ヨーロッパ学 ICT 講義テキスト（IA/IB）

永 田 善 久

目次

図目次

はじめに

インターネットが社会における不可欠のインフラとなった現在，どのような専門分野を学ぶにせよ ICT（Information and Communications Technology）の習得が必須とされる時代がすぐそこまで来ている。実際，諸外国の先例に倣う形で，我が国でも「2020 年から小学校でプログラミング教育が必修化」されることが決まった。

この事実は，社会的存在としての人間が習得すべき「読み，書き，計算」に並ぶ新たな基本的能力として，「プログラミング」（あるいはプログラミング的思考法）が「時代を超えて必要とされる普遍的な能力」として認知された，ということに他ならない。

つまり，これからの社会の変革と発展は「プログラミングを含む創造的な ICT 技能」抜きでは成り立たなくなってしまう，ということだ。プログラミングや ICT に関する知識がなければ，想いだけが空回りして具体的な「形」にすることができない，あるいは時代に相応しいアイディアが全く湧いてこない，ということになりかねない。

そうは言っても，では，人文学部で学ぶ我々は具体的に何をすれば良いのだろうか。今すぐ C や PHP あるいは Ruby といったプログラミング言語の学習を始めるべきなのか。それも良いだろう。しかし，何のためにプログラミングをするのかというはっきりとした「目的」なくしては，勉強が長続きしないかも知れない。

そこで「ヨーロッパ学 ICT」講義では，人文学部でドイツ語やフランス語を学ぶ我々の実情に沿うべく，先ずはドイツ語，フランス語，英語，日本語等の「自然言語のテキスト」を自由自在に処理できる技能を身に付ける「テキスト主義 ICT」をテーマに掲げる。「テキスト主義 ICT」の習得は本格的なプログラミング言語学習のためのまたとない助走ともなる。プログラミング言語のソースコードは全てテキストで記述するため，互いに大きな親和性があるからだ。

「ヨーロッパ学 ICT」講義は 2 年間連続するカリキュラム（2 年次： ヨーロッパ学 ICT IA/IB，3 年次： ヨーロッパ学 ICT IIA/IIB）から成る。つまり 2 年（4 学期）間を通じて，一つのまとまった体系をなす「テキスト主義 ICT」技能が身に付くよう，連続的かつ段階的にカリキュラムを組んである。具体的には，GNU Emacs（以下，Emacs）という高機能テキストエディタ（図 1, 4 ページ参照）を用いて高レベルの汎用的「テキスト編集」技能を体系的に学び，その上で TEX（技術・芸術を意味するギリシア語の τέχνη を語源とし，テヒ，テック，テフと発音される。TEX というロゴを出力できない場合，TeX のように e だけを小文字で表記することになっている。本「ヨーロッパ学 ICT 講義テキスト」も TEX を使って組版している）と HTML (Hypertext Markup Language)

/ CSS (Cascading Style Sheets) というマークアップ言語を習得するが，部分的受講だと目指すレベルの技能習得は極めて困難である。段階を経ない受講（ヨーロッパ学 ICT IA を学ばずにいきなり IB を受講する，とか，IA/IB を受講せずに IIA/IIB から受講を開始する，等々）は，そこで学ぶ技能を既に自ら修得しているのでない限り，不可能である。

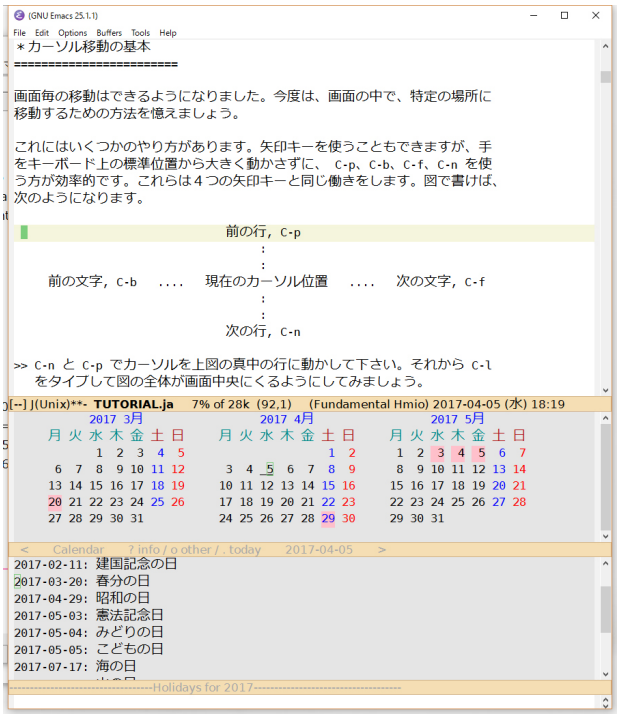


図 1: Emacs（日本語チュートリアルとカレンダー）

Emacs は 1970 年代の中頃 MIT（Massachusetts Institute of Technology）人工知能研究所で誕生し，現在も開発が続けられている古参のテキストエディタで，フリーソフトウェア財団 FSF（Free Software Foundation）による GNU（GNU's Not Unix! の自己言及型頭字語）プロジェクトを象徴するソフトウェアでもある。著名なユーザに Richard Stallman（Emacs 作者），Guy Steele（C 言語開発者），Michael Widenius（MySQL/MariaDB 作者），Linus Torvalds（Linux 作者），Donald Knuth（TEX 作者），まつもとゆきひろ（Ruby 作者），Guido van Rossum（Python 作者），Mark Zuckerberg（Facebook 創始者），Julian Assange（WikiLeaks 創始者）等々がいる。

TEX は 0.000005mm 精度で文字配置を制御できる高機能の文書作成ソフトかつプログラミング言語で，論文・レポートといった「構造化」された文書の処理，また，数式処理や多言語テキスト処理を特に得意とする。TEX でテキストを組版処理すれば，そのまま商業印刷の版下となり得る「美文書」を作成できる（図 2，4 ページ参照）。


Français: Le Petit Prince

Lorsque j'avais six ans j'ai vu, une fois, une magnifique image, dans un livre sur la forêt vierge qui s'appelait *Histoires vécues*. Ça représentait un serpent boa qui avalait un fauve. Voilà la copie du dessin.

On disait dans le livre : « Les serpents boas avalent leur proie tout entière, sans la mâcher. Ensuite ils ne peuvent plus bouger et ils dorment pendant les six mois de leur digestion. »

J'ai alors beaucoup réfléchi sur les aventures de la jungle et, à mon tour, j'ai réussi, avec un crayon de couleur, à tracer mon premier dessin. Mon dessin numéro 1. Il était comme ça :

J'ai montré mon chef-d'uvre aux grandes personnes et je leur ai demandé si mon dessin leur faisait peur.

Elles m'ont répondu : « Pourquoi un chapeau ferait-il peur ? »

Mon dessin ne représentait pas un chapeau. Il représentait un serpent boa qui digérait un éléphant. J'ai alors dessiné l'intérieur du serpent boa, afin que les grandes personnes puissent comprendre.

Elles ont toujours besoin d'explications. Mon dessin numéro 2 était comme ça :

2 j'ai vu] vu < voir: Indicatif passé composé

II sah] sah < sehen: Indikativ Präteritum

Deutsch: Der kleine Prinz

Als ich sechs Jahre alt war, sah ich einmal in einem Buch über den Urwald, das *Erlebte Geschichten* hieß, ein prächtiges Bild. Es stellte eine Riesenschlange dar, wie sie ein Wildtier verschlang. Hier ist eine Kopie der Zeichnung. V

In dem Buche hieß es: „Die Boas verschlingen ihre Beute als Ganzes, ohne sie zu zerbeißen. Daraufhin können sie sich nicht mehr rühren und schlafen sechs Monate, um zu verdauen." X

Ich habe damals viel über die Abenteuer des Dschungels nachgedacht, und ich vollendete mit einem Farbstift meine erste Zeichnung. Meine Zeichnung Nr. 1. So sah sie aus: XV

Ich habe den großen Leuten mein Meisterwerk gezeigt und sie gefragt, ob ihnen meine Zeichnung nicht Angst mache.

Sie haben geantwortet: „Warum sollen wir vor einem Hut Angst haben?" XX

Meine Zeichnung stellte aber keinen Hut dar. Sie stellte eine Riesenschlange dar, die einen Elefanten verdaut. Ich habe dann das Innere der Boa gezeichnet, um es den großen Leuten deutlich zu machen. XXV

Sie brauchen ja immer Erklärungen. Hier meine Zeichnung Nr. 2:

図 2: TEX を使った対訳組版例

HTML は Web ページにおける論理構造を，CSS は視覚レイアウトに関わる箇所を，それぞれ記述するマークアップ言語で，いわゆる「ホームページ」は HTML/CSS で書く。

なお，本講義で TEX を使うといった場合，実際には「素」の TEX に「マクロ」と呼ばれる高機能な一群の命令体系が付け加えられた（Leslie Lamport によって開発された）LATEX を使うが，両名称は，以下，特に区別しないで用いることとする。

TEX，HTML/CSS のソースコードは全てテキストで記述するので，Emacs を使えば全く同じ使い勝手，同じワークフローで，一元的にテキスト編集作業を行える（例： 入力補完，履歴機能，自動整形，等々が使える）。Emacs は，全てのプログラミング言語，ほぼ全ての自然言語の文字入力を取り扱える（シンタクスハイライト，自動インデント，矩形編集，辞書検索，スペルチェック，約 200 種類もある各言語固有のインプットメソッド，等々が使える）。ゲームもできる（Tetris, Solitaire, Bubbles, Pong 等々で遊べる）[*1]。

なお，Emacs や TEX のユーザインタフェース（ヘルプやマニュアルも含む）は全て「英語」だから，我々がこ

[*1] 人文学部で学ぶ者であれば，ここで Johan Huizinga の著書 *Homo Ludens* (1938) を想起したいところだ。

れらのソフトウェアを用いる場合，日常的に英語に触れる環境に身を置くことになり，「英語をツールとして使う」良い実践ともなる。

Emacs はキーボード使用を前提としてデザインされたソフトウェアである。各種修飾キー（Ctrl キーや Alt キー，Mac では Command キーや Option キー等）も駆使して作業するので，タッチタイピング技能は必須である。これができない者は「急がば回れ」，先ずは確実にタッチタイピングができるようにしておくこと。それから，PC やファイルに関する基本操作（コピー，ペースト，削除，等々）を普通にこなせることも前提となる。

これ以上の前提知識は不要だが，ICT 技能は「理論」(講義) と「実践」(受講者自身による反復練習，技術の習得）という両輪の上に成立するものなので，講義時にのみ PC を使う，という姿勢では全くモノにならない。講義外においても不断に PC を「いじり倒す」意気込みが求められる。自分の PC に Emacs と TEX をインストールし，これらを自らの電子文房具として日常的に使いこなすことが，「テキスト主義 ICT」習得における一番の理想形かつ近道である。

PC 教室のコンピュータには，英語や日本語は言うまでもなく，ドイツ語，フランス語の高度なテキスト処理のために特別にカスタマイズした Emacs と TEX をインストールし，2017 年度から Windows 教室, Mac 教室のどちらでもほぼ全く同じように使えるよう環境を整えた。講義サポート Web には，こうした特別仕様の Emacs & TEX 環境を受講者自身の PC（Windows または Mac または Linux）にもインストールして自宅で勉強・反復実践できるよう，解説ページを設けてある。なお，「ヨーロッパ学 ICT」用にカスタマイズした Emacs や TEX を，以後，EURO ICT Emacs や EURO ICT TEX と呼称する。

Emacs も TEX もフリーかつオープンソースソフトウェアであるから，無償でこうした高度多言語テキスト処理用 ICT 環境を受講者自身の PC 上に構築できる。大いに活用して欲しい。なお，ここでいう「フリー」とは，実は， 無償というだけではなく「自由」というもっと高次の意味もが込められた概念である。詳しくはインターネット等を使って自ら検索すること。

講義サポート Web

「講義サポート Web」の URL は以下の通り（本来は改行せず各 1 行表記)。

- http://apapa.hum.fukuoka-u.ac.jp/~ynagata/2017/euro_ict1_ab.html［講義サポート Web］
- http://apapa.hum.fukuoka-u.ac.jp/~ynagata/euro_ict_materials.html［ヨーロッパ学 ICT 課題ファイル置場］
- http://apapa.hum.fukuoka-u.ac.jp/~ynagata/euro_ict_installation_win.html［EURO ICT Emacs & TEX のインストール法（Windows)］
- http://apapa.hum.fukuoka-u.ac.jp/~ynagata/euro_ict_installation_mac.html［EURO ICT Emacs & TEX のインストール法（Mac)］
- http://apapa.hum.fukuoka-u.ac.jp/~ynagata/euro_ict_forum/wforum.cgi［EURO ICT Emacs & TEX 掲示板］

URL にある「2017」および ict「1」の箇所は実際の講義（年度）に応じて可変。講義サポート Web には「課題ファイル置場」ページ，「インストール法」ページ (Windows/Mac)，「掲示板」ページへのリンクが張ってある。

どのような技能が身につくのか

「ヨーロッパ学 ICT IA」(2 年次前期）では Emacs の操作法をゼロから体系的に学ぶ。具体的には

- 文字・語・文・段落・セクショニング（章，節，小節，等々）単位での移動および編集，単語検索・置換
- 正規表現[*2] を用いた検索・置換，矩形編集，複数ファイルの高速処理（ファイル名や拡張子の一括変換，連番処理等)
- 各種文字コード・改行コードの制御および相互変換（いわゆる「文字化け」の原因を理解し，これを解消できるようになる)
- ユニコードによる独・仏・日・英文入力，独・仏・日・英語による自動日付入力

[*2] 形式言語理論分野で用いられる概念。Emacs では正規表現を使えるので，例えば，あるファイルから「be 動詞（be, am, are, is, was, were, been, being 等)」を「一度に全て抽出」といったことができる。

- バイナリファイルである Word ファイルと Excel ファイルをテキストファイルへ変換，およびその逆変換
- 古典ギリシア語（アクセントと気息記号を含む），『旧約聖書』ヘブライ語（右から左書き，ニクードを含む）入力
- Google Translate を呼び出し，多言語テキストを瞬時に相互自動翻訳（独 ⇄ 日，仏 ⇄ 日，独 ⇄ 英，仏 ⇄ 独，等々）

といったことができる技能の習得を目指す。

「ヨーロッパ学 ICT IB」（2 年次後期）では Emacs の応用的操作法を深く学び，終盤ではいよいよ TEX の学習に入る。具体的には

- 多言語スペルチェック（ドイツ語は「旧正書法」および「新正書法」の双方でスペルチェックができる）
- マルチカーソル編集（カーソルを分身させ，複数個所を同時一括編集）
- ダイアリーとカレンダー（日の出入り・月齢・閏年・日独仏の祝祭日チェック，日独仏英の月・曜日名の動的変換，暦の変換[*3]）
- アウトライン編集（セクショニングの昇・降格やツリーの一括移動等，文書全体における論理構造の高速編集）
- スケジューリング（デッドライン，TODO リスト，アジェンダ，ノート等々[*4]）
- 擬似ビッグデータ処理[*5]
- 同一テキストをそのまま HTML および TEX ソースコードに「変換（エクスポート）」
- TEX による組版（文書整形）の初歩

といった技能を習得する。「ヨーロッパ学 ICT IA/IB」で Emacs を縦横無尽に使いこなせるようになれば，さらに「ヨーロッパ学 ICT IIA/IIB」（3 年次前前期・後期）において TEX の高度な使い方と HTML/CSS を学ぶ。

- テキストと高度に戯れる（ドイツ旧字体，ドロップキャップ，写本書体を用いた古文書（「ヒルデブラントの歌」等）処理，多段組み，複数行取り，行番号自動振り，縦書き，隔字体，牛耕式，ロンゴロンゴ，多言語テキスト混植処理，等々）
- 文書の「論理構造」と「視覚レイアウト」を区別し，前者に注力して論文やプレゼンテーションファイル等の「美文書」を作成する
- 目次・図目次・表目次・索引・脚注・各種クロスリファレンスの自動生成，参考文献データベースの作成・読み込みと文献の自動ソート，独・仏・英語の正書法処理，異体字処理（ユニコードでは「同一文字」として同じコードポイントが振られてしまっている「辻・辻」，「葛・葛」，「吉・吉」といった文字も正しく区別して処理できる）
- ユニコードに採用された Emoji（😂😱😻🙀💢💩👎等）をテキストとして処理する
- 電子文書内自動ハイパーリンク処理
- 文字コード（文字集合および文字符号化方式）を正しく理解し，HTML5/CSS3 に則った多言語 Web ページ（プログラミング言語 JavaScript を用いた簡単な「動的」ページも含む）を作成し，インターネットで公開する

といった技能を身に付ける。

諸注意

Emacs および TEX は，仕様が全く異なるハードウェアやオペレーティングシステム上でも同じように動作させることのできるクロスプラットフォーム対応のアプリケーションプログラムである[*6]。このため，ある端末の GUI（Graphical User Interface 対概念は CUI: Character-based User Interface）環境では「（シングル）クリック」で済むところを，別端末の GUI 環境では「ダブルクリック」しなくてはならない，ということがある。

こうした事情を講義テキストの中にいちいち盛り込まねばならないのは煩瑣に過ぎるので，本テキストの中では「クリック」と「右クリック」のみイベントとして区別して用いることにする（つまり，「クリック」と説明してあっても，受講者の端末によっては「ダブルクリッ

[*3] 例えば「音楽の父バッハはユリウス暦 1685 年 3 月 21 日に誕生した。現代グレゴリオ暦に直すと何月何日となるか」，「テルミドール反動はフランス革命歴 9 Thermidor an II de la Révolution に起きた。現代グレゴリオ暦に直すといつになるか」といった問いに瞬時に答を出せるようになる。

[*4] 高度な GTD (Getting Things Done) 用タスク管理ツールとしての Emacs の使用法も紹介する。

[*5] 例えば，「七限」の他に「一，二，・・・，九」限という地名は日本に存在するか，といった問いに対するアプローチ法を学び，短時間で正確な答を導き出せるようになる。

[*6] 講義担当者は，実際，Windows, Mac, Linux の各端末で動作を検証・確認している。

ク」しなくてはならないことがあるかも，ということ）。

受講者の用いる PC 端末は Windows または Mac であると想定している。双方に同様の記述が妥当する場合は一々断らないが，Windows 限定あるいは Mac 限定という場合には，これを明示しておいた。同じ UNIX 系 OS であるから，Linux ユーザであれば Mac に関する記述箇所を参照すれば事足りるだろう。

端末エミュレータは，全て「ターミナル」という名称で呼ぶことにする（Windows のコマンドプロンプトや Windows PowerShell もターミナルと表現する）。alias は別コマンドを併記する際に用いている。[ ] ブラケット内の記述は「省略可」を意味する。

本講義テキストでは「フォルダ」という言葉を使わず，UNIX 流儀の「ディレクトリ」という名称で呼ぶ。こちらの方が Emacs や TEX，さらにはターミナルを用いる際に相性が良いためだ[*7]。.（ドット）で始まる名前を持つファイルは，Mac 教室では（残念ながら）「不可視」設定となっているので，Finder では見えないことに注意[*8]。ターミナルで確認するしかない。Windows では「. ファイル」は見えはするが，例えばエクスプローラーでは .emacs という名前のファイルを作成またはリネームすることはできない。こういった場合も，やはりターミナルで作業するしかない。

なお，本講義では「テキスト処理」という観点から高い有用性が認められる場合，UNIX 系の OS（Linux や Mac）にデフォルトで付随するシェルプログラム Bash（Bourne-Again Shell）をターミナルで利用する方法（シェルコマンドやシェルスクリプトの活用）についても紹介している。Linux/Mac のみならず，Windows においても 2017 年秋の Windows 10 Fall Creators Update からは Windows Subsystem for Linux 上で動作する Ubuntu, openSUSE, Fedora（いずれも著名な Linux ディストリビューション）等の Bash が使えるようになった。

\（正式名称は Reverse Solidus）と ¥ 記号について一言。文字集合の符号化（エンコーディング）における歴史的経緯もあり，かつて日本語の PC 環境では「Backslash \」と「円記号 ¥」を正確に区別して入力することができなかった（あるいは困難だった）。Unicode 文字集合の符号化方式として UTF-8（Unicode Transformation Format 8）が普及してからは，これら 2 つの記号には別々のコードポイントが振られることになり，原理的には明確に区別することが可能となったが，現在でもなお，日本語の PC 環境ではこれら 2 つの記号の意図した入力がままならないケースもある。

EURO ICT Emacs では「日本語キーボードの ¥ キー押下で \ が入力される」設定にしてある。TEX で用いる制御コマンドは \ であり，TEX のソースコードを書く場合は \ を多用するため，こちらの方が便宜が良いからだ。従って，EURO ICT Emacs で \ ではなく ¥ を入力するにはインプットメソッドに ucs を選択し，u に続けて 16 進数コードで直接 00A5 と打ち込む必要がある。なお，Mac では Option と ¥ キーを押すことで円記号を入力することもできる。

最後に受講生への御願いがある。「ヨーロッパ学 ICT」講義では，技能習得の過程において受講生間の積極的な交流を大いに期待している。相互扶助の精神に則り，教え合い，助け合い，互いに高め合って欲しい。個々の技能の習得において学友より先行したという自覚が少しでもある者は，是非教える側にも回って欲しい。古代ローマの思想家・政治家・詩人である小セネカ（Lucius Annaeus Seneca, BC1–AD65）は次のように言っている：「人は教えることで学ぶ」（Homines dum docent discunt. 『ルキリウスへの倫理的書簡』VII, 8）。

ヨーロッパ学 ICT IA

第 1 講

PC 教室での準備（Windows 端末）

PC 教室の Windows 端末で EURO ICT Emacs & TEX 環境を準備するためには以下の作業（この作業は年度始めに「一度」行うだけで良い。その際，旧環境はリネームして保存しておくか，削除すること）を済ませておくこと。

- ローカルディスク C:\usr 以下にある全てのディレクトリ，ファイル（ドットファイルも含め）をホームフォルダ H:\cygwin 以下へコピーする

[*7] 例えば cd というコマンドが change directory に由来する，ということは，「フォルダ」という名称を用いていては分かりにくい。つまり，この名称選択は「記憶術」にも関わる。

[*8] 自分用の Mac では「可視」化しておくと良い。ターミナルで default write com.apple.finder AppleShowAllFiles -boolean true というコマンドを実行し，次に killall Finder と打ち込み Finder を再起動する。ただし，自己責任で。

PC 教室での準備（Mac 端末）

Mac 端末では，しかし，ログインする毎に以下の作業を行わねばならない（Mac PC 教室では個々のユーザのホーム環境を保存できないため）。

- start.command（このスクリプト・プログラムはデスクトップで「ホームフォルダ／アプリケーション／ Emacs」と辿ったところ，絶対パスで書けば /Applications/Emacs/start.command にある）をクリックする
- するとログインユーザのホームディレクトリに EURO ICT Emacs & TEX を利用するための関連ファイル群および諸設定ファイルがコピーされ，Emacs も自動的に起動する

Emacs の起動

EURO Emacs の起動法は以下の通り。GUI/CUI ともに複数の方法で起動してみよ。& の有無による挙動の違いも学ぶこと。

- GUI（Graphical User Interface）で起動
  - アイコンをクリック（Windows/Mac）
  - ターミナルで以下のコマンドを打ち込む（Windows）
    emacs
  - ターミナルで以下のコマンドを打ち込む（Mac）
    emacs [&]
  - ターミナルで以下のコマンドを打ち込み（EURO Emacs ではない）素の Emacs を起動（Windows/Mac）
    emacs -q
- CUI（Character-based User Interface）で起動
  - ターミナルで以下のコマンドを打ち込む（Mac）
    emacs -nw（EURO ICT Emacs では alias: emacsnw）
  - ターミナルで以下のコマンドを打ち込み（EURO Emacs ではない）素の Emacs を起動（Windows 上の Bash/Mac）
    emacs -nw -q

Windwos 上の Bash を使った CUI では EURO ICT Emacs は使えないが，「素の Emacs」であれば利用できる。nw（no window）は window を使わない，q（quick）は「EURO ICT 用初期設定ファイルを読み込ませない」オプション。q の代わりに Q としても良い。Q は q オプションに加え「さらにスプラッシュ画面も省略」して Emacs を起動する。

Emacs の終了

Emacs が備える全コマンドは Lisp というプログラミング言語による関数となっており，これらは「英単語をハイフンで連結したコマンド名」を持つ。これらの中でよく使われるコマンドには少ない打鍵数でコマンドを呼び出せる「キー」が割り当てられており，これを Key Bindings と呼ぶ。

以下，Ctrl, Alt, Shift, F1, F2, …, Tab, Space-Bar, Esc, Enter, Delete, Backspace, PageUp, PageDown, ←, →, ↑, ↓, Windows キーはそれぞれ，C-, M-, S-, <f1>, <f2>, …, TAB, SPC, ESC, RET, <del>, <backspace>, <prior>, <next>, <left>, <right>, <up>, <down>, <win> と記し，Emacs のコマンドを記す際は「Key Bindings（コマンド名）」のように書くこととする。（コマンド名）のみが記されている場合は，対応する Key Bindings がデフォルトでは存在しないことを意味する。

Emacs の終了法を学べ。コマンドは次の通り。

- C-x C-c (save-buffers-kill-terminal)

なお，Emacs の全てのコマンドは M-x save-buffers-kill-terminal のように M-x を前置してキーボードでコマンドを叩くことで実行できる。実際，デフォルトで Key Bindings が存在しないコマンドの場合は，このようにコマンド入力して実行することになる。M は Meta Key を指すが，これは Windows では Alt キーであったり，Mac では Command キーや Option キーであったりするので，使用環境毎にその都度，どのキーが Meta Key となっているかを確かめること。これらのキーが Meta Key として機能しない場合は，ESC キーが使える。

ファイル操作の基本

所与のテキストファイルがあれば，通常のソフトウェア同様，PC 教室ではそのアイコンを右クリックして（個人 PC における設定次第ではアイコンを直接クリックして）Emacs から開くことができる。Emacs が既に起動している場合は，以下に掲げる種々のコマンドを

用いたファイル操作の基本をよく学ぶこと。開く対象に（ファイルではなく）ディレクトリを選ぶと，Dired（Directory Editor）という Emacs のファイラー操作画面となるが，これについては後の講義で詳しく学ぶので，今は Dired の操作法についてあまり気にしなくて良い。なお，EURO ICT Emacs においては，C-x C-f の Key Bindings をデフォルトの (find-file) から，FFAP パッケージが提供するより使い勝手のよい (find-file-at-point) へ置き換えている。

- ファイルを開く： C-x C-f (find-file-at-point)
- 新規作成： 同上コマンドで「新規ファイル名」を与える
- 上書保存： C-x C-s (save-buffer)
- 別名保存： C-x C-w (write-file)
- カーソル位置にファイル挿入： C-x i (insert-file)

第 1 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. PC 上に，適当な作業用ディレクトリ work を作成せよ（例： Windows の場合は H:\work, Mac の場合は /Users/ynagata/Documents/work 等）
2. 上記ファイル 4 点をダウンロードし，work に保存せよ
3. Emacs を起動し，4 つのファイルをそれぞれ開け（ヒント： TAB を押せば Completion 機能が働き，入力補完できる）
4. work 内に my_first.txt を作成し，中身を「Emacs を使った最初のテキストファイル作成演習。This is my first plain text file using Emacs.」とし，保存せよ
5. khm53_ja_utf8_unix.txt ファイルを開き，末尾に「英・独・仏語」テキストを，この順で，それぞれ追加せよ
6. このようにして新たに作成したファイルに khm53_utf8_unix.txt というファイル名を付けて，work 内に保存せよ

Emacs の操作体系は独自の内部整合性・完結性を持つ。これに習熟するまではキーの押し間違いはつきものなので，必要に応じて

- コマンド取り消し： C-g (keyboard-quit)
- アンドゥ： C-/ or C-x u or C-_ (undo)

を使うこと。Übung macht den Meister! C'est en forgeant qu'on devient forgeron ! Practice makes perfect!

発展： Emacs の強制終了法

意図せず Emacs が「固まってしまう」（アプリケーションが全く反応しなくなる）ことがある。こういった場合は，Windows/Mac とも以下に記す 3 つのキーを同時に押し「タスクマネージャー（Windows）」あるいは「アプリケーションの強制終了（Mac）」を起動して，ウィンドウリストに挙がっている Emacs を選択してから「タスクの終了（Windows）」もしくは「強制終了（Mac）」をクリックする。

- Ctrl-Alt-Delete (Windows)
- Commnad-Option-Esc (Mac)

第 2 講

Emacs Jargon

Emacs は正確には GNU Emacs という名称を持ち，これは the GNU incarnation of the advanced, self-documenting, customizable, extensible editor であると説明される。また free/libre text editor – and more とも言われる。Emacs を描写するこうした説明を吟味せよ。

Emacs でグローバルに用いられる以下の基本的諸概念（Emacs Jargon）に慣れよ。file, buffer（file 内のデータ等を一時的に読み込んでおく作業領域），frame（独立した GUI ウィンドウ），window（Emacs の作業画面領域，編集中の buffer がここに表示される），menu bar, tool bar, major mode（対象に特化させて提供される編集環境。Emacs で編集を行う場合，必ず一つの major mode が選択される），minor mode（major mode 内でオプションとして用いられる編集環境，複数選択が可能），mode line, minibuffer（echo area とも），key sequence（複数のキーを押す場合），command, key bindings, complete key（コマンドとして機能する key sequence），prefix key（complete key の前置部分），cursor（buffer 中の現在位置），point（cursor と直前文字間），fill（文字列を整形すること）。

Cursor 移動

テキストを高速に編集するためには，先ずは，文字，語，行，文，段落，画面，buffer, といった異なる Syntax 単位での確実な移動ができなくてはならない（つまり，狙った箇所に「瞬時に飛べる」ということ）。

以下の「移動系コマンド」を習得せよ。その際，f, b, p, n といったキーを用いる Emacs の Mnemonic System にも留意すること。また，prefix key の C- および M- による動作の差異にも注意を払うこと。

- 1 文字先へ： C-f (forward-char)
- 1 文字前へ： C-b (backward-char)
- 上行へ： C-p (previous-line)
- 下行へ： C-n (next-line)
- 1 語先へ： M-f (forward-word)
- 1 語前へ： M-b (backward-word)
- 行頭へ： C-a (beginning-of-line)
- 行末へ： C-e (end-of-line)
- 1 文先へ： M-e (forward-sentence)
- 1 文前へ： M-a (backward-sentence)
- 1 段落先へ： M-} (forward-paragraph)
- 1 段落前へ： M-{ (backward-paragraph)
- 1 画面先へ： C-v (scroll-up)
- 1 画面前へ： M-v (scroll-down)
- buffer 先頭へ： M-< (beginning-of-buffer)
- buffer 末尾へ： M-> (end-of-buffer)
- 画面再描画： C-l (recenter)
- buffer 何行目へ： M-g g (goto-line)
- 行番号表示： <f5> (linum-mode)

Emacs が認識する「語」は原則として欧文単語（両端に space あり）であり，日本語文書内では M-f/M-b により cursor が想定外の箇所に飛ぶことがある。「文」は欧文では period (.) によって，邦文では句点（。）によって，それぞれ識別される。「段落」は空行で隔てられることで認識される。linum-mode コマンドの Key Bindings <f5> は「toggle キー」（同一操作で on/off 切り替え）となっている。

発展： C-M- コマンドでの移動

移動系コマンドには，自然言語が持つ構造「以外」の Syntax 単位での cursor 移動を可能としてくれるものも存在する。これらは通常 major mode と連動し，対象ファイルの編集に際し，さらなるより使い勝手の良い cursor 移動手段を提供してくれる。例えば，プログラミング言語のソースファイルを編集する場合は，対応する式や関数の先頭・末尾に瞬時に移動できるコマンド群が用意されている，という具合である。

通常のテキストでは，例えば，C-M-f (forward-sexp), C-M-b (backward-sexp) で，それぞれ「対応するカッコ」に飛ぶことができるので試してみよ。

French/Non-French Spacing

グーテンベルクによる活版印刷術の発明以来，ヨーロッパでは「文（さらには約物）間スペース」をどれほど取るのか，について言語毎にそれぞれの因襲が確立されてきた（詳しくは，“history of sentence spacing” で Google 検索してみると良い）。

例えば，英語では省略符として用いられる Mr. の . とは異なり，「文末」の . の後には「広目の space」（Non-French Spacing）を置くことになっているが，フランス語やドイツ語では双方ともに「同じ長さの space」（French Spacing）を置くのが慣習となった。タイプライターが使われるようになると，英文を打つ場合，文末には「2 つの spaces」を挿入することが一般的となる。

ここで，Emacs における「文単位での cursor 移動」を考えてみよう。アメリカ生まれの Emacs は，デフォルトでは，「文」を「period に続く 2 つの spaces」によって識別する。従って，

> This is Mr. Brown. And that is Mrs. Brown. They live in Fukuoka.

という英文（実際には 1 行，また，文末 period 後には 2 spaces）があった場合，デフォルトの Emacs ではこの行の上で C-a, M-e と打つと，Mr. をすっ飛ばして，いきなり Brown. の後まで cursor が移動することになる。

しかし，EURO ICT Emacs では「ドイツ語，フランス語を優先」し，French Spacing に対応した振舞をするように Emacs をカスタマイズしてある。こうしたカスタマイズ箇所は，全て Emacs Lisp というプログラミング言語を用いて「設定ファイル」（テキストファイル。Windows では C:\home\.emacs.d\init.el，Mac では /Users/ynagata/.emacs.el）の中に記述してある（ynagata 部分は Mac におけるユーザ名）。

Emacs を起動する際には，常にこの設定ファイルが読み込まれることになるので，EURO ICT Emacs では

French Spacing 対応となる。この振舞を抑制したい場合，言い換えれば，French Spacing 対応するように記述された設定ファイルを「読み込ませないで素の Emacs を起動」したい場合には，ターミナル上で emacs -q もしくは emacs -Q と打ち込めばよい。

第 2 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. 演習用ファイルを使って，欧文テキスト，邦文テキストとも「cursor 移動系コマンド」を実際に検証せよ
2. (help-with-tutorial-spec-language) コマンドを実行し，Emacs が備える「多言語 Tutorial」を buffer に呼び出せ
3. Japanese, English, German または French を言語に選び，それぞれ Tutorial を熟読せよ（中身は全て同じなので，多言語の速読練習ともなる！）
4. その際，特に「cursor 系移動コマンド」の動作を検証すること
5. Tutorial 内にある未習箇所については，今は無視して良いが，今後，折に触れて Tutorial に立ち返ること
6. Tutorial（に限らないが現 buffer）を消すには C-x k (kill-buffer) と打つ
7. 「素の Emacs」を起動し，Non-French Spacing の動作を検証せよ
8. 逆に EURO ICT Emacs を立ち上げ，French Spacing の動作を検証せよ

発展： スマートフォンから Emacs を遠隔操作

Emacs は，もう一つの有名なテキストエディタである Vi[m] 同様，UNIX 系 OS には大抵デフォルトで搭載されている。GUI のみならず CUI でも動作するため，リモート先のテキストファイルをネットワーク越しに遠隔操作する，などということも可能である。

図 3（11 ページ）は，Android Smartphone からリモート先の Linux サーバに SSH（Secure Shell）通信により接続し，サーバ上にある HTML ファイルを Emacs で編集している様子を示したものである。もちろん，スマートフォンには物理キーボードが付いていないので実際の編集作業に使うことは稀であるが，ここでは Emacs を「広範に活用できる事例」の一つとして紹介しておく。

図 3: スマートフォンから Web サーバ上にある HTML ファイルを Emacs でリモート編集

第 3 講

テキスト編集： 変換，交換

buffer に読み込まれたテキスト内を Syntax 単位で自由自在に移動でき，目的箇所に素早く辿り着けるようになれば，次は，テキストを「編集」することを学ぶ。

「編集」とは元のテキストに「手を入れ」て「中身を変更する（元の姿を変えてしまうこと）」ことであり，Emacs を学習し始めた頃は，不慣れのためキーの押し間違いをすることは不可避であるため，想定外のトラブルに見舞われて，元テキストを「破壊」して取返しのつかないことをしてしまうのではないか，と不安を覚える向きもあろうが，心配は無用である。

先ず，Emacs は編集中にテキスト（ファイル）そのも

のには手を付けない（のでオリジナルを破壊することはない）。Emacs はテキストの「コピー」を buffer に読み込み，編集は全て buffer 上のコピーに対して行われるからである。編集した部分を「上書保存」するときに始めて元テキストが書き換えられる。次に，Emacs は自動的にバックアップファイルを作成してくれている（ファイル名末尾に ~ が付いたもの）。さらに，編集作業中に突然電源が落ちてしまった，などという（本来であれば壊滅的な）事故の場合でも，編集途上の buffer を自動保存してくれている（ファイル名が # で挟まれたもの）ため，ここからファイルを recover できるし，あれやこれやの編集を「御破算」にして元ファイルに revert することもできる（後述）。

もっとも，テキスト「編集」の段階に学習が進めば，「万一の場合に備え，適宜小まめに，自分から能動的に編集済み buffer をファイルに保存する」という癖を身に付けておくのが良い。

以下の「変換」系および「交換」系編集コマンドを習得せよ。実際にはこれらのコマンドを用いる場合，C-g (keyboard-quit) や C-/ or C-x u or C-_ (undo) コマンドも併せて使うケースが増えるだろう。region 概念は後で学ぶが，同系のコマンドをまとめておく。

- cursor 位置文字を大文字へ： M-c (capitalize-word)
- cursor 位置文字以降を大文字へ： M-u (upcase-word)
- cursor 位置文字以降を小文字へ： M-l (downcase-word)
- region 内文字列を大文字へ： C-x C-u (upcase-region)
- region 内文字列を小文字へ： C-x C-l (downcase-region)
- region 内の語頭のみ大文字へ： (cap-r or capitalize-region)
- 前後文字交換： C-t (transpose-chars)
- 前後語交換： M-t (transpose-words)
- 前後行交換： C-x C-t (transpose-lines)
- 前後文交換： (t-sen or transpose-sentences)
- 前後段落交換： (tr-p or transpose-paragraphs)

なお，C-x C-u および C-x C-l コマンドはデフォルトでは無効となっている。C-x C-u をはじめて実行すると，“Disabled Command” buffer が開き “You have typed C-x C-u, invoking disabled command upcase-region …” というメッセージが表示される。この場合，y と答えてやり，さらに “Please answer y or n. Enable command future editing sessions also? (y or n)” と尋ねてくるのでこれにも y と答える。すると，設定ファイルに (put ’downcase-region ’disabled nil) が追加され，これ以降この設定が有効となる。C-x C-l についても同様。

入力支援： auto-complete, electric-pair

EURO ICT Emacs では「テキスト編集」の効率を上げるため，様々な入力支援機能を追加しているが，ここでは，先ず auto-complete.el と electric-pair.el パッケージを紹介する。

auto-complete とは，buffer 上で文字入力を行った瞬間から入力支援が開始されるもので，自動的に pop up する補完候補から <up>, <down> を使って選択し，RET で確定することで，素早く目的の語を入力できる。補完されたくない場合は， C-g あるいは <right> を押せば良い。

electric-pair は，( ), [ ], { }, （ ），" ", 「」 といった対応するカッコ類を，開始カッコを入力するだけで終了カッコも同時に入力してくれるものである。

EURO ICT Emacs では，auto-complete, electric-pair ともテキストファイル編集時に自動的に on となるよう設定してあるが，以下のコマンドにより，これを off とすることができる。

- (auto-complete-mode)
- (electric-pair-mode)

いずれも toggle コマンドである。

半角／全角変換

邦文テキスト内に欧文文字を混在させる場合は，原則として全ての欧文文字を「半角」で入力する。数字も同様に半角で打つ。半角・全角文字が混在しているテキストを編集する場合には，次のコマンドが使える。3 番目のコマンドは設定ファイル内で定義している。

- region 内文字列を全て半角（カナ）へ： (ja-ha or japanese-hankaku-region)
- region 内文字列を全て全角へ：(ja-ze or japanese-zenkaku-region)
- 全角英数字を半角へ，半角カナを全角へ： C-c h

(normalize-chars)

2 番目のコマンドにおいて，英数字は半角のままにしておきたいときは，前置引数 C-u を入力してからコマンド入力（C-u M-x ...）する。

第 3 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. 演習用ファイルを使って，欧文テキストまた邦文テキストとも，上に挙げた編集用「変換」系コマンドと「交換」系コマンドの動作確認をせよ
2. 望む結果を得るためにはどこに cursor を置くのが良いか，最適箇所を見つけ出せ
3. auto-complete, electric-pair の挙動を検証せよ
4. 半角・全角変換を試してみよ

第 4 講

テキスト編集： 削除

以下の Emacs Jargon に習熟せよ： kill（文字列を buffer から削除して，それを kill ring に保存する），kill ring（kill された文字列が保存される場所），delete（単純削除。kill ring に保存しない），yank（kill ring から文字列を引っ張り出してきて，buffer に戻す）。

以下の「削除」系編集コマンドを習得せよ。

- cursor 前文字または region 内文字列を delete: <del>[*9], <backspace> (delete-backward-char)
- cursor 後文字を delete: C-d (delete-char)
- cursor 後の語を kill: M-d (kill-word)
- cursor 前の語を kill: M-<del> or M-<backspace> (backward-kill-word)
- cursor 以降行末までを kill: C-k (kill-line)
- cursor（がどこにあってもその）行を（改行コードも含め）kill: C-S-<backspace> (kill-whole-line)
- cursor 以降の文を kill: M-k (kill-sentence)
- cursor 前の文を kill: C-x <del>, <backspace> (backward-kill-sentence)
- cursor 前後の空白を delete: M-\ (delete-horizontal-space)
- 同上, ただし one space 残しながら toggle：M-SPC (cycle-spacing)
- 現在行または現在行以下の空白行を delete: C-x C-o (delete-blank-lines)
- 現在行と前行を連結： M-^ (delete-indentation)
- kill ring に最後に入った文字列を yank: C-y (yank)
- kill ring に格納されている他の文字列を選んで yank: M-y (browse-kill-ring)
- 対応カッコをペアで削除： C-c p (delete-pair)

EURO ICT Emacs では，素の Emacs に browse-kill-ring.el という拡張パッケージを組み込み，M-y の Key Bindings を オリジナルの yank-pop から browse-kill-ring コマンドへ置き換えている。こちらの方が使い勝手が良いためだ。C-c p を使う場合は，「開始カッコ」に cursor を置く。

Mode Line での編集状況確認

buffer の編集状況は，mode line で確認できる。以下の表示の意味を学べ。

- - -: 変更なし
- * *: 変更あり（未保存）
- % %: 読込専用 buffer
- % *: 読込専用 buffer に変更あり

編集対象ファイルを，意図せず「読み込み専用」で開いてしまった場合は，C-x C-q (read-only-mode) で編集可能（読み込み専用を解除）状態に戻せる。C-x C-q は toggle キーである。

第 4 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. 演習用ファイルを使って，欧文テキストまた邦文テキストとも，上に挙げた「削除」系編集コマンドの動作確認をせよ
2. その際，C-/ or C-x u or C-_ (undo) コマンドを使ってみよ
3. C-y, M-y コマンドも検証せよ

[*9] Windows では Delete キーが (delete-forward-char) として機能するので注意。

Homo Ludens 1: カエサル暗号と (rot13-region/other-window)

次の文は「英文」であるとする。さて，何と書いてあるか分かるだろうか。Ur ybirf Pyrbcngen.

共和政ローマ期の政治家であり武将であり文筆家であり，そして何より英雄であったカエサル（Gaius Iulius Caesar, BC100–BC44）は，機密書を草する際，アルファベットを 3 文字後ろにずらした暗号文を使ったという（スエトニウス『ローマ皇帝伝』より）。カエサル暗号と呼ばれるこの方式は単一換字式暗号（Monoalphabetic Substitution Cipher）の一種で，現在のセキュリティ水準から見た場合あまりに単純であるため実際の用途には適さないが，それでも暗号技術において現在でもなお基本となる「規則（アルゴリズム）」（＝各文字を辞書順に特定数ずらす）と「鍵」（ずらす際の数値を指定する）という 2 要素が既に完全な形で含まれているという事実には，感嘆せざるを得ない。

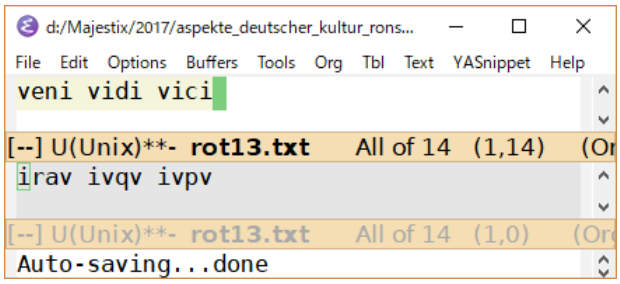


図 4: カエサル暗号と (rot13-other-window) コマンド

そして Emacs はカエサル暗号に基づいたコマンドを持っている（英字限定）。

- region 内の各英字を 13 字ずらす： (rot13-region)
- 分割した別 window にリアルタイムで rot13 を表示： (rot13-other-window)

英字の総数は 26 字だから，ある文字列を対象に (rot13-region) コマンドを「1 回」打つと「暗号化」され，「2 回」発行すると「復号化」されることが分かる。(rot13-other-window) コマンドを用いれば，英字入力テキストをリアルタイムで 13 字ずつずらしながら表示させることができる（図 4，14 ページ）。単純であるとはいえ，(rot13-region/other-window) は古代ローマのカエサルを偲ぶオマージュとなっているコマンドである。

なお，単一換字式暗号を解析するためのコマンド (decipher) も Emacs には用意されている。さらに，region 部文字列をモールス符号に変換するコマンド (morse-region) やモールス符号を文字列に変換するコマンド (unmorse-region) も備わっている。試してみよ。

Homo Ludens 1: 演習

1. Ur ybirf Pyrbcngen. を (rot13-region) で復号せよ
2. 同じ文字列を使って (rot13-other-window) の動作を検証せよ

カエサルは自分のことを記す際 1 人称を使わず「3 人称」を用いた（『ガリア戦記』参照），というエピソードも良く知られている。

第 5 講

テキスト編集： Region 操作

以下の Emacs Jargon に習熟せよ： region（選択領域，mark と point で囲まれた部分），mark（領域指定の始点），point（既出。region 関連では領域指定の終点），copy（文字列を buffer から削除することなく，それを kill ring に保存する）。

delete したものは単純削除されるだけだが，kill されたものは全て kill ring に格納されており，これらはいつでも yank で取り出せることに留意すること。

以下の region 系編集コマンドを習得せよ。mark をセットして cursor 移動すると，そこが point となり，これにより region を指定できる。region 指定すれば，テキストの任意領域を編集対象とすることができる。また，C-@ と「移動」系コマンドを組み合わせることで，テキスト編集効率が一段と高まる。EURO ICT Emacs では，region 指定領域はハイライトされるように設定済みである。

- mark をセット： C-@ (set-mark-command)
- mark と point を交換して可視化： C-x C-x (exchange-point-and-mark)
- region を kill: C-w (kill-region)
- region を copy: M-w (kill-ring-save)
- 現段落を region 指定： M-h (mark-paragraph)
- 現 buffer 全体を region 指定： C-x h (mark-hole-buffer)
- cursor 位置にブックマークをセット，かつ，カーソルのある行をハイライト： C-SPC (bm-toggle)

- 次のブックマークへ： M-] (bm-next)
- 前のブックマークへ： M-[(bm-previous)

set-mark-command コマンドは，オリジナルの Emacs では C-SPC にも key bind されているが，EURO ICT Emacs では C-SPC を 拡張パッケージの bm.el が備えるコマンド bm-toggle に key bind し直している。bm.el は buffer 内に「ハイライトされる book mark をセット」するプログラムで，テキスト編集における便宜性をさらに高めてくれるはずである。C-SPC は toggle キーとなっている。

なお，Mac においては，C-SPC の Key Bindings が別アプリケーションのキーボードショートカット（例えば Spotlight）に使われてしまっていることがあるため，bm.el を使いたい場合は，自分で「システム環境設定」に手を入れてこれをカスタマイズするか，あるいは，コマンド入力で凌ぐ。

CUI 環境で C-@ で mark をセットすることができない場合は，C-SPC を使えば mark をセットすることができる。

編集破棄とファイルの復元

これまではテキストの「変更」に力点を置いた編集コマンドを学んできたが，場合によってはあまりにもごちゃごちゃと手を加え過ぎた結果，いっそのこと全てを御破算にして，元のファイルにあった通りのテキストに戻したい，ということもあろう。

次のコマンドで，前回ファイルを保存した時の状態まで buffer を復元することができる。

- (revert-buffer)

逆に，突然電源が切断された，とか，Emacs が異常終了した，といった場合のように，「自らの意志に反して失われてしまった」編集内容を復元したいこともある。こういった場合は，Emacs が自動保存しているファイルから以下のコマンドでテキストを取り戻せることがある。

- (recover-file)

広範囲に及ぶ編集作業を行うようになれば，次のコマンドの有用性も理解できるようになる。

- cursor のある段落の自動整形： M-q (fill-paragraph)
- region にある各段落の自動整形： (fill-region)
- 折り返し文字数の設定： C-x f (set-fill-column)

EURO ICT Emacs では半角 60 文字で 1 行を折り返す設定にしてある。改行文字数を変更したい場合は C-x f で適当な数値を指定した後で M-q すれば良い。なお，buffer 先頭に Time-stamp: <> と記しておけば，C-x C-s した際に「自動タイムスタンプ」が記録される。これも便利な機能であるから，活用して欲しい。

英・独・仏・日本語での「今日の日付」挿入

EURO ICT Emacs では，以下の key-sequence で，それぞれ英語，ドイツ語，フランス語，日本語による「今日の日付」を挿入することができる。前置引数（Prefix Argument）である C-u を前に打てば，表記形式を変えられる。英・独・仏では 2 回まで前置引数を取れる。日本語は 1 回のみ（Linux では 2 回の前置引数を与えることで「元号」による紀年表記もできる）。

- 英語で今日の日付： C-c e (insert-date-e)
- ドイツ語で今日の日付： C-c d (insert-date-d)
- フランス語で今日の日付： C-c f (insert-date-f)
- 日本語で今日の日付： C-c j (insert-date-j)

第 5 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. 演習用ファイルを使って，欧文テキストまた邦文テキストとも，上に挙げた「region 操作」系編集コマンドの動作確認をせよ
2. 段落を 1 つ copy し，buffer 末尾に yank せよ
3. 段落を 3 つ copy し，buffer 先頭に yank せよ
4. 文を 1 つ kill し，適当な位置に yank せよ
5. 文を 2 つ kill し，適当な位置に yank せよ
6. テキストを様々に kill し，C-y, M-y の動作を検証せよ
7. M-q を使って適当に整形せよ
8. テキストの適当な位置に book mark を複数セットせよ
9. セットした book mark 間を移動せよ
10. 2 つの book mark 間を region 指定し，その部分を copy し，適当な箇所に yank せよ
11. セットした book mark を解除せよ
12. 変更を加えた buffer を，それぞれ別名で保存せよ
13. 自動タイムスタンプを活用せよ
14. 英・独・仏・日本語で今日の日付を挿入してみよ

Homo Ludens 2: La Tour d'Hanoï

フィボナッチ数列やメルセンヌ（素）数の研究で知られる 19 世紀のフランスの数学者 François Édouard Anatole Lucas は，1883 年に自身で考案したゲーム「ハノイの塔」(La Tour d'Hanoï) を発売した。ハノイの塔は，別名，バラモンの塔，ブラフマーの塔，リュカ（ルーカス）の塔とも呼ばれる。

当時同梱されたリーフレットには，ハノイの塔にまつわる「それらしい（創作された）伝説」も添えられていた。詳細は Google 等で検索できる。現代では，PC あるいはスマートフォン用にフリーの「ハノイの塔」ゲームも配布されているから，誰でもこのゲームを簡単に試すことができる。

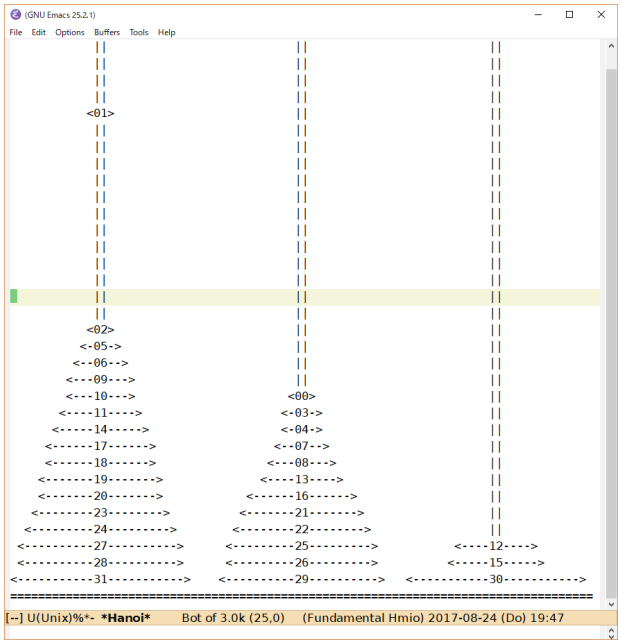


図 5: 32 枚の円盤からなるハノイの塔

さて，ハノイの塔とは，3 本ある軸のうち一番左の軸に「大から小の順」に刺さっている円盤を，1 枚ずつ動かし，最終的には一番右（または真中）の軸に「元の形のまま」に移動させるゲームであるが，移動の際，「小さな円盤の上に大きな円盤は乗せられない」というルールに従わねばならない。

実はハノイの塔は再帰的アルゴリズムの問題そのものとなっており，n 枚の円盤全てを移動させるには「最低 $2^n - 1$ 回の手数がかかる」ことが証明できる。

Emacs はハノイの塔に関連する 3 つのコマンドを持つ。

- 3 枚の円盤からなるハノイの塔： (hanoi)
- 32 枚の円盤からなるハノイの塔： (hanoi-unix)
- 64 枚の円盤からなるハノイの塔：(hanoi-unix-64)

いずれも自らゲームをプレーできる形は取っておらず，「Emacs がルールに従って円盤を動かす様」をじっと眺めていることになる（図 5，16 ページ参照）。C-u 10 M-x hanoi のように Prefix Key を叩いてから整数値（例では 10）を打ち込めば，初期値の 3 に限らず，自分の好きな「円盤の数」を指定できる。

ところで，32 枚の円盤から成るハノイの塔を別軸に移すにはどのくらいの時間がかかるだろうか。

1 つの円盤を動かす際に 1 秒かかる（実際 Emacs でもそのように動く）と仮定すれば，所要時間は手数と全く同じ「$2^n - 1$」秒となる。(hanoi-unix) では，従って約 136 年，(hanoi-unix-64) に至っては，何と，5849 億年以上も時間がかかる（我々が住むこの宇宙の始まりとされるビッグバンでさえ，今から 138 億年前の出来事とされている）ことが分かる！　リュカのリーフレットには「天地創造のときに神は 64 枚の純金の円盤を大きい順に重ねて置いた・・・司祭達は昼夜を通して円盤を別の柱に移し替えている・・・全ての円盤の移し替えが完了すると，世界は崩壊し終焉を迎える」と書かれていた。

Emacs の (hanoi-unix), (hanoi-unix-64) の双方において，1970 年 1 月 1 日 0 時 0 分 0 秒（いわゆる UNIX 時刻の起点時）から 1 枚 1 秒で円盤を動かしたときの進行の様子を見ることができる。

第 6 講

検索： インクリメンタル検索

本格的にテキストを編集したい，という場合は，Emacs が備える多彩な検索機能と置換機能を利用することになる。

先ずは Incremental Search から習得せよ。検索文字列の 1 文字目を入力すると同時に cursor 位置以降のテキスト検索が開始されるので，この検索法を使えば，最小の打鍵数で目的の語に到達できる。

- cursor 以降を I-search: C-s (isearch-forward)
- cursor より前を I-search: C-r (isearch-backward)
- I-search 終了： RET
- I-search 取り消し： C-g (keyboard-quit)
- I-search で input method を toggle: C-\ (isearch-

toggle-input-method)
- I-search で input method を指定： C-^ (isearch-toggle-specified-input-method)
- I-search 時に Migemo を on/off: M-m
- cursor 位置「単語」を検索対象とする： C-s C-w (isearch-yank-word-or-char)
- 検索対象単語末から 1 文字ずつ消去： C-M-w (isearch-del-char)
- 検索対象単語末から 1 文字ずつ伸長： C-M-y (isearch-yank-char)
- 前回の検索を順方向へ再開： C-s C-s
- 前回の検索を逆方向へ再開： C-r C-r
- 検索履歴を前にたどる： C-s M-p
- 検索履歴を後にたどる： C-s M-n

人文学部でドイツ語，フランス語を学ぶ我々が起草したり編集したりする文書は，普通に multilingual documents である。つまり，1 文書内に日本語，ドイツ語，フランス語，英語等の多言語が混在する。特に欧文テキストを主に編集している場合，日本語を検索する際その都度一々「かな漢字変換の日本語入力システム」に input method を切り替えるのは煩わしく感じられる。

そこで，EURO ICT Emacs では「ラテン（ローマ）字のまま日本語（かなカナ漢字）を検索できる」便利なプログラム Migemo を使えるようにしてある。C-s, C-r と打ち込むだけで Migemo が機能する Incremental Search となる。Migemo の詳しい使い方については Google 検索すること。基本的にはそのまま小文字のラテン文字を入力すれば良いが，「吾輩は猫である」というような複数の文節にまたがる言葉を検索するときは wagahaiHaNekoDearu というように文節の先頭を大文字で区切る。C-s, C-r では，もちろん，日本語（かなカナ漢字）のままでも検索できる。input method がプログラムに奪われて minibuffer に日本語を入力できない場合は，C-\ あるいは C-^ を打って対処する。

Incremental Search において Migemo を off にしたい場合は，続けて M-m を押せばよい。これは toggle キーとなっている。

目指す検索文字列にたどり着けたら RET を押して検索を終了する。この場合，最後の検索位置に cursor が移動する。検索を取り消す場合は，C-g を押す。すると，検索開始位置まで cursor が戻る。この 2 つの挙動の違いに注意すること。

なお，Emacs の version 25 からは，区分的発音符（diacritical marks）等を正確に入力せずとも「母字」だけでこうした「区分的発音符付きの文字」（例えば a ä á à など）までをも Incremental Search できる機能（Emacs Jargon ではこれを character folding と呼ぶ）が追加されたが，残念ながら，現時点では Migemo の初期設定と両立できないため，EURO ICT Emacs ではこの機能を on にしていない。

一時的にこの character folding 検索を行いたい場合は，C-s M-s ' (isearch-toggle-char-fold) と打てば良い。character folding 検索では，引用符号の検索時にも同様の振舞をしてくれる。

第 6 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. 演習用ファイルを使って，欧文テキストまた邦文テキストとも，Incremental Search の動作確認をせよ
2. 欧文において lower-case 文字だけを検索語に指定する場合と，upper-case 文字も打ち込む場合の違いはなにか（ヒント：case-[in]sensitive）
3. I-search において欧文単語「間」の空白はどのように認識されているか
4. minibuffer に打ち込んだ通り (literally 反対は loosely）の whitespaces で検索する場合は I-search 時に M-s SPC (isearch-toggle-lax-whitespace) を押す
5. ただしこの場合，Migemo は off にしておく必要がある
6. 上の動作を検証せよ（M-s SPC は toggle キー）
7. character folding 検索を検証せよ

第 7 講

検索： 単純検索

Emacs では文字列を全て打ち込んでから検索を開始させる単純検索（Nonincremental Search）も可能である。以下のコマンドを学べ。*string* は文字列を指す。

- 順方向へ単純検索： C-s RET *string* RET
- 逆方向へ単純検索： C-r RET *string* RET

続けて C-s/C-r を複数回押下すれば，順／逆方向への単

純検索を繰り返すことができる。

検索： 単語検索

単語検索（Word Search）は欧文限定（日本語では不可)。語の連なりがあった場合，句読点や改行を無視して検索する。以下のコマンドを学べ。

- M-s w (isearch-forward-word)

Incremental Seach 中に M-s w コマンドを打てば，Word Seach に移行できる。この場合， M-s w は toggle キーとなる（iseach-toggle-word)。かつては，検索したい語句が 2 行にわたっていた場合，この語句を見つけるには Word Search を使うしかなかったが，現在の Emacs では Incremental/Nonincremental Search でも行をまたぐ語句の検索ができるようになっている（ただし，その際 Migemo はオフにしておく)。

逆に，行をまたぐ語句「のみ」を検索したい場合には，C-q C-j を打ち，明示的に改行コードを入力してやればよい。

第 7 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. 演習用ファイルを使って，欧文テキストまた邦文テキストとも，Nonincremental Search および Word Search の動作確認をせよ
2. その際，自分で適当なテスト語句を考え，そこでも検証せよ

第 8 講

置換： 一括置換

buffer 中のある文字列を，一つ一つ確認する必要もなく，一気に「全て」別の文字列に置換したい，という場合には，以下のコマンドを用いる。

- *string* を *newstring* で一括置換： (r-st or replace-string) RET *string* RET *newstring* RET

この際気を付ける点は，このコマンドは「point から buffer 末尾まで」の文字列を置換対象とする，ということである。従ってこのコマンドを用いる際は，常に M-< で cursor を buffer 先頭に移動してから編集作業に入る，という習慣を身に付けると良い。

一括置換が行われると，cursor は最後に置換された文字列の後に来る。これを，置換開始前の位置に戻したい場合は，C-u C-@ と 打てば良い。

置換： 問い合わせ置換

問答無用の一括変換ではなく，Emacs と対話しながら行うのが「問い合わせ置換」である。必要に応じて cursor を buffer 先頭に移動するのは一括変換時と同様。以下のコマンドを学べ。

- 問い合わせ置換： M-% (query-replace) *string* RET *newstring* RET

既に問い合わせ置換を行っていた場合は，最後の置換パタンがデフォルトとして提示されることに注意。Emacs からの「問い合わせ」に対する「答え」としては y（置換する)，n（置換しない)，,（文字列が置換されたことを確認してから y で先に進める)，!（残りは確認せずに置換する)，q（置換を終了）などのキーを押下すれば良い。なお，? を打てば「これら以外の答え方」についても知ることができる。

置換： 再帰編集

問い合わせ置換実行中は，buffer 内での移動を目の前に見ているわけで，その際，「あ，ここも変更しておきたい」という箇所が出てくることは容易に考えられる。

Emacs はこのための機能も提供してくれる。「再帰編集」がそれだ。問い合わせ置換作業の途中で一旦置換を保留し，buffer 内移動中に目に留まった他の箇所を編集してから，再度保留したところに戻り置換を続けることができる。

以下のコマンドを学べ。

- 問い合わせ置換中に再帰編集： C-r (recursive-edit)
- 再帰編集を終了し, 問い合わせ置換に戻る: C-M-c (exit-recursive-edit)
- 再帰編集だけでなく，問い合わせ置換も併せて終了する： C-] (abort-recursive-edit)

なお，「全角」の再帰編集をした場合は，問い合わせ置換に戻る前に「半角」に戻しておくこと。

第 8 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. 演習用ファイルを使って，欧文テキストまた邦文テキストとも，一括置換および問い合わせ置換の動作確認をせよ
2. 問い合わせ置換時の「これら以外の答え方」を検証せよ
3. 再帰編集に入ると mode line に変化があったはずだが，それは何か

第 9 講

インプットメソッド： Latin-Prefix

ここまでは，欧文文字であれ，かなカナ漢字であれ，Emacs における個々の文字入力方法については特に説明してこなかった。EURO ICT Emacs では，OS に備わる標準の文字入力方式（Windows では「Microsoft IME」，Mac では「Mac 標準 IM」等）も，もちろんそのまま使えるようにしてある。

しかし，将来的な拡張性を持たせた「自然言語もプログラミング言語も編集対象とするテキストの高速処理」を考えた場合，こうした入力方法だけに頼っていると，どうしても不便を感じるケースに遭遇する。

なぜか。それは OS の文字入力方式が，標準日本語キーボードのキー配列を大きく変更してしまうことがあるからだ。例えば，今，インプットメソッドを「ドイツ語キーボード」配列にしてみたものと仮定しよう。すると，Z と Y の位置が入れ替わってしまっていることに気付く。さらに，ドイツ語キーボード配列のまま \ や @ を入力するのは，さほど簡単ではないことも分かってくる。

日本語キーボード配列をベースとして「日本語とドイツ語とフランス語とプログラムのソースコードを入力する」作業を考える場合，言語やプログラム毎にインプットメソッドを切り替えねばならない編集方法では，効率的であるとは言えまい。

Emacs は，OS とは別に，独自のインプットメソッドを約 200 種類も備えており，その中には，我々の目的である「日本語とドイツ語とフランス語とプログラムのソースコードを簡便に入力する」のに打ってつけ――つまり日本語キーボード配列のままキーを打てる――のインプットメソッドもある。Latin-Prefix である。

インプットメソッドに関する以下のコマンドを学べ。併せて window 操作法も覚えよ。同時に使うことになる。

- インプットメソッドの切り替え： C-x RET C-\ *method* RET (set-input-method)
- 選択したインプットメソッドをオン・オフする： C-\ (toggle-input-method)
- インプットメソッドの説明を見る： C-h I or C-h C-\ *method* RET (describe-input-method)
- 全インプットメソッドのリストを見る： (list-input-methods)
- 別 window に移る： C-x o (other-window)
- 選択中の window を消す： C-x 0 (delete-window)
- 選択した window を除き frame 内にある全 window を消す： C-x 1 (delete-other-windows)
- 別 window を scroll する： M-<next> or C-M-v (scroll-other-window)
- 別 window を逆方向に scroll する： M-<prior> or C-M-S-v (scroll-other-window-down)

method を入力せずに SPC を叩くと対象となるメソッドのリストが現れるので，これを見ながら minibuffer にて望むメソッドを入力することもできる。もちろん TAB を押せば入力補完機能も使えるし，適当なところで SPC を打ちながら作業すれば，目当てのメソッドを段々と絞り込んでも行ける。

行数・語数・文字数のカウント

region に対して，以下のコマンドを実行することで，region 内の「行数・語数・文字数」をカウントすることができる。

- 行数・語数・文字数のカウント：(cou-w or count-words)

ただし，日本語における「語数」は，大抵の場合，正しく計上されない。Emacs が分節する位置は M-f/M-b で移動する箇所と一致する。なお space や引用符も 1 文字としてカウントされる。

発展： Say コマンドで外国語テキストを読み上げさせる（Mac）

Mac には say という「ドイツ語，フランス語，イギリス英語，アメリカ英語，日本語，・・・」ネイティブを模した発話プログラムが内蔵されており，ターミナルから任意のテキストをそれぞれの言語の発音に相応しく読み上げさせることができる。読み上げさせるテキストは，

ターミナル上で直接打ち込んでも良いし，テキストファイルとして予め用意しておいても良い。1 分間あたりの発話語数を指定することにより，「速聴・遅聴」にも使える。

ネイティブとしては，例えばドイツ語では Anna, フランス語では Thomas, イギリス英語では Daniel, アメリカ英語では Alex, 日本語では Kyoko, がそれぞれ使える。

say 関連の以下のコマンドを試してみよ。

- システムに用意されている「各言語ネイティブ」の名称確認： say -v ? [| less]
- Quotation Marks で囲まれたテキストを -v (voice) で指定したネイティブに読み上げさせる： say -v Daniel "I am speaking British English."
- 上記と同じ，ただし 1 分間に 100 語の発話ペース（rate）で： say -v Alex -r 100 "I am speaking American English."（標準レートは 175–180）
- 準備したドイツ語テキスト deutsch.txt（file）を，読み上げ箇所をハイライトさせながら（interactive）Anna に発話させる： say -v Anna -i -f deutsch.txt
- 上記の音声をファイルに保存する： say -v Anna -f deutsch.txt -o deutsch.aiff
- say の詳しい使い方をマニュアルで参照する： man say

音声ファイル形式としては，他にも m4a 等色々と指定できるので，マニュアルを参照すること。

第 9 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. Emacs のインプットメソッドは正確にはいくつあるか数えよ
2. latin-prefix の説明を良く読め
3. ウムラウトやアクサン等の入力法を理解したら，khm53_de/fr_utf8_unix.txt を手本に latin-prefix メソッドを用いてキーボード操作し，「全く同じテキストを作成し，別名でファイル保存」せよ
4. これらの作業を行う際，上に挙げたコマンドの動作を全て検証せよ
5. 各言語テキストについて，行数・語数・文字数をカウントせよ
6. 各言語テキストを，Mac の say コマンドを使って読み上げさせてみよ
7. 英語テキストを Kyoko に読み上げさせるとどうなるか，検証せよ

第 10 講

Buffer, Window, Frame の操作

第 9 講演習に取り組んだ際，恐らく誰もが「frame を左右に window 分割できた方が作業が捗るのだが」という印象を持ったことだろう。もちろん Emacs ではそれもできるし，full frame 表示することもできる。

buffer, window, frame の操作に関する以下のコマンドを習得せよ。

- 上下に window 分割：C-x 2 (split-window-below)
- 左右に window 分割： C-x 3 (split-window-right)
- 左右分割された同一 buffer を仮想的な 1 つの window として扱う： (follow-mode)
- （C-x C-f 派生形）別 window で開く：C-x 4 f (find-file-other-window)
- （C-x C-f 派生形）別 frame で開く： C-x 5 f (find-file-other-frame)
- 選択した window の高さを増やす： C-x ^ (enlarge-window)
- 行数が少ない buffer の高さを適当に縮める： C-x - (shrink-window-if-larger-than-buffer)
- 全 window の高さおよび幅を同じに： C-x + (balance-windows)
- 選択した window 横幅を広げる： C-x } (enlarge-window-horizontally) frame の縁をマウスでドラッグしても良い
- 選択した window 横幅を縮める： C-x { (shrink-window-horizontally) frame の縁をマウスでドラッグしても良い
- frame を最大化表示（toggle）： M-<f10> (toggle-frame-maximized)
- frame を full screen 表示（toggle）： Windows は <f11>, Mac では「緑ボタン」を押す (toggle-frame-fullscreen)
- buffer 内文字を拡大／縮小／リセット： C-x C-+/-/0 (text-scale-adjust)
- buffer リストを表示： C-x C-b (list-buffers)
- buffer リスト中で次の buffer へ移動：C-x <right> (next-buffer)

- buffer リスト中で前の buffer へ移動： C-x <left> (previous-buffer)
- buffer を削除： C-x k (kill-buffer)
- 対話しながら個々の buffer を削除： (kill-some-buffers)
- 目指す buffer に移動： C-x b (helm-mini)

(follow-mode) は toggle コマンドとなっている。window の高さを変更する際，C-x ^ を繰り返して打鍵するのが面倒だと感じれば，C-u (universal-argument) の後に適当な数字（デフォルトは 4）を打ち込んでから C-x ^ としてやれば良い。なお，EURO ICT Emacs では，smartrep.el パッケージを用いた追加設定をしているので C-x }/C-x { を用いる際 Prefix Key を繰り返し打鍵する必要はない。}, { だけ叩けばよい。

デフォルトの Emacs では C-x b は (switch-to-buffer) に key bind されているが，EURO ICT Emacs では使い勝手を高めるため追加インストールした helm パッケージの (helm-mini) コマンドに bind し直している。こちらであれば，今開かれている buffers のみならず「最近開いたファイル」をもリストアップしてくれるからだ。

なお，マウスを使わずにウィンドウを切り替えたい場合には，Windows では Alt + TAB を Mac では Command + TAB を使う。ただし Mac においては，同一アプリケーション内の切り替え（Emacs の frame を 2 つ開いている場合等）には Command + Fn + F1 を用いる。

第 10 講演習

演習用ファイル： khm53_de/en/fr/ja_utf8_unix.txt

1. 第 9 講で学習したコマンドも含めて，今回学習したコマンドを様々に検証せよ
2. window を上下分割せよ，左右分割せよ，上下分割せよ，・・・
3. 分割した window を元に戻せ
4. buffer を様々に操作してみよ

第 11 講

正規表現： 検索

「正規表現 (Regular Expression, 正則表現とも)」とは，そもそもは形式言語理論分野で用いられる術語で，文字列の集合を一つの文字列で表現する方法の一つである。

Emacs のようなテキストエディタでは，あくまでも実用的な観点から正規表現を利用するが，それは「特定のパタンが出現しているかどうかを判定」する（Pattern Matching）目的で文字列を検索したい場合に行われる。

具体的に見てみよう。例えば，バッハと同年にドイツのハレで生まれたヘンデルという作曲家がいる。彼の名前はそもそも Georg Friedrich Händel であったが，イギリスに帰化してからは George Frideric Handel と英語名表記になった。

さて，あるテキストの中で「ヘンデル」を表すのに Handel あるいは Händel またさらには Hendel, Haendel などという表記が混在してしまっていた場合，これらの「表記の揺れ」を「一つのパタン」で「マッチ」させることのできる表記法があれば，これらの文字列を「一度に，漏れなく」検索できることになる。

つまり，Emacs での正規表現は，こういった検索や置換を行いたい場合に用いられるのである。

表記の揺れを持つ検索対象文字列のパタンを正確に表す正規表現式を与えるのは，しかし，それほど簡単ではない。それなりの時間をかけて試行錯誤しつつ経験を積むことが必要である。そこで本講義テキストでは，いくつか「具体的な検索法」を検討するに留め，それ以上の検証は学習者自らの意欲に任せる。

正規表現検索に関する以下のコマンドを学べ。

- 順方向に正規表現インクリメンタル検索： C-u C-s or C-M-s (isearch-forward-regexp)
- 逆方向に正規表現インクリメンタル検索： C-u C-r or C-M-r (isearch-backward-regexp)

先の「ヘンデル」に戻ると，揺れのある 4 つの表記にマッチする正規表現式は，以下のように書ける。

- `H\(e\|ae?\|ä\)ndel`

実際に試してみよ。次に，なぜこのような記法となるのか，少々考えてみよ（ヒント：「エスケープ文字」,「または」,「直前文字が 0 もしくは 1 個」)。この記法に含まれる Syntax については講義時に解説するが，Emacs が内蔵するマニュアルに全て詳しく説明されているので，そちらを参照すること。

Emacs のマニュアルは，次のようにして呼び出せる。

- C-h i m Emacs

実際にマニュアルを表示してみよ。そして，Search, Regexp Search, Regexps, Regexp Backslash あたりを参

照せよ。

正規表現： 一括置換

今の学習段階で用いることはない（後に学習する TEX や HTML/CSS のソースを書く際には，当然のように使用することになる）が，ICT 術語として使われる「コメント」とは，プログラミング言語やマークアップ言語で書かれたソースコードの中に挿入された「人間が読むための注釈」を指し，この箇所の処理はコンピュータによって無視される（コメントアウト）。

コメント記号はプログラミング言語毎に異なる（例えば，C では #, Lisp では ;, TEX では %, HTML では <!--> 等々）。これをその都度意識しているのは面倒くさいので，Emacs では major mode に対応したコメント記号を，次のコマンドで自動的に挿入してくれる。コメント記号を外すことは「アンコメント」という。

- 行または region 全体をコメント・アンコメント： C-x C-; (comment-line)
- cursor 行末にコメント記号付加，または region 全体をコメント・アンコメント： M-; (comment-dwim)

この 2 つのコマンドを上手に使い分けるには「慣れ」が必要である（特に後者。dwim: Do What I Mean）。

さて，ここで，将来の TEX 学習を見据えて，あるテキストの「行末」に「出典を示すページ番号」を「コメント」として挿入することを考えてみる。

先ず, 正規表現による一括置換方法のコマンドを学べ。

- 正規表現による一括置換： (replace-regexp)

行末は正規表現を使って $ で表す（ちなみに行頭は ^）。従って，対象箇所を region 指定し，上記コマンドを打ち込み，「$」を「 % P.53」などに全置換する指定をすれば良い。

次に，既にこのようなコメントが施されたテキストがあると仮定する。今度は逆に，1 行毎に埋め込まれたコメントを全て削除し，さらに（例えば，編集済みのテキストをワードプロセッサで編集するために）改行箇所を全て取り去り，段落を（長い）1 行にまとめたい。どうしたら良いか。

段取りとしては，先ず「コメント部分」を「(空白)」で全置換をし，「行末に存在するかもしれない space を除去」した上で，「改行」を「(空白)」に全置換する作業を行えばよい。

「改行」や「Tab」といった制御文字を挿入するコマンドが C-q (quoted-insert) である。以下のコマンドを学べ。

- 改行の入力： C-q C-j
- Tab の入力： C-q TAB

今回仮定したコメントは「出典ページ」であり，例えば P.98 や P.102 のように桁数が異なることも有り得る。このような場合においても「一度で処理」をしたい場合は，次の正規表現を学べ。

- `% P\.\w*`

式の意味を考えてみよ（ヒント： 「正規表現における . そのものは任意の 1 文字を表すから \ でエスケープする」,「単語を構成する任意の文字（数字も含む）」,「直前文字の 0 個以上の繰り返し」）。

そのままでは space は見落としがち（space と全角空白は別物であることに注意！）なので，以下の toggle コマンドも知っておくと良い。

- space, Tab 等を可視化： C-c w (whit-m or whitespace-mode)
- buffer 内行末 space を全削除： (delete-trailing-whitespace)

正規表現： 問い合わせ置換

正規表現による問い合わせ置換のやり方も学べ。

- 正規表現で問い合わせ置換： C-M-% (query-replace-regexp)

後は，実践あるのみ。経験を積め。

第 11 講演習

演習用ファイル： khm53_ja_utf8_unix.txt

1. ファイルを開き，空白行を除く各行末に「出典ページ番号」を「TEX コメント」の形で挿入せよ
2. 「出典ページ番号」コメントを全て削除し，かつ，改行を取り去り，1 行 1 段落とせよ
3. be, sein, être の全活用形を表現する正規表現式を考えよ
4. それらを記したファイルを作成し，正規表現によ

る問い合わせ置換を行え

第 12 講

Dired

Emacs はテキスト編集用モードのみならず，ディレクトリ編集に特化した major mode である Dired (Directory Editor) をも備える。

Dired は各種ファイル操作を簡便にしてくれる filer プログラムの一種であるが，実に多機能であるため，本講義で扱えるのはそれらの内の一部に過ぎない。詳しくは Emacs マニュアルを呼び出し（C-h i m Emacs）Dired (Directory and file manager) の項目を参照すること。

Dired の起動法は

- C-x d (dired-at-point)

であるが，C-x C-f (find-file-at-point) コマンドで（ファイルではなく）ディレクトリを指定しても同様に Dired が立ち上がる。なお，EURO ICT Emacs では，C-x d の Key Bindings を，デフォルトの (dired) からより使い勝手のよい FFAP (Finding Files at Point) パッケージが提供する (dired-at-point) に変更し，さらに，Dired において条件を満たすファイルのみを絞り込んで表示できる機能を追加する dired-filter もが同時に作動するように設定してある。

また，素の Dired では，テキスト・バイナリといったファイル種類の違いに拘らず，RET キーを叩くと対象ファイルを全て Emacs で開こうとするが，この挙動はやや不便であるから，EURO ICT Emacs では openwith.el パッケージを組み入れることで，*.pdf, *.ods, *.xls[x], *.doc[x] といった拡張子を持つファイルを開く際には，システム（Windows, Mac, Linux）がこれらの拡張子にデフォルトで関連付けているアプリケーションで開くように変更してある。

以下の Dired 関連コマンドを学べ。Dired モードに入れば，一般的なテキスト編集モード時とは異なり，Dired モード内独自の Key Bindings となっていること，また，大文字・小文字の区別にも注意せよ。

- Dired を抜ける： q (quit-window)
- Dired のリフレッシュ: g (revert-buffer)
- Dired のヘルプを表示： h (describe-mode)
- 下行へ移動： n (dired-next-line)
- 上行へ移動： p (dired-previous-line)
- 次のディレクトリへ移動： > (dired-next-dirline)
- 前のディレクトリへ移動： < (dired-prev-dirline)
- 日付・ファイル名順に sort（toggle キー）: s (dired-sort-toggle-or-edit)
- 削除対象ファイルにフラグ立て： d (dired-flag-file-deletion)
- auto-save ファイルにフラグ立て： # (dired-flag-auto-save-files)
- backup ファイルにフラグ立て： ~ (dired-flag-backup-files)
- その他自動生成ファイルにフラグ立て： % & (dired-flag-garbage-files)
- フラグの立ったファイルを削除： x (dired-do-flagged-delete)
- マーク付け： m (dired-mark)
- cursor 位置のマーク・フラグ解除： u (dired-unmark)
- Dired 内全マーク・フラグ解除: U (dired-unmark-all-marks)
- マークを toggle: t (dired-toggle-marks)
- マークの付いたファイルに対し正規表現問い合わせ置換： Q (dired-do-query-replace-regexp)
- コピー： C (dired-do-copy)
- ファイル・ディレクトリの移動および名称変更： R (dired-do-rename)
- ディレクトリ作成： + (dired-create-directory)
- 全ファイル・ディレクトリ表示： / /
- ディレクトリで絞り込み： / d
- 拡張子で絞り込み： / .
- 直前条件の反転： / !

矩形編集

普通に region 指定すると図 6（24 ページ）のように指定範囲が選択されるが，region を「矩形（長方形，ブロック）」で選択したい場合もある。こういう場合のために，Emacs には「矩形編集モード」が付いている。以下のコマンドを学べ。

- 矩形編集モードへ移行（toggle キー）： C-x SPC (rectangle-mark-mode)
- 矩形選択 region の前に空行挿入： C-o (open-rectangle)

- 矩形選択文字列を *string* で置換： C-t *string* (string-rectangle)

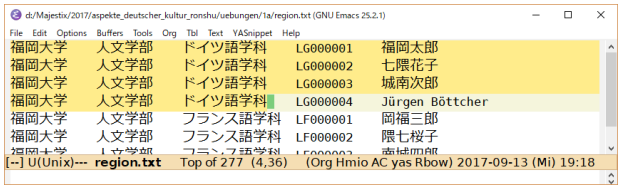


図 6: 通常の region 指定

矩形編集モードに入ると，図 7（24 ページ）のように region 選択ができ，後は通常の編集作業となる。

EURO ICT Emacs では Wdired (Writable Dired) も有効にしている。Dired モード中に「e」と打つことで Wdired (wdired-change-to-wdired-mode) が呼び出されるように設定してあり，ファイル名や拡張子の編集のみならず，ファイルパーミッションも編集可能となる。C-c C-c で編集結果を保存， C-c ESC で編集を中止して元の Dired モードに復帰する。

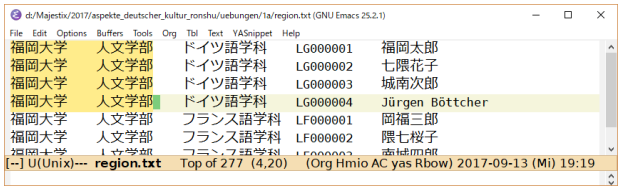


図 7: 矩形で region 指定

発展： カレントディレクトリ以下のファイル内文字列を再帰的に一括置換

Emacs では，カレントディレクトリ以下のファイル (サブディレクトリ，サブサブディレクトリ，・・・，にあるファイルも含む）内の文字列を一括置換することもできる。ただし，正確には外部プログラムである Shell から find というコマンドを利用するので，Linux/Mac ではそのままで大丈夫であるが，Windows では Bash Shell が使えるようになっていなくてはならない。

Mac や Linux で以下のコマンドを検証してみよ。

- (find-name-dired)

このコマンドを打ち込むと minibuffer で Find-name (direcroty): と尋ねてくるので，置換対象ファイル群が含まれる親ディレクトリ（カレントディレクトリ）を指定する。すると次には Find-name (filename wildcard): と尋ねてくる。ここで言う wildcard は Emacs のそれではなく shell の wildcard なので *.txt（拡張子が txt），あるいは，*（全ファイル）のように指定する。

Dired が立ち上がれば，t を押して対象ファイルを全選択した上で，Q を押すと Query replace regexp in marked files と尋ねてくるので，一つ一つ確認する必要がなければ Y を押してやれば一括置換される。最後に C-x s ! (save-some-buffers) 打ち込んでやれば作業は終わり。

発展： シェルスクリプトを用いた再帰的一括置換

この技法も Mac, Linux であればそのまま使えるが，Windows の場合は，Windows Subsystem for Linux 上で Bash が使えるようになっていなければならない。

以下のような内容を持つシェルスクリプトを用意すれば，「表記の揺れ」を含むテキスト template.txt の中身は同じだが名前が dummy_001.txt から dummy_100.txt まで異なるファイルを 100 個自動生成させ，さらにこれらを一気にサブディレクトリ，サブサブディレクトリにもコピーすることができる。do と done の間の処理には「コマンド置換」機能を使っている。

```
#!/bin/bash

for i in `seq -f%03g 1 100`
do
  `cat template.txt > dummy_$i.txt`
done
mkdir -p subdir/subsubdir
cp dummy_*.txt subdir
cp dummy_*.txt subdir/subsubdir
```

シェルスクリプトは，ターミナルを開き，そのシェルスクリプトに実行属性を付与してから，実行する。シェルスクリプトが script.sh というファイル名であったとすれば，具体的には以下のように打ち込む。

```
chmod +x script.sh
./script.sh
```

そして以下のシェルスクリプト（find で始まる行は，本来は改行なし）を走らせてやれば，サブディレクトリ，サブサブディレクトリにまたがって存在するテキスト内の表記の揺れを一度で統一（一気に全置換）できる。丸カッコ（Parentheses）で囲った箇所は，正規表現置換により「後方参照」される。

```
#!/bin/bash
```

```
find . -name "dummy_*.txt" | xargs sed -i
 's/\(福大\|福岡大\|福岡大学\)\(附属\|付属\)
 大濠\(高校\|高等学校\)*/\1\2（大濠）/g'
find . -name "dummy_*.txt" | xargs sed -i
 's/\(福大\|福岡大\|福岡大学\)\(附属\|付属\)
 若葉\(高校\|高等学校\)*/\1\2（若葉）/g'
find . -name "dummy_*.txt" | xargs sed -i
 's/h\(e\|a\|ae\|ä\)ndel/Händel/gi'
```

上記シェルスクリプトは，Mac でも GNU sed が使えるようになっていれば，sed 箇所を gsed と書き換えるだけでそのまま使える。なお，GNU sed のような UNIX 系ソフトウェアは，Mac では Homebrew 等のパッケージ管理システムを用いて簡単に導入できるので，Homebrew をインストールしておくと良い。Homebrew 本体のインストールが済ませてあれば，brew install gnu-sed とターミナルに打ち込むだけで GNU sed（プログラム名は gsed）が使えるようになる。GNU sed の方が Mac の sed より遥かに使いやすい。

GNU sed ではなく Mac の sed を使う場合は

```
#!/bin/bash

find . -name "dummy_*.txt" | xargs sed -E -i
 "" "s/(福大|福岡大|福岡大学)(附属|付属)
 大濠 (高校|高等学校)*/\1\2（大濠）/g"
find . -name "dummy_*.txt" | xargs sed -E -i
 "" "s/(福大|福岡大|福岡大学)(附属|付属)
 若葉 (高校|高等学校)*/\1\2（若葉）/g"
find . -name "dummy_*.txt" | xargs sed -E -i
 "" "s/h(e|a|ae|ä)ndel/Händel/g"
```

とする。これで同様の処理が可能となる。

第 12 講演習

演習用ファイル：ohori_wakaba.txt, region.txt, mk_file_serial_padding_subdir.sh, *.jpg, *.png, replace_regexp_subdir.sh

1. 上で解説した Dired 関連コマンドを全て検証せよ
2. 上で取り上げられていない Dired 関連コマンドを調べ，動作を検証せよ
3. ohori_wakaba.txt は「福岡大学，福岡大，福大，附属，付属，高校，高等学校」等々，表記の揺れを含むファイルであるが，これらの表記を全て「福岡大学附属大濠高等学校」に統一せよ
4. 次に「福岡大」，「福岡大学」，「附属」，「付属」といった原表記はそのままにして，「福岡大学附属（大濠）」といった表記に統一せよ
5. 適当な作業ディレクトリを Current Directory とし，ここに ohori_wakaba.txt の中身を持つファイルを，ファイル名に連番を振り，100 個作成せよ
6. 上記作業ディレクトリの中にサブディレクトリ subdir を作り，上と全く同様に，100 個ファイルを作成せよ
7. subdir の中にさらに subsubdir ディレクトリを作り，ここにも上と同じように，100 個ファイルを作成せよ
8. シェルスクリプト mk_file_serial_padding_subdir.sh を走らせてみよ
9. シェルスクリプト replace_regexp_subdir.sh を走らせてみよ
10. region.txt を使って，矩形編集を試してみよ
11. 画像ファイル（拡張子が jpg, png 等）を images ディレクトリに格納せよ
12. Wdired を用いて，images ディレクトリ内にある全画像ファイルの拡張子を小文字 3 文字に，ファイル名を picture_1, picture_2, のように統一せよ

第 13 講

多言語インプットメソッド

Emacs は

- Full Unicode support for nearly all human scripts
- Support for displaying and editing bidirectional text, including right-to-left scripts such as Arabic and Hebrew

と謳われており，実際，広範囲の国際文字集合を編集できる。手始めに，以下のコマンドを打ち込み，世界の「こんにちは」ファイルを表示してみよ。

- C-h h (view-hello-file)

ドイツ語やフランス語テキストの入力法については既に学んだ。多言語テキスト編集に関して Emacs が備える潜在力を概観した後は，実際に「ロシア語，古典ギリシア語，古典ヘブライ語（『旧約聖書』ヘブライ語）」によるテキストの入力を試してみよう。

ドイツ語・フランス語テキスト入力に latin-prefix インプットメソッドを用いた様に，ロシア語入力には

cyrillic-yawerty, 古典ギリシア語入力には greek-babel, 古典ヘブライ語入力には hebrew-biblical-sil を使って，図 8（26 ページ）の「非ラテン文字アルファベットによる多言語テキスト」を実際に入力してみよ。ロシア語はトルストイ『アンナ・カレーニナ』(「幸福な家庭はどれも似たものだが，不幸な家庭はいずれもそれぞれに不幸なものである」）から，古典ギリシア語はプルタルコス『対比列伝』（もし余がアレクサンドロスでなかったら，余はディオゲネスでありたい）から，古典ヘブライ語は『旧約聖書』創世記（はじめに神は天と地とを創造された）から取った。

1. Лев Толстой: Анна Каренина

Все счастливые семьи похожи друг на друга, каждая несчастливая семья несчастлива по-своему.

2 Πλουτάρχου Βίοι Παράλληλοι

ἀλλὰ μὴν ἐγώ, εἰ μὴ Ἀλέξανδρος ἤμην, Διογένης ἂν ἤμην.

3 בְּרֵאשִׁית

בְּרֵאשִׁית בָּרָא אֱלֹהִים אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃

図 8: 非ラテン文字多言語テキスト

ヘブライ語は right-to-left 表記の言語である。Emacs ではヘブライ語入力時に語がどのように入力されるか，また，カーソルはどのように移動するか，よく観察せよ。なお，ロシア語，古典ギリシア語，古典ヘブライ語，・・・，を含む多言語テキストは，TEX を使えば各言語の正書法を反映させて自由自在に組版（typeset）できる。実際，図 8（26 ページ）は TEX を用いて組版処理したものである。

frame を左右に 2 分割し，左 window に各インプットメソッドの説明 buffer を表示させつつ，右 window で作業する，というように以前学んだことを応用しつつ，工夫して作業せよ。

Cursor 位置文字情報取得

cursor 位置にある「文字 (character)」の正確な情報を minibuffer に表示させるための以下のコマンドを学べ。

- C-x = (what-cursor-position)

前置引数（Prefix Argument）である C-u を前打しておけば，より詳細な文字情報が別 window に現れる。

当該文字の正式名称（英語），10 進数，8 進数，16 進数での文字コード（Character Code），キャラクタセット中のコードポイント，モニタ表示時の使用フォント，等々といった正確な情報を得られるが，特に対象文字の Unicode 16 進数コードが分かっていれば，Emacs では C-x 8 RET の後に 16 進数コードを直接打つことでも，全 Unicode 文字を入力できるようになっている（当該文字のモニタ上での表示の可否は，対応フォントがシステムに組み込まれているかどうかに依存する)。

第 13 講演習

演習用ファイル： multilang_ru-gr-he.pdf/tex

1. 図 8（26 ページ）にある「ロシア語，古典ギリシア語，古典ヘブライ語」テキストを入力せよ
2. フランス語引用符 « Français » の正式名称および 16 進数コードを調べ，これを入力せよ
3. ドイツ語引用符 „Deutsch“ の正式名称および 16 進数コードを調べ，これを入力せよ
4. 英語引用符 “English” の正式名称および 16 進数コードを調べ，これを入力せよ
5. latin-prefix でフランス語引用符，ドイツ語引用符，英語引用符はそれぞれ入力できるか
6. rfc1345 インプットメソッドを検証せよ
7. rfc1345 を使って，英語・ドイツ語・フランス語それぞれの引用符を入力してみよ
8. 長い s, つまり，ſ を rfc1345 を用いて入力してみよ（Wachſtube vs. Wachstube）

第 14 講

文字コード・改行コードの変換

「文字コード Character Code」とは，コンピュータで文字（Characer）を扱うために個々の文字に割り振られた固有の番号（Code）をいう。

その際，どのような文字をどれだけ取り扱うのか，予め「文字集合 Character Set」を定めておく必要がある。そして，文字集合内の各文字には，重複が起きないよう番号を付与せねばならない，つまり対応が一意（unique）となるように「文字符号化方式 Characer Encoding Scheme」を決めねばならない，が，この文字符号化方式は複数存在する。

実際，例えば，日本語表記，地名，人名等に用いられ

る文字集合を規定する日本工業規格の一つに JIS X 0208 があるが，この文字集合に対しては歴史的な事情から，ISO-2022-JP（JIS コードとも呼ばれ，古い電子メール規格ではこのコードが使われる），EUC-JP（Extended Unix Code for Japanese, UNIX 系のコンピュータで用いられていた），Shift_JIS（日本語 Windows 標準の文字コードだった）といった異なる文字符号化方式が存在しており，アプリケーションソフトウェア側で文字コード（文字集合および文字符号化方式）を正しく処理できないと，いわゆる「文字化け」を引き起こす。さらに，多様な文字コードの存在は，「テキストデータの互換性・同一性」という観点から，時に，少なくない問題を出来させることがある。

近年では，こうした（特に）互換性問題を回避するために，世界中で使われている多くの文字を単一の文字集合として包含した Unicode という規格が普及し，多くの OS やアプリケーションで用いられるようになっている。そして，Unicode への文字符号化方式も沢山存在する中で，ASCII（American Standard Code for Information Interchange）文字コードの上位互換であることから de facto standard として用いられるのが UTF-8（Unicode Transformation Format 8）である。

Unicode では，文字集合中の文字を符号位置（Code Point）で，U+ の後に 16 進数コードを付加して表す。BMP（Basic Multilingual Plane, 基本多言語面）内の符号位置は U+0000 から U+FFFF の 4 桁で表すことができ，SMP（Supplementary Multilingual Plane, 補助多言語面）以降は 5 桁または 6 桁で表される。

今後，我々が新たに「多言語（混在）テキスト」を作成する場合は，文字集合として Unicode, 文字符号化方式として UTF-8 を選択すれば良いが，現実の仕事においては，古い文字コードによって作成されたテキストに遭遇することも想定される。

また，改行コード（Newline, Line Ending, EOL: End of Line, Line Break とも。改行を表す制御文字）にも注意を払う必要がある。改行コードには CR: Carriage Return と LF: Line Feed の 2 種類があり，Linux や Mac など UNIX 系のシステムでは LF が，Windows では CR+LF が用いられる。なお，Mac では Mac OS の version 9 までは CR が用いられていた。Perl や PHP で書かれたプログラムの改行コードが（LF ではなく）CR+LF であったがために，Linux 上の Web アプリケーションが動作しない，ということも有り得る。

buffer に読み込まれるテキストの文字コード・改行コード情報は，mode line に表示される。U = Unicode, S = Shift_JIS, J = JIS (ISO-2022-JP), E = EUC-JP, 1 = Latin-1 (ISO-8859-1), (Unix) or : = LF, (DOS) or \ = CR+LF, (Mac) = CR.

文字コード・改行コードに関する以下のコマンドを学べ。

- ユーザが指定する文字コード・改行コードへ buffer のテキストを変換： C-x RET f *coding* RET (set-buffer-file-coding-system)
- ユーザが指定する文字コード・改行コードで buffer のテキストを開き直す：C-x RET r *coding* RET (revert-buffer-with-coding-system)

LF と CR+LF が混在してしまっている（ある意味では「壊れた」）テキストを Emacs で開くと，CR 部分が ^M と表示されるが，こうした場合は，一括置換コマンドを用いて ^M を除去すれば良い。^M は C-q C-M で入力できる。

発展： シェルコマンドを用いた再帰的一括コード変換

以下のようなシェルコマンド（実際は 1 行で入力）を実行すれば，current-/sub-/subsubdirectory にまたがる全てのテキストファイルの文字コード・改行コードを UTF-8 (LF) に一括変換できる。この技法も Mac, Linux であればそのまま使えるが，Windows の場合は，Windows Subsystem for Linux 上で Bash が使えるようになっていなければならない。

```
find . -name "*.txt" | xargs
nkf -w -Lu --in-place
```

nkf (Network Kanji Filter) コマンドオプションの第 1 引数（文字コード指定）には j (ISO-2022-JP), s (Shift_JIS), e (EUC-JP), w (UTF-8) を，第 2 引数（改行コード指定）には Lu (LF), Lw (CR+LF), Lm (CR) を指定できる。nkf の詳しい使い方について調べる場合は，ターミナルで man nkf と打つ。f キーで画面を下へ，b キーで上へスクロールできる。q キーを叩けばマニュアルを抜けられる。

Google Translate

EURO ICT Emacs では，Google Translate パッケージを組み入れ，Emacs から直接 Google Translate サービスを利用できるようにしてある。Google Translate は，

2017 年 8 月 31 日現在で，103 もの言語のテキスト相互翻訳サービスを提供しているが，我々の Emacs では，「ギリシア語・ラテン語・アイスランド語・デンマーク語・スウェーデン語・英語・フランス語・日本語からドイツ語へ」，「ドイツ語から英語・フランス語・日本語へ」，という組み合わせを初期設定とした。設定ファイルに手を入れれば，もちろん，簡単に相互翻訳言語を増やせる。Google Translate を使う際は，当然のことであるが，端末がインターネットに接続されていなくてはならない。

Emacs の buffer に Google Translate を呼び出す以下のコマンドを学べ。

- C-c t (google-translate-smooth-translate)

翻訳したい箇所を region 指定し，C-c t と打てば [Latin German] Translate: のように source language と target language が提示されるので，C-n や C-p を続けて打鍵し，自分の求める source と target を選択し，RET を叩く。分割された window に buffer 全体が収まりきらず，buffer 先頭に記されている source/target languages が見えなくなってしまう場合は， M-< (beginning-of-buffer) を使って buffer を先頭までスクロールすると良い。

第 14 講演習

演習用ファイル： khm53_de/fr/ja/en_*.txt

1. 演習用ファイル UTF-8 (LF) ドイツ語テキストを Latin-1 (CR+LF) に変換し（ヒント：latin-1-dos），別名で保存せよ
2. 演習用ファイル Latin-1 (CR+LF) フランス語テキストを UTF-8 (LF) に変換し（ヒント：utf-8-unix），別名で保存せよ
3. 演習用ファイル EUC-JP (LF) 日本語テキストを UTF-8 (LF) に変換し（ヒント： utf-8-unix），別名で保存せよ
4. 演習用ファイル Shift_JIS (CR+LF) 日本語テキストを JIS (LF) に変換し（ヒント： iso-2022-jp-unix），別名で保存せよ
5. khm53_ja_utf8_unix.txt を Shift_JIS (CR+LF) で開き直してみよ（ヒント： sjis-dos），その後，buffer を元の UTF-8 (LF) に戻せ（ヒント： utf-8-unix）
6. khm53_fr_latin1_dos.txt を UTF-8 (LF) で開き直してみよ（ヒント： utf-8-unix），その後，buffer を元の Latin-1 (CR+LF) に戻せ（ヒント： latin-1-dos）
7. khm53_de_utf8_unix.txt を Latin-1 (CR+LF) で開き直してみよ（ヒント： latin-1-dos），その後，buffer を元の UTF-8 (LF) に戻せ（ヒント： utf-8-unix）
8. 適当な作業ディレクトリ・サブディレクトリ・サブサブディレクトリ内に khm53_ja_sjis_dos.txt をそれぞれ 100 個（ファイル名箇所は連番を振る）ずつコピーし，これら全ての文字コード・改行コードを UTF-8 (LF) に変換せよ
9. Google Translate を用いて khm53_en_utf8_unix.txt をドイツ語，フランス語，日本語に翻訳させてみよ

第 15 講

バイナリとテキスト

文字データのみで構成されたファイルをテキストファイル（Text File）と呼び，この対概念がバイナリファイル（Binary File）となる。MS-Word や MS-Excel 等で作成されたファイルは，いずれもバイナリファイルである。

一般的にバイナリファイルは，それが作られたソフトウェア上という狭い範囲内にやり取りが限定されるのに対し，テキストファイルは原則的にどのようなソフトウェアでも取り扱うことができ，コンピュータ間での圧倒的な互換性を誇る。バイナリに比べファイルサイズも極めて小さいため，テキストエディタを使って中身を高速かつ高レベルに編集できるという点が，「テキスト主義 ICT」から見た場合の大きな利点となる。

一方で，テキストファイルは「文字情報だけしか取り扱えない」のであるから，テキストレベルで「文字を大きく，中央に配置し，色を付ける」といったような「視覚レイアウト」に関わる事柄を処理することはできない。もっとも，視覚レイアウト要素は，テキストにそれを実現するための「命令を書き込む」（markup）ことで如何様にも処理可能である。後で学ぶ TEX と HTML はこうしたマークアップ言語（Markup Language）処理系のシステムである[*10]。

[*10] ただし，最近では HTML においても TEX においても，文書作成時には主に「文書の論理構造」に注力し，視覚レイアウトは CSS や各種スタイルファイルに委ねる，言い換えると「論理デザインと視覚デザインを分離して文書作成を行う」という方式が主流となっている。

Weblio: インターネットでの串刺し検索

Emacs を文書作成ツールとして常用する際，Emacs 上で辞書検索もできるととても便利である。EPWING 形式の電子辞書が用意できれば，これを Emacs で利用するための辞書検索用プログラム Lookup と組み合わせることで，有償・無償を問わず Emacs から辞書を引くことができるようになる。図 9（29 ページ）は，「広辞苑第 6 版」（有償）と「和独辞典」（無償）をセットした Windows 上の Emacs から「有償」という語を引いている様子。

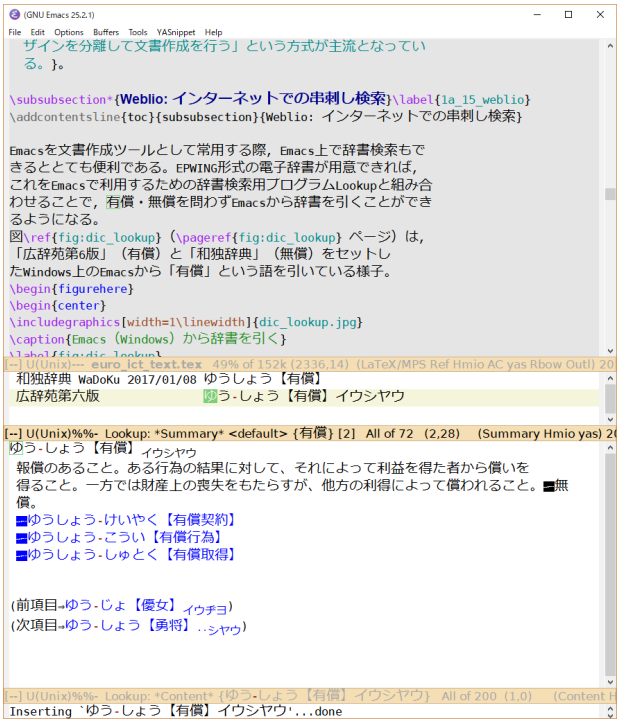


図 9: Emacs（Windows）から辞書を引く

Mac には標準で 10 を超える数の辞書を含む「辞書.app」が付いており，その中には国語辞典「スーパー大辞林」そして「ウィズダム英和／和英辞典」の他にも，Oxford Dictionary of English（英英辞典），Duden-Wissensnetz deutsche Sprache（独独辞典），Multidictionnaire de la langue française（仏仏辞典）等が入っている。

「Ctrl + Command + d」というキーボードショートカットを使えば，マウスポインタのある単語の意味をすぐに辞書.app で調べられるし，適当に設定すれば「トラックパッド上での 3 本指タップ」で同様の辞書引きを行える。無償の辞書（例えば「和独辞典」）を追加して，辞書.app を拡張することもできる。図 10（29 ページ）は，Mac 上の Emacs から辞書.app を使って「有償」という語を引いている様子。

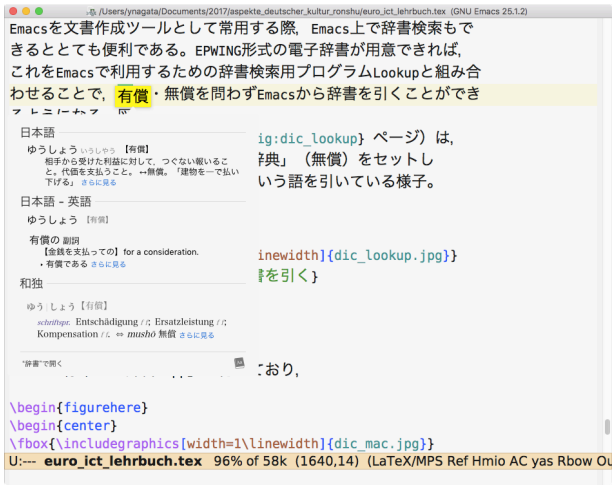


図 10: Emacs（Mac）から辞書を引く

このような，いわば standalone 状態で使用する「辞書」以外にも，Emacs ではネットワーク上の「辞書サイト」を利用することができる。EURO ICT Emacs では，統合型オンライン百科事典サイトである Weblio に接続して，650 種類を超える辞書・事典・用語集サイトの「串刺し検索」（一括検索）を可能としてくれるパッケージ eww-weblio.el を組み入れてある。

このパッケージは EWW を Weblio や Wikipedia 検索に特化させたものである。EWW (Emacs Web Wowser) とは，Emacs が内蔵する「テキスト閲覧に最適化させたブラウザ」である。buffer に読み込まれた Web ページは，「閲覧専用」であることを除けば，これまで学んできた Emacs の操作法と全く同様に扱うことができる。

Emacs から Weblio や Wikipedia, EWW を利用するための，以下のコマンドを学べ。

- EWW で Weblio に接続：(weblio)
- EWW で Wikipedia に接続：(wikipedia)
- EWW の起動：(eww)
- EWW マニュアルの参照： C-h i m eww

Weblio が提供するサービスは「日本語圏」向けであるから，英語「以外」の言語を検索する場合，Weblio を経由せず，最初から EWW で検索語を打ち込んだ方が，より早く目当ての情報にたどり着けるかもしれない。また，実際の検索時には，window を左右 2 分割しておくと作業がやり易いだろう。

weblio コマンドを使う場合は，前もって region 指定

しておいた語を検索することができる。EWW の詳しい使い方については，マニュアルを参照すること。さしあたっては，以下のコマンドを覚えておけばよいだろう。

- SPC/S-SPC: 画面のスクロール
- TAB/S-TAB: 次の／前のリンク
- q: EWW を閉じる
- g: 画面のリフレッシュ
- l: 前のページに戻る
- r: 次のページへ進む
- b: ブックマークする
- B: ブックマークの確認
- d: ダウンロードする
- &: 外部ブラウザを起動

第 15 講演習

演習用ファイル： text_word.docx, text_excel.xlsx, text.html, text.tex

1. text_word.docx, text_excel.xlsx, text.html, text.tex の中身を調べよ
2. text_word.docx, text_excel.xlsx, text.html, text.tex を「メモ帳」（Windows）や「CotEditor」（Mac）で開き，バイナリとテキストの違いを感得せよ
3. *.html, *.tex ファイルを開いたとき，Emacs の major mode は何になっているか
4. *.html, *.tex ファイルを「メモ帳」（Windows）や「テキストエディットまたは CotEditor」（Mac）等で開き，Emacs で開いた場合との差異を見出せ（Cf. Syntax Highlighting, Emacs Jargon では Font Lock）
5. text_word.docx について「ファイルの種類」を「書式なし (*.txt)」として保存せよ, その際,「ファイルの変換」を尋ねられるが，デフォルトのエンコード方法では「Windows Shift_JIS」および「CR+LF」で保存されてしまうため，将来的なファイル互換性を担保するためにも，「その他 Unicod (UTF-8)」および「改行の挿入 LF のみ」で保存する癖を身に付けると良い
6. 上記のようにしてできた text_word.txt を MS-Word で開き，新たに text_word_txt.docx というファイル名で保存せよ
7. text_word.docx をデフォルトのまま「書式なし (*.txt)」で保存してみよ，その際，ドイツ語・フランス語記述箇所で何が起きるか
8. text_excel.xlsx について「ファイルの種類」を「Unicode テキスト（*.txt)」として保存せよ
9. 上記のようにしてできた text_excel.txt を MS-Excel で開き，新たに text_excel_txt.xlsx というファイル名で保存せよ
10. text_excel.xlsx を「CSV（Comma-separated Values: カンマ区切り）(*.csv)」で保存してみよ，その際，ドイツ語・フランス語記述箇所で何が起きるか
11. Windows で単に Unicode と表記される場合の文字符号化方式は UTF-16 であり，また，改行コードは CR+LF となるので，必要に応じて意識的に UTF-8 (LF) に変換すること
12. text_excel.txt を「UTF-8 (LF)」に変換せよ
13. Emacs から Weblio および Wikipedia を利用し「Emacs」および「TeX」を検索してみよ
14. 直接 EWW を起動し「Froschkönig」そして「Le Petit Prince」を検索してみよ

ヨーロッパ学 ICT IB

第 1 講

Ispell/Aspell スペルチェッカ

英文，独文，仏文テキストを編集する際，入力した各単語が「正しく綴られているか」をチェックしてくれて，間違っている場合は「訂正候補」をも提示してくれるような機能を備えたソフトウェアがあると便利である。

EURO ICT Emacs にはこの目的のための「スペルチェッカ」（spell check をするソフトウェア）である Ispell/Aspell を組み込んである。Ispell/Aspell は major mode と連動し，それに相応しく動作するので，例えば，TEX 文書や HTML 文書の中に埋め込まれた「命令（Control Sequence)」や「タグ（Tag)」（これらも広義の「テキスト」と見做せる）は一切無視し，テキスト本文に現れる単語のみをチェックさせることができる（図 11, 31 ページ参照)。

Aspell は Ispell の後継スペルチェッカであり，一般論として，後継ソフトウェアは旧来のものを機能や使い勝手等において凌いでしまっているのが常であるが，Ispell には Aspell にはない利点もある。

それは，Ispell がスペルチェック時に Babel 式記法（Babel は TEX システムにおける多言語処理用パッケージ）をも識別してくれる，という点だ（図 12，31 ページ参照）。Babel を使えば ä ß の代わりに "a "s，é è ê の代わりに \'e \`e \^e と入力するだけで良い（つまり，日本語と混在させる場合，ウムラウトやアクサン記号を直接打ち込むのに比べ，入力がとても楽になる）が，このように入力されたテキストを TEX で処理すると，通常のウムラウトやエスツェット，アクサン記号が正確に出力される。まだまだ 7bit（つまり 128 文字）レベルで文字コードが定義されていた 1990 年代前半頃まで，ヨーロッパ語圏では ASCII 7bit コード表に存在しない文字をこのような方式で処理することが稀ではなかったのである。

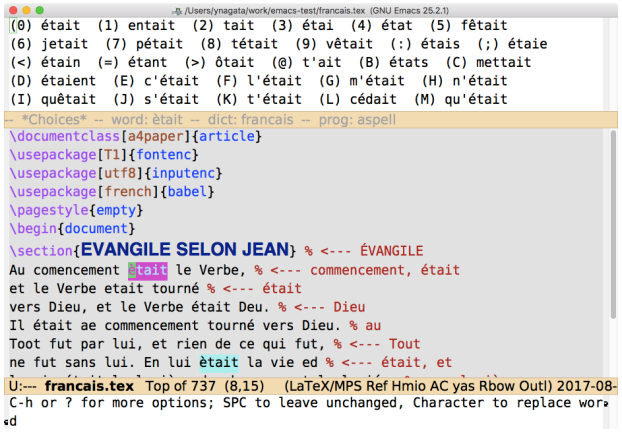


図 11: Aspell でフランス語の TEX ソースファイルをスペルチェック

なお，Emacs には (iso-gtex2iso) といったコマンドが内蔵されているので，Babel 式記法のウムラウトおよびエスツェット箇所を「後から一気に本来の文字へと変換する」ということも簡単にできる。

もちろん，現在のドイツ語圏やフランス語圏で Babel 式記法を用いてウムラウトやエスツェットやアクサン記号を入力する者は（その必要性がほぼ全くないので）ほとんどゼロであろう。そのため，今では，Babel 式記法に対応した Ispell 用辞書は配布されていない。

講義担当者は，かつて配布されていた Babel 式記法対応辞書（テキストファイル）から自分で Windows 用にコンパイルした「バイナリ」ファイルのみ所有しており，Mac 用のソースは持ち合わせていないので，残念ながら Mac で動作するバイナリを作成することができない。従って，Babel 式記法を含むテキストをスペルチェックできるのは Windows 上のみ，ということになる。

こうした事情から，Windows 教室では Ispell を，Mac 教室では Aspell を，それぞれデフォルトのスペルチェッカとして設定してある。もちろん，設定ファイル内の当該箇所を書き換えることで，Ispell と Aspell はいつでも交換できる。Ispell/Aspell のコマンドは共通である。

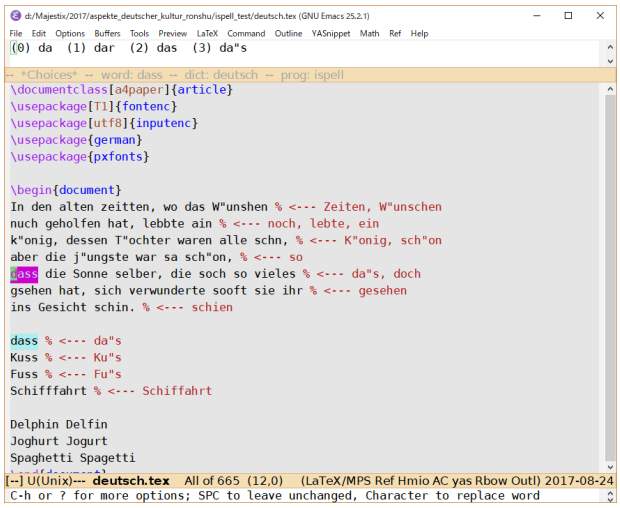


図 12: Ispell で Babel 式記法のドイツ語ソースファイルを旧正書法でスペルチェック

以下の Ispell/Aspell コマンドを学べ。

- スペルチェック用辞書の指定：(i-chan or ispell-change-dictionary)
- buffer に対してスペルチェック：(isp-b or ispell-buffer)
- region に対してスペルチェック：(isp-r or ispell-region)

Ispell (Windows) 用の辞書としては，american（米語＝アメリカ英語），english（英語＝イギリス英語），deutsch（旧正書法ドイツ語，7bit の Babel 式記法），deutsch8（旧正書法ドイツ語），german（新正書法ドイツ語，7bit の Babel 式記法），german8（新正書法ドイツ語），francais（フランス語），francais-tex（フランス語，7bit の Babel 式記法）が使える。

Aspell (Mac) 用の辞書としては，american（米語），british（英語），deutsch-alt（旧正書法ドイツ語），german（新正書法ドイツ語），francais（フランス語）が使える。

buffer または region に対してスペルチェックをかける際には，チェックされた語について，さしあたり SPC: そのままにしておく，r(eplace): 明示的に訂正語を入力する，a(ccept): このセッションのみ暫定的に認める，i(nsert): 個人辞書に登録する，(e)x(it): スペルチェックを

抜ける，といったキーを覚えておけばよい。? を打ち込めば，さらに詳しい使用法に導かれるので，必要に応じてそちらを参照すること。

r および i を打鍵した場合は，minibuffer で Personal dictionary modified. Save? (y or n) と聞かれるので y と答えると，ユーザのホームディレクトリに .ispell_german (german の箇所は「使用した辞書名」) というファイルが自動作成され，その中に語が登録されていく。

第 1 講演習

演習用ファイル： khm53_de_fr_en_misspell.txt

1. 演習用ファイルを用いて，ドイツ語（旧正書法／新正書法とも）スペルチェックをせよ
2. ドイツ語旧正書法スペルチェックをせよ
3. ドイツ語新正書法スペルチェックをせよ
4. German TEX 式ドイツ語（旧正書法／新正書法とも）スペルチェックをせよ（Windows 端末のみ）
5. フランス語スペルチェックをせよ
6. French TEX 式フランス語スペルチェックをせよ（Windows 端末のみ）
7. 英語／米語スペルチェックをせよ

第 2 講

マルチカーソル編集

マルチカーソルとは何か。一言で言えば，cursor を分身させ，これらを同時に動かすことで，複数行にまたがって散在する文字列を一括編集したり，複数の行頭・行末に文字列を追加したり，そこから削除したり，といった作業を効率よく行うことを可能とする機能である。

もちろん，これまでに学んだ「置換・正規表現置換」や「矩形編集」等の機能を用いてもマルチカーソル編集と同じようなことができるが，マルチカーソルならではの利点として，目の前で編集作業を「リアルタイムに確認」できるということが挙げられる。場合によっては，こちらの方がはるかに使い勝手が良い，ということもある。

EURO ICT Emacs では，multiple-cursors.el と smartrep.el パッケージを組み込むことで，この機能を実現している。

マルチカーソル編集に関する以下のコマンドを学べ。

- region 内の全ての行に cursor 作成： C-M-g (mc/edit-lines)
- region 内で対話的にキーワードを入力して cursor 作成： C-M-x (mc/mark-all-in-region)
- region 指定したキーワードを連打により次々と選択： C-^ (mc/mark-next-like-this)

カーソルを分身させた後は，通常の編集作業となる。マルチカーソルを解除するには C-g を複数回打鍵する。マルチカーソル状態で C-v/M-v を打つと，通常状態における画面のスクロールではなく，マルチカーソルの point を行き来するようになる。

他にも C-^ の後で s を押すとスキップして次のキーワードに移動 (mc/skip-to-next-like-this), u を押すと選択を解除し一つ前の選択済みキーワードに戻る (mc/unmark-next-like-this), * を押すと buffer 内にある全てのキーワードを選択する，といった機能も備えているので，必要に応じて使い分けると良い。

発展： マルチカーソルで連番挿入

例えば，次のようなケースを想定してみよう。『グリム童話集』に収められている各メルヒェンのタイトルを 1 行ずつ入力したテキストファイル（全話だと 200 行以上）がある。後から，各行頭に KHM 001, KHM 002, KHM 003, …, といった「連番」を振る必要が生じた（KHM: Kinder- und Hausmärchen の initial word)。どうしたら単調な逐一入力を回避できるか。

KHM という文字列だけの入力であれば，先に解説したマルチカーソル機能の範囲内で処理できるが，併せて (my/mc/insert-numbers) コマンドを使うと「連番処理」も可能となるので，是非，一緒に覚えておきたい。

マルチカーソルを設置してから上記のコマンドを呼び出すと，まず Start from (default 0): と尋ねられる。これは「開始番号」なので，今の場合，「1」と打ち込む。次に Increment by (default 1): と「増分」を確認されるが，これはデフォルトのままで良いから単に RET を叩く。最後に Padding (%01d): と聞かれるので「%03d」等と入力する。数字は「桁」を，0 は「埋込文字」(例えば 7 を 3 桁表記すると 007 となる）をそれぞれ現わしている。0 を省略すると代わりに数字分だけ spaces が入る。「%d」とすれば「連番数のみ」が振られる（0 あるいは空白は埋め込まれない)。

なお，C-^ を行頭で連打しても window は連動スク

ロールしない。こういった場合は，C-x SPC で矩形編集モードに入って region 指定してから C-M-g した方がやり易い。

第 2 講演習

演習用ファイル： multiple-cursors_test.txt, insert-numbers.txt

1. multiple-cursors_test.txt の第 1 段落にある「全行」に cursor を設置せよ
2. その上で，各行頭に 5 つ分の spaces を挿入せよ
3. さらに全 cursors を行末に移せ
4. その上で，各行末に「% ここはコメント」といった文字列を挿入せよ
5. 「マルチカーソル」という語にマルチカーソルを設置せよ
6. その上で，各 cursor 間を移動せよ
7. さらに，「マルチカーソル」という文字列を「Multiple-Cursors」という文字列で置き換えたファイルを multiple-cursors_modified.txt という名前で保存せよ
8. insert-numbers.txt に記載されている各メルヒェンタイトルの前に「KHM 01: , KHM 02: , KHM 03: , …」を追記せよ
9. 同様に，今度は「KHM 1: , KHM 2: , KHM 3: , …」のようにせよ

なお，ファイルを C-x C-w で別名保存する際には，その前に C-g でマルチカーソルを解除しておくこと。

Homo Ludens 3: Tetris

「テトリス」とはロシア人のアレクセイ・レオニードヴィチ・パジトノフ（Алексей Леонидович Пажитнов, 1956–）がソビエト連邦時に開発したコンピュータゲームで，いわゆる「落ち物ゲーム」の代表である。

Emacs にもテトリスが付随する。

- テトリス起動： (tetris)

起動後は，<left> で「テトリミノ」（ブロックピース）を左へ，<right> で右へ移動させられる。テトリミノは <up> で時計回りに回転し，SPC を押せば即座に底部へ落下する。n で新ゲーム開始，q で終了，p で中断・再開となる。仕事や勉強に疲れたら，息抜きとしてやってみると良い。

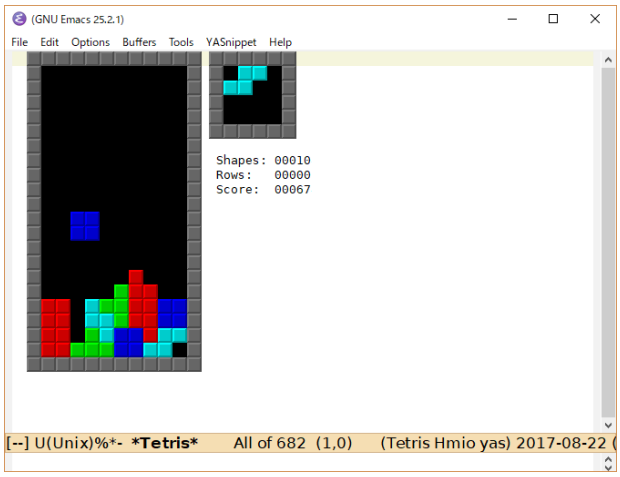


図 13: Tetris を楽しむ

第 3 講

Emacs は狭義にはテキストエディタであるが，広義では「ユーザにとっての作業環境そのもの」という形容もされる。本講からしばらくは，時間管理ツールあるいは個人情報管理ツールとしての Emacs の活用法を学ぶ。

カレンダー

カレンダーの起動法を学べ。このコマンドを実行すると，別 window に「先月，今月，来月」の 3 か月分カレンダーが表示される。なお，PC 教室（Windows, Mac とも）における初期設定は「ドイツ語月名・曜日名，月曜はじまり，土日色分け表示，日独祝祭日ハイライト表示」としてあるが，後（IB 第 5 講，36 ページ）で説明するように，この挙動は EURO ICT Emacs を立ち上げたまま動的に変更（ドイツ語，フランス語，英語，日本語環境）できるし，「EURO ICT Emacs & TEX のインストール法」Web では，最初から LG/LF 学生用に最適化した「ドイツ語／フランス語」環境を整えた「Emacs 初期設定ファイル」をダウンロードできるようにしてある。

- カレンダーを起動： (calendar)

カレンダーを起動すると，cursor が「今日」の上に移動し，mode line 上に「YYYY-MM-DD」形式で「今日の日付」が表示される。1 週間の表示を「日曜はじまり」にしたければ，M-x set-variable RET calendar-week-start-day RET 0 RET を実行し，カレンダーを再起動する。常に「日曜はじまり」としたければ，設定ファイルに記述さ

れている (setq calendar-week-start-day 1) とある行をコメントアウトして，設定ファイルを保存すれば良い。

次に，カレンダー内での移動コマンド等を学べ。

- 今日の日付へ移動： . (calendar-goto-today)
- 1 日後へ： C-f (calendar-forward-day)
- 1 日前へ： C-b (calendar-backward-day)
- 1 週間後へ： C-n (calendar-forward-week)
- 1 週間前へ： C-p (calendar-backward-week)
- 1 か月後へ： M-} (calendar-forward-month)
- 1 か月前へ： M-{ (calendar-backward-month)
- 1 年後へ： C-x] (calendar-forward-year)
- 1 年前へ： C-x [(calendar-backward-year)
- 週の始めへ： C-a (calendar-beginning-of-week)
- 週の終わりへ： C-e (calendar-end-of-week)
- 月の始めへ： M-a (calendar-beginning-of-month)
- 月の終わりへ： M-e (calendar-end-of-month)
- 年の始めへ： M-< (calendar-beginning-of-year)
- 年の終わりへ： M-> (calendar-end-of-year)
- 指定日へ： g d (calendar-goto-date)
- 1 か月先へスクロール: C-x > (calendar-scroll-left)
- 1 か月前へスクロール： C-x < (calendar-scroll-right)
- 3 か月先へスクロール： C-v (calendar-scroll-left-three-months)
- 3 か月前へスクロール： M-v (calendar-scroll-right-three-months)
- 別 window をスクロール： SPC (scroll-other-window)
- 別 window にを逆方向へスクロール： S-SPC (scroll-other-window-down)
- カレンダー buffer を再表示： C-c C-l (calendar-redraw)
- region 内の日数をカウントする： M-= (calendar-count-days-region)
- カレンダーから抜ける： q (calendar-exit)

「指定日へ」移動する際，対話式に Month name が尋ねられるが，このとき「当該言語環境」での「月名」を打ち込まねばならない（例えば，日本語：3 月；ドイツ語：März; フランス語：Mars)。もっとも，この場合でも TAB キーによる補完入力が効くので，入力はさほど面倒ではない。

C-c C-l コマンドは，カレンダー buffer の縦幅が意図せず広くなってしまった場合の再描画に用いると良い。C-x - (shrink-window-if-larger-than-buffer) を使っても良い。

region 内の日数を数える M-= コマンドは，mark および point で指定した日を「含んだ」日数を表示するので注意すること。

第 3 講演習

1. 上で取り上げた Calendar 関連コマンドを全て検証せよ
2. ヴォルフガング・アマデウス・モーツァルトが生まれた日（1756 年 1 月 27 日）は，今日から数えて何日前となるか計算せよ

第 4 講

日出入時刻，朔弦望

EURO ICT Emacs では「福岡大学文系センター棟」を観測地点（北緯 33 度 32 分 52 秒・東経 130 度 21 分 50 秒）に設定してあり，この地点における日出入時刻や朔弦望（新月，上弦，満月，下弦）を簡単に調べることができる。以下のコマンドを学べ。

- カレンダー内 cursor 位置の日付の日出入時刻を表示： S (calendar-sunrise-sunset)
- （カレンダーの外からでも）今日の日出入時刻を表示： (sunrise-sunset)
- カレンダー buffer に表示されている 3 か月間における朔弦望の日付および時刻の一覧を表示： M (calendar-lunar-phases)
- （カレンダーの外からでも）今月を中央月とする前後 1 か月の朔弦望の日付および時刻の一覧を表示： (lunar-phases)

前置キー C-u を打ってから (sunrise-sunset) コマンドを発行すると，指定した日付の日出入時刻を表示できる。C-u を 2 回打ってから同コマンドを発行すると，世界中の任意の場所（緯度・経度は 10 進数で入力する）での任意の日付の日出入時刻を表示できる。ただし，後者の場合，「協定世界時」からのズレを「分単位」で入力する必要がある。

前置キー C-u を打ってから (lunar-phases) コマンドを発行すると，指定する月と年を中央月とする前後 1 か月の朔弦望の日付および時刻の一覧を表示できる。

閏年

EURO ICT Emacs では，指定する年が閏年であるかどうかを簡単にチェックできる関数を定義してある。以下のコマンドを学べ。

- 任意の年が閏年かどうかをチェック：C-c y (leap-year-check)

上記コマンドを打ち込むと minibuffer で Year: と尋ねられるので，調べたい年を西暦で入力する。答えは「英語で」帰ってくるようにプログラムしてある。

ダイアリー機能： 誕生日，授業時間割

Emacs はカレンダーと連携するダイアリーをデフォルトで包含しており，特定の日付に対しメモ等を記入することができる。EURO ICT Emacs では，しかし，デフォルトのダイアリーよりも遥かに使い勝手の良い高機能パッケージ Org を組み込んであるので，ダイアリー機能を用いる場合も Org を使うこととする。

そもそも「情報」は，時間に関する概念に従って分類すれば，ひとまず「フロー情報」と「ストック情報」に分けて考えることができる。例えば，特定の日付に様々な形を取る（締切期限を設ける，締切間近あるいは超過時に警告通知を受け取る，等々）任意のメモを書き込む（Org では Capture という術語で呼ばれる）という作業では「フロー情報」が扱われることになるが，このやり方については，後（IB 第 9 講，41 ページ）で解説する。本講では，一種のデータベース的な性質をも持ち合わせている「ストック情報」に属する「誕生日，授業時間割」を Org でどのように取り扱えるのか，を見る。

Org では独自のフォーマットに従った「階層構造」を取るテキストを扱うので，ダイアリー機能を用いる場合でもこのフォーマットに倣い，第 1 階層に「誕生日，授業時間割，西暦年号」を設定する。

EURO ICT Emacs においては，ストック・フローのいずれであっても，ダイアリー関連の情報データは全て一元的に ~/org/Memo.org ファイル内に書き込むことにしてある。以下の書式に従い，「誕生日，授業時間割，西暦年号」を設定せよ。~/org/Memo.org のパスは Windows では C:\home\org\Memo.org, Mac では /Users/ynagata/org/Memo.org となる（ynagata 部分は Mac におけるユーザ名）。

```
* Birthday
%%(diary-anniversary 1685 03 31) J. S.
   Bach's birthday: %d years old
%%(diary-anniversary 1755 04 01) J. A.
   Brillat-Savarin's birthday: %d years old

* Class Schedule
** ヨーロッパ学 ICT IB
*** <2017-09-14 Do 13:00-14:30>
     ヨーロッパ学 ICT IB
*** <2017-09-21 Do 13:00-14:30>
     ヨーロッパ学 ICT IB
*** <2017-09-28 Do 13:00-14:30>
     ヨーロッパ学 ICT IB
*** <2017-10-05 Do 13:00-14:30>
     ヨーロッパ学 ICT IB
*** <2017-10-12 Do 13:00-14:30>
     ヨーロッパ学 ICT IB

* 2017
```

%% や * で始まる行は，実際は改行せず「1 行で」記述する。(diary-anniversary …) フォーマットの中は西暦（グレゴリオ暦）で YYYY MM DD 形式で書く。同様にして「誕生日」以外の記念日もエントリできる。%d の部分は Emacs のダイアリが持つ「S 式（S-expression）」による記法で，ここは Org により自動展開され出力される（後述）。

「時間割」のエントリにおけるフォーマットは見ての通りであるが，「年月日」にハイフンを入れること，「曜日」箇所は「当該言語で」記述することに留意する。「月，火，水，木，金，土，日」はドイツ語，フランス語ではそれぞれ „Mo, Di, Mi, Do, Fr, Sa, So“, « Lu, Ma, Me, Je, Ve, Sa, Di » とする。この部分はカレンダーの初期設定言語と揃える必要があるので注意すること。

以上の設定を済ませ，EURO ICT Emacs を再起動すると，Week-agenda buffer を閲覧できるようになる。f キーや b キーを押下し buffer を行き来すると，ドイツ語環境では例えば以下のような出力を確認できる。

```
Donnerstag 14 September 2017
 Memo: 13:00-14:30 ヨーロッパ学 ICT IB

Samstag 23 September 2017
 Diary: 秋分の日

Samstag 31 März 2018
 Memo: J. S. Bach's birthday: 333 years old
Sonntag 1 April 2018
 Memo: J. A. Brillat-Savarin's birthday:
```

```
      263 years old
Diary: Ostersonntag
```

何か別のファイルを開いている場合は，C-c a a (org-agenda-list) を打てば 別 window に Week-agenda buffer を表示させることができる。

第 4 講演習

1. 上で取り上げた日出入時刻と朔弦望に関する全コマンドを検証せよ
2. 今日の日出入時刻を調べよ
3. 次に満月を拝めるのはいつか
4. 次の新月，上弦・下弦の月はそれぞれいつか
5. 今年は閏年か
6. 次の閏年はいつか
7. Memo.org を編集し「ヨーロッパ学 ICT IB」の時間割エントリを完成させよ
8. 同様に「ヨーロッパ学 ICT IB」以外の授業科目も全て入力せよ
9. Week-agenda buffer で「誕生日」や「授業科目」を確認せよ

第 5 講

カレンダーの多言語表示

講義担当者はドイツ語を専門とするので，自分の Emacs 環境におけるカレンダーでは「ドイツ語による月名・曜日名」が表示されるように設定してある。もちろん，連動して「ドイツの祝祭日」もカレンダー上ですぐ参照できるように german-holidays.el パッケージも組み込んである。

しかし，日本国内で Emacs を使う場合，当然「日本の祝祭日」も併せて参照できる方が使い勝手が良いため，japanese-holidays.el パッケージも同時に読み込ませている。EURO ICT Emacs でも，この設定をデフォルトとした。

つまり，EURO ICT Emacs のカレンダーの初期設定では「月名・曜日名はドイツ語表示，ドイツおよび日本の祝祭日はピンク色でハイライト表示」されている。もっとも，この設定は Emacs を立ち上げたまま「動的」に変更できる。

以下のコマンドを学べ。「フランス語」または「日本語」を選択すると「月名・曜日名がフランス語または日本語表記となり，フランスまたは日本の祝祭日のみピンク色でハイライト表示される」ようになる。「英語」では「月名・曜日名が英語表記となり，ドイツ・フランス・日本の祝祭日の全てがピンク色でハイライト表示される」設定となっている「ドイツ語」は EURO ICT Emacs におけるデフォルトのカレンダー表示と同一。

なお，「設定ファイル」(Windows: init.el; Mac: .emacs.el) に記述してある holiday-german-holidays という箇所を holiday-german-BW-holidays のように書き換えれば，「バーデン・ヴュルテンベルク州のみ」の祝祭日を表示させることもできる。同様に，BW の部分を「BY（バイエルン州），BE（ベルリン州），BB（ブランデンブルク州），HB（ブレーメン特別市），HH（ハンブルク特別市），HE（ヘッセン州），MV（メクレンブルク・フォアポメルン州），NI (ニーダーザクセン州），NW (ノルトライン・ヴェストファーレン州），RP（ラインラント・プファルツ州），SL（ザールラント州），SN（ザクセン州），ST（ザクセン・アンハルト州），SH（シュレスヴィヒ・ホルシュタイン州），TH（テューリンゲン州）」と書き換えることで，各州に特化させた祝祭日表記に変更できる。

- カレンダーの言語環境を設定： C-c k (set-locale-with-calendar)

コマンドを打ち込むと locale: と尋ねられるので, Windows では French, Japanese, English, German と，Mac では fr_FR.UTF-8, ja_JP.UTF-8, en_US.UTF-8, de_DE.UTF-8 と答える。入力の際には，もちろん，TAB キーによる補完が効く。カレンダー buffer の window の高さがおかしくなった場合は，C-c C-l を打つと良い。

以上を踏まえた上で，祝祭日に関する以下のコマンドを学べ。

- cursor のある日付の祝祭日を表示： h (calendar-cursor-holidays)
- 表示中の 3 か月に含まれる祝祭日を別 window 表示： a (calendar-list-holidays)
- （カレンダーの外からでも）今月を中心とした 3 か月に含まれる祝祭日を別 window 表示： (holidays)
- （カレンダーの外からでも）指定した範囲の年の祝祭日を別 window に表示： (list-holidays)

C-u M-x holidays のように前置キーを付すと任意の

中央月（年と月名を答えてやる）を指定できる。list-holidays コマンドでは始年と終年を指定する。その際 List (TAB for choices): と聞かれるので All と答えてやれば良い。

他暦変換

Emacs のカレンダーは，現在世界中で広く用いられている「グレゴリオ暦」(「新暦」とも）で，西暦 1 年 1 月 1 日以降のみ（紀元前は不可），グレゴリオ暦が存在しなかった時代についてもグレゴリオ暦に換算した暦で，表示される。もっとも，グレゴリオ暦がそれまでヨーロッパで用いられていた「ユリウス暦」（グレゴリオ暦との対比で「旧暦」とも）に置き換わり世界的に普及したのは 20 世紀の初頭になってからである。

Emacs はグレゴリオ暦以外のカレンダーを表示することはできないが，指定した日付と他の暦の日付とを交互に変換することはできる。「ISO 商用暦」，「ユリウス暦」，「天文日（ユリウス日）」，「ヘブライ暦」，「イスラム暦」，「フランス革命暦」，「マヤ暦」，「コプト暦」，「ペルシャ暦」，「旧暦（中国暦）」，「バハイ暦」を取り扱えるが，本講義では「ユリウス暦とフランス革命歴」に関するコマンドのみを扱う。

「ユリウス暦」とは，共和政ローマ期のカエサルにより紀元前 45 年 1 月 1 日から施行された（1 年を 365.25 日とする）太陽暦。共和政ローマおよび帝政ローマ時代に用いられ，西ローマ帝国滅亡（紀元 476 年）後もヨーロッパで広く使用されたが，1582 年にはローマ教皇グレゴリウス 13 世が「太陽年」（回帰年とも）との誤差を修正した「グレゴリオ暦」を制定し，現代では「グレゴリオ暦」が世界中で広く用いられている。

「フランス革命歴」（Calendrier révolutionnaire Français,「共和暦」Calendrier républicain とも）とは，フランス革命期にフランスで用いられた 10 進法に基づく独自の暦法。12 年ほど使われただけで，間もなく廃止された。

他暦変換に関する以下のコマンドを学べ。

- cursor のある日付をユリウス暦に変換表示： p j (calendar-julian-print-date)
- cursor のある日付をフランス革命歴に変換表示： p f (calendar-french-print-date)
- ユリウス暦で指定した日付をグレゴリオ暦で確認： g j (calendar-julian-goto-date)
- フランス革命暦で指定した日付をグレゴリオ暦で確認： g f (calendar-french-goto-date)

第 5 講演習

1. 今年および来年の「海の日」は何月何日か
2. 今年および来年の „Ostersonntag“ は何月何日か
3. 今年および来年の « Ascension » は何月何日か
4. 音楽の父と称えられるヨハン・ゼバスティアン・バッハはユリウス暦 1685 年 3 月 21 日に誕生したが，グレゴリオ暦では何月何日となるか
5. フランス革命歴 II 年熱月（テルミドール）9 日に起きたフランス革命時のジャコバン派独裁に対立する勢力によるクーデター「テルミドール 9 日の反動」は，グレゴリオ暦では何月何日となるか
6. ナポレオンが総裁政府を倒した軍事クーデターはグレゴリオ暦 1799 年 11 月 9 日に起きた（フランス革命はここに終わったとされる）が，この日はフランス革命歴ではいつとなるか

calendar-french-goto-date コマンドを用いる際は，Année de la Révolution: そして Mois ou Sansculottide: 最後に Jour (1–30): とフランス語で尋ねられるので正しく回答すること（TAB キーによる補完も効く）。

第 6 講

アウトライン編集

ダイアリー機能の箇所でも言及した Org は Emacs 標準添付の多機能パッケージである。実際，Org はアウトライン編集，メモ管理，スケジュール管理，プロジェクト管理，HTML/XML/LaTeX フォーマット変換，表作成・表計算，グラフ作成，ハイパーリンク・インライン画像処理，スマートフォンとの連携，等々に用いることができる major mode で，これら全ての機能を「テキストファイル」システムに基づいて高速に動作させている点が特筆できる。

なお，Org の Web ページでは高らかに次のように謳われている：“Org mode for Emacs—Your Life in Plain Text”. (Org mode is for keeping notes, maintaining TODO lists, planning projects, and authoring documents with a fast and effective plain-text system.)

さて，Org が備える全機能を検証することがヨーロッパ学 ICT の目的ではない。「テキスト主義 ICT」にとっ

て特に役立つと考えられる機能に絞って紹介したい。

先ずは，「アウトライン編集」を取り上げる。

レポートであれ論文であれ，あるいはまたメモのレベルであれ，我々が文書を起草する際，その文書を「構造化」しておくと作業能率が格段に上がることは言うまでもない。ここで言う構造化とは，文書内容や項目を「階層化」することを指す。文書の構造化・階層化は文書のみならず，思考そのものを整理することにも役立つ。

Org では階層の違いを「アスタリスク * の個数」で表し，* で始まる行を「見出し（headline）」と呼ぶ。

```
* 第 1 階層の見出し
  ここにテキストがあっても良い。
** 第 2 階層の見出し
   ここにもテキストがあって良い。
*** 第 3 階層の見出し
    ここにテキスト。
*** 別の第 3 階層の見出し
    ここにテキスト。
* 別の第 1 階層の見出し
```

「見出し」を付けた Org 文書のイメージは，上の例で簡単に理解できると思う。こうしておけば「見出し」以下を自由に畳んだり（隠したり）展開したりできるのが Org を用いる利点の一つである。この機能の実現には単に TAB または S-TAB キーを用いるだけである。しかも Org は見出し行に色（階層によって異なる）まで付けてくれる（Syntax Highlighting）。

以下のコマンドを学べ。「見出し」を「ツリー」（Tree）構造と見做している点にも注意。見出しにぶら下がる見出しは「サブツリー」（Subtree）となる。

- cursor のある見出し以下を順々に展開し，また畳む： TAB (org-cycle)
- buffer にある全ての見出し以下を順々に展開し，また畳む： S-TAB (org-global-cycle)

次に「見出し」間の移動に用いる以下のコマンドを学べ。

- 次の見出しへ： C-c C-n (outline-next-visible-heading)
- 前の見出しへ： C-c C-p (outline-previous-visible-heading)
- 同階層の次の見出しへ： C-c C-f (org-forward-heading-same-level)
- 同階層の前の見出しへ： C-c C-b (org-backward-heading-same-level)
- 1 つ上の階層の見出しへ： C-c C-u (outline-up-heading)

最後にアウトラインの「構造編集」に関する以下のコマンドを学べ。

- 同レベルの見出しをすぐ下の行に挿入： M-RET (org-insert-heading)
- 見出しに中身（Body）があれば，すぐ下の行にではなく，サブツリーを含む中身の後に同レベルの見出しを挿入： C-RET (org-insert-heading-respect-content)
- TODO エントリを含む同レベルの見出しをすぐ下の行に挿入： M-S-RET (org-insert-todo-heading)
- TODO エントリを含む同レベルの見出しをサブツリーを含む中身の後に挿入： C-S-RET (org-insert-todo-heading-respect-content)
- 見出しの中身がまだ無い場合，見出しレベルを変更する： TAB (org-cycle)
- 見出しの昇格（promote）： M-<left> (org-do-promote)
- 見出しの降格（demote）： M-<right> (org-do-demote)
- サブツリーを伴う見出しの昇格： M-S-<left> (org-promote-subtree)
- サブツリーを伴う見出しの降格： M-S-<right> (org-demote-subtree)
- 見出しをサブツリーごと同レベルの見出しの前に移動： M-<up> (org-move-subtree-up)
- 見出しをサブツリーごと同レベルの見出しの後に移動： M-<down> (org-move-subtree-down)
- cursor のある見出し（およびサブツリー）以外は不可視とする： C-x n s (org-narrow-to-subtree)
- 不可視となっている他の見出し（およびサブツリー）を元に戻す： C-x n w (widen)

M-RET/M-S-RET を打つ際は cursor を headline 行末に置く。cursor が headline 行頭にあると「前行に」同レベル見出しが挿入され，見出し「テキスト」の上にあれば cursor 以下のテキストが改行されて新たな同レベル見出しとなる。C-RET/C-S-RET では cursor は headline 行頭以外に置けば良い。行頭にあれば前行に同レベル見出しが挿入される。TODO については後述（IB 第 8 講，

40 ページ）。

見出しの昇・降格の際は，cursor は headline 上のどこにあっても良い。サブツリーの前後移動も同様。

C-x n w (org-narrow-to-subtree) コマンドで不可視となった（Narrowing）部分は元ファイルから消去された訳ではないので混乱しないこと。

Org が備える充実した「アウトライン編集」機能は，構造化された文書（論文やレポート）の草稿執筆およびアイデアの錬成に際し，強力なツールとなる。EURO ICT Emacs では，拡張子 .org 以外に，通常の .txt でも Org が major mode となるように設定してある。

第 6 講演習

演習用ファイル： fairy-tales.org

1. 演習用ファイル fairy-tales.org を用いて，上で説明したアウトライン編集に関するコマンドを全て検証せよ
2. EURO ICT Emacs で fairy-tales.org を開き，C-c C-e h o と叩くと何が起こるか
3. 同様に C-c C-e l o と打ち込むとどうなるか

Org 文書で /string/ と記しておけば, string という文字列がイタリック体 *(Italic Shape)* となる。同様に *string* で太字体 **(Boldface Series)**, _string_ でアンダーラインが施される。

なお，本講演習の 2, 3 は Org のエクスポート機能を使って元ファイルを HTML 文書および LaTeX & PDF 文書に変換してオープンするコマンドである（45 ページ参照）。

第 7 講

リスト，チェックボックス

Org では「リスト」も簡単に作成，編集できる。リストは項目の下にサブ項目を作るといった「入れ子」構造にすることもできる。同レベルのリストは同じインデント（字下げ）を持つ必要がある。逆に言えば，同じインデント量を持つ項目は同レベルのリストと見做される。最初にリストを作成する場合は C-j を打つと良い。

「番号なし」リスト（Unorderd Lists）は - もしくは + の後に one space 置いて，「番号付き」リスト（Ordered Lists）は 1. もしくは 1) の後に one space 置いて作成する。この際，各記号の後に one space を置き [ ]（ブラケットの中も one space）と書くと，これがチェックボックスとなる。また，リストおよびそのリストを含む見出しの末尾に [/] あるいは [%] と記しておけば，前者は項目数を母数とする分数形式で，後者はパーセンテージで，「達成度」が自動表示される。リストが含まれる見出しを超えての達成度表示はできない。なお，達成度に関する数字は完遂までは「赤」色で，完遂すれば「緑」色で表示される。

リストおよびチェックボックス作成に関する以下のコマンドを学べ。

- 適当なインデントを設ける： C-j (org-return-indent)
- リストを畳む・展開する： TAB (org-cycle)
- 新たにリスト項目を挿入する： M-RET (org-insert-heading)
- チェックボックス付きのリスト項目を新たに挿入する： M-S-RET (org-insert-todo-heading)
- リスト項目をサブ項目ごと前に移動： M-<up> (org-move-item-up)
- リスト項目をサブ項目ごと後に移動：M-<down> (org-move-item-down)
- 項目の昇格： M-<left> (org-metaleft)
- 項目の降格： M-<right> (org-metaright)
- サブ項目ごと項目を昇格： M-S-<left> (org-shiftmetaleft)
- サブ項目ごと項目を降格： M-S-<right> (org-shiftmetaright)
- 同レベルリスト内の項目記号を変更する： C-c - (org-cycle-list-bullet)
- チェックボックスにチェックを入れる／外す： C-c C-c (org-ctrl-c-ctrl-c)

TAB/M-RET/M-S-RET/M-<up>/M-<down> キー押下時の動作は「アウトライン編集」時の「見出し」の場合と同様。見出しにおける「Tree/Subtree」とパラレルに，リストでは「Item/Subitem」（項目／サブ項目）という捉え方をしていることに注意。「番号付き」リストの場合，項目の前後移動では番号が自動的に振り直される。

M-<left>/M-<right> のいずれも，リスト最初の項目であれば使えない。M-<left> は「親」項目の存在を前提とする（つまり，親項目と同レベルの項目に昇格させる，ということ）。M-S-<left>/M-S-<right> のいずれも，リスト最初の項目であれば使えないことがあるので注意す

ること。

C-c C-c は toggle キーとなっている。

表

Org を使えばテキストレベルでの表作成・編集も簡単にできる。それどころか表計算までさせることができるが，これは本講義の目的を逸脱する事柄であるから，本講ではそこまでは扱わない。

Org を用いた表作成・編集は極めて直観的であり，何よりテキストレベルでの素早い作業が可能となる。表（罫線）に用いる記号は，| と + と - のみである。

表作成・編集に関する以下のコマンドを学べ。

- 表作成開始： C-c | (org-table-create)
- cursor がセルを移動することなく表を再描画： C-c C-c (org-ctrl-c-ctrl-c)
- 次のセルに cursor 移動して表を再描画： TAB (org-cycle)
- 前のセルに cursor を戻して表を再描画： S-TAB (org-table-previous-field)
- 同列内の次行に cursor 移動して表を再描画：RET (org-return)
- cursor のある列を左へ移動： M-<left> (org-table-move-column-left)
- cursor のある列を右へ移動： M-<right> (org-table-move-column-right)
- cursor のある列を削除： M-S-<left> (org-table-delete-column)
- cursor のある列の左に新しい列を挿入： M-S-<right> (org-table-insert-column)
- cursor のある行を上へ移動： M-<up> (org-table-move-row-up)
- cursor のある行を下へ移動： M-<down> (org-table-move-row-down)
- cursor のある行を削除：M-S-<up> (org-table-kill-row)
- cursor のある行の上に新しい行を挿入： M-S-<down> (org-table-insert-row)
- cursor のある行の下に新しい行を挿入： C-u M-S-<down> (org-table-insert-row)
- cursor のある行の下に横罫線を引く： C-c - (org-table-insert-hline)
- cursor のある行の下に横罫線を引き，さらに cursor を次行へ移動させる： C-c RET (org-table-hline-and-move)

C-c | コマンドを打つと Table size Columns x Rows [e.g. 5x2]: と尋ねられるので，「3x4」といった具合に「列数 x 行数」を指定する。既に存在する文字列を region 指定して C-c | コマンドを発行すると，単語間にある space を列の区切りとして，自動的に表作成してくれる（org-table-convert-region）。

TAB/RET キーを押すと，必要に応じて新しい行を作ることもある。M-S-<down> コマンドの前に前置キー C-u を付ければ，cursor のある行の「下」に新しい行が挿入される。C-c - コマンドの前に前置キー C-u を付ければ，cursor のある行の「上」に横罫線が引かれる。

第 7 講演習

演習用ファイル： lists.org, tables.org

1. 演習用ファイル lists.org を用いて，上で取り扱ったリスト・チェックボックスに関するコマンドを全て検証せよ
2. 同様に tables.org を用いて，表に関するコマンドを全て検証せよ

第 8 講

TODO/DONE/WAIT/SOMEDAY/CANCELED

Org は高度な GTD (Getting Things Done) 用タスク管理ツールとしての機能を豊富に提供してくれる。言い換えると，Emacs はテキストベースでのスケジューリング用ツールとしても大いに活用できる。

具体的に考えてみよう。我々には，レポートの提出であったり，ゼミコンパの企画であったり，何であれ「なすべきこと」が日々あるものだ。Org では「見出し」に TODO（この語は自動的に赤色で強調される）というキーワードを付しておくことにより，この「なすべきこと」を Emacs 上で管理することができるようになる。

さて，この「なすべきこと」の状態は刻々と変わっていくのが常である。例えばゼミコンパの参加者を募るようなケースであればゼミ生からの「回答待ち」となり，その結果を受けてから参加費を決めたり店を予約したりと，次のアクションを起こすことになる。「やるべきこと」については「完遂」が理想だが，拠無い事情により「延期」や「中断」せざるを得ないことも有り得る。

Org では TODO（なすべきこと）に関連するこれらの要素を WAIT（回答待ち），DONE（完遂），SOMEDAY（延期），CANCELED（中断）といった計 5 つの「状態」として捉え，これを簡単に交替（rotate）させることができる。しかも，DONE, SOMEDAY, CANCELED とした場合は CLOSED（停止）というキーワードとともに「締め」られたタイムスタンプが [2017-08-14 Mo 13:29] といった具合に自動的に刻印される。

なお，タイムスタンプにおける曜日表記は選択言語（ドイツ語，フランス語，英語，日本語）に依存する。Emacs でカレンダーを経由する自動処理を行わせる場合は，ドイツ語ならドイツ語というように 1 言語に固定して置かねばならないので注意すること。後で解説する「DEADLINE, MEMO, AGENDA, NOTE」がこれに該当する。

単にカレンダー表記言語を動的・暫定的に変更する場合は，この限りでない。

TODO は「見出し」にのみ設定できる。TODO に関する以下のコマンドを学べ。

- TODO, DONE, WAIT, SOMEDAY, CANCELED と交替させる： S-<right> (org-shiftright)

DEADLINE

TODO には通常 Deadline（締切）が付き物であり，実際 Org では見出しに年月日指定（時刻印を欠いたタイムスタンプ）の DEADLINE を設定できる。DEADLINE の設定はカレンダーと連携して行われるが，時刻の付加は，必要に応じて，手動で行う。

Deadline に関する以下のコマンドを学べ。

- Deadline を設定： C-c C-d (org-deadline)

C-c C-d コマンドを打つと別 window にカレンダーが現れるので，S-<left>/<right>/<up>/<down> キーを使って cursor を移動させ，任意の年月日を選択する。時刻を付加する場合は 2017-08-14 Mo 15:26 といったように HH:MM フォーマットで曜日の後に one space 置いて表記する。前置キー C-u の後に C-c C-d コマンドを発行すれば，設定済みの Deadline を削除できる。

タイムスタンプの数字（年，月，日，時，分）や曜日（選択言語依存）の上に cursor を置き S-<left>/<right>/<up>/<down> キーを押せば「年，月，日，曜日，時，分」を変更できる。その際，年月日と曜日は正しく連動して変化するので，曜日の転記ミスから解放されるという利点がある。

なお，TODO 項目は（従ってまた Deadline も）次講で解説する Agenda（予定表）と連携させるのが良い。このためには TODO を含む見出しのあるファイルは（他のやり方もあるが）~/org/ 以下に保存する。また，TODO を Agenda に反映させるにはファイル拡張子を *.org とすること（*.txt では不可）。

第 8 講演習

演習用ファイル： todo.org

1. 演習用ファイル todo.org を用いて「見出し」に TODO を設定せよ
2. TODO を DONE/WAIT/SOMEDAY/CANCELED に rotate させよ
3. 適当な日時で DEADLINE を設定せよ
4. DEADLINE に時刻を付加せよ
5. DEADLINE のタイムスタンプを色々と変更させてみよ

第 9 講

Memo

「なすべきこと」は，大抵の場合，安閑時ではなく，何か仕事をしているときにこそやって来たり，それに気付いたりするものだ。我々が Emacs を使ってレポートや論文の執筆や編集作業に従事しているとき，その作業を一旦中断して TODO ファイルを新たに作成する，などという手順を取っていては効率が悪いし，何より思考の流れを邪魔されることで生産性も落ちてしまう。つまり，作業中に「メモを取る」ような感覚で TODO を処理したい。

Org では *.org ファイルに TODO 項目を追加することを Capture と呼んでいるが，これらキャプチャされる項目は，EURO ICT Emacs では原則として全て ~/org/Memo.org ファイルの中に書き込まれ蓄積されていくように設定してある。また，キャプチャされた項目はすぐ後に説明する Agenda（予定表）buffer において Memo: エントリの下に整理されるようにしてある（「祝祭日」は Diary: エントリ）。

キャプチャ（Emacs 作業中にメモを取るような感覚で TODO 項目を追加すること）に関する以下のコマンドを

学べ。

- キャプチャ開始： C-c c m (org-capture)
- キャプチャを保存し，元 buffer に復帰： C-c C-c (org-capture-finalize)
- キャプチャのプロセスを中断し，元 buffer に戻る： C-c C-k (org-capture-kill)

キャプチャされた TODO 項目は，Memo.org ファイル中で第 4 階層（第 1 階層は西暦年，第 2 は西暦年月，第 3 は西暦年月日と曜日；曜日表記は言語依存）に自動的に書き込まれる。キャプチャであるからデフォルトではもちろん TODO キーワードが付されているが，これを除去しても（「状態」を変更しても）一向に構わない。つまり，TODO ではない文字通り単なるメモ書き項目としての記録もできるということである。

なお，C-c c m でキャプチャする際，場合によっては「リンク」の張られた「注記」（Annotation）が自動的に付加されることがある。Org のリンク機能については（極めて便利なものであるが），本講義では取り扱わないので，これらの注記は無視して（その部分を削除しても）良い。

Agenda

TODO 項目を一覧することのできる機能が Agenda（予定表）である。Euro ICT Emacs では Week-agenda という 1 週間単位での時系列 Agenda を一覧できるよう設定してある。TODO に限らず，~/org/*.org ファイルに「活性タイムスタンプとともに記録されている」項目であれば同様に一覧できる。Deadline 日には赤色で項目が強調される。しかも Deadline の 7 日前から In 4d.:（後 4 日）のように警告まで発してくれる。Deadline を過ぎた場合も赤色で 1 d. ago（締切は 1 日前）のように警告を発し続けてくれる（TODO 状態をを DONE/SOMEDAY/CANCELED のいずれかに変更するまで）。

Org では <2017-08-19 Sa 19:00> という表記形式を「活性タイムスタンプ」と呼び，[2017-08-19 Sa 19:00] という形の「非活性スタンプ」から区別している。前者は Agenda に反映されるが，後者は，たとえそれが TODO 項目であったとしても，反映されない（Agenda に挙がってこない）。

先ずは，ここで Org におけるタイムスタンプの生成法についてまとめておこう。

- 年・月・日・曜日から成る活性タイムスタンプを生成： C-c . (org-time-stamp)
- 同様の非活性タイムスタンプを生成： C-c ! (org-time-stamp-inactive)
- 活性・非活性の変更：(org-toggle-timestamp-type)
- 1 日前へ： S-<left>: (org-timestamp-down-day)
- 1 日後へ： S-<reight>: (org-timestamp-up-day)
- 年，月，日，曜日，時，分を減らす： S-<down> (org-timestamp-down)
- 年，月，日，曜日，時，分を増やす： S-<up> (org-timestamp-up)

C-c . そして C-c ! とも，前置キー C-u の後にコマンドを打てば，「時，分」を加えたタイムスタンプを生成できる。また，既にあるタイムスタンプ上で C-c ./C-c ! を叩くことで，活性・非活性の変更にも使える。

活性・非活性に拘らず，タイムスタンプ生成コマンドを「続けて 2 回」発行すると，「期間」が設定される。Org における「期間」表記は <2017-08-15 Di>--<2017-08-16 Mi> のように「ハイフン 2 つ」でなされ，「時間範囲」は [2017-08-15 Di 13:00-14:30] のように「ハイフン 1 つ」で表される。

S-<left>/<right> コマンド使用時は，cursor がタイムスタンプ上のどこに位置していても良い。曜日も日にちに連動して変更される。

S-<left>/<right> コマンド使用時には，増減させたい項（例えば「年」）の上に cursor を置く。この際，必要に応じて曜日も変更される。「分」の増（減）分は（先ずキリのよい「5 の倍数」分に丸められてから）「5」分ずつとなっている。なお，「時間範囲」が設定された項の「開始時刻」（「時」でも「分」でも）に cursor を置いてこれらのコマンドを用いると，最初の時間範囲で定められた時間「量」を保ったまま「終了時刻」も連動して変更される。

以上を踏まえ，次に，Agenda に関する以下のコマンドを学べ。なお，EURO ICT Emacs では，ファイルを指定せず Emacs を起動した場合には，Week-agenda buffer が表示されるように設定してある。

- Week-agenda を呼び出す： C-c a a (org-agenda-list)
- 次行へ移動： n (next-line)
- 前行へ移動： p (previous-line)
- 次週へ移動： f (org-agenda-later)

- 先週へ移動： b (org-agenda-earlier)
- 今日へ移動： . (org-agenda-goto-today)
- 任意の年月日へ移動： j (org-agenda-goto-date)
- エントリ項目のある元ファイルへ移動： TAB (org-agenda-goto)
- Agenda buffer をリフレッシュ： g (org-agenda-redo)
- TODO 状態をリモート編集： t (org-agenda-todo)
- 日出入時刻を minibuffer に表示： S (org-agenda-sunrise-sunset)
- 別 window に他暦（ユリウス，イスラム，ヘブライ，等々）変換一覧表示：C (org-agenda-convert-date)
- エントリ項目にノートを追記する： C-c C-z (org-agenda-add-note)
- Agenda buffer を抜ける： q (org-agenda-quit)

Agenda でタイムスタンプが使われる場合の曜日表記は言語に依存するので留意すること。

Agenda の一覧表示法には，他にも様々な形式がある。C-c a a と打ち込むのが基本だが，C-c a と打った段階で Press key for an agenda command: というプロンプトメッセージが出るので必要に応じて色々と試してみると良い（t: List of all TODO entries; n: Agenda and all TODOs: set of 2 commands 等々。n の方は，Week-agenda に加えて週をまたいだ全ての TODO を一覧表示できる）。

j を打つと，望みの年月日をカレンダー上で選択するよう求められるので，S-<left>/<right>/<up>/<down> キーを用いて移動する。

TAB を使って元ファイルへ移動し，再度 Agenda buffer に戻った際は，元ファイルの編集いかんに拘らず g（このキー操作は Dired や EWW におけるリフレッシュと同じ！）を打ち Agenda buffer をリフレッシュしておくこと。そうしないと，次に TAB を打っても同じ動作を期待できない。

TODO 状態をリモートで編集するには t の後にさらに t(TODO), d(DONE), w(WAIT), s(SOMEDAY), c(CANCELED) のキーを打つ。これにより Agenda buffer はリアルタイムで TODO 状態が rotate する。元ファイルにおける当該エントリの TODO 状態も同様に変更されているが，こちらは C-x C-s で元ファイル（Memo.org）を上書保存してやる必要がある。

既存の TODO 項目（キャプチャしたエントリ項目）に，後から追記したい事柄が出てくることがある。Org ではこれを Note と呼ぶ。Agenda buffer で当該エントリ項目に cursor を置き，C-c C-z キーを押して必要事項を入力し C-c C-c と打てば（破棄する場合は C-c C-k），- Note taken on [2017-08-15 Di 11:53] といったフォーマットの「ノート」が元ファイル（Memo.org）の当該エントリ項目に追記される。元ファイルは C-x C-s で上書保存する。なお，NOTE では非活性のタイムスタンプが打たれるので，ノート・エントリは Agenda に挙がっては来ない（つまり，元ファイルを参照する必要がある）。

以上，Emacs における多機能な巨大パッケージ Org が備えるごく一部の機能に絞って解説してきた。他にもある魅力的な Sparse trees, Footnotes, Hyperlinks, Progress logging, Priorities, Tags, Properties, Clocking work time, Refile, Archive 等々といった機能については，それぞれの興味に応じて，各自で学んで欲しい。

第 9 講演習

1. Memo をキャプチャで作成・保存せよ
2. Memo.org ファイルの中に今キャプチャした Memo がエントリされていることを確認せよ
3. Week-agenda を呼び出せ
4. Week-agenda 内を様々に移動せよ
5. Week-agenda から Memo.org をリモート編集し，Note を追加せよ
6. タイムスタンプの活性・非活性を変更せよ
7. タイムスタンプの各項を変更してみよ
8. 「期間」設定した TODO 項目は Week-agenda buffer ではどのように表示されるか，確認せよ

第 10 講

擬似ビッグデータ処理

次のような問題を考えてみよう。「東京都千代田区には永田町（隣接する霞が関とともに日本の国家中枢機能が集中している）という地名があり，これは日本における国会の代名詞ともなっているが，東京都以外に永田町という地名は存在するか」。

本講では，この問題を「正確に」かつ「素早く」解く方法を検討する。

先ず，大前提として「日本全国の地名が漏れなくリストアップされたデータ」が必要であることがすぐに分か

る。このようなデータは存在するのか。

幸いなことに,「郵便局」(日本郵便株式会社) の Web からこの種のデータを取ってくることができる。「住所の郵便番号 (CSV 形式)」がそれだ。「読み仮名データの促音・拗音を小書きで表記するもの」のページから「全国一括」データ (ken_all.zip) をダウンロードする。ZIP 形式で圧縮されたファイルを展開すると KEN_ALL.CSV という CSV フォーマットのテキストファイルが現れる。

KEN_ALL.CSV ファイルを Emacs で開くと,当該ファイルは「124,132 行から成り,11.7MB のファイルサイズを持つ」テキストとしてはそれなりに巨大なファイルであることが分かる。しかしもちろん,テキストであるから Emacs で自由自在に処理できる。

冒頭の問題に戻ろう。KEN_ALL.CSV ファイルは「日本全国の郵便番号と住所等を 1 行ごとに区切って対応させたテキスト・データベース」となっている。この膨大なテキスト・データベースから「永田町」という文字列を含む行のみを抽出できれば,我々は「東京都以外に永田町が存在するかどうか」を正確に調べることができる。

そして Emacs には,この用途に打ってつけの Occur というコマンドが用意されている。Occur を使えば「カレント buffer において正規表現にマッチした行を全て表示」させることができる。

Occur に関連する以下のコマンドを学べ。

- カレント buffer において正規表現にマッチした行を表示させる: M-s o (occur)
- 正規表現にマッチしない行を表示: (flush-lines)
- 正規表現にマッチする行を表示: (keep-lines)

KEN_ALL.CSV ファイルを buffer に読み込み,M-s o コマンドを打つと List lines matching regexp: と尋ねられるので,「永田町」と入力し RET する。すると別 window に 53 matches for "永田町" in buffer: KEN_ALL.CSV と表示される。53 行程度であれば,一つ一つ目で追って確認していくこともできなくないが,ここは作業速度を重視し,(flush-lines) コマンドを使ってさらに絞り込むことにする。

その前に,Occur の結果 buffer の mode line を良く見てみると「%%」と表示があり,この buffer は「読み込み専用」となっていることが分かる。このままでは (flush-lines) コマンドを使えないので,C-x C-q と打ち込み (これは toggle キー),当該 buffer を編集可能な状態にしておくことが必要となる。

準備が整えば,M-x flush-lines と打ち込む。すると Flush lines containing match for regexp: と尋ねられるので,「東京都」と入力し RET する。結果として,「永田町」という地名は東京都以外には「栃木,埼玉,新潟,岐阜,静岡,長崎,鹿児島」の 7 県に存在することが分かった (ただし「ナガタチョウ」と読むのは静岡県富士市と鹿児島県奄美市にある永田町のみ。その他は「ナガタマチ」と読む。奄美市にある地名は正確には「名瀬永田町 (ナゼナガタチョウ)」)。

第 10 講演習

演習用ファイル: KEN_ALL.CSV

1. 「福岡」という地名は福岡県福岡市以外に日本のどこに存在するか
2. 「七隈」以外に,漢数字を伴う「一隈,二隈,・・・,九隈」という地名は日本に存在するか

ただし,本講演習 1 では「上福岡」(新潟県阿賀野市) のような地名は除くものとする (ヒント: 正規表現では \< で単語の先頭,\> で単語の末尾,をそれぞれ表すことができる;「一隈あるいは二隈」は正規表現を用いて \(一隈\|二隈\) と書ける)。

第 11 講

エクスポート: TEXT UTF-8/TEX/HTML

これまで我々は Emacs を使ったテキスト処理に関する技法を様々な観点から学んできた。ただし,学習の力点は意図的に「Input」(入力) と「Edit」(編集) に置き,「Output」(清書・整形を伴う出力) については特に触れて来なかった。

出力については,もちろん,Emacs を一種のプリプロセッサ (前処理プログラム) として利用し,Org のアウトライン編集機能を駆使してレポートや論文の草稿を徹底的に練り上げ,その中身を最終段階でワードプロセッサ等にコピーした上で完成原稿として仕上げ,ワードプロセッサから出力 (例えばプリントアウト) する,というやり方もある。現実解としてこうした出力方法を取るのも,決して悪くはないだろう。

だが,Emacs はもっと遥かにスマートな Output 法と連携できる。それが TEX であり HTML である。いずれも「マークアップ言語 (Markup Language)」の一種で,

通常のテキストに「メタレベル」の「命令」(HTML では Tag, TEX では Control Sequence と呼ばれる。「制御綴」とも）を埋め込むことで視覚表現や文章構造等を記述することができる。なお，TEX は組版用のプログラミング言語でもある。

TEX や HTML の「ソースファイル」(ファイルの内容を読み込ませて何らかの処理や変換などを行い，結果を別のファイルに保存するシステムや作業等において，処理にかけるファイルのことをこのように呼ぶ。インプットファイルとも）はいずれも「テキストファイル」であるから，テキストエディタである Emacs とはそもそも相性が良い。さらに，Emacs には HTML や TEX 専用の major mode までもが存在する。

「ヨーロッパ学 ICT 講義」では，これ以降，テキスト主義 ICT が持ち得る強力な潜在能力の一例として TEX と HTML を学び，マークアップ言語との連携がもたらしてくれる更に豊かなテキスト表現の沃野を概観する。

本講では，手始めに，Org を使ってテキストファイル（ソースファイル）を UTF-8 文字コードの簡易整形テキスト（*.txt)，そして TEX のソースファイル（*.tex)，さらには HTML ファイル（*.html）へと変換する作業を体験してみる。Org ではこれを「エクスポート」と呼んでいる。TEX のソースファイルはバックグラウンドで自動的に「コンパイル」(ここではテキストファイルから PDF ファイルへ変換すること）までされるから，結果としてアウトプットファイルである PDF ファイル（*.pdf）も作成される。

Org のエクスポートに関する以下のコマンドを学べ。これらのコマンドが使えるファイル形式は *.org および *.txt のみである。Key Bindings で用いられる英字との関連で，ここでは TEX の代わりに LATEX という名称を使っていることに注意。

- UTF-8 簡易整形テキストへ変換： C-c C-e t u: (org-ascii-export-to-ascii)
- LATEX ソースファイルおよび PDF ファイルへ変換し，PDF ファイルを開く： C-c C-e l o (org-latex-export-to-pdf)
- HTML ファイルへ変換し，ブラウザで HTML ファイルを開く： C-c C-e h o (org-html-export-to-html)

Org のエクスポートはとてもスマートで便利な機能であるが，そもそもなぜこのようなことが可能なのか。ここでは詳しく立ち入らないが，それは Org の文書が決められたシンタクス（例えば * の数によって「見出し」の階層構造を定義，- や + 記号の前置によって「番号なし」箇条書きを，数字の前置によって「番号付き」箇条書きを定義，等々）によって，通常テキストにメタレベルで「マークアップ」を施しているからである。いずれもマークアップ・テキストであればこそ，TEX/HTML/Org 文書間での相互変換も可能となるのである。

「Org のエクスポートを使えるようになれば TEX/HTML の学習は不要では」と考えるのは誤りである。なるほど Org のエクスポート機能は Org における文書構造をできるだけ正確に TEX/HTML に変換してくれようとするが，TEX/HTML におけるフォーマットの種類の方が Org より遥かに豊富であるため，万全ではない。それに，そもそもエクスポートの段階でエラーが生じれば文書の出力は論外となるし， 生成された TEX/HTML ソースファイルに不具合があった場合，TEX/HTML の知識がなければ不具合箇所を訂正（Debug）することが出来ない。

第 11 講演習

演習用ファイル： fairy-tales.org, lists.org, tables.org, todo.org, export.org, css.zip, jpg.zip, txt.zip

1. 演習用ファイルを全て UTF-8 簡易整形テキスト， LATEX ソースファイルおよび PDF ファイル，HTML ファイルへエクスポートせよ
2. 自動生成された fairy-tales.txt/tex/html, lists.txt/tex/html, tables.txt/tex/html, todo.txt/tex/html の中身を点検せよ
3. export.pdf 以外の *.pdf ファイルにおけるドイツ語・フランス語表記の不具合を見つけよ
4. export.org/tex/html の中身を点検せよ

export.org ファイルでは，本講義で取り扱わなかった Special Lines, Hyperlinks, Tags, Properties, Inline Images といった Org の機能も用いている。ソースファイルがどのようになっているか, 必要に応じて参照して欲しい。

第 12 講

Emacs パッケージのアップデート

EURO ICT Emacs には標準添付以外の様々なパッケージをも組み込んで，「テキスト主義 ICT」用ツールとし

ての使い勝手を高めている。これらのパッケージは，あらゆるソフトウェアがそうであるように，絶えずアップデートされるのが常である。

Emacs の諸パッケージを安定利用できているのであれば，そしてこれらのパッケージが提供してくれる諸機能の現状に満足しているのであれば，パッケージ・アップデートがセキュリティホールと無関係である限り，敢えてアップデート作業を行う必要はないのかも知れない。しかし，一般論として，アップデートされたパッケージには（新機能の追加も含めて）何らかの「改善」がなされていることも事実である。なお，アップデートの際はインターネット接続が前提となる。

パッケージのアップデートに関する以下のコマンドを学べ。

- パッケージのアップデートを行う：(list-packages)
- アップデート buffer を抜ける： q (quit-window)

(list-packages) コマンドを打ち込むと，2 packages can be upgraded; type 'U' to mark them for upgrading. といったメッセージが minibuffer に表示されるので, U を叩く。するとさらに 2 Packages marked for upgrading と出るので，x を押す。今度は畳みかけるように Upgrade these 2 packages (js2-mode-20170815.546, helm-20170815.1123)? (y or n) といったようなメッセージが現れるので（アップデートを望むなら）y を押す。

この一連の手順を終えると，必要なアップデート用データがダウンロードされ，Emacs 内部でのファイルの読み込みを速くするための「バイトコンパイル」がバックグラウンドで実行される。パッケージ・アップデートが終わると Package menu: Operation finished. というメッセージが現れ，新パッケージを利用できるようになる。

アップデートが存在しない場合は，(list-packages) コマンドを打ち込んだ際，Package refresh done というメッセージだけが出る。念のため U と叩いてみると確かに No packages to upgrade. と返ってくるので q キーを打ってアップデート buffer を抜ければ良い（U で確認せず，すぐに q を打って抜けても良い）。

Emacs の設定ファイル

既に言及した通り，Emacs の設定ファイルは Windows では C:\home\.emacs.d\init.el に，Mac では /Users/ynagata/.emacs.el にある（ynagata 部分は Mac におけるユーザ名）。そして，この中に「素の Emacs」とは異なる「EURO ICT Emacs 独自の動作」を規定する全てのプログラムが Emacs Lisp というプログラミング言語で記述してある。

"the World's Most Extensible, Customizable Editor" とも称される Emacs は，ユーザ自身で文字通り如何様にも機能拡張が可能であるが，そのためには（程度にもよるが）Emacs Lisp 言語でプログラムを書かねばならない。EURO ICT Emacs の init.el/.emacs.el には，講義担当者が現時点で「ヨーロッパ学 ICT のための最適設定」と考える事項を全て書き込んである（そしてこれからも書き込んで行く）ので，通常は手を入れる必要はない。

従って本講義では，Emacs 設定ファイルの書き方そのものについての解説は割愛するが，設定ファイル内の知っておくと良いいくつかの箇所についてのみコメントしておきたい。

Windows 限定であるが（Mac の「辞書.app」にはデフォルトで国語辞典「スーパー大辞林」が付いている），電子版の『広辞苑』を Emacs で使えるようにしてみる。先ずは有償の『広辞苑第六版 DVD-ROM 版』を購入する。次に設定ファイル init.el にある以下の箇所（ndeb … は実際は 1 行）

```
(setq lookup-search-agents '(
;  (ndeb "C:/Program Files (x86)/Kototoi Project/
    Data/KOJIEN")
   (ndeb "C:/usr/local/dict/wadoku")
   (ndeb "C:/usr/local/dict/openthde")
   (ndeb "C:/usr/local/dict/demorph")
;  (ndic "C:/DWB")
   )
)
```

で (ndeb "C:/Program Files …) 行頭にあるコメント記号を削除する。『広辞苑』の辞書データがある箇所のパスも，実際のものと揃えておく。

最後に Emacs を再起動し M-x lookup と打つ。すると Lookup buffer が表示され，「ndeb:C:/Program … furoku 付属資料」や「ndeb:C:/Program … kojien 広辞苑第六版」といった項目が新たにエントリされたことが確認できる。フリーの wadoku（和独辞典），openthde（ドイツ語類語辞典），demorph（ドイツ語変化形辞典）については既に「選択」されて Emacs 上で使えるようになっているが『広辞苑』についてはまだなので，「付属資料」と「広辞苑第六版」に cursor を移動し，それぞれ m を

押す。するとこの 2 項目も「選択状態」(* が付く）となり，以降，Emacs から「広辞苑を引ける」ようになる。Lookup buffer は q キーを押せば抜けられる。

Windows の Emacs から「和独辞典」や『広辞苑』を引くには，調べたい語の上に cursor を置き，M-? (lookup-word) と打ってやればよい。別 window に検索結果 buffer が表示されるが，これも q キーを打てば抜けられる。

Google Translate については既に解説した（27 ページ）が，Emacs の設定ファイルには次のようなフォーマットで相互翻訳言語を指定している。

```
(setq google-translate-translation-directions
 -alist '(("de" . "ja") ("ja" . "de") ...
 ("de" . "fr") ("fr" . "de")))
```

例えば ("de" . "ja") という箇所は「ドイツ語から日本語へ（翻訳）」を意味する記述なので，同様の「言語セット」を自分が必要とする分だけ追記していけば，翻訳対象言語を好きなだけ増やせる（もちろん Google Translate がサポートしている言語に限られるが)。「言語」は「ISO 639-1 言語コード」によって定められている「2 文字」コードで記す（例： el 現代ギリシア語，la ラテン語)。設定ファイルを変更したら，それを保存し，Emacs を再起動することを忘れずに。

Emacs の「外見」を変更したい場合があるかも知れない。そのための一番簡単な方法は，「テーマ機能」を使うことだ。EURO ICT Emacs では tsdh-light テーマをデフォルトとしているが，

```
(load-theme 'tsdh-light t); gut
```

という箇所をコメントアウトし，代わりに他のテーマ adwaita, deeper-blue, dichromacy, … が記載されている行を一つ有効にして設定ファイルを保存し，Emacs を再起動すると，雰囲気ががらりと変わった外見で Emacs が立ち上がるようになる。試してみよ。

なお，Emacs そのもののインストール法については Web（5 ページ参照）にまとめておいたので，そちらを参照すること。

第 12 講演習

1. Emacs パッケージをアップデートせよ
2. Windows 上の Emacs から「和独辞典」を引いてみよ
3. EURO ICT Emacs の「テーマ」を色々と変更してみよ

Mac 教室の Mac 端末ではアップデート結果が保存されないことに注意。現時点では，残念ながら，Windows 上の Emacs と連携できるフリーのフランス語辞書を講義担当者は探し出せていない。

第 13 講

TEX とは

本講からいよいよ TEX の学習に入る。

TEX とは何か。一言で言えば，それは アメリカの数学者 Donald Knuth（1938–）によって開発された Typesetting System（あるいは Formatting System）である。この TEX の上にアメリカのコンピュータ科学者 Leslie Lamport (1941–) によって構築された Document Preparation System が LATEX である。

我々が本講義で学ぶ対象は，実際にはほぼ LATEX のみであるが，本講義では必要な場合を除いて両者の呼称を敢えて区別しない。つまり，TEX/LATEX の違いを細かく詮索するような “The Game of the Name” には深入りしない。そうしないと肝心の “The Name of the Game” を掴み損ねてしまうことになるからだ。

医学の父と呼ばれる古代ギリシアの医者ヒポクラテス（Ἱπποκράτης, BC460–BC370）は，「人生は短く，技術（医術）は長い」(ὁ βίος βραχύς, ἡ δὲ τέχνη μακρή）と言ったとされる。この格言は後にラテン語訳され，「芸術は長く，人生は短い」(ars longa, vita brevis）という意味合いで人口に膾炙した。TEX という名称は，ヒポクラテスの格言の中にもある τέχνη という語から取られた（故にテックスではなく，テフ，テヒ，テックのように発音される)。そして実際 TEX は技術ではあるが，芸術の側面をも併せ持っている。「ヨーロッパ学 ICT」講義の回数は限られているので，我々は TEX を用いたテキスト処理をこそ実践的に学んで行くこととしたい。

つまり，EURO ICT Emacs & TEX の学習に際しては，古代ギリシアの哲学者アリストテレス（Ἀριστοτέλης, BC384–BC322）が述べた次の言葉に倣うということだ：「竪琴を弾くことを学ぶ者は，竪琴を弾くことによって，竪琴を弾くことを学ぶ」(ὁ γὰρ μανθάνων κιθαρίζειν κιθαρίζων μανθάνει κιθαρίζειν.『形而上学』IX, 8)。

さて，TEX 記法は数式をテキストで記述する際の標準となっていることからも分かるように，TEX は特に数式

を含むテキスト処理に定評があり，実際，

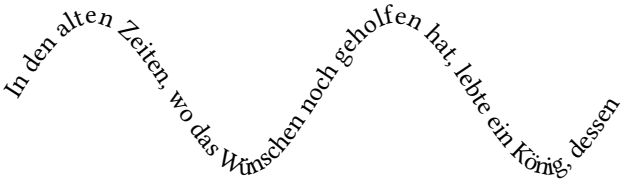
$$\left(\int_o^\infty \frac{\sin x}{\sqrt{x}}\,dx\right)^2 = \sum_{k=0}^{\infty} \frac{(2k)!}{2^{2k}(k!)^2}\frac{1}{2k+1} = \prod_{k=1}^{\infty}\frac{4k^2}{4k^2-1} = \frac{\pi}{2}$$

や

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{\cdots}}}}}$$

といった数式を簡単に入出力できる。

一方，テキストを正弦波に添って



のように配置させるといったことも自由自在にできる。他にも，複雑な物理数式や化学式の入出力，さらには音楽記譜（図 16，57 ページ参照）やチェス，バックギャモン，将棋，碁の棋譜，クロスワード・パズルの作成，数独（Sudoku, Number Place とも）の作題および解答（！）等々も可能となっており，およそテキストに関わるありとあらゆる組版処理（そしてまたプログラミング）の可能性を私たちに提供してくれる。

しかし，人文学部でドイツ語・フランス語を学ぶ我々にとって，それ以上に注目したいのは，TEX が優れた多言語処理能力をも備えているという点だ。実際，TEX はワードプロセシングを超えた「テキストプロセシング」のレベルで多言語混在テキストを組版処理してくれる。具体的には，各言語に応じた「正書法」をも反映した処理をしてくれるということであるが，詳しくは追々学んでいく。

ターミナルで用いる基本コマンド

先ずは，一般のワープロソフトと TEX の処理方式の違いについて慣れること。前者では WYSIWYG (What You See Is What You Get) という「画面表示と印刷イメージが同じ」方式で文書を作成していくが，後者では「各種の命令を埋め込んだテキストファイル」（これを Source File と呼ぶ）を「一括処理」（Batch Processing とも）することで最終的な「出力」（現在では通常 PDF ファイル）を得る方式を取る。この作業手順に関し「ソース（ファイル）を TEX にかける，タイプセットする」という言い方をすることもあれば，プログラミング言語で書かれたプログラム（Source Code とも）をコンピュータが直接実行可能な形式のプログラム（これを Object Code と呼ぶ）に変換する作業になぞらえて「ソースをコンパイルする」とも言う。なお，ソースファイル作成時には視覚的レイアウトのことは特に考慮せず，ただ「文書の論理的構造」にのみ注力しテキストを作成していくのが TEX における文書作成の原則である。

ソースのコンパイルは基本的にターミナル上で行う。従って，ターミナルを用いるに際して必須となるコマンドを以下にまとめておく。

- pwd: カレントディレクトリの絶対パスを表示
- ls: カレントディレクトリ内に含まれるファイルやディレクトリ情報を表示
- cd: ディレクトリ移動（.. で 1 階層上へ）
- mkdir *string*（string ディレクトリを作成）
- rm *string*（string ファイルを削除）
- mv *string*（string ディレクトリまたはファイルを移動）
- cp *string*（string ディレクトリまたはファイルをコピー）
- <up> 1 つ前のコマンドを再実行する
- <down> 1 つ後ろのコマンドを再実行する

pwd コマンドを打つと「絶対パス」でカレントディレクトリが表示される。Windows のコマンドプロンプトでは pwd が使えない（PowerShell であれば大丈夫）ので，代わりに cd とだけ打つ（引数を付けない）。

ls コマンドでは ls -a とオプションを付ければ，隠しファイルである「ドット・ファイル」を含む全てのファイルやディレクトリが表示される。ls -ah のように併せて h オプションを付加すれば，KB/MB/GB といった表示単位を人間にとって読みやすいよう適当に切り替えてくれる。

cd コマンドでは絶対パスあるいは「相対パス」で移動先ディレクトリを指定できる。相対パスではカレントディレクトリを . で，ホームディレクトリを ~ で表す。例えば，EURO ICT Emacs の設定ファイルの在処を絶対パスで記すと，Windows では C:\home\.emacs.d\init.el（Mac では /Users/ynagata/.emacs.el）となるが，今カレントディレクトリが C:\home\.emacs.d であったと仮定すれば，cd ../../texlive と打つことでも C:\texlive に移動

できる。相対パス表記は HTML ソースを書く際にも頻繁に用いることになるから，慣れておくと良い。

Windows のコマンドプロンプトでは rm の代わりに del を使う。rm ではワイルドカードが使えるので，rm *.aux *.log *.dvi などと打てば拡張子が aux, log, dvi であるファイルを一括削除することもできる。ディレクトリごと（サブディレクトリも含め再帰的に）削除したい場合は rm -rf とすれば良い。

mv, cp（コマンドプロンプトでは move, copy）では絶対パスあるいは相対パスで移動・コピー先を指定する。cp -r とすれば対象ディレクトリを（中のファイルも含めて）再帰的にコピーできる。

<up>/<down> は「ヒストリ（履歴）機能」を呼び出すキー操作となる。これにより，ターミナル上で既に打ち込んだコマンドを簡単に再利用できる。なお，UNIX 系 OS のターミナルであれば 例えば man cp のように打ち込むことで cp の詳しい使い方を知ることができる。

ターミナルにおける基本コマンドについてはインターネット上に多くの解説があるので，適宜検索して調べると良い。

Homo Ludens 4: SL

ターミナルは賑々しいユーザインタフェースでは決してないので，そこに無味乾燥な趣（端的に地味であるということ）を感じてしまいがちなのは仕方のないことかもしれない。このような場合，思わずくすっと微笑んでしまうようなコマンドがあると嬉しい。

こうした微風を運んでくれるかも知れないコマンドが sl である。sl は数多ある UNIX 系ジョークコマンドの一つである（図 14，49 ページ参照）。

ターミナルで用いる頻度の高いコマンドに ls（ファイルやディレクトリ情報の表示）があるが，sl は丁度その逆綴りとなっている。つまり，イライラしている等などで，ターミナル上で ls と打つべきところを誤って sl と叩いてしまった場合，突如モニタ画面に Steam Locomotive が現れ，我々はそれが通り過ぎていく様をただぼーっと見ていなくてはならなくなるのである（その間，作業は中断される）。

オプション引数として -a, -l, -F, -e が使える。それぞれ違いを楽しんでほしい。man sl でマニュアルを参照できる。ただし，sl コマンドを覚えたからといって，ヨーロッパ学 ICT に関する技能が向上するわけではない。

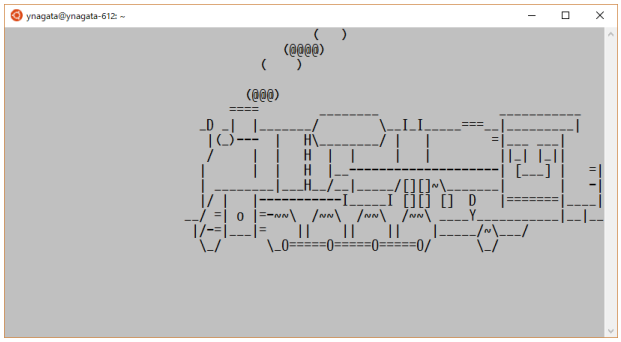


図 14: ターミナル画面を走る SL

TEX Jargon と欧文ソースのコンパイル法

以下の TEX Jargon を学べ（第 13 講演習で用いる各演習用ファイルの中身も比較参照せよ）。Control Sequence（\で始まる TEX のコマンドを正確にはこう呼ぶことがある），Preamble（\begin{document} コマンドの前に TEX への様々な処理指定を記しておく前口上箇所）Argument（コマンド部の {…} 内に記す引数，必須），Option（コマンド部の […] 内に記すオプション指定，任意），Environment（\begin{…} … \end{…} で囲まれた環境型コマンド），Declaration（宣言型コマンド），Scope（宣言型コマンドの有効範囲），ノンブル（Nombre, 各ページに振る Page Number），柱（各ページの上 Header か下 Footer に出力する章や節の名前等）。

TEX のソースファイルは，「欧文」のみから成る（あるいは欧文を基底言語とし，それ以外の邦文等が例外的に少々現れるような）テキストを作成する場合と，「邦文」を基底言語としその中に欧文も含まれるようなテキストを作成する場合とで，プリアンブルの書き方が変わってくる。また，それぞれの処理プロセスも変わる。先ずは，欧文のみを含む TEX のソースファイルの書き方を「演習用ファイル」を手本として学べ。

& $ # % _ { } ^ ~ \ の 10 文字は，いずれも「1 文字から成る TEX コマンド」であるから，TEX ソースの中では「そのままでは使えない（つまり，エスケープする必要がある）」。これらを出力するにはどのように入力すべきであるのか，演習用ファイルの中身を精査せよ。

演習用ファイルの中身を精査した後は，欧文のみから成るテキストを TEX で処理するための以下のコマンドを学べ。これらのコマンドはターミナルで打ち込む。

- pdflatex *string*（欧文のみを含む string.tex ソースを TEX でコンパイルし，出力ファイル string.pdf を得る）

- x（エラーが生じた場合に打ち込む）
- h（エラーに関する TEX の助言を求める）
- e（エラー行に cursor を移動させた状態で Emacs を起動）

「相互参照」（Cross Reference）表記を含む入力ファイル（例えば「目次」や「参考文献」等を自動出力させる）を処理する際は，pdflatex コマンドを複数回発行する（最初あるいは 2 回目の処理でページ数等の情報を補助ファイルである *.aux (auxiliary) 等に書き出しておき，最終プロセスではこうした情報を読み込んだ上でソースファイルを処理することにより正確な相互参照を得る仕組み）。

ソースファイルに誤りがあれば，TEX はその箇所で処理をストップし，エラーが起こった行番号とともにエラーメッセージを返してくる。こうした場合は (e)x(it) を打って TEX から抜け，ソースファイルの当該行にあるエラー箇所を訂正し，ファイルを上書保存した後，再び TEX で処理する，というプロセスを繰り返す。

x の代わりに h(elp) と打ってやれば，TEX は考え得る限りの助言を寄越してくれるが，h を押し続けていくと Sorry, I already gave what help I could…Maybe you should try asking a human? An error might have occurred before I noticed any problems. “If all else fails, read the instructions.” という最終メッセージを発して匙を投げてしまうので，いずれにせよ x を打って TEX から抜けなくてはならない。

なお，e(dit) を打つことで，既に立ち上げている Emacs はそのままで，別 frame あるいは別 CUI の Emacs を立ち上げ（このとき cursor はエラー行に移動している）すぐさまエラーの訂正に取り掛かることもできる。ただし，新しく起動した Emacs buffer 上でソースファイルの編集・保存を行った場合は，元から立ち上げている Emacs 側でも (revert-buffer) コマンドを打って，編集内容を「反映」（更新）させる必要があるので注意すること。

演習用ファイル euro.tex では Babel パッケージを用いた多言語処理をしているので，Babel についても説明しておく。

Babel パッケージ

Babel とは TEX において international/multilungual document を処理するためのパッケージである。ここで言う international document とは英語，ドイツ語，フランス語，等々，といった様々な言語で，しかも「原則として 1 言語のみ」で，書かれた文書を意味し，一方，multilingual document とは「様々な言語が混在する多言語」で書かれた文書を意味する。いずれにおいても Babel は，目次，参考文献，図，表，等々のキャプションや日付表記をそれぞれの言語で自動出力し，さらには，ハイフネーションを含む各言語固有の正書法までをも反映させて，正確に組版を行ってくれるため，TEX でドイツ語やフランス語を処理する際には必須のパッケージとなっている。Babel は，現在，84 の言語をサポートしている。

Babel を使うためには，プリアンブルに\usepackage[polutonikogreek,french,ngerman,english]{babel} のように記す。オプション部には文書内で用いる言語をコンマで区切って全て挙げておく。この際，オプション部の最後に記された言語が「基底」言語となり（あるいは main=english といった書き方でも良い。この場合は必ずしも最後に記す必要はない），当該言語に応じた組版処理がなされる。例えば，上例のように基底言語が「米語」となっていれば，目次，参考文献，図，表のキャプションはそれぞれ Contents, Bibliography, Figure, Table と出力される（ドイツ語であれば Inhaltsverzeichnis, Literatur, Abbildung, Tabelle のように，フランス語では Table des matières, Bibliographie, Figure, Table となる）。

オプション部にある polutonikogreek, ngerman とは，それぞれ「複式アクセント記法」のギリシア語（古典ギリシア語等）および「新正書法」ドイツ語を指す。english は「米」語であるが，「英」語で処理したい場合は british に替える。同様に「旧正書法」でドイツ語を処理したい場合には ngerman ではなく german とし，現代ギリシア語を扱う場合には polutonikogreek ではなく greek とする。

これらのオプション間にはどういう違いがあるのか。

例えば，english/british ではハイフネーション位置や日付出力が anal|y|sis/ana|lysis; August 31, 2017/31st August 2017 のように変わるし，german/ngerman では Bäk|ker/Bä|cker; Mei|ster/Meis|ter のようになる。他にも [n]austrian（オーストリアドイツ語）を指定すれば，1 月は（標準ドイツ語の）Januar ではなく Jänner と出力されるし，french（標準フランス語）と canadian（カナダ・フランス語）でも，それぞれ日付形式が異なって

くる。

Babel の使い方としては，基本的に，処理対象言語を切り替えたい箇所に \selectlanguage{russian} のように記すだけで良い。これは宣言型コマンドであるので，これ以降のテキストは「ロシア語用」に処理されることになる。ラテン文字アルファベット表記言語の中で一時的に短めにロシア語やギリシア語を用いたい場合には，\textcyrillic{…}, \textgreek{…} として引数内にロシア語やギリシア語を記せば良い。

さて，Babel は多機能パッケージであり，その使用法を詳しく知りたい場合はどうしたら良いのか。Emacs が自身の内部にヘルプを含む各種ドキュメントを備えていたように，TEX システムにも膨大な数のパッケージに関するマニュアルがデフォルトで含まれている。これらのマニュアル（原則的に英文 PDF 文書）を呼び出すには，ターミナルで texdoc babel のように打てば良い。babel の代わりに germanb, ngermanb, frenchb とすれば Babel における旧正書法ドイツ語，新正書法ドイツ語，フランス語に関するマニュアルを参照できる。

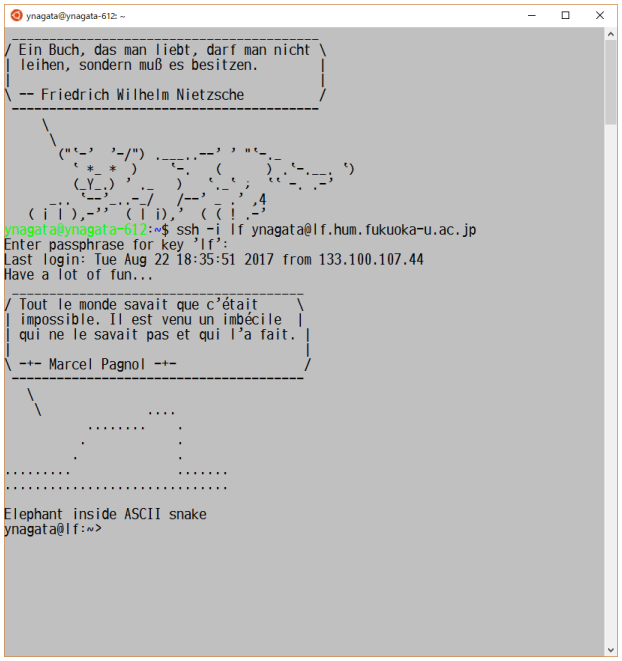


図 15: ドイツ語とフランス語の Fortune

演習用ファイル euro.tex では，Babel の他に，多言語対応引用符処理用パッケージ Csquotes (Context Sensitive Quotation Facilities) や国際音声記号処理用パッケージ Tipa (TEX/Tokyo International Phonetic Alphabet) も用いている。適宜ターミナル上で texdoc csquotes（または tipa）のように打ち込んで詳しい使用法を確認すると良い。

ところで TEX システムには一体どういったパッケージがどれほど同梱されているのだろうか。これらを概観したい，あるいは「全部」チェックしたい，という場合にはどうすれば良いのか。

そのためには TEX システムが内包する doc.html（TEX Live documentation: This document lists links to all HTML and PDF files for packages and guides contained in TEX Live, sorted by package name.）を見れば良い。当該ファイルは texlive/2017/doc.html（Windows），/usr/local/texlive/2017/doc.html（Mac）にある（2017 の箇所は TEX Live の配布年に応じて可変）。例えば，l2kurz.pdf, latex2e-fr.pdf などは，それぞれドイツ語，フランス語で書かれた TEX に関する入門的マニュアルとなっている。これらも，もちろん，texdoc l2kurz のようにして呼び出せるので参照して欲しい。TEX Live 2017 では 2766 個のパッケージが挙がっている。

Homo Ludens 5: Fortune/Cowsay/Lolcat

ターミナルを使うことが楽しくなってくるかもしれない工夫をもう一つ紹介しよう。UNIX 系のプログラム fortune（格言等を引用したメッセージを無作為に表示する Fortune Cookie を模したプログラム），cowsay（任意のメッセージを牛をはじめとする様々な ASCII Art キャラクタの吹き出しの中に表示させるプログラム），lolcat（プログラムの実行結果を虹色で彩色して出力するプログラム）を組み合わせることで，ターミナルを立ち上げる度に，ドイツ語やフランス語の名言をそれなりに賑々しく味わうことができるようになる（英語はもちろん，名言を収めたテキストデータベースさえあればどのような言語でも可能）。

図 15 (51 ページ) は，講義担当者の研究室にある Windows PC 上で動く Bash (on Ubuntu on Windows) を起動し（この段階でフリードリヒ・ニーチェの格言が表示された），続けて SSH によりフランス語学科の Web サーバに接続した（この段階でマルセル・パニョルの名言が表示された）様を表している。講義担当者の Bash on Ubuntu on Windows ではドイツ語の fortune が，フランス語学科サーバのターミナルではフランス語の fortune が，それぞれ表示されるように設定してあるが，このためには簡単なプログラム（スクリプト）を .bashrc か .profile に書いておくだけで良い。詳しくは授業サポー

トページで解説してあるのでそちらを参照すること。

なお，図 15（51 ページ）における fortune の出力は，『研究部論集』への掲載を考慮し，lolcat プログラムを除いて実行した結果となっている。

Mac 端末の PC 教室では，ターミナルを起動する度に「ドイツ語・フランス語・英語」いずれかの言語による Fortune が楽しめる（Cowsay, Lolcat も連動する）ように設定してある。ただし前提として，EURO ICT Emacs & TEX 環境を使えるように start.command を走らせておかねばならない（「PC 教室での準備（Mac 端末）」，8 ページを参照）。

第 13 講演習

演習用ファイル： small2e.tex, euro.tex

1. 英語のみで書かれた small2e.tex の中身を精査せよ
2. small2e.tex を TEX で処理せよ
3. 入力ファイル small2e.tex と出力ファイル small2e.pdf を良く比較せよ
4. 英・独・仏語テキストを含む euro.tex の中身を精査せよ
5. euro.tex を TEX で処理せよ（複数回）
6. 入力ファイル euro.tex と 出力ファイル euro.pdf を良く比較せよ
7. euro.tex ファイル内のタイトルをドイツ語に切り替え，さらに，\usepackage コマンド・オプション部の最後に ngerman が来るように書き換え，ファイル名を euro_de.tex のようにリネームせよ
8. euro_de.tex を TEX で処理せよ（複数回）
9. euro.pdf と euro_de.pdf を比較せよ
10. 同様にフランス語用に調整した euro_fr.tex を作り TEX で処理し，euro_fr.pdf を確認せよ
11. euro.tex ファイル内プリアンブルの \documentclass のオプション部に「,twocolumn」を追記して保存したファイルを TEX で処理させるとどうなるか

euro.tex は TEX で 2 回処理する必要がある（「目次」出力のため）が，ターミナルで続けて同じコマンドを打ち込む場合は，ヒストリ機能を使えば良い。

出力ファイル small2e.pdf と euro.pdf における欧文フォントが異なっていることに気付いただろうか。前者では Knuth 自身が設計・開発した CM (Computer Modern) というラテン文字フォントが用いられている。これはラテン文字のフォントエンコーディング（各文字への番号の振り方）を OT1 (Old Text 1) とする TEX デフォルトのフォントであり，TEX では特にフォント指定をしなければ欧文ラテン文字として CM が用いられ，また，特にフォントエンコーディング指定をしなければ OT1 フォントエンコーディングが用いられることになっているためである。

フォントのデザイン（に対する好悪）を度外視すれば，欧文テキストとして「英文のみ」を組版処理する場合は OT1/CM 使用で何の問題もない。しかし，英字に加えウムラウト，エスツェット，アクサン記号を含むフランス語，ドイツ語を当たり前に処理したい我々としては，7 ビット（つまり 128 文字）という制約のある OT1 エンコーディングでは力不足となる。

具体的には，例えば，OT1 でも ä や ô といった文字を一応それなりに出力することはできる（入力は \"a や\^o となる）が，この場合，実は「1 文字」として処理しているのではなく，それぞれ a と ¨ および o と ^ の合成字として出力している（つまり ä や ô 文字自身に独立したコードポイントが振られていない）ので，ä や ô を含む文字列を PDF 上で検索できないのである。

一方，euro.pdf では Times/Palatino 系フォント（正確には Times/Palatino 相当のフリーフォント TEX Gyre Termes/Pagella）を 8 ビット（つまり 256 文字）の T1 (Text 1) フォントエンコーディングで用いており，入力ファイルのエンコーディング（文字コード）も UTF-8（ユニコード）を明示的に指定している。ä や ô といった欧文特殊文字を含む文字列も，PDF 上で正確に検索することができる。

結論: 英字の 26 文字ラテン文字を超えてドイツ語やフランス語を扱う我々は，さしあたり TEX ソース内に必ず \usepackage[T1]{fontenc} と\usepackage[utf8]{inputenc} を指定しておくこと。

また，TEX では無数のフォントが使えるが，論文・レポート文書作成に重点を置く本講義では，ラテン文字としてさしあたり Times/Palatino の使用を原則とする。具体的にはプリアンブルに \usepackage{newtxtext,newtxmath} のように記しておく（Palatino の場合は tx の部分を px とする）。数式を一切使わない場合は，後者の引数は不要である。

なお，CM の T1 拡張フォントは Latin Modern であるから，T1 フォントエンコーディングで CM 系フォ

ントを使いたければ \usepackage{newtxtext,newtxmath} の代わりに \usepackage{lmodern} とすれば良い。

発展： fontenc/inputenc とは何か

先ほど「さしあたり」と圏点を付けてやや強調しておいたように，現段階での欧文（のみ）処理は fontenc を T1 とし inputenc を utf8 として作成した入力ファイルに対し pdflatex コマンドを発行することで（pdfTEX エンジンを使って）行う。その際使用するラテン文字は Times/Palatino である。しかし，pdflatex コマンドで用いられる pdfTEX エンジンは，後で解説する upTEX エンジン同様，デフォルトではそもそも UTF-8 での入力を受け付けていない。

こうした制約を回避するため，ASCII 文字コードに存在しない欧文文字を 8 ビット拡張してエンコーディングさせるのが fontenc であり，こうした欧文文字を UTF-8 で直接入力できるようにするのが inputenc パッケージである。こうすることで TEX エンジンが ASCII コードを超える文字（言語）を取り扱えるようになる。

例えば，fontenc のオプション部には T1 の他に，T2A（ロシア語などのキリル文字），LGR（ギリシア語などのギリシア文字）等々といった引数を，inputenc のオプション部には latin1（ラテン文字第 1 部と呼ばれる主に西欧諸国で用いられるラテン用字系，ISO-8859-1 文字コードと同じ），koi-8r（主にロシア語で用いられるキリル文字を取り扱う文字コード，ブルガリア語も扱える）等々といった引数を，それぞれ与えることができる。

しかし，実際に我々が「日本語，ドイツ語，フランス語，英語」を基底語とする文書を作成する場合，fontenc や inputenc のオプション引数をいちいち意識する必要はほぼ全くない。この間の事柄は，Babel という多言語処理パッケージが全て自動的に処理してくれるからだ。

さらに言えば，現在では TEX そのものを根底からユニコード拡張した XƎTEX や LuaTEX といった TEX の代替エンジンの使用も徐々に広まりつつある（Lua は軽量の汎用スクリプト言語）。XƎTEX や LuaTEX 用の多言語処理パッケージ Polyglossia の開発も進められている（ただし XƎTEX や LuaTEX においても Babel は使える）。（近い）将来的には，TEX を fontenc/inputenc といった助けを借りながら用いるよりも，XƎTEX や LuaTEX を使って「ユニコード・ネイティブ」での多言語処理を行う方式が優勢となっていくように思われる。

邦文に関しては，これまで TEX を土台とした日本語処理の仕組みや知見が膨大に蓄積されてきており，それにより高度な邦文組版レベルも実現されてきたという経緯もあり，XƎTEX や LuaTEX を使って従来レベルでの邦文組版処理を行うことは中々難しかったが，現在では LuaTEX-ja プロジェクトにより LuaTEX を用いてこれまでと遜色のない日本語組版ができるようになりつつある。いずれは，欧文（を基底語とするテキスト）処理のみならず，邦文（を基底語とするテキスト）処理においてもユニコード・ネイティブでの多言語処理が当たり前となっていくことだろう。

しかし，ユニコード・ネイティブでの多言語処理を，出力時における「美文書」という観点から考える場合，「フォント」（同じデザインで統一された一揃いの書体データ）の整合性をも顧慮することが不可避となる。具体的にはラテン文字から成る A フォントおよびギリシア文字から成る B フォントがあった場合，各フォントがそれぞれ取り扱う文字集合の範囲内でどれほど美しく包括的であるとしても，A と B を組み合わせた場合にはどうなのか，さらには，邦文と共に用いる場合はどうなのか，ということが問題になってくるということだ。

世界中の言語表記に用いられる広範な文字の全てを，UTF-8 エンコーディングの同一書体で，しかもウェイト（文字の線の太さ）やシェイプ（Upright, *Italic*, Small Caps といった文字変種）まで併せて取り扱えるようなフォントは，まだ存在していない。しかし包括的なフリーの欧文フォント（ラテン文字，ギリシア文字，キリル文字，ヘブライ文字）の提供を目指す Linux Libertine（フリーのインターネット百科事典 Wikipedia の欧文題字にも用いられているフォント）や，欧文を超えて CJK (Chinese/Japanese/Korean) をはじめ世界中の言語表記文字をサポートすることを目標に Google によって開発が進められているオープンソースのフォント（ファミリー）である Noto (No more Tofu) は明らかにこうした方向を目指している。

Linux Libertine や Noto を XƎTEX や LuaTEX で用いる方法については，ヨーロッパ学 ICT IIA の講義で詳しく取り扱う。因みに，本「ヨーロッパ学 ICT 講義テキスト」の欧文（ラテン文字，キリル文字，ギリシア文字，ヘブライ文字）には，少数の例外を除いて Linux Libertine フォントを用いている。欧文フォントにおけるデザイン上の統一感はそこから来ている（邦文にはヒラギノを用いて，原稿のソースファイルは upTEX でコンパイルしている）。

第 14 講

邦文ソースのコンパイル法：セクショニング，ディスプレイ表示，箇条書き

TEX における欧文の入出力の基本を学んだ後は，邦文テキスト処理の仕方を学ぶ。従来 TEX で邦文テキストを扱う場合は，日本語用の機能を追加した pTEX（Publishing TEX の略）が広く用いられてきた。pTEX では漢字・かな・和文記号として JIS 第 1 第 2 水準（JIS X 0208）の範囲しか扱えないため，これを漢字・かな・CJK（日中韓）記号・ハングルとして Unicode の範囲を扱えるように pTEX 内部をユニコード拡張したものが upTEX である。本講義での邦文テキスト処理（ここでいう邦文テキストとは「基底語が邦文」ということであり中身には欧文多言語をも含み得る）では，原則として，upTEX を用いる（正確には upTEX に e-TEX 拡張が施された e-upTEX を使う。e-TEX では「左から右書き」および「右から左書き」言語の混植をサポートする機能 TEX–XETEX が備わっているためである）。

日本語を基底語とする邦文テキストを TEX で処理する場合のソースファイルの書き方を，先ずは演習用ファイル ja.tex を手本として学べ。

documentclass には jsarticle (Japanese Standard Article) を，オプション部には必ず uplatex, dvipdfmx を指定する。現段階では ja.tex に欧文テキストを一切挿入していないが，いずれは日本語を基底語としながらもドイツ語，フランス語，英語等々を混在処理させることになるので，fontenc (T1) と inputenc (utf8) 指定も常用する癖を身に付けておくと良い。

okumacro パッケージを読み込むことで，日本語組版特有の「ルビ」も使えるようにしてある。ルビの入力法も良く確認すること。出力用欧文フォントとしては Times（あるいは Palatino），和文フォントとしては MS 明朝・ゴシック（Windows 端末），ヒラギノ（Mac 端末）が用いられる。

邦文出力では各段落の頭を一字下げる「段落字下げ」を行うのが大原則であるが，TEX で邦文テキスト処理をする場合はプログラムが自動的に段落字下げを行ってくれるので，書き手は「空白行を設けることで段落区切りを指示する」ことだけに意識を払えば良い。

また，書き手が section, subsection 部に番号を振る必要はない（振ってはならない）。これらは TEX が論理的階層に応じて自動処理してくれる。実際のレポートや論文を書く際には section の位置が前後で入れ替わったり，subsection へと降格したり（逆に subsection が section に昇格したり）ということは頻繁に起こることであるから，書き手は文書の論理および階層構造（ここは section レベル，ここは subsection レベル，等々）の違いにのみ注力し文書を作成する。

itemize 環境では記号付き箇条書きが，enumerate 環境では番号付き箇条書きが，それぞれ処理される。いずれも各項目は \item コマンドの後に記す。番号付き箇条書きでは自動連番が振られる。コマンドの後には必ず one space 設ける（コマンドによっては {} をコマンド直後に添えなくてはならない場合もある）ことによってコマンドと本文（地の文）との区切りを明確にする。itemize や enumerate は「入れ子」(Nesting) にしても使える（itemize の中でさらに enumerate を使う，等々）。

レポートや論文中で「引用」を行う場合，短い引用であれば「かぎ括弧」を用いれば良いが，ある程度まとまったテキストの引用をする場合は「ディスプレイ表示」(Displaying) をして，本文（地の文）とは異なる箇所であることを視覚的にも分かるようにする。この場合，引用の中に段落を含まない短めの引用に対しては quote 環境を，引用の中に段落が含まれるような長めの引用に対しては quotation 環境を，それぞれ用いる。

演習用ファイル ja.tex の中身を精査した後は，邦文テキストを TEX で処理するための以下のコマンドを学べ。欧文の場合とは異なり，2 パス処理（Two-Pass Procedure）となっていることに注意。

- uplatex *string*（邦文テキストを含む string.tex ソースを TEX でコンパイルする）
- dvipdfmx *string*（上記コマンドによって生成された string.dvi ファイルを string.pdf ファイルへ変換する）

第 14 講演習

演習用ファイル：ja.tex, ja_euro_uptex.tex, ptextest.tex

1. 邦文のみから成るテキスト ja.tex の中身を精査せよ
2. ja.tex を TEX で処理せよ
3. 入力ファイル ja.tex と 出力ファイル ja.pdf を良く比較せよ

4. ja.tex に英・独・仏・露・希語テキストを追加した ja_euro_uptex.tex の中身を精査せよ
5. ja_euro_uptex.tex を TEX で処理せよ（複数回）
6. 入力ファイル ja_euro_uptex.tex と 出力ファイル ja_euro_uptex.pdf を良く比較せよ
7. ja_euro_uptex.tex ファイルに手を入れて，ノンブルを非表示にせよ
8. 同様にヘッダ（ページ上部）にノンブルと柱を出力させよ
9. ja_euro_uptex.tex ファイルに手を入れ，目次や処理日付の出力をドイツ語やフランス語にしてみよ
10. 以下のように ptextest.tex を upTEX ではなく pTEX で処理し，ロシア語および（複式アクセント記法）ギリシア語箇所に不具合が出ることを確認せよ
11. platex ptextest（複数回）
12. dvipdfmx ptextest

ja_euro_uptex.tex では otf パッケージを用いる設定にしてある。otf には OpenType と呼ばれる（従来の PostScript Type 1 と TrueType 形式を包含する）新しいフォント形式の機能を TEX で「全て使い切る」ための仕組みが備わっている。現段階ではその有難みを享受することはない（そこまでの多文字・異体字処理をしないため）が，遅かれ早かれ，いずれはこのパッケージの御世話になるため，今のうちから慣れておきたい。オプションの deluxe は明朝体 3 ウェイト（線の太さ），ゴシック体 3 ウェイト，丸ゴシック体の 7 書体に加えプロポーショナル組みも使えるようにする指定である（もちろん全ての機能を使うためには，これら全ての書体を備えるヒラギノのようなフォント使用が前提となる）。expert を指定すると縦組み，横組み，ルビのそれぞれに「専用の仮名グリフ」（字形）が用いられる。本講義では扱わないが，オプションに multi を指定すれば簡体字（简体），繁體字（繁體），ハングル（한글）も使えるようになる。

また ja_euro_uptex.tex では，Babel で指定する基底言語に拘らずキャプション等の出力を「日本語」とするパッケージ pxbabel を使用し，プリアンブルに \usepackage[japanese]{pxbabel} 指定を書き加えていることに注意。併せてラテン文字，ギリシア文字，キリル文字といった欧文アルファベットを（全角ではなく）欧文（半角）として出力するためのパッケージ pxcjkcat を使用し，オプション部に prefernoncjk 指定をしている。プリアンブルにおける後者の指定は（もし otf パッケージを使用するのであれば）otf 指定よりも「後」に追加すること。

プリアンブルに記した \title, \author, \date コマンドの引数が，本文にある \maketitle, \tableofcontents コマンドによってどのように出力されることになるか，じっくりと確認すること。なお \date コマンドの引数には \today コマンドを用いているが，これによりソースファイルを TEX で処理した日付が Babel の基底言語に応じた言語で出力される。

発展： pTEX および upTEX における邦文処理

pTEX で取り扱える邦文は JIS 第 1 第 2 水準漢字（JIS X 0208）にある文字に限られる（この範囲内にある文字を全て邦文として扱う）が，入力のエンコーディングを UTF-8 とすることで，事実上これらを超える一部のユニコード文字（全部ではない）を「欧文」として処理することもできる。具体的には，ウムラウトやアクセント記号付きラテン文字は JIS X 0208 に含まれていないので，これらの文字が含まれる邦文テキストを fontenc (T1) および inputenc (utf8) 指定により pTEX で処理すれば，「欧文文字」として正しく出力されるのである。しかし，JIS X 0208 にはギリシア文字やキリル文字は含まれてしまっているため，これらの文字は欧文ではなく常に邦文として処理されてしまう（いわゆる全角文字となってしまう）。

一方，邦文処理の内部文字コードをユニコードに拡張した upTEX は，デフォルトで全てのユニコード文字を邦文扱いする。つまり，ギリシア文字やキリル文字はもちろん，ウムラウトやアクセント記号付きラテン文字までをも upTEX では原則として全角文字として出力する。

ただし，upTEX には「各文字を邦文・欧文のどちらとして取り扱うのか」を制御できる機能が備わっているため，実際には，ギリシア文字，キリル文字，ウムラウトやアクセント記号付きラテン文字等々を正しく欧文として処理させることができるのである。このためのパッケージが pxcjkcat で，prefernoncjk オプションを指定することで望む出力が得られる。逆に欧文扱いを一時的に回避したい場合は \withcjktokenforced{…} コマンドを用いる。

ラテン文字表記を超える欧文をも含む邦文処理を視野に入れた場合，pTEX よりも upTEX の優位性は明らかである。本講義で upTEX を邦文（欧文を含む）処理エンジンとして用いる根拠はここにある。

発展： upTEX における欧文の行送り

邦文を文書の基底言語としている ja_euro_uptex.tex では，当然，本文の「行送り」量（TEX では Baselineskip と呼ぶ）は文書全体にわたり邦文テキストに最適化されて処理される。しかし，これでは欧文箇所も邦文用の行送りで組版されることとなるため，まとまった欧文が出現する箇所では，この有様はやはり美的であるとは言えない。

そこで ja_euro_uptex.tex では新しい環境コマンド euro_narrow を定義し，この環境内では欧文に相応しい行送りでテキストが処理されるようにしてみた。ドイツ語，フランス語，ロシア語，（複式アクセント記法）ギリシア語の小節箇所にこの仕掛けを施し，対照のため英語の小節箇所は邦文用行送りとしてある。それぞれの部分の出力を比較してみよ。ただし，段落冒頭における字下げ（Indent）量は邦文仕様（全角 1 文字）のままである。ここをも欧文仕様とするためには，さらなる仕掛けが必要となる。

なお，TEX における新しいコマンドや環境の定義の仕方や既存コマンドの再定義の仕方，およびそれらの用い方については，ヨーロッパ学 ICT IIA の講義で扱う。

発展： 縦書き

ヨーロッパ学 ICT 講義で取り扱うテキスト処理は，欧文多言語と邦文との混在文書を対象とするため「横書き」（横組み）文書を原則とする。邦文箇所における句読点は，欧文とのバランスをも顧慮し，「，（全角コンマ）」と「。（全角マル）」がお勧めである。

もっとも pTEX や upTEX ではソースファイルを「縦書き」（縦組み）で処理させることもできる。この場合は jsarticle の代わりに utarticle をクラス指定する。otf パッケージを使えば，縦組み処理時に，ソースファイル中にある全角コンマと全角ピリオドを自動的に「、（全角テン）」と「。（全角マル）」に変換してくれる。句読点の変更をするだけのためにソースファイルに手を入れる必要は全くない。

発展演習（縦書き）

演習用ファイル： ja_tate.tex

1. ja_tate.tex を TEX で処理してみよ（横書き時と同じコマンド，2 パス処理）

TEX を使った縦組みについては，本講義では「紹介」に留める。実際の縦組みを行う際には，節（小節）番号等を算用数字ではなく漢数字とする必要があるし，「縦中横」で組まれてしまう半角数字を「縦中縦」とする「連数字」処理させる等，色々と工夫をせねばならない点も多いからである。

Homo Ludens 6: MusiXTEX

TEX を使えば美しい楽譜も書ける。具体的には TEX で楽譜を書くための拡張マクロパッケージ MusiXTEX を用いるが，MusiXTEX のマークアップ体系は少々煩雑であるため，実際には前処理プログラム（プリプロセッサ）である PMX を使ってソースファイルを作成することが広く行われている。楽曲に合わせて「歌詞」（多言語も可）をも処理したい場合は，別のプリプロセッサである M-Tx を使う。

もちろん，本格的な記譜法や楽譜作成法を学ぶことが我々の目的ではないので，ここでは「ジュピター」の愛称で知られる「モーツァルト作曲交響曲第 41 番ハ長調第 1 楽章」(K.551) の冒頭 (図 16，57 ページ) 部分を PMX 記法で入力したソースファイルを使い，これを実際にコンパイルしてみることで，11 段から成る同じ総譜を出力してみよう。なお，処理過程で MIDI (Musical Instrument Digital Interface) ファイルも作成されるから，生成物を「音」（電子音ではあるが）で楽しむ（確認する）こともできる。なお，TEX エンジンとしては 24 段までの総譜作成に対応可能な e-TEX 拡張された TEX を用いる。

Homo Ludens 6: 演習

演習用ファイル： k551.pmx

1. ソースファイル k551.pmx がテキストファイルであることを確認せよ（PMX によるソースの書き方を勉強していないので中身は理解できなくてよいが，ここには音符を始めとする最終的な総譜出力に至る全てのデータが書き込まれていることに留意せよ）
2. k551.pmx のあるカレントディレクトリでターミナルを開き，pmxab k551 と打て
3. この段階で k551.mid も作成されるので「音楽」を音で確認してみよ
4. 同ディレクトリ内には k551.tex ができているので etex k551 と打て

5. 次に musixflx k551 と打て（この作業により音符の横幅の割付を自動処理させる）
6. 次に，もう一度，etex k551 と打て
7. 今度は dvips k551 と打て（中間ファイル k551.dvi を PostScript ファイルに変換）
8. 最後に ps2pdf k551.ps と打ち込み PostScript を PDF へ変換する（k551.ps と拡張子も含めて打て）
9. k551.pdf の中身を確認せよ

ターミナルで texdoc musixtex/pmx276/mtx などと打ち込めばそれぞれのマニュアルを参照できる。

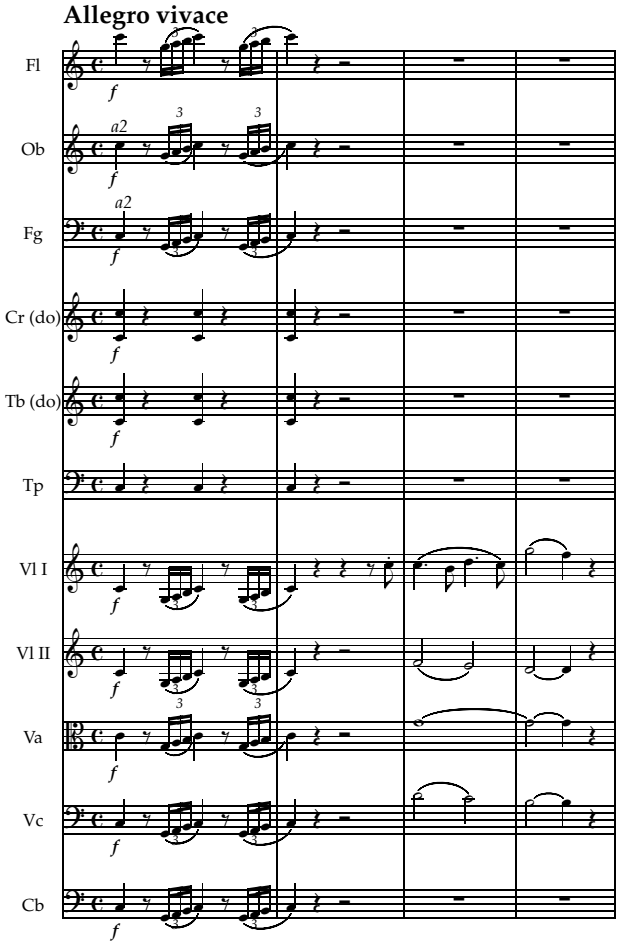


図 16: 交響曲第 41 番「ジュピター」第 1 楽章の冒頭（モーツァルト）

第 15 講

脚注，参考文献

ja_euro_uptex.tex では「引用」のディスプレイ表示に関するコマンドは扱ったが，本来であれば同時に処理すべき「出典」の挙げ方については説明してこなかった。出典は「脚注」で取り扱うことが多いので，本講では「参考文献」と「脚注」を含めた TEX における「文献参照」の仕方について解説する。現段階で紹介する文献参照の仕方は，書き手がソースファイル内に参考文献リストを作り，文献の参照番号はプログラムによって自動的に振らせるという方式であるが，ソースファイルとは別の文献データベース（これもテキストファイル）を用意し，BIBTEX という TEX 関連プログラムを用いて，文献のソート（邦文文献 50 音順，欧文文献アルファベット順）を含め，参考文献リストを自動的に作成させるというもっとスマートかつ高度なやり方も存在する。BIBTEX を使った文献処理の仕方はヨーロッパ学 ICT IIA で扱う。

先ずは，演習用ファイル ja_euro_uptex_bib.tex の末尾を参照せよ。そこには thebibliography という環境コマンドが用いられているが，その横にある 9 という引数は参考文献が「1 桁」冊以内であることを示す参照数値である。同様に 2 桁以内であれば 99，3 桁以内であれば 999 とする（実際には「桁数」のみが重要であるので，9/99/999 の代わりに 3/74/285 などとしても全く同じ効果となる）。

参考文献は，そのままでは目次に載らないが，\addcontentsline{toc}{section}{参考文献} としてやれば section レベルに「参考文献」という見出しをエントリすることができる。小節レベルに載せたい場合は section の代わりに subsection とすれば良い。

因みに，\section や \subsection コマンドを用いると自動的に番号が振られ，これが目次にも載ることになるが，番号を振らずに，しかし見出しは目次に載せたいという場合，同じ記述を用いることができる。この場合は，\section の方も代わりに \section* とアスタリスクを付けておく。

各文献は \bibitem というコマンドの後に記す。この命令は引数として「文献キー」を取り，この文献キーを本文中（脚注も含む）で参照することにより，対応する参考文献「番号」が自動出力される仕組みとなっている。従って，正しい番号出力を得るために文献キーは一意でなくてはならない。

Emacs は thebibliography 環境に入ると，M-RET を押すことで \bibitem を自動入力するという入力支援をしてくれる。その際，同時に，(Optional) Bibitem label: と聞いてくるので（特に必要なければ）そのまま RET を叩く。次に Add key (default none): と尋ねられるので，ここで一意の文献キーを入力する。

邦文書籍のタイトルは『2 重かぎ括弧』で括り，欧文書籍のタイトルは引用符号で括るか「イタリック体」で記すのが原則である。ja_euro_uptex_bib.tex ファイルにおいては，プリアンブルで新コマンド \booktitle を定義し，引数に記した欧文文献タイトルをイタリック体で出力させる仕掛けを施してある。

次は「脚注」についての説明である。脚注は \footnote コマンドを用いて，引数に注を書き入れるだけで良い。後は TEX が脚注の自動連番振りやレイアウトを含め，全て自動で処理してくれる。脚注で文献参照する場合は，\cite コマンドの引数に文献キーを記しておく。こうしておけば TEX で処理する際，この箇所が対応する文献番号で自動置換される。ただし，出力はブラケットで括られた文献番号（つまり数字）のみなので，有意の文言とするためには \footnote{『聖書』\cite{bib_ja} 新約聖書「ヨハネによる福音書」p.189 参照。} や\footnote{Grimm~\cite{khm7}, S.132.} のように書かねばならない。欧文文献関連の注で用いる ~ 記号は，この箇所での改行を抑制する one space を挿入するコマンドである。こうしておかないと，当該箇所で改行が生じた場合，ブラケット付きの文献番号だけが著者名等から切り離されて次行に送られることとなり，読み手にとって読み辛い不親切な書き方（出力）となってしまう。

第 15 講演習

演習用ファイル： ja_euro_uptex_bib.tex

1. 入力ファイル ja_euro_uptex_bib.tex の中身を精査せよ
2. ja_euro_uptex_bib.tex を TEX で処理せよ（複数回）
3. 入力ファイル ja_euro_uptex_bib.tex と出力ファイル ja_euro_uptex_bib.pdf を良く比較し，脚注や参考文献の処理の仕方を学べ

発展： 少しだけ LuaTEX

入力ファイル ja_euro_uptex/_bib.tex を TEX および dvipdfmx で処理することで最終的に得られる出力ファイル ja_euro_uptex/_bib.pdf をよくよく眺めていると，日・英・独・仏語における節・小節番号に用いられている数字フォントと露・希語におけるそれとが異なっていることに気付いたかも知れない。もちろん，英・独・仏語で用いられている欧文フォントと露・希語のそれも異なっている。これは前者では TEX Gyre Termes/Heros フォントが，後者ではロシア語に CM-Super フォントが，ギリシア語に CBgreek フォントがそれぞれ用いられているためである。

我々が作成する実際の原稿においては，基底語を日本語あるいは英独仏等の西欧語としながら 1 節ないしは 1 小節を丸々ロシア語あるいはギリシア語とする，などということはまず考えられないため，これらの多言語を混植する場合でも大きな不具合が出来するとまでは言えない。しかし，フォントの不整合という瑕疵が残ったままとなっている，という点も事実である。

そこで，ここに ja_euro_luatex/_bib.tex というソースファイルを用意してみた。英・独・仏・露・希語で用いられる字体を包括する Linux Libertine という OpenType フォントを使うようにプリアンブルで設定している。このソースをコンパイルするためには LuaTEX のようなエンジンが必要である。upTEX を使って処理する場合とはプリアンブルの書き方も少々異なってくる。現段階では「LuaTEX を少しだけ試してみる」という趣旨であるから，ソースファイルの中身の説明は特に行わない。

ソースファイルを LuaTEX で処理するための以下のコマンドを学べ。欧文，邦文とも同じコマンドである。

- lualatex *string*（LuaTEX 用の string.tex ソースを LuaTEX でコンパイルし string.pdf ファイルを得る）（必要に応じて複数回）

発展演習（LuaTEX）

演習用ファイル： ja_euro_luatex/_bib.tex

1. ja_euro_luatex.tex を LuaTEX でコンパイルせよ
2. ja_euro_luatex_bib.tex を LuaTEX でコンパイルせよ
3. ja_euro_luatex/_bib.pdf を 第 14, 15 講で用いた ja_euro_uptex/_bib.pdf と比較してみよ

（ヨーロッパ学 ICT 講義テキスト（IIA/IIB）に続く）