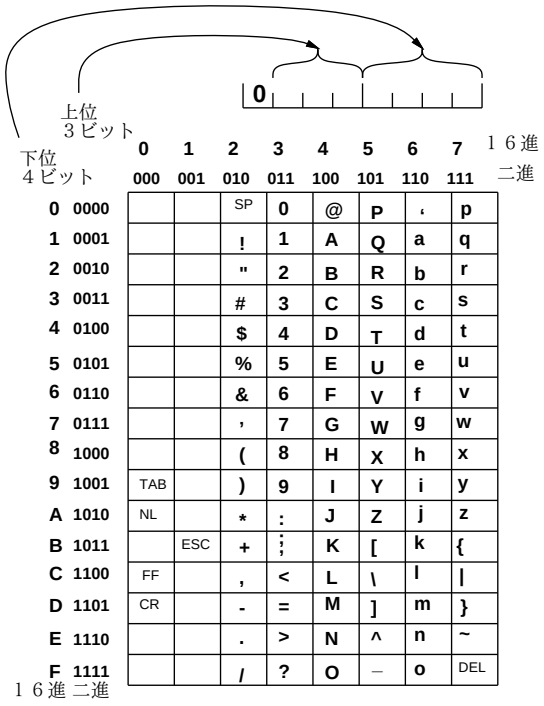
文字の種類は有限ですから、その文字の一覧表を作り、それぞれの文字に重複がないように番号を割り当てます。この番号を**文字コード** (character code) と呼びます。次に、実際にファイルに文字を入れるときに、その文字コードをビット列でどのように表現するかを決めます。これをその文字の**符号化** (encoding) と呼びます。あるひとまとまりの文字群 (英語の文字とか日本語の文字とか) について文字コードと符号化を併せたものを**文字コード体系** (character coding system) と呼びます。

コンピュータの初期には、扱える文字の種類が限られていたので、文字コードと符号化の区別をする必要がありませんでした。このころ作られ、現在でも使われている文字コード体系に **ASCII コード**があります (図 1)。これは英数字・記号と制御文字から成ります。

ASCII の各文字は 7 ビットで表せますが、ファイルはバイト (8 ビット) 単位で扱うので、「0」のビットを追加し、1 バイトに 1 文字を入れます。これが ASCII のテキストファイルです。テキストファイルとは、何らかの文字コード体系のビット列が入っているファイルなのです。

実際にファイルにどのようなビット列が入っているかを調べるには「`od -t x1` ファイル」というコマンドが利用できます。やってみましょう。

```
...$ echo 'This is a pen.' >test.txt
...$ cat test.txt
This is a pen.
...$ od -t x1 test.txt
0000000 54 68 69 73 20 69 73 20 61 20 70 65 6e 2e 0a
0000017 ←ここの表示はファイルの長さ（ただし 8 進表記）
```




| | 0 000 | 1 001 | 2 010 | 3 011 | 4 100 | 5 101 | 6 110 | 7 111 |
|---|---|---|---|---|---|---|---|---|
| 0 0000 | | | SP | 0 | @ | P | ‘ | p |
| 1 0001 | | | ! | 1 | A | Q | a | q |
| 2 0010 | | | " | 2 | B | R | b | r |
| 3 0011 | | | # | 3 | C | S | c | s |
| 4 0100 | | | $ | 4 | D | T | d | t |
| 5 0101 | | | % | 5 | E | U | e | u |
| 6 0110 | | | & | 6 | F | V | f | v |
| 7 0111 | | | ’ | 7 | G | W | g | w |
| 8 1000 | | | ( | 8 | H | X | h | x |
| 9 1001 | TAB | | ) | 9 | I | Y | i | y |
| A 1010 | NL | | * | : | J | Z | j | z |
| B 1011 | | ESC | + | ; | K | [ | k | { |
| C 1100 | FF | | , | < | L | \ | l | \| |
| D 1101 | CR | | - | = | M | ] | m | } |
| E 1110 | | | . | > | N | ^ | n | ~ |
| F 1111 | | | / | ? | O | _ | o | DEL |

図 1: ASCII コードの表

図 1 と照合すると、確かに先頭の「54」は「T」ですし、以下同様で、最後の方の「2e」は「.」です。一番最後の「0a」は改行文字ですね。

**演習 0** 各自が英数字 15 文字までの 1 行のファイルを作り、3〜4 名のグループで互いに「`od -x t1` ファイル」の結果を見せて内容を (紙に書いて) 当てるまでの時間を競う競技を開催してみなさい。最も解読に時間が掛かる問題を出した人の優勝とする。[1]

## 2.2 日本語の文字コード exam

さて、英字は分かったとして、では日本語の文字はどうでしょうか。その前に少し歴史的な話をします。ASCII では前述のように左端のビットは「0」ですから、そこを「1」にした場合に、1 文字を 8 ビットで表すというところは変えないままで、ASCII とは別の文字を最大 $2^7 = 128$ 種類、追加する余地があります。ヨーロッパ各国の言語は英語にない文字 (や、英語の文字にアクセント記号を追加した文字) があるので、それをこの部分に追加した文字コードを使用しました。

日本では、漢字を表示できる画面や打ち出せるプリンタが当初なかったので、カタカナの 50 音と「゛」「゜」をこの部分に追加した文字コードを使用しました。これを「**8 ビットカナ**」や (俗称ですが)「半角カナ」と呼びます。今でも幅が英字と同程度に狭くて濁点・半濁点が別の文字になっているものを見掛けますが、これが 8 ビットカナ文字です。

やがて漢字の表示や印刷ができるようになると、こんどは漢字を含めた日本語文字コード体系が必要になり、**日本工業規格** (Japan Industrial Standard、JIS) として規格 **JIS C6226** が定められました (現在では改訂されて **JIS X0208** となっています)。これを **JIS 漢字コード**と呼びます。

漢字は 256 よりはるかに個数が多いため、日本語の 1 文字は 2 バイトで表す必要があります。そこでこの規格では、1〜94 の「区番号」と 1〜94 の「点番号」を組み合わせた「**区点番号**」で 1 つの文字を表すようになっています (区番号が 1 バイト目、点番号が 2 バイト目に対応)。なぜ 94 かというと、ASCII の図形文字 (制御文字でない、目に見える文字) が「！」〜「〜」までの 94 あるので、それに対応させるという意図があったためです (図 1 を見てください)。

[1] ヒント: 記号を沢山入れたり、普通の英文に見えてわざと違うスペルを入れるなどの技があると強いかもしれない。

## 2.3 日本語のエンコーディング exam

JIS 漢字コードを実際にファイルに格納するときには、ASCII の文字と混ぜて使用することが必要とされました。そこで次の 3 つのエンコーディングが開発され使用されるようになりました。

- **iso-2022-jp**(いわゆる JIS) — ISO-2022 というのは制御文字 ESC で始まる 3 バイトでさまざまな文字コードを切替えるという規格であり、それを用いて「ESC-$-B」で JIS2 バイトコード、「ESC-(-B」で ASCII コードに切替える。
- **euc-jp**(日本語 EUC) — 一番左のビットが 0 のものは ASCII、1 のものは 2 バイトずつ JIS2 バイトコード (の最左端のビットを 1 にしたもの) とする。
- **shift-jis**(シフト JIS) — 日本語 EUC に類似しているが、8 ビットカナと共存させるようにずらしたもの。このため 2 バイト文字の 2 バイト目に最左端のビットが 0 のものもある。

iso-2022-jp は初期の電子メールなど 8 ビット文字が使えない場面で、euc-jp は Unix 系のシステムで、shift-jis は PC 系 (Windows、Mac) で使われてきました。さらに面倒なことに、行末の文字も Unix 系では LF(0a)、Windows 系では CF+LF(0d0a) の 2 バイト、Mac では CR(0d) が使われて来ました。そして今は UTF8(後述) が多くのシステムでも使われるようになりました。改行文字も Mac では OS-X 以降、Unix と同じになってきています。つまり、日本語の文字コード (とエンコーディング) は大変混乱しているというのが現状です。

## 2.4 UNICODE と UTF8 exam

以前は、複数の国の文字を扱うとき、ISO-2022 規格に従い、言語ごとに ESC で始まる 3 バイトで切替えていく、という方法しかありませんでした。しかしこれではコンピュータによる処理が面倒なため、「すべての国のすべての文字に 16 ビットの文字コードを割り当てて扱おう」という考え方が現れました。これが UNICODE で、この考え方に基づく国際規格 **ISO 10646** が制定されています。[2]

表 1: UTF8 による符号化の形式

| UNICODE の値 | bits | ビット列 (UTF8 による符号化) | bytes |
|---|---|---|---|
| 00〜7F | 7 | `0xxxxxxx` | 1 |
| 80〜07FF | 11 | `110yyyxx 10xxxxxx` | 2 |
| 0800〜FFFF | 16 | `1110yyyy 10yyyyxx 10xxxxxx` | 3 |
| 10000〜1FFFFF | 21 | `11110zzz 10zzyyyy 10yyyyxx 10xxxxxx` | 4 |

UNICODE における符号化には複数の方式があり、そのうちで、1 バイト (8 ビット) ずつの処理を前提としたものが **UTF8** です。UTF8 は、00〜7F の範囲は ASCII と完全に一致しており、さらにすべての国の文字をまとめて扱えるため使いやすく、その使用が広まっています。UNICODE の文字コードの範囲は 21 ビットで表現でき、000000〜1FFFFF までの範囲になります。そしてそれを UTF8 で符号化すると 1 文字が 1〜4 バイトで表現されます (表 1)。

## 2.5 Emacs による文字コードの切り替え

本節以下では Emacs のコマンド列が多数出てきますが、読みやすさのため、Ctrl-X (`^X`) の代わりに C-x のように記します。M-x は Meta-X ですが、常に [ESC]x でも同等に使えることに注意。

ここでは Emacs における文字コード切り替えの操作を説明します (他のさまざまな機能は次節)。前述のように、システムにより日本語の文字コード/符号化は様々です。そこで Emacs では、次の 3 つについてそれぞれ別個に「どの文字コード、どのような改行規則」を設定できます。

[2]ただし実際にやってみるとすべての文字を 16 ビットのコードに収録するのは無理だとわかり、現在では一部の文字は 16 ビットを 2 つつなげた値として表現しています。

f. ファイル読み書き — ファイルを読む/書く時の文字コード

t. 画面 — 画面表示をおこなうときの文字コード

k. キー — キー入力をおこなうときの文字コード

なお、Emacs を no window モードで使うときにはこれら 3 つともが問題になりますが、GUI で直接窓を開いているときは、画面表示もキーの入力も Emacs が直接おこなうので、問題になるのはファイル読み書きだけになります。[3]

- C-x [RET] f コード [RET] — ファイル読み書きのコード設定
- C-x [RET] t コード [RET] — 端末画面のコード設定
- C-x [RET] k コード [RET] — キー入力のコード設定

設定できるコードは「`iso-2022-jp`」「`euc-jp`」「`shift_jis`(これだけ語のつなぎが下線!)」「`utf-8`」のいずれかで、さらに改行コードを指定する場合にはその直後に「`-unix`」(LF)、「`-dos`」(CFLF)、「`-mac`」(CR)」のいずれかを指定します (指定しない場合はこれまでと同じになる)。

**演習 1** 日本語のひらがな一式を Emacs でファイルに書き込み、「`od -t x1` ファイル」で表示してみなさい。続いてファイル読み書きの符号化 (文字コード) を変更してから保存し直し、変化があったことを確認しなさい。確認できたら、次の課題から 1 つ以上やってみなさい。

a. 自分の好きな符号化を 1 つ選び、その符号化でのひらがなのビット表現一覧表を作る。

b. 自分の好きな符号化を 1 つ選び、その符号化で「記号」「ひらがな」「カタカナ」「漢字」などがどのあたりにあるかを調べて表にまとめる (全部は大変なのでできる範囲でよい)。

c. ひらがな一式について、文字コードを iso-2022-jp、euc-jp、shift-jis、utf-8 と変更したとき、ビット表現がどのように違うかを整理してまとめる。

**演習 2** 日本語の文章を Emacs で「iso-2022-jp」の符号化で書き込み、「`tr -d '\033' <入力 >出力`」で変換し、日本語のないファイルに化けていることを確認する。できたら、次の課題から 1 つ以上やってみなさい。

a. この化けたファイルから日本語として読めるファイルを復元してみなさい。[4]

b. 単純に復元するかわりに、ひらがなをカタカナに、カタカナをひらがなに入れ換えてから復元してみなさい。[5]

c. (チャレンジ問題)1 バイト文字 (いわゆる半角) と 2 バイト文字 (いわゆる全角) に同じ文字があるものがいくつかある。そのどちらを使うかは自分の好みだが、自分の好みでないファイルを好みであるように変換するシェルスクリプトを書いてみなさい。

# 3 Emacs の詳細

## 3.1 ファイル読み書き等

先に Emacs の基本を説明した際ファイル関係で学んだものに、C-x C-f (ファイルを編集)、C-x C-s (保存)、C-x C-w (名前をつけて保存)、C-x i (ファイルを挿入)、C-x C-c (終了) がありました。

一般に複雑なコマンドを途中で中止するのは C-g (中止) でできます。また、実行し終ったばかりのコマンドの効果を取り消すのには C-x u (アンドゥ) が使えます (取り消せない動作もありますが)。

[3]ただし GUI であっても、Emacs の画面からコピーペーストするときは、コピーペーストする先の文字コードに合わせる必要があります。

[4]ヒント: ASCII と日本語の切替えは ESC-$-B、ESC-(-B であったところを ESC を削除したために文字化けしている。ファイルに現れない文字 X を 1 つ選び、`sed` で$-B や (-B を X-$-B や X-(-B に置き換えてから `tr` でこの X を'\033' に置き換えれば戻るはずである (が、間違った置き換えが起きる可能性もあるかも)。

[5]ヒント: `tr` でひらがなの 1 バイト目をカタカナの 1 バイト目に、またカタカナの 1 バイト目をひらがなの 1 バイト目に置き換えればよい。ただし、2 バイト目にそれらの文字が出て来るのにも対応するとしたら少しやっかいかも。

### 3.2 カーソル移動

C-p C-n C-f C-b (上下左右に 1 文字) は以前学びましたが、左右については 1 単語ずつ移動する M-f M-b も使えます。また行頭/行末に移動するのに C-a C-e が使え、文の先頭/末尾に移動するのに M-a M-e が使え、バッファの先頭/末尾に移動するのに M-< M-> が使えます。表示の制御として、C-l によりカーソルのある行が画面中央になるようスクロールし、1 画面進む/戻るのに C-v M-v が使えます。

現在いるのが何行目かはモードラインに表示されていますが、特定行にカーソルを移動するには「M-x goto-line [RET] 行数 [RET]」でできます。また、一般にコマンドには「C-u 数値」で数値プレフィクスをつけることができ、多くの移動コマンドではこれは反復数を意味します。たとえば「C-u 5 C-n」で 5 行下に行けます。

### 3.3 日本語入力

日本語入力モードは C-\ または C-￥ で ON/OFF できます。ON のときは打ち込んだものはローマ字かな変換されるので、入力したい単語の読みが打ち込めたら [SP] で漢字変換します。漢字変換中は次の候補に進む/前の候補に戻るのに C-n C-p が使えます。正しい漢字になったら [RET] によって確定します。単語によっては変換する単位が (文節) 複数に分かれることがあります。このとき、文節の区切りが正しくない場合は C-i C-o で文節を 1 文字ずつ伸ばす/縮めることができます。現在の文節 OK で先の文節に進むのは C-f また前の文節に戻るのは C-b です (図 2)。

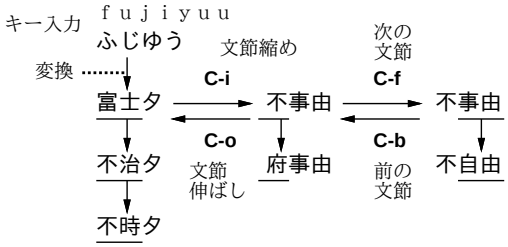



図 2: 文節伸ばし/縮めと文節移動の使い方

### 3.4 削除とコピーペースト

1 文字消すのに [BS] [DEL] C-d がありましたが、大量に消すのは大変です。そのときは、C-k が「行末まで消す、既に行末なら改行を消す」なので、これを連打することで行単位で消していけます。

Emacs では**キルリング** (kill ring) と呼ばれる見えないバッファがあり、ここにテキストを一次保存して別の位置に移したりします。C-k の連打で消したテキストは複数の連打の (他の操作が間に入らない) 範囲でまとまってキルリングに入ります (図 3 上)。

Word などで沢山消すには消す範囲をマウスでドラグして塗りますが、Emacs では (1) カーソルの現在ある位置を「マーク」し、(2) 次にカーソルを任意の場所まで移動することでマークとの間が範囲として指定できます。この範囲のことを**リージョン** (region) と呼びます。マークを設定するコマンドは正規には C-[SP] ですが、演習室の環境ではこのキーが漢字変換に割り当てられているので代わりに C-@ を使う必要があります。リージョンの範囲を確認するのには、マークの位置とカーソルの位置を交換するコマンド C-x C-x が使えます。

リージョンの内容は C-w で削除されキルリングに入ります。また M-w ではリージョンは削除されずにその内容がキルリングにコピーされます。C-y はキルリングの先頭 (最後に消したりコピーした範囲) がカーソル位置に挿入されるので、これを用いてテキストを移動できます。そして、C-y に続いての M-y は、いま挿入されたものをリングの「1 つ前に」あった範囲に置き換えます (図 3 下)。M-y を連続して使うことで、さらに 2 つ前、3 つ前…とできます。

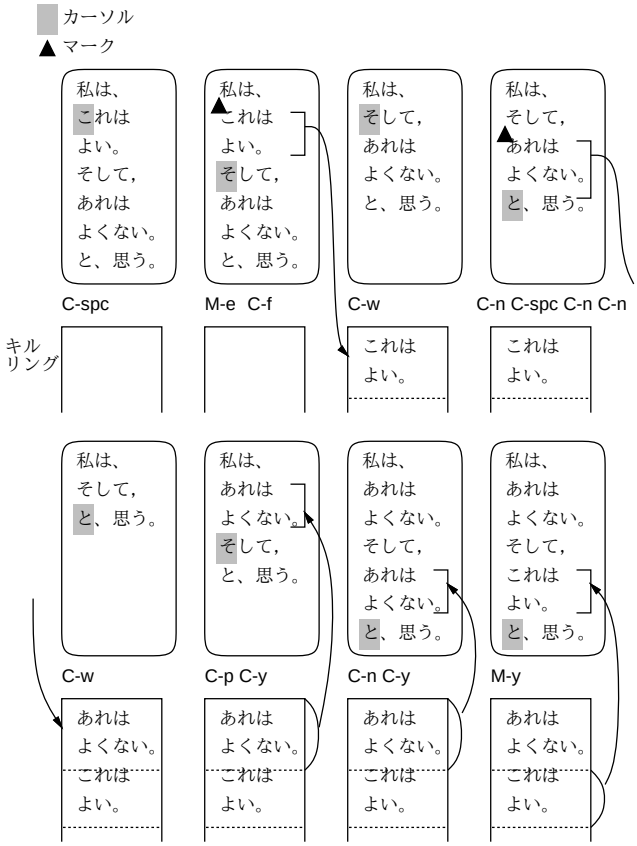



図 3: マーク・リージョンとキルリング

### 3.5 探索と置換

Emacs で英語テキストを扱っているときは**イクリメンタルサーチ** (incremental search) 機能が使えます。これは前方向なら C-s 後方向なら C-r で開始し、その後文字を打つごとにそこまでの文字のある位置にカーソルが移動します (そして入力を [BS] で取り消すとカーソルもそのぶんだけ前の位置に戻ります — 図 4)。めざすものが見つかった時は [ESC] で終了できます。せっかく便利なのですが、日本語の場合は漢字変換と一緒に使えないので、もっと原始的な M-x search-forward [RET] 文字列 [RET] や M-x search-backward [RET] 文字列 [RET] を使ってください。

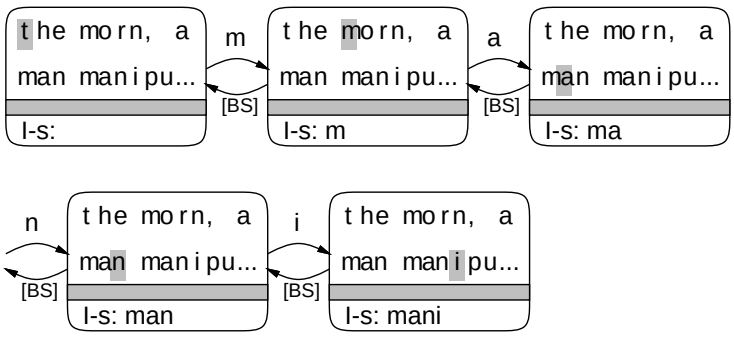



図 4: インクリメンタルサーチ

文字列の置換には M-x replace-string [RET] 文字列$_1$ [RET] 文字列$_2$ [RET] が使えます。ただしこれは一気に全部置換してしまうので、M-x query-replace [RET] 文字列$_1$ [RET] 文字列$_2$ [RET] が使いやすいかも知れません。こちらは 文字列$_1$ が見つかるごとに止まってどうするか聞いてくるので、

[SP] または y で「置換して次の場所」、[BS] または n で「置換せずに次の場所」、q で「置換せず終了」、. で「現在位置を置換して終了」、! で「残りを全部置換」、^ で「誤って置換したので 1 つ前に戻る」のいずれかが行えます。

上の 2 つはいずれも文字列を探索して置換しますが、文字列の代わりに正規表現を探索するコマンド M-x replace-regexp [RET] パターン [RET] 文字列 [RET]、M-x query-replace-regexp [RET] パターン [RET] 文字列 [RET] も使えます。これらではマッチした範囲の一部を`\(...\)` で囲み、置き換え文字列側で`\1` 等で参照する機能が使えます。

文字列やパターンの入力字に制御文字などはそのままでは入れられませんが、C-q を前置することで入れられます。たとえば改行文字はコントロール記法だと C-j になるので、改行をすべて削除するのは M-x replace-string [RET] C-q C-j [RET] [RET] でできます。

### 3.6 窓・フレーム・バッファの操作

Emacs では C-x 2 と C-x 3 で画面を水平/垂直に 2 分割できます (そのそれぞれの部分を窓と呼びます)。その状態でカーソルを移動すると同じバッファの違う場所を見ながら編集することができます。カーソルを他の窓に切替えるのは C-x o でできます。現在いる窓を少し大きくしたい場合は C-x ^ でできます。最後に、C-x 1 と C-x 0 でカーソルの無い側/ある側の窓を消すことができます (図 5)。

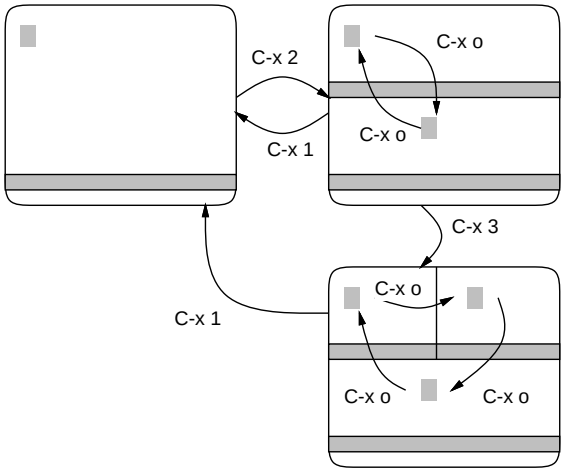



図 5: Emacs の窓分割機能

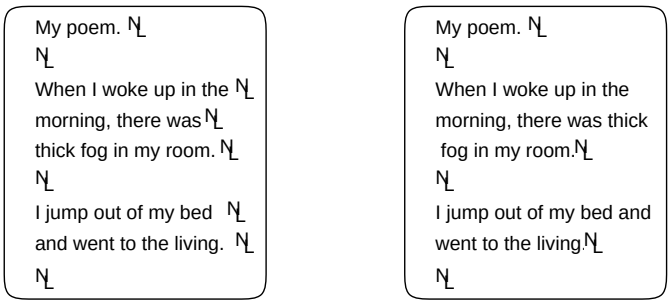
GUI の環境では、ウィンドウ (Emacs の用語ではフレーム) を増やすこともできます。C-x 52 で新しいフレームを作ることができ、 C-x 50 で現在カーソルのあるフレームを閉じます。

前に説明したように、Emacs では内部に複数のバッファを持ちそれぞれで別のファイルを編集できます (そのほか Emacs の作業領域としても使ったりしています)。C-x C-f でファイルを指定すると、そのファイル名と同名のバッファができて、それが現在の窓に表示されます。C-x C-b で現在存在しているバッファの一覧が見られます。そして C-x b バッファ名 [RET] で現在の窓を指定したバッファに切替えられます。

**演習 3** Emacs を使って以下の課題をやってみなさい。

a. 世の中のワープロソフト文化では改行文字は「行の終わり」「段落の終わり」を兼ねています。それに対し、Unix 文化では段落も 1 行ずつ改行されていて、段落の終わりは「1 行アキ (改行文字 2 文字以上連続) で表されています (下図、Word 風の方では画面の端まで来たら NL が無くても次の行に折り返されていることに注意)。Emacs の文字列置換を用

いて、Unix 文化のファイルをワープロソフト文化の形に変換してみなさい。[6]




b. `man` ページを「`man` コマンド名>ファイル」で保存して Emacs で見ると、太字の部分は「`x^Hx`」の形になっています。これはどういう意味かというと、昔のタイプライタでは C-h は「1 文字印字位置を戻す」だったので、「x を打ち (1 文字ぶん進んだ) 印字位置を戻し、もう 1 回 x を打つ」ことで 2 重打ちして太字にしていたわけです。しかしこれでは検索しづらいので、Emacs の文字列置換を用いて普通に読めるように (`x^Hx` をただの `x` に) してみなさい。なお、コントロール文字を「そのまま」入力したいときは「Ctrl-Q Ctrl-H」のように Ctrl-Q を前につけてから入力すればできます。

c. (自由課題) Emacs の置換機能の「面白くて役に立つ」利用例を 1 つ考案して報告しなさい。

## 本日の課題 9A

本日の課題は「演習 1」「演習 2」「演習 3」に含まれる小問 (合計で 9 個) の中から 1 つ以上を選択し、結果をレポートとして報告して頂くことです。LMS の「レポート # 9」の入力欄に直接入力してください (別途作成したものをコピーペーストで貼っても構いません)。以下の内容がこの順に含まれるようにしてください。

- 冒頭に題名「コンピュータリテラシレポート # 8」、学籍番号、氏名、提出日付を書く。(グループでやったものはグループのメンバー全員の氏名も別途書く。)
- 課題の再掲を書く (どんな課題であるかをレポートを読む人が分かる程度に要約する)。
- レポート本体の内容 (やったこととその結果) を書く。
- 考察 (課題をやった結果自分が新たに分かったことや考えたこと) を書く。
- 以下のアンケートに対する回答。

  Q1. テキストファイルや文字コードについてどれくらい知っていましたか。面白いところはありましたか。

  Q2. Emacs の新たに知った機能で興味深かったものはありましたか。または別のエディタの方がいいと思っていますか (それはなぜですか)。

  Q3. リフレクション (今回の課題で分かったこと) ・感想・要望をどうぞ。

なお、課題はグループでやって構いません。その場合も、(メンバー氏名を明記した上で) レポートは必ず各自で執筆してください。レポート文面が同一 (コピー) と認められた場合は同一であると認めた全員について点数にペナルティを科すことがあります。

[6] ヒント: 改行の連続を適当な (文章中に現れない) 印の文字列にし、次に改行文字を空文字列 (または 1 個の空白がいいかも) に変換し、最後に印を 2 つの改行文字に直す。