計算機科学'11 ＃1 — テキストと文書

久野　靖*

2011.5.24

1　文書作成

1.1　テキストと文書の違いと位置付け

計算機はその初期においては、文字通り数値を大量に「計算」するための装置として使われて来ました。しかし、人間の知的活動のうち、数値によって表される部分は実はごく一部分であって、ほとんどの知的活動の表現は**文書** (書籍、レポート、手紙、…) によっています。ですから、計算機の適用範囲が広がり、各種の「情報」を取り扱う装置として位置付けられるようになるにつれ、計算機で文書を取り扱いたいという要求が現われて来たのは当然のことです。

たとえば、この科目で我々が題材として扱っている Unix システム自体も、1970 年ごろにベル研究所で「美しい文書を作成し印刷するための」ソフトを作る土台として開発されたものです (それだけが目的だったというわけではありませんが)。現在でも、整った文書を作成するためのソフトである**ワープロソフト** (や、後で説明する**文書作成系**) は、計算機において最も多く使われているアプリケーションソフトの 1 つだといえるでしょう。

また、作成ということだけでなく、情報の流通や利用という面まで考えると、マルチメディア (音声、画像、動画など) が普及した今日の計算機システムやネットワークにおいても、取り扱う情報の大部分は依然として**テキスト**の情報です (図 1)。

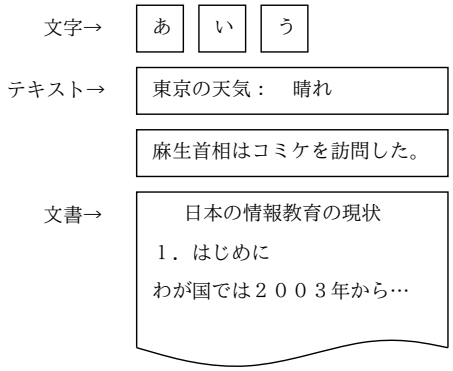


図 1: 文字/テキスト/文書

テキスト情報とは「文字で表された情報」つまり意味のある言葉や文章など、自然言語 (日本語、英語など) で表現された情報を意味します。我々は多くの情報を言葉で表し扱いますから (思考、伝達、記録など)、そのためテキスト情報が大きな位置を占めるのは当然なわけです。

*経営システム科学専攻

次に、テキストと**文書 (ドキュメント)** の違いはどうでしょうか。文書もテキストの情報が土台となっていることは確かですが、まとまった内容について扱うもので、分量が多く、そのため見やすい整形や章立てなどが必要な程度のもの、と考えるのがよさそうです。本、書類、この資料など、ほとんどのまとまったテキスト情報は文書です。[1]

では、コンピュータ上でテキスト情報を扱うソフトウェアとしてはどのようなものがあるでしょうか (図 2)。真っ先に「ワープロソフト」が思い浮かぶかも知れませんが、ワープロソフトと後で出て来る整形系は「文書」を作成するためのソフトということでちょっと別格と考えておきます。

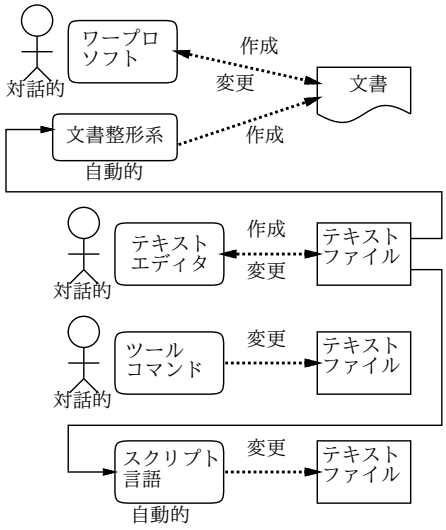


図 2: テキストを扱うソフトウェア

単純なテキスト情報を対話的に入力/編集するためのソフトウェアを**テキストエディタ**と言います。我々が使っている Emacs がその代表格ですが、Windows であれば「メモ帳」、Macintosh であれば「シンプルテキスト」などそれぞれの環境で扱えるエディタが 1 つ以上備わっているはずです (これがないと設定ファイルを変更するなどの作業ができないので困ります)。

テキストエディタでできることは何でしょうか? 単純なテキスト情報は要するに文字の並びですから、これらの文字を挿入/削除/置換/移動するなどの機能があれば一応の用事は足ります。メモ帳などの基本的なエディタはこれらを行う最低限の機能だけを持っています。一方、Emacs には非常に多くの機能が備わっていますが、これはひとくちにテキストファイルといってもさまざな用途のものがあり、それらの用途ごとにその用途に合わせた「便利な」機能を追加しようとするとどんどん機能が多くなる、という面があるからです。

計算機科学基礎で Unix のいくつかのフィルタを扱いましたが、これらの多くはテキストを加工するものですから、テキスト情報を扱うプログラムということになります。ただし残念ながら、Unix オリジナルのものは日本語に対応していないためと、単語が空白で区切られているという英語の形式を前提としていることから、我々日本人にはいまいち使いでがないという面があります。

また、計算機科学基礎では**シェルスクリプト**について扱いましたが、今日ではそこから発展した、**スクリプト言語**と呼ばれる簡便な言語による記述でさまざまなデータの加工を行うことが一般的です。スクリプト言語とテキスト処理については回を改めて扱います。

[1] 電話帳や辞書はまとまったテキスト情報ですがまとまった 1 つの内容というわけではないので、文書とは考えないのが普通でしょう。

1.2 文書作成系

テキストについて述べましたが、実際に我々がまとまった情報を入手する場合はテキストというより文書を読むことになりますから、それが美しく読みやすいことは大切だと言えます。

そこで、以下では計算機で文書を扱うに際して「美しい文書とはどういうものか」「それをうまく作るにはどのようにシステムを設計したらいいのか」について検討してみましょう。そうすることで、考えずにひたすらワープロソフトを使うだけの人よりもずっとスマートにことを運ぶことができるはずだからです。

では最初に、「美しい文書」について考えてみることにしましょう。次の問いに対してあなたはすぐに答えを思いつきますか?

- そもそも、美しい文書とはどういうものを言うのか?
- なぜ美しい文書が望ましいのか?

1 番目の問いに対して「文字がきれいな字で打たれている」とか「多様な字形 (フォント) が使われてる」などの回答が浮かんだ人もいるかと思います。しかしそもそも、文字がきれいでもそれがでたらめな規則で並べられていたりしたら (たとえば 1 文字ごとにフォントが切り替わっているなど)、1 文字 1 文字を見て「ほう、この『あ』という文字は美しいですねえ」と鑑賞 (?) はできても、その文書を読んで理解する上ではちっとも嬉しくないはずです。

そこで 2 番目の問いが問題になります。なぜ美しい文書が望ましいのでしょう? それは (美しさを鑑賞するというあまりありそうにない回答はおいておいて)、文書が美しいとその内容を見て取る効率がよいから、ということになります。それには、文字の大きさが読みやすいなどの特性に加えて、「ここは見出し」「これは図」などの文章の構造が見てとりやすく、必要なことがらが (たとえば見出しをざっとスキャンしていって) すぐ探せる、などの点も重要になります。これを整理すると次のようにまとめられます:

- 文書の構造が読み手に的確に伝わること。

逆の面から見ると、「美しい文書」を扱うことができたとすれば、そこに含まれる情報は、その内容を表すテキスト (文字) だけではなく、その文書を美しくあらせるための**付加情報**までを含めたものになっているはずです。なぜなら、この付加情報があってはじめて、どの部分はどういう風にする (たとえば見出しだから目立つ字体にする)、という処理が可能になるからです。同じ大きさの文字だけから成るプレーンテキストファイルと、ワープロなどで美しく仕上げた文書との違いは、この付加情報の部分にあるといえます。

1.3 文書整形のアルゴリズム

ワープロソフト (や文書整形系) が「どのように」各文字を紙面に配置しているかを言葉で説明できますか? たとえば、初期のワープロソフトなどでは「原稿用紙」モデル、つまり画面や紙の上に縦横のサイズが決まった枡目があり、そこに 1 文字ずつ文字を詰めて行く、というアルゴリズムが使われていました。それだとどんな美しくないことが起きるか分かりますか?

- 禁則処理 (行頭に「。」などが来ては行けない等の処理) を行った結果、右端が「でこぼこ」になる。
- 英字を日本語と同じ文字間隔で詰めるとえらく間延びになる。かといって日本語 1 文字ぶんに英字 2 文字を詰めると窮屈 (しかも英字が奇数だと半分のアキが…)。

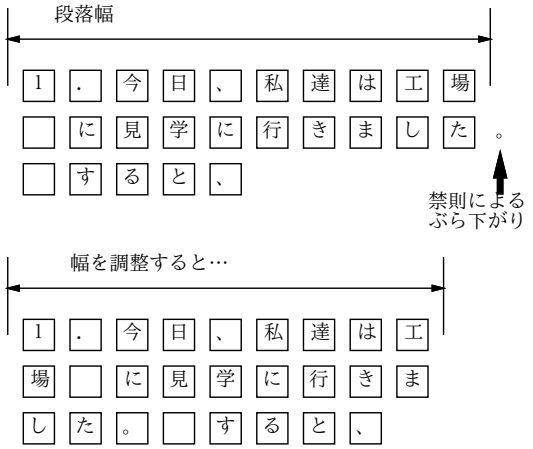


図 3: 「原稿用紙方式」の困った点

さらに箇条書きなど字下げがある段落を書いてから文字詰め幅を変更すると、空白がずれて悲惨なことになります (図 3)。

今日のソフトではこのようなことはなく、段落は「1 行目の左マージン」「2 行目以降の左マージン」「右マージン」の位置で形が指定されており、文字を普通に打って行くことでこれらのマージン内で自動的に文字が詰め合わされて行くようになっています (図 4)。

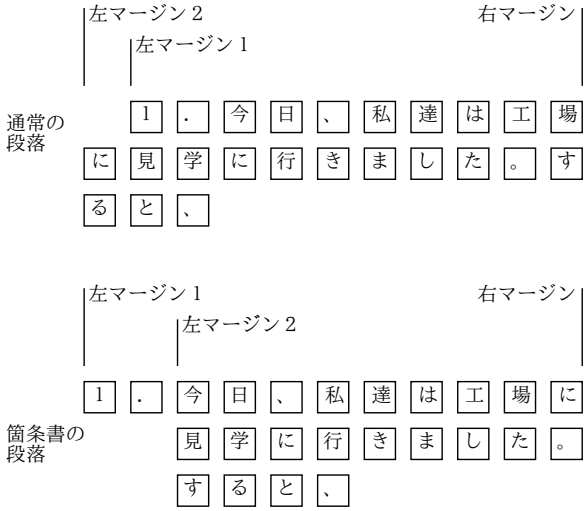


図 4: マージン指定による段落の詰め合わせ

ただし、これをちゃんと使うためには、段落の形を変える時に「ルーラ」を出してマージンを設定しないといけません。もしかして、あなたは今でも字下げをするのに空白を挿入していますか? (とすると、今でも「悲惨な目」に逢っていることになりますね…)

さらに、禁則などの扱いも含めてきれいに整形するため、現在では次のようなアルゴリズムが使われています。

- 段落ごとに「詰め合わせる幅」があり、それぞれの文字は「四角い箱」と考えてその箱を幅一杯まで詰めていく。
- 文字と文字の組み合わせごとに適切な「あき」が変わってくるので、組み合わせごとに適切な「にかわ (のり)」を詰めていく。
- 禁則処理などでその位置で行かえできないなら、少し余分に詰めるか詰めたものを取り除く。

- 右端が綺麗に揃うように全体を詰めたり伸ばしたりする。伸ばすとにかわの部分が伸びて全体が長くなる。
- このとき、すべてのにかわが均等に伸びるわけではなく、「(」の左、「)」の右、「。」の右など、伸びてもおかしくないところが余分に伸びるようにする。
- このようにして各行が出来てくると、今度はページにその行を上から詰めていく。ここでも「にかわ」がはさまれ、ページの切れ目の禁則 (節タイトルがページ下端に来ないとか) に応じて同様の処理が行われる。

次節で説明する TeX 系列のソフトウェアも、このような整形アルゴリズムが使われているわけです。

ここで重要なのは、「文字どうしのアキ (にかわの幅や伸び)」「段落間のアキ」「マージンの値」などをどう調整すれば見やすい文書になるか、ということです。これについては、長い歴史のある印刷・出版界が多くのノウハウを持っており、それを駆使して読みやすい書籍を作っています。そこで、latex などの整形システムでもこれらのノウハウを借りて来てパラメタを調整することで、できるだけ見やすい整形となるように努めているわけです。

ここでなぜ後述する「意味づけ方式」が重要になるかがお分かりだと思います。ユーザがいちいちこれらのパラメタを指定して調整しているようでは、そのユーザが上記のような出版のノウハウを持っていない限り、美しい整形レイアウトは困難です (そもそも素人になんとなくでできるなら出版レイアウトの専門家など存在しないはずですから)。しかし、ソフト側では「ここは箇条書き」「ここはタイトル」などの情報をユーザに教えてもらわなければ、どこが何なのかは分からない (「ここはたまたま強調のため太字にしただけかも知れない?!」) ため、せっかくのノウハウが活かせないわけです。これに対し、意味づけ方式で「何であるか」が記述されていれば、ソフト側でその情報をもとに適切なパラメタを使用して見やすい整形が行えるわけです。

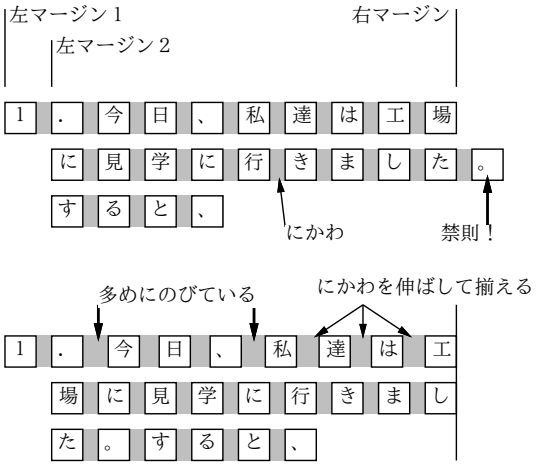


図 5: 整形のアルゴリズム

1.4 文字の配置に関するノウハウ

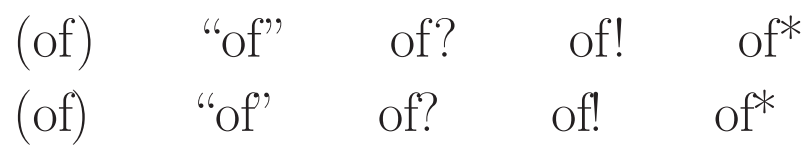
整形のノウハウの中で、とくに英語など少数のアルファベットを組み合わせる言語に重要なものとして、文字の「くっつき方」に関する次の 2 つがあります。

- **カーニング** (kerning) — 2 つの文字の間のくっつき具合を調整すること
- **リガチャ**(ligature、合字) — 特別な複数文字の組み合わせの場合、それらが「合わさった」文字を使うこと

カーニングについては、次の例を見てください。

W や T は上が広く、A は下が広い文字なので、これらを普通に並べると間延びして見えます。そこで、これらの文字が並んだときは「食い込んで」配置するわけです。右側に縦線を挿入したものをつけたので、どれくらい食い込んでいるか分かると思います。

また、このような文字どうしだけでなく記号と隣接する場合もカーニングが問題になります。たとえば小文字の「f」は右上が出っぱっているので、次の記号の形によってはうまく離さないと下のように「くっついて」しまいます。



リガチャは「f+i」「f+l」「f+f」などの特定の文字の組にくっついた文字を使うことを言います。

よく見ると、左では「くっついた文字」が使われているのが分かると思います。右側はリガチャを止めた (普通の文字を隣接させた) 場合です。

それが何か? と思ったかも知れませんが、英語圏の出版文化ではこれらの処理をきちんと行ってはじめて「まっとうな」文書と考えられるので、けっこう重要です。ちなみに、latex などの組版ソフトはまさにそのために頑張っているので、きちんとカーニングやリガチャを扱っています。一方、Word などのワープロソフトは場合によってはかなり問題があります (その筋の人たちがよく文句を言っています)。

1.5 見たまま方式とマークアップ方式

美しい文書を作成するためには、テキストと付加情報をともに入力したり修正するなどの必要があることまでは分かりました。ではそれを具体的にどのようにして行なったらよいでしょうか? 計算機の世界では、その方法として次の 2 種類が考えられ、現に使用されています (図 6)。

- 見たまま方式、または **WYSIWYG**(What You See Is What You Get — 「あなたが見るものがあなたの得るものである」) 方式。この方式では、実際にプリンタで印刷したイメージにできるだけ近い画像を生成し、画面で表示します。そこでマウスやメニューなどを用いて、画面上に見える文書を対象として修正を施します。配置やフォントを変更したり、文章を直したりすると、その結果は直ちに画面に反映されます。多くの人が使っている MS Word などもこの仲間です。
- **マークアップ方式** — これはエディタを用いて作成するファイルの中に、テキストに混ぜて特別な「印」(マークアップ) を入れておき、これに従って**整形系** (**文書フォーマッタ**) が整形を実行するものです。この方式では、さまざまなマークアップを指定することで非常に細かい制御まで行なうことができますが、マークアップを覚えるまでに手間がかかりますし、整形系を経て紙に出すまでどういう出力になるか見えないという弱点があります (紙に出す代わりに画面で確認する**プレビューア**を使えることが普通ですが)。マークアップ方式は、さらに次の 2 種類に分かれます。

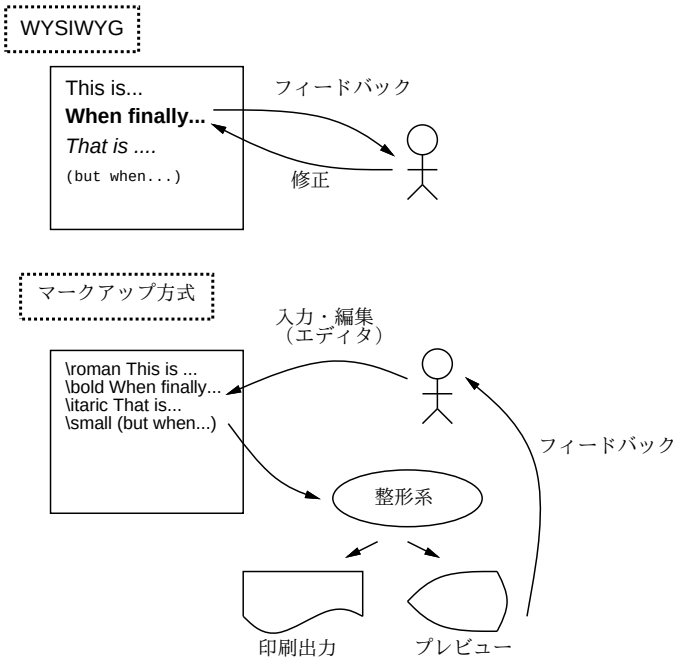


図 6: WYSIWYG 方式とマークアップ方式

- **コマンド方式** — マークアップが「ここからこういうフォント」「ここで改ページ」などのコマンドになっていて、その位置で命令を実行するようにして整形を行うもの。非常に古くからあり、**troff**、**TeX** などがこの代表です。
- **意味づけ方式** (semantic encoding) — 同じマークアップでも、「どう整形する」というコマンドではなく「ここは章の切れ目」「ここからここまでは箇条書き」など、文書の**意味**を表す印を入れておくものです。コマンド方式より覚えやすく、またひと通りの印つけに対して複数通りの整形方法を対応させることも可能です (たとえば 2 段組用、大きな文字用など)。この後でとりあげる **latex** や WWW のところで説明した HTML はともにこれに属します。

MS Word の隆盛を見れば分かるように、世の中のワープロソフトでは見たまま方式が主流なのに、この章ではマークアップ方式を取り上げる、その理由について説明しておきましょう。

- 見たまま方式は、文書の個別の個所ごとに「こういうスタイル」という指定を繰り返し行うのにマウスやメニューと格闘してかなりの手間が掛かり、かなり大変です。マークアップなら、ここはこういうマークアップ、ということを覚えればあとは単に「打ち込む」作業の一部なので手間はそれほどではありません。
- 見たまま方式では、(大抵は)「こういう体裁にしよう」ということを人間がデザインして制御するのでそういうセンスがないと美しくできません。意味づけ型では、スタイルはある程度標準化されているのでそのようなことに頭を悩ますことはありません。(WYSIWYG でもスタイルシート機能を活用すればいいはずなのですが、それを使っている人を見たことがありません。それはたぶん、WYSIWYG ではどうしても「ここはこのフォントでこの大きさ」と指示したくなるからではないかと思います。)
- 見たまま方式は、ある 1 つの体裁に合わせて文書を作ってしまうので、あとから体裁を変更するのは大変です。意味づけ型のマークアップなら、整形時のスタイルを複数種類用意することで、多様な体裁に対応できます。

- 見たまま方式は、ファイル形式がある特定のツールに依存するので、ツール間でテキストを流通しにくいという弱点があります。テキストファイルにしてやればもちろん他のツールに持っていけますが、付加情報はすべて失われてしまいます。マークアップはもともと全部テキストファイルなので、(印を統一的に書き換えるなどの処理は必要になるとしても) テキストの流通が容易です。[2]

「単に文書を作るのにいちいちコマンドを調べるなどとても耐えられない」「そんな面倒な方式は旧態依然だ」と思ったでしょうか? しかし実は、大量に文書を作成するプロほど、latex のような整形系をメインに使っています。それは「計算機科学基礎」でやったコマンド対メニューのように、ワープロソフトだと一見分かりやすくても操作に時間が掛かってしまうので、大量に文書を作る人にとっては結局コマンドを打つ方が楽で効率的だからです。また、latex は美しいスタイルのためのチューニングが徹底しているので、「おまかせ」で美しい文書ができるという利点も見逃せません (Word ではカーニングなどの処理に問題があることは先に述べました)。

というわけで、今回は latex を「とにかく」体験してみて頂くことにします。今後使い続けるかどうかは各自の選択ですが、どういうものかは知っておくべきだと思いますので。

1.6 latex を動かしてみる

latex のコマンドの書き方は少し後回しにして、先にサンプルファイルを「そのまま」整形する方を体験していただきましょう。latex に入力するファイルは「`.tex`」で終わる名前をつけますから、たとえばこのファイル名が「`sample.tex`」であるものとして以下説明します。

整形のためのコマンドは **platex** ですので、[3]入力ファイルを指定してこれを起動します:

- `platex` ソースファイル — pLaTeX による整形

実際に動かすといろいろ細かいメッセージが出力されますが、エラー以外は当面無視して構いません。

```
% platex sample.tex | nkf      ← nkf はメッセージ日本語表示に必要
This is pTeX, Version 3.141592-p3.1.9 (euc) (Web2C 7.5.4)
(./sample.tex
pLaTeX2e <2006/01/04>+0 (based on LaTeX2e <2003/12/01> patch level 0)
(/usr/local/ptex-3.1.9/share/texmf/ptex/platex/base/jarticle.cls
Document Class: jarticle 2002/04/09 v1.4 Standard pLaTeX class
(/usr/local/ptex-3.1.9/share/texmf/ptex/platex/base/jsize10.clo))
(./sample.aux)
 (/usr/local/ptex-3.1.9/share/texmf-dist/tex/latex/base/omscmr.fd) [1]
[2] [3] (./sample.aux) )
Output written on sample.dvi (3 pages, 7456 bytes).
Transcript written on sample.log.
%
```

後で自分でコマンドを打つときは、もしエラーがあると途中で「`?`」というプロンプトが出て処理が止まります。その時はとりあえず「`x[RET]`」を打ち込んで実行を終わらせ、エラーメッセージをよく見て間違っているところを直し、再度走らせてください。整形が成功すると`.dvi` で終わる名前のファイル (**DVI 形式ファイル**、この場合だと `sample.dvi`) ができているはずです。

これをいきなり紙に出力してもよいのですが、普通はまず画面でできばえを確認します。それにはプレビューア **xdvi** を使ってください:

- `xdvi` ファイル — DVI 形式のプレビュー

[2]この弱点に対しては、見たまま方式のツールでも保存したファイルが後述する XML などマークアップ型の形式であるようにする、という方法が今後普及する可能性があります。

[3]ほかに **jlatex** などのコマンドを使うサイトもあります。

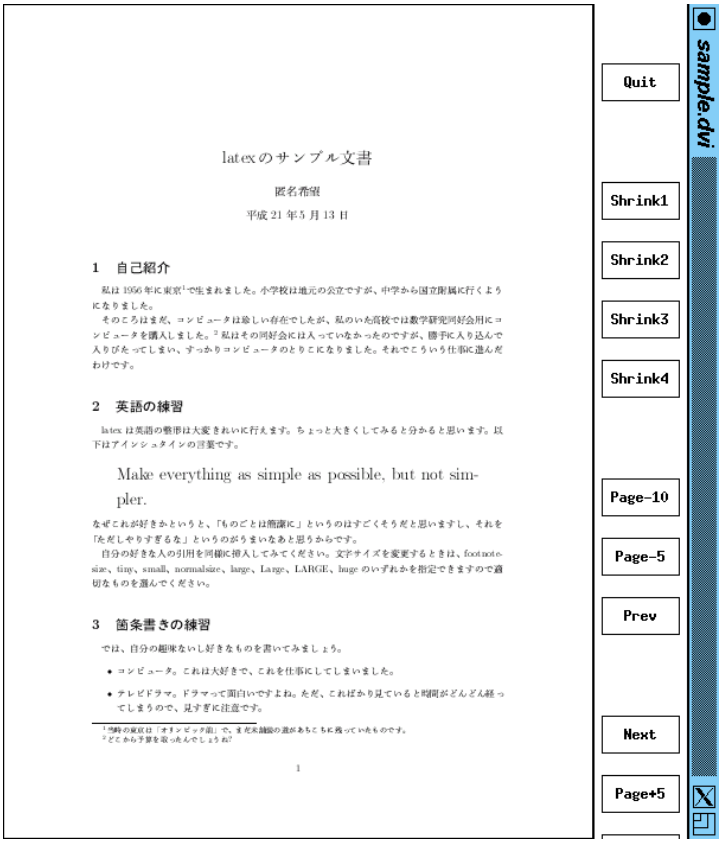


図 7: XDVI によるプレビュー

xdvi の画面を図 7 に示します。右側に縮尺やページめくりを指定するボタンが並んでいるので、これらのボタンをマウスで操作して各ページが意図どおりかどうか確認します。

プリンタに出す場合には DVI 形式から PostScript 形式に変換するために **dvips** コマンドを使います: [4]

- `dvips -o PS ファイル DVI ファイル` — DVI 形式を PostScript に変換

PostScript 形式ファイルはそのまま PostScript プリンタに送れば印刷されます。そのほか、PostScript ビュアで見たりすることもできま

```
% dvips -o sample.ps  sample    ← PS に変換 (「.dvi」は省略可)
This is dvips(k) p1.7 Copyright 2005 ASCII Corp.(www-ptex@ascii.co.jp)
based on dvips(k) 5.95a Copyright 2005 Radical Eye Software
(www.radicaleye.com) ' TeX output 2006.04.13:1544' -> sample.ps
<tex.pro><texps.pro>. <cmr10.pfb>[1]
% gv sample.ps &                ← PS ビュアで見る
% lpr -Pmyprinter sample.ps     ←プリンタ出力
```

また、**dvipdf** コマンドにより **PDF** 形式に変換することもできます。

- `dvipdf DVI ファイル` — DVI 形式を PDF に変換

```
% dvipdf sample          ← PDF に変換 (「.dvi」は省略可)
% acroread sample.pdf    ← Acrobat Reader で見る
```

[4]なお、日本語が扱える latex としては platex のほかに **jlatex** というものもあり、こちらでは **dvi2ps** という変換コマンドを使います。

ここに示すように、latex の文書はまずこの文書がどんなスタイルの文書であるかを宣言する部分から始まる。スタイルの例としては「記事 (jarticle)」、「本 (jbook)」、「報告 (jreport)」などがある。ここでは一番簡便な jarticle を例に使用している。これに加えて字の大きさ、図形の取り込み機能などをオプションとして指定できる。

続いて「文書開始」の宣言があり、この中に文書の本体が入る。この部分には様々な記述が可能だが、一番簡単にはここに示すように段落ごとに 1 行あけて次々に文章を書いて行くだけでも地の文が普通にできる。つまり、「特に指定がない」ならば「地の文」である。

あとは文書の本体が終わったら最後に必ず「文書終了」の宣言がある。最低限必要なのはたったこれだけである。

図 8: pLaTeX の出力

PDF 形式は Windows や Mac でも広く使われていますので、これならどこの PC に持って行っても見たり印刷できます。

大変だったでしょうか? 本章冒頭でも書いたように、最初はとっつきにくいと思いますが…ではたとえば、一番最初の行を次のように直すとどういうことが起きるか試してみてください。

```
\documentclass[12pt,a4j]{jarticle}       ←フォントサイズ変更
\documentclass[twocolumn,a4j]{jarticle}  ← 2 段組みに変更
```

このような取り換えが簡単なのが意味づけ方式の利点だということが理解して頂けるかと思います。

2 latex 入門

2.1 文書の基本構造

この節では、意味付け方式の文書整形システムの一つである **latex** (コマンド方式の TeX を拡張する形で構成されています) について、実例中心で一通り解説することを通じて、意味付け方式 (と広義の指令方式) というのはどんなものかの感触を持っていただこうと思います。まずともかく、latex の文書に最低限必要な基本構造の例を見ていただきましょう。次のようなものは、一つの完結した latex 文書です (内容も読んでみてください):

```
\documentclass{jarticle}
\begin{document}
ここに示すように、latex の文書はまずこの文書がどんなスタイルの文書である
かを宣言する部分から始まる。スタイルの例としては「記事 (jarticle)」、
「本 (jbook)」、「報告 (jreport)」などがある。ここでは一番簡便な
jarticle を例に使用している。これに加えて字の大きさ、図形の取り込み
機能などをオプションとして指定できる。

続いて「文書開始」の宣言があり、この中に文書の本体が入る。この
部分には様々な記述が可能だが、一番簡単にはここに示すように段落ごとに 1 行
あけて次々に文章を書いて行くだけでも地の文が普通にできる。つまり、
「特に指定がない」ならば「地の文」である。

あとは文書の本体が終わったら最後に必ず「文書終了」の宣言がある。
最低限必要なのはたったこれだけである。
\end{document}
```

これを latex に掛けて打ち出すと図 8 のようになります。ところで、ここで少し補足しておくと、まず宣言 (指令) は

```
\指令名 [オプション指定]{パラメタ}
```

のような形をしています。ここでオプション指定がないときは [...] はなく、またパラメタがないときはさらに{...}も不要です。ということは\はそのままでは文書に含められないわけです。LaTeXでは、このような特別な (そのままでは使えない) 記号としては次のものがあります。[5]

```
# $ % & ~ _ ^ \ { }
```

これらはとりあえず「そのままでは使わない」ようにしてください。

2.2 表題、章、節

いきなり本文が始まって延々と続くだけではあんまりですから、表題と章立てをつけましょう。その場合の例を次に示します:

```
\documentclass{jarticle}
\begin{document}

\title{あなたがつけた表題}
\author{書いた人の名前}
\maketitle

\section{表題について}
表題をつけるには title、author など必要な事項を複数指定した後、最後
に maketitle というとそれらの情報をもとに表題が生成される。その際指
定されなかった事項は出力されないかまたは適当なものが自動生成される
(たとえば日付を指定しないと整形した日付が入る。)

\section{章立てについて}
ここにあるように section 指令を使って各節の始まり、およびその表題を
指定する。節の中でさらに分ける場合には section というのも使え、
さらに subsection というのまで可能である。ちなみに jbook/jreport スタイル
では section の上位に chapter があるが jarticle では section からである。
ところで、節番号 etc. は自動的に番号付けされるのに注意。

\section{より細かいことは}
より細かいことは、参考書を参照してください。

\end{document}
```

これを整形すると、文書の冒頭にタイトル、著者、日付 (整形した日付) がそれなりの形式で用意され、見出しも前後にアキを取ってそれなりのフォントで出力されます。

2.3 いくつかの便利な環境

表題と章建てができればこれだけで結構普通の文書は書けてしまうはずですが、しかし全部地の文ではめりはりがつきません。それに、プログラム例など行単位でできているものまできれいに詰め合わされてしまうのでは困りますね。このように、部分的にスタイルが違う部分を指定するのには次のような形 (**環境**と呼びます) を使います:

```
\begin{環境名}
....
....
\end{環境名}
```

では、よく使う環境について、説明していきましょう。まず、**verbatim 環境** (そのまま) というのは文字通り入力をそのまま整形せずに埋め込みます。たとえば

[5] この他に<、>など記号自体に特別な意味はないのですが、TeX の標準フォントの関係で出力すると別の字になってしまうものがいくつかあります。

```
\small\begin{verbatim}
This is a pen.
That is a dog.
\end{verbatim}\normalsize
```

は、つぎのような出力になります:

```
This is a pen.
That is a dog.
```

つまり、これはプログラム例などを入れるのに適していますし、その他よく作り方が分からないスタイルはすべて手で整形してこれで用意してしまってもとりあえずは形になります。また、この内側では先に注意した「特殊記号」もそのまま出力されるので使うことができます。

わざわざ別の行にするのでなく、文章のなかに一部「そのまま」を埋め込みたい場合もあります。そのような時には verbatim の類似品で`\verb|....|`という書き方を使うことができます。これで縦棒にはさまれた部分がそのまま出力できます (中に縦棒を含めたい時は、両端に縦棒以外の適当な記号を使ってください)。これも特殊記号が含められます (ただし脚注の中には`\verb|...|`は入れられません)。

つぎに、**itemize 環境** (箇条書き) について説明しましょう。この場合は、環境のなかに複数「`\item ....`」というものが並んだ格好になっていて、その一つずつが箇条書きの 1 項目になります。たとえば

```
\begin{itemize}
\item あるふぁはギリシャ文字の一番目です。
\item ベータはギリシャ文字の二番目です。
\item ガンマはギリシャ文字の三番目です。
\end{itemize}
```

は、つぎのような出力になります:

- あるふぁはギリシャ文字の一番目です。
- ベータはギリシャ文字の二番目です。
- ガンマはギリシャ文字の三番目です。

次に、この itemize を **enumerate 環境** (数え上げ) に変更すると、出力の際に項目ごとに 1, 2, 3... と自動的に番号付けされるようになります。入力はほとんどまったく同じだから出力のみ示します:

1. あるふぁはギリシャ文字の一番目です。
2. ベータはギリシャ文字の二番目です。
3. ガンマはギリシャ文字の三番目です。

点や番号でなくタイトルをつけたい場合には **description 環境** (記述) を使います。これは

```
\begin{description}
\item[あるふぁ] これはギリシャ文字の一番目です。
\item[ベータ] これはギリシャ文字の二番目です。
\item[ガンマ] これはギリシャ文字の三番目です。
\end{description}
```

のように、各タイトルを `[]` の中に指定するもので、上の整形結果は次のようになります:

あるふぁ これはギリシャ文字の一番目です。

ベータ これはギリシャ文字の二番目です。

ガンマ これはギリシャ文字の三番目です。

この他にもいくつか環境がありますが、とりあえずこれくらいで十分使えると思います。

2.4 脚注

脚注の作り方はとても簡単で、単に好きなところに`\footnote{.....}`という形で注記をはさんでおけばそれがページの下に集められて脚注になり、そこへの参照番号は自動的につけられます。[6]

2.5 数式

実は、TeX 属の整形系はもともとは数式を美しく打ち出したい、という目的のもとに開発されたものなので、数式機能はとても充実しています (そのため凝り出すと大変ですから、ここでは簡単に説明します)。まず、数式を文中にはさむときは`$`で囲みます。たとえば、

…と書くと`$x^2 - a_0$`のような具合になります。

と書くと $x^2 - a_0$ のような具合になります。ここで出てきたように、肩字は`^`、添字は`_`で表せます。その他、数学に出てくる記号はたとえば

```
\equiv \partial \subset \bigcap
```

などのように書くと $\equiv \partial \subset \bigcap$ のように出てきます。また、文中ではなく独立した行にしたければ、`$`の代わりに`$$`ではさむか、`\begin{displaymath}` …`end{displaymath}`で囲みます。たとえば

…と書くと`$$f(t) = \sum_{j=1}^m a_j e^{i\lambda_j} t$$`になります。

と書くと

$$f(t) = \sum_{j=1}^{m} a_j e^{i\lambda_j} t$$

になります。このように、肩字や添字のグループ化には`{}`を使います (丸かっこは数式本体のために使います)。もっと網羅的な記号一覧などは TeX の入門書などを参照してください。

あと、複数行にわたる数式を書きたい場合は displaymath 環境の代わりに eqnarray*環境を使います (つまり、`\begin{eqnarray*}` …`end{eqnarray*}`で囲みます。この中では数式の指定に加えて次の 2 つの機能が使えます。

- `\\` 改行 — この印の箇所で改行。
- `&` タブ — 各行のこの印の箇所を縦にそろえる。

たとえば

```
\begin{eqnarray*}
  (x - 1) (x^2 + x + 1) & = & x^3 - 1 \\
  x^3 - 1 & = & (x - 1) (x^2 + x + 1)
\end{eqnarray*}
```

のようにすると

$$\begin{aligned} (x-1)(x^2+x+1) &= x^3-1 \\ x^3-1 &= (x-1)(x^2+x+1) \end{aligned}$$

のようになるわけです (縦に「=」の位置が揃っていることに注意)。

[6] たとえば、こんな具合になりますね。

2.6 表

表は情報を整理して提示する強力なツールです。latex では表は **tabular 環境**で作り出します。その先頭では、

- 表のカラム数
- それぞれのカラムを左/中央/右揃えのどれにするか
- 各カラムの境界および左右に罫線を引くかどうか

を 1 つの文字列で指定します。すなわち、1 つのカラムごとに揃え方を `l`/`c`/`r` のうち 1 文字で指定します (`l`/`c`/`r` の文字数がカラム数になります)。また、これらの文字の間や左右端に「`|`」を挿入すると、そこに縦罫線が入ります。具体例で見てみましょう (**center 環境**は表全体を中央揃えするために使っています):

```
\begin{center}          ←見ばえのため
\begin{tabular}{c|ll}
AND 演算 & 0 & 1 \\
\hline                  ←横罫線
0       & 0 & 0 \\
1       & 0 & 1 \\
\end{tabular}
\end{center}
```

のようにすると、次のように表示されます。見れば分かる通り、表の内部では「`&`」がカラムの区切り、「`\\`」が 1 行終わり、「`\hline`」が横罫線を表します。

AND 演算	0	1
0	0	0
1	0	1

3 HTML+CSS

3.1 WWW とマークアップ言語

LaTeX について学んで、理屈は分かったけれど、やっぱり WYSIWYG で「見たまま」のものが表示される方が分かりやすいし使いやすい、と思われたかも知れません。しかし実は、WYSIWYG が**使えない**ような場合というのも存在します。しかも、皆様がいつも目にしているもの — Web ページがまさに「そういう場合」なのです。どうしてだか分かりますか?

つまりワープロであれば、出力する紙のサイズが決まっていて、その紙に合わせて配置を決めて、文字の大きさなども決めて、その通りに打ち出せばいいので「見たまま」を自由に調整できます。しかし、Web ページはどうでしょうか? Web ページには「紙の大きさ」は存在しませんから、「1 行何文字詰めで文書を作る」という設計がそもそも不可能です。また、マシンによって使えるフォントも文字サイズも変ってきますから、「この表題は MS 明朝の 24 ポイント」と指定しても、そのフォントがないかも知れません。

ではどうすればいいのでしょうか? できることは「ここは表題」「ここは段落」「ここは箇条書き」という指定をしておいて、**ブラウザが画面に表示する瞬間に**窓の幅や使えるフォントに合わせて整形してくれるようにお願いする、つまり意味づけ方式でマークアップするしか方法がないわけです。

「でも私は WYSIWYG ツールで Web ページを作っているが」という人もいるかも知れません。しかし、実はそれは「WYSIWYG みたい」なだけで、本当の WYSIWYG ではないのです。というのは、あなたが使っているマシンと表示能力や画面サイズの違うマシンに行ったら、どのみち「その通りに」表示することは不可能なのですから。

なお、WYSIWYG ツールが無意味だというつもりはありません。とりあえず「自分のマシンならこんな仕上り」という様子を見ながら編集できるのはそれなりに便利だと思います。しかし、Web アプリケーションなどでプログラムから HTML を生成する場合には、どのみち直接 HTML を扱う必要がありますから、HTML+CSS についてどのようなものかを知っておくことはやはり必須だと考えます。

3.2 HTML の概要

HTML(HyperText Markup Language) は、WWW の生みの親 Tim Berners-Lee が設計した言語ですが、実はそれ以前から存在している **SGML**(Standard Generalized Markup Language) の枠組みを利用しています。SGML というのは、troff や Scribe(いずれも TeX より前から存在している整形系) が世の中に出たころ、「このままプログラムごとに全部違ったマークアップ言語が作られていたら世の中はバベルの塔になってしまう」と思った人たちが相談して「どのようなマークアップ言語でも共通に使えるカタチを用意した」ものであり、ISO 規格および JIS 規格になっています。

HTML そのものは SGML を基に作られた「Web ページマークアップ用の言語」であり、HTML 1.0 → HTML 2.0 → HTML 3.2 → HTML 4.0 → HTML 4.01 と変遷してきています。ここでは最終版である HTML 4.01 のみを扱います。

SGML では (そして HTML では) マークアップの部分を**タグ**と呼び、「<名前…>」のように「<」と「>」で囲んであらわします。ここで「名前」はマークアップの種類 (「段落」「見出し」などの区別)、「…」の部分にはそのオプションが指定できます。また、見出しや段落は「範囲」があるので、その範囲の終りを表すのに「</名前>」という形のタグ (終了タグ) を使います。このような囲まれた範囲全体を**要素** (element) と呼びます。まとめると、HTML の要素は次のような形をしています:

```
<名前 オプション…> …内容… </名前>  ←タグが対になる要素
<名前 オプション…>  ←単独タグだけの要素
```

3.3 Web ページを作る

ここからは、実際に HTML の機能を見ていきましょう。「計算機科学基礎」でも HTML については簡単にやりましたが、復習を兼ねて前提なしで最初から説明します。例題は基本的に

```
http://w3in/~kuno/ecs09/html/
```

に置いてあるものと同等ですから (紙面の都合で一部変更)、こちらのサイトを表示して「View → Page Source」でソースも見ながら読むのがおすすめです (今回は Lesson 1 と Lesson 2 をカバーしています)。

はじめての HTML

まずは、一番簡単なページの例からです。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
</head>
<body>
<h1>HTML と特別な文字</h1>

<p>これは HTML のサンプルです。HTML ではタグと呼ばれる「印」で
「どの範囲が」「何であるか」を指定します。</p>

<p>HTML では「&lt;」「&gt;」「&amp;」の 3 つの文字は特別な意味を
```

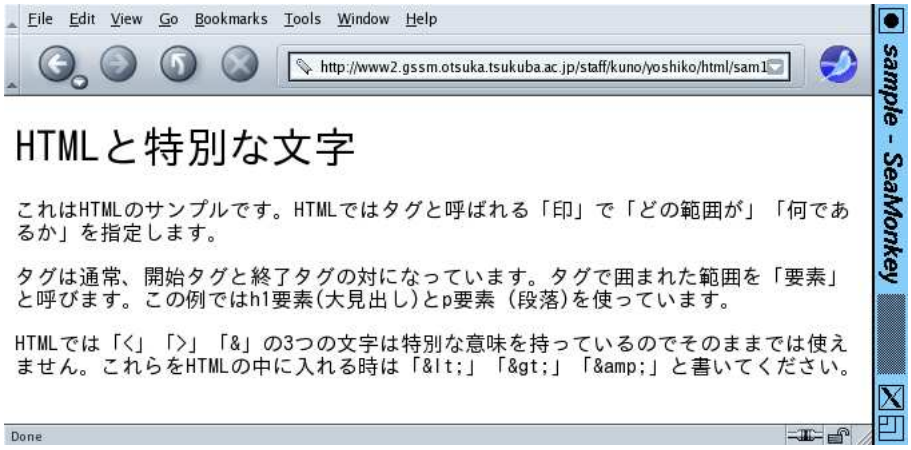


図 9: 最初の練習ページの表示

```
持っているのでそのままでは使えません。これらを HTML の中に入れる
時は「&amp;lt;」「&amp;gt;」「&amp;amp;」と書いてください。</p>
</body>
</html>
```

このファイル内容の説明をしておきます:

- 1 行目は **DOCTYPE 宣言**といい、このファイルが HTML のどのバージョンで記述されているかを明示する。ここでは HTML 4.01 版を指定してある。
- `<html>…</html>` — **html 要素**は、この範囲が 1 つのページ記述であることを表す。
- `<head>…</head>` — **head 要素**は、この範囲がヘッダ情報 (このページがどんなページであるかを示す情報) の記述部分だということを表す。
- `<title>…</title>` — **title 要素**は、この範囲がページのタイトル (ブラウザのタイトルバーなどに表示されたりブックマークに現われる) であることを表す。
- `<body>…</body>` — **body 要素**は、この範囲がページ本体 (ブラウザの窓の内容として見える部分) であることを表す。
- `<h1>…</h1>` — **h1 要素**は、この範囲が第 1 レベルの見出し (大見出し) であることを表す。見出しは`<h1>…<h1>`〜`<h6>…</h6>`の 6 レベルある。
- `<p>…</p>` — **p 要素**は、この範囲が段落であることを表す。

このように、HTML では「要素の中に別の要素が入る」という入れ子構造を多く使います。指定 (マークアップ) の内容は LaTeX とよく似ています (文書のマークアップだから当然といえば当然ですが)。

HTML では「そのままでは使えない特殊記号」は、「<」、「>」、「&」の 3 つです (規格上は「"」もこの仲間なのですが、実際にはこれが問題になることはないようです)。これらの文字を使いたい時は、それぞれ「`<`」、「`>`」、「`&`」、「`&quot;`」のように打ち込んでください。このような記法を HTML(SGML) では**文字エントリ**と呼びます。

ファイルができたら、ブラウザの「ファイルを開く」機能で直接開くか、または自分の Web ディレクトリに置いて Web サーバ経由で眺めてみましょう。確かに見出しは大きい文字で表示されていますし、段落はそれらしく詰め合わされています。ここで、ブラウザの窓の幅を狭くするとどうなるでしょうか? 当然、ブラウザは段落の内容を詰め直してくれます。もちろん、ブラウザや環境が違えば詰め合わせや見え方も変ってきますが、それぞれの環境での見え方を調整するのはブラウザの仕事なわけです。

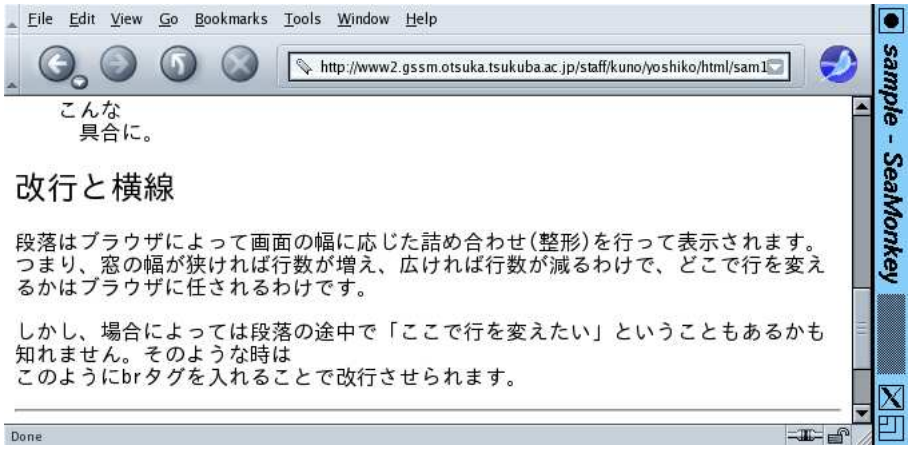


図 10: pre 要素と hr 要素と br 要素

もう少し HTML だけ要素を学んでおきましょう。LaTeX の verbatim(そのまま) のようなものは、HTML では **pre 要素**になります (図 10)。また、ここまで出てきた要素はすべて開始タグと終了タグがあって「範囲を囲む」ものでしたが、そうでないもの、つまり 1 つのタグだけで 1 つの要素になるもの (単独のタグ) もいくつかあります。たとえば、**hr 要素** は 1 つのタグだけで区切りの横線が引かれます。**br 要素**は段落の途中などで強制的に行替えを行うのに使います。

```
<p>プログラムの記述例などは自動的に整形されてしまうと困るので、
pre 要素を使うのが適しています。</p>
<pre>
  たとえば
    こんな
      具合に。
</pre>
<h2>改行と横線</h2>
<p>必要な時は<br>このように br タグを入れることで改行させられます。</p>
<hr>
```

注意しておきますが、あくまでも段落等はブラウザによって自動的に整形するのが基本です。強制改行はどうしても必要がある場合に限って使うようにしてください。

3.4 構造と表現の分離、スタイルシート

ここまで「HTML は文書の構造を規定する」と説明してきましたし、実際これまでに学んだ HTML 要素はほとんどがそのような目的に沿ったものでした (少しだけ違うものがありましたが、どれだか分かりますか?)。[7]

しかし、実際に世の中の Web ページを見ていると、色や配置などの「表現」がざまなまに工夫されていて楽しいものが多数あります。自分のページにもこれらの表現を行なうにはどうしたらいいのでしょうか? 実は HTML にも「色をつける」「フォントを変える」「中央そろえ」など表現を指定する要素がいくつか用意されています。しかし、HTML 4.0 からはこの種の機能はすべて「非推奨」になり、代りに「スタイルシート」と呼ばれる指定方法でさまざまな表現を指定することになりました。

なぜでしょうか? たとえば HTML では「大見出しを全部集めて来て一覧を作る」といった作業は (grep などのツールを使って) 簡単に行なえますが、そのとき見出しの中に「ここは青い色」といった別のタグが混ざっているとうまく取り出せなかったり、または取り出したものにタグが混ざっているなど、面倒なことが起きます。

[7] br 要素は「ここで行を変える」という指示なので意味の指定というよりは見え方を制御する要素です。

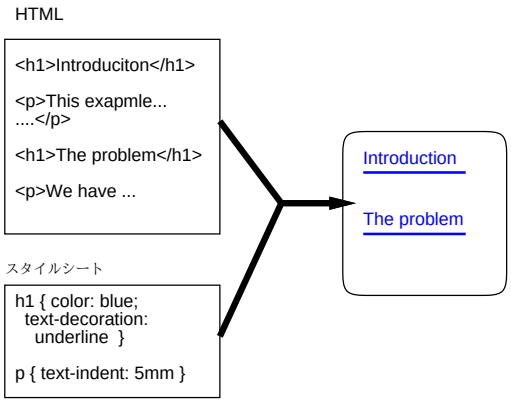


図 11: スタイルシートの概念

それに、大見出しを青い色にするとしたら、全部の大見出しをそのように統一したいわけですが、すべての大見出しの所に余分に「ここからここまで青」というタグをつけて行くのも無駄な話です。計算機で処理するのだから、「すべての大見出しは青」と「ひとこと」言えば済むようであるべきではないでしょうか? (図 11)。スタイルシートとはちょうどそのように、つまり文書の構造のそれぞれについて「このような部分はこのような表現」という形で表現を指定する機能なわけです。

3.5 HTML に CSS 記述を追加する

HTML と組み合わせるスタイルシート指定言語としては **CSS**(Cascading Style Sheet) が使われます。HTML に CSS の指定を追加する方法としては、次の 3 通りがあります:

(1) CSS 指定を次のような **style 要素**の内側に書く。style 要素はヘッダ部分に入れる必要がある。

```
<style type="text/css">
CSS 指定…
…
</style>
```

(2) CSS 指定を別のファイルに入れ、HTML のヘッダ部分 (<head>…</head>の内側) に次のような **link 要素**を入れる (ここでは CSS 指定が mystyle.css というファイルに入っているものとしました)。

```
<link rel="stylesheet" href="mystyle.css" type="text/css">
```

この方法はやや繁雑だけれど、1 つの CSS ファイルを複数の HTML ページに適用させられる。

(3) HTML の各要素に **style 属性**を指定し、その値として CSS 指定の本体部分を書く。たとえば次のようになる。

```
<p style="color: blue">この段落は青い。</p>
```

この方法はまあ「緊急避難用」ですね。多用するとわけがわからなくなります。

以下では (1) の方法を使うようにします。先のページに CSS 指定を入れたものを示します (図 12)。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
<style type="text/css">
```

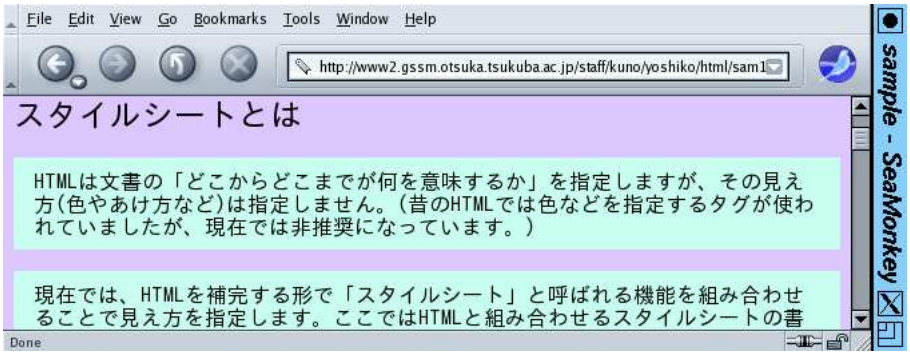


図 12: スタイルシートを使ったページ

```
body { background: rgb(220,200,255) }
p  { background: rgb(200,255,240) }
p  { padding: 3mm 5mm }
</style>
</head>
<body>
<h2>スタイルシートとは</h2>
<p>HTML は文書の「どこからどこまでが何を意味するか」を指定します
が、その見え方 (色やあけ方など) は指定しません。</p>

<p>現在では、HTML を補完する形で「スタイルシート」と呼ばれる機能
を組み合わせることで見え方を指定します。</p>
</body>
</html>
```

3.6 CSS の指定方法

順序が逆になりましたが、CSS の指定方法について説明しましょう。まず、CSS の指定は「規則」の集まりで、1 つの規則は次の形をしています:

セレクタ { プロパティ: 値; プロパティ: 値; … }

「セレクタ」としてはとりあえず HTML のタグを考えればよいでしょう。つまり「この要素はこういうふうに表現する」という指定だということになります。プロパティについては上で出て来たように、色や字下げなどさまざまなものがあります (すぐ後で説明します)。そして、それに対する値を指定します。値の指定方法については次の通り:

- 文字サイズの指定方法: 12pt (ポイント数)、x-small、small、medium、large、x-large など
- 長さの指定方法: 1px(画面上の点)、1cm(センチ) など。
- 百分率: %をつけた数値。ページ全体の幅に対する割合や、本来のフォントサイズに対する割合が指定できる。
- 色の指定方法: black、blue、gray、green、maroon、navy, olive、purple、red、silver、white、yellow、rgb(赤, 緑, 青) ただし「赤」「緑」「青」は 3 原色の強さを 0〜255 の数値で表す
- ファイルや URL: url(ファイル名)、url(URL)

CSS プロパティの代表的なものとしては次のものがあります:

- color: 色 — 文字色を指定。

- `background:` 色 — 背景色を指定。
- `margin:` 長さ — 要素の周囲のマージン (余白) 幅を指定。4 つの長さを指定すると「上、右、下、左」の長さを指定したことになる。2 つだと「上下、左右」、1 つだと 4 周全部がその長さに。
- `padding:` 長さ — 要素の周囲のパディング (詰めもの) 幅を指定。指定方法は margin と同様。
- `border:` 形状 色 長さ — 要素の枠を指定。形状として、`solid`(均一)、`dashed`(点線)、`double`(2 重線)、`ridge`(土手)、`groove`(溝)、`inset`(くぼみ)、`outset`(出っぱり) 等が指定できる。色は枠の色、長さは枠の幅を指定。
- `text-indent:` 長さ — 段落先頭の字下げ幅を指定。
- `text-align:` 種別 — `left`、`right`、`center` で左そろえ、右そろえ、中央そろえを指定。
- `text-decoration:` 文字飾り。`underline`(下線)、`blink`(点滅) 等を指定。
- `font-size:` 文字の大きさを指定。
- `width:` 長さ、`height:` 長さ — この要素を整形する幅や高さを指定できる。

3.7 ID 属性と class 属性による指定 option

ここまで説明してきた方法では、「すべての段落をこうする」「すべての見出しをこうする」といった指定しかできませんでした。しかし、場合によっては「この段落だけこうしたい」ということもあるはずです。その場合は次の 2 通りの方法が使えます:

(1) 「`<p id="p01">...</p>`」のように要素の開始タグに **ID 属性**を指定し、スタイル指定で

```
#p01 { color: red }
```

という形で「この ID が指定されている要素について」指定する。なお、HTML ではこのように開始タグ内につける付加指定を**属性**と呼びます。

(2) 「`<p class="important">...</p>`」のように要素の開始タグに **class 属性**を指定し、スタイル指定で

```
p.important { color: red }
```

という形で「このクラスが指定されている要素について」指定する。

両者の違いですが、ID 属性は「すべての要素について異なる値を指定する」ことになっているので「特定のこれ」という形での指定に使い、class 属性は複数の要素に同じものを指定できるので「このクラスを指定したものすべてに適用」という指定に向いています。上の例では「important 指定の段落 (p) は赤」でしたが、

```
.important { color: red }
```

のように要素名の指定をなくせば「class が important のものは何であれすべて」という形でも使うことができます。

3.8 さまざまな HTML 要素

順序が前後したようですが、CSS について分かったところで、HTML の要素についてもう少し学んでおきましょう。黒丸つきの箇条書きは全体が **ul 要素**、その中の各項目が **li 要素**になります:

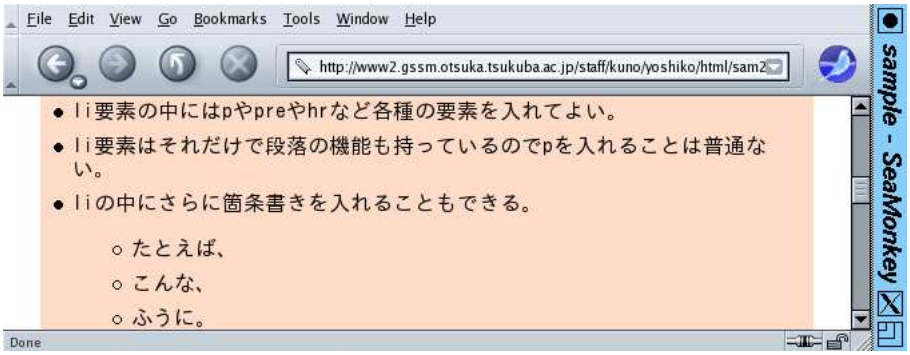


図 13: 箇条書きの例

```
<ul>
<li>li 要素はそれだけで段落の機能も持っているので p を入れることは
普通ない。</li>
<li>li の中にさらに箇条書きを入れることもできる。
  <ul>
  <li>たとえば、</li>
  <li>こんな、</li>
  <li>ふうに。</li>
  </ul>
</li>
</ul>
```

黒丸でなく番号を振るには ul 要素の代りに **ol 要素**を使います:

```
<ol>
<li>自分が考えたことが、きっちりその通りに動く (ちゃんと考えれば)</li>
<li>一度正しく作れば、何回でもいつでもその通りに動く</li>
<li>沢山の考えたことを「積み重ねて」行くことができる</li>
</ol>
```

タイトルつきの箇条書きは全体が **dl 要素**、項目タイトルが **dt 要素**、項目本文が **dd 要素**になります:

```
<dl>
<dt>正確さ</dt>
<dd>自分が考えたことが、きっちりその通りに動く (ちゃんと考えれば)</dd>
<dt>自動化</dt>
<dd>一度正しく作れば、何回でもいつでもその通りに動く</dd>
<dt>階層性 (抽象化)</dt>
<dd>沢山の考えたことを「積み重ねて」行くことができる</dd>
</dl>
```

ここまでに出て来たタグをまとめておきます:

- `<pre>…</pre>` — 整形ずみテキスト (そのまま) を表す (preformatted text)。
- `<ul>…</ul>` — 黒丸の箇条書きを表す (unordered list)。
- `<ol>…</ol>` — 番号つきの箇条書きを表す (orderd list)。
- `<li>…</li>` — 箇条書きの 1 項目を表す (list item)。
- `<dl>…</dl>` — タイトルつき箇条書きを表す (definition list)。
- `<dt>…</dt>` — タイトルつき箇条書きの項目タイトルを表す。
- `<dd>…</dd>` — タイトルつき箇条書きの項目本文を表す。
- `<hr>` — 区切りの横線を表す (horizontal rule)。
- `<br>` — 行替えを表す (break)。

図 14: 表組みの例

3.9 HTML の表

HTML の表は、細かい機能を省略して概要だけ説明すると、**table 要素**の内側に各行を表す **tr 要素**、その中に各セル (箱) を表す **th 要素**または **td 要素**を入れるという 3 重の入れ子構造を書けば済みます。

- `<table summary="説明" border="幅">…</table>` — 1 つの表を示す。簡単な説明と罫線の幅 (指定しなければ罫線なし) を指定する。
- `<tbody>…</tbody>` — 表の内側本体を示す (table 要素の中にこれを 1 つ入れる)。
- `<tr>…</tr>` — 表の 1 つの行を示す。
- `<th>…</th>`、`<td>…</td>` — 表の 1 つの箱 (セル) を示す。th は見出しセル、td はデータセルで、見え方が多少違う。

表は情報を整理して示すのに有効です (図 14)。

```
<table summary="sample" border="2"><tbody>
<tr><th>魚</th><th>肉</th><th>デザート</th></tr>
<tr><td>伊勢海老</td><td>ステーキ</td><td>シャーベット</td></tr>
</tbody></table>
```

さらに、th と td では「`colspan="数"`」、「`rowspan="数"`」を指定することで、セルを N 個ぶん横/縦に「ぶち抜き」にできます。

3.10 リンク

次は Web の重要な機能であるページ中へのリンクの埋め込みを説明しましょう。リンクは次の 2 つの情報から成っています:

- リンクテキスト — ページの内容の一部として見えていて、そこを選択するとリンクがたどられるようなテキスト
- リンク先 — リンクを選択した時取り寄せて表示する情報のありかを表す URI

このため、リンクを表す HTML の **a 要素**は次のような形をしています:

```
<a href="http://www.jaxa.jp/">JAXA のページ</a>
```

ここで「`http://www.jaxa.jp/`」がリンク先 URI、「JAXA のページ」がリンクテキストになります。なお、リンクのつけ方で、こういうのはよくないとされています:

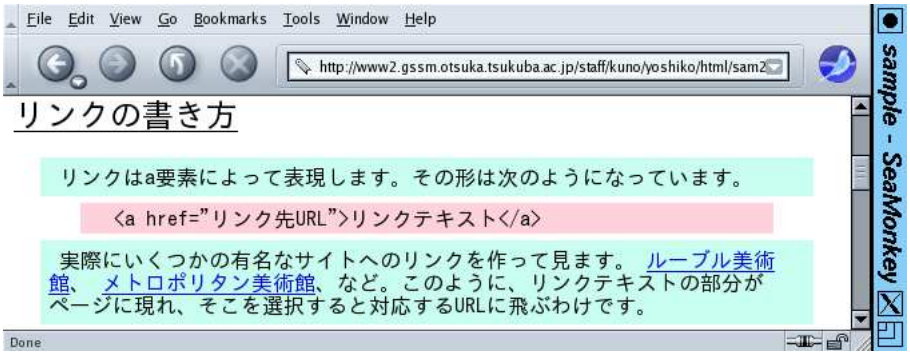


図 15: リンクの例

```
JAXA のページに行くには<a href="http://www.jaxa.jp/">
ここ</a>をクリックしてください。
```

なぜでしょうか? それは、リンクテキストとして「ここ」だけしか表示されていないと、実際にリンクをたどって見ないと行った先がどんなページなのか分からないという問題があるからです。それよりは、1 つ前の例のように行った先の内容を表す語がリンクになっていた方が読み手にとってずっと親切なわけです (この例ではもっと前の文章を見れば分かりますが、そうやって前後を探さなければならないのも不親切のうちです)。あと、a 要素は実際には段落等の中に入れて使うことに注意してください (図 15)。

```
<p>実際にいくつかの有名なサイトへのリンクを作って見ます。
<a href="http://www.louvre.or.jp/">ルーブル美術館</a>、
<a href="http://www.metmuseum.org/">メトロポリタン美術館</a>、
など。</p>
```

上の例ではスキームまで含めた長い URI(**絶対 URI**) を指定して外部のページへのリンクをつけていましたが、単にファイル名だけを指定することで自分の作った別のページへのリンクを用意することもできます。

```
<a href="enshuu2.html">演習の 2 番目のページ</a>
```

このように、スキームから始まらない URI は**相対 URI** と呼ばれ、次のような操作によって絶対 URI に変換されます: [8]

- スキームがなければ、現在見ているページを取り寄せた絶対 URI のスキームがそのまま使われる。
- ディレクトリ部分が「/」で始まらなければ、現在見ているページの絶対 URI のディレクトリ部分を挿入する。

そして、ブラウザはリンクがたどられた時、リンク先の URI の内容が HTML であればそれを直接表示し、ブラウザに表示できる以外のコンテンツであれば適宜それを表示/再生するアプリケーションを起動したり、そのようなアプリケーションが不明であればファイルに保存するかどうか尋ねて来る、などの動作を行うわけです。

ここでふたたび、WYSIWYG とマークアップについて考えてみましょう。たとえば、WYSIWYG ツールでページを作成している場合、このようなリンクの情報は「見たまま」では編集できません (というのは、もともと見える情報ではありませんから)。そのため、リンク個所で何かウィンドウを開いてフィールドに打ち込むといった操作が必要になります。マークアップであれば、見える情報 (テキスト) も付加情報も一緒に編集しているわけですから、その一部としてリンク先などの情報を編集するのも何ら特別な手間は必要としないわけです。

[8] 込み入った規則なので省略して概要だけ示しています。

図 16: 埋め込み画像の例

3.11 画像の利用 option

WWW が普及した原動力の 1 つに画像が表示できるという点があったことは説明しました。Web ページで画像を使う場合、ファイル形式として GIF、JPEG、PNG のいずれかを使ってください (次回に詳しくやります)。とりあえず、画面に出ているものは何でも次のコマンドで GIF 形式取り込めます (コマンドを起動し、カーソルが「+」になったら取りたい領域をドラグします)。

```
import myimg.gif
```

画像を用意したとして、Web ページで画像を使う方法としては次の 3 つがあります:

(1) リンク画像 — リンクの宛先として画像ファイルの URI を指定することができます。この場合、そのリンクを選択するとブラウザはその画像を単独で表示します。大きい画像などの場合に適する方法で

(2) 埋め込み画像 — 一番多い方法で、ページの中に画像を埋め込んで表示します (図 16)。そのためには、**img 要素**を使います。

- `<img src="画像ファイル" alt="説明文">` — 画像を埋め込む

`img` も `hr` や `br` と同様、単独の要素であることに注意。alt による説明文は、画像を表示できない環境や目の見えない人のための手がかりになるので、必ず指定しましょう。

また `img` 要素は `a` 要素と同様、段落の中に入れてください (段落の中に入れたくない場合は `<p><img ...></p>`のように単独の段落とするか、次に述べる div 要素を使って`<div><img ...></div>` としてください。

(3) 背景画像 — CSS で `background:` プロパティの値として色のかわりに「`url(画像ファイル)`」を指定することで、その要素の背景画像を指定できます。

Web では画像は見た目を引き付ける有用な手段ですが、なにごとも「使いすぎ」には注意してください。

3.12 div と span: 範囲指定のためのタグ option

ここまでに見て来た方法でさまざまなスタイルはつけられるようになりましたが、まだ「段落の中のこのフレーズだけ色を変える」とか「見出しと段落をまとめて囲む」という目的には不十分です。このような、新しい範囲を指定するためには、HTML 4.0 で追加された **span 要素**と **div 要素**を使います:

- `<span>...</span>` — 段落内部などに含まれる範囲を指定する。

図 17: div と span の活用

- `<div>...</div>` — 見出しや段落などをを含む範囲を指定する。

これらを使うことで、次のようにして部分強調やカコミなどが作れます (図 17)。

```
<div id="box1">
CSS で文字の斜体/立体を設定するには font-style: italic、
font-style: normal の指定を使います。
</div>
<p>なお、<span id="span1">文字範囲を囲むいちばん多い用途</span>
は「強調」でしょうが、強調のためには em 要素 (emphasis) が用意され
ています。</p>
```

div 要素を使ってページの一部を囲んだ場合、それを本文とは分けて独立配置したり、雑誌のカコミ記事のように左右に本文を流し込んだりするのに使うこともできます。そのための CSS 指定も追加しておきましょう: [9]

- `float:` 流し込みの指定。`left`(左に寄せて右に流し込む)、`right`(右に寄せて左に流し込む)、`none` が指定できる。
- `position:` 位置指定。`absolute`(絶対位置を指定)、`fixed`(画面上での絶対位置を指定)、`relative`(本来あてはまる位置からのズレを指定) のいずれかが指定できる。
- `top:`、`left:`、`bottom:`、`right:` 要素の上端、左端、下端、右端の位置を指定。

4 まとめと演習

今回はまず計算機におけるテキスト情報の位置付けについて考え、テキストと文書の違いを学びました。続いて、文書 (ドキュメント) を作成する際、付加情報を含めて扱う方法として WYSIWYG とマークアップの 2 つの方式があることを学び、意味づけ方式のマークアップの例として LaTeX と HTML を取り上げました。また、HTML では文書構造のみを記述し、その表現については CSS によって分けて指定することも扱いました。

1-1. latex のサンプルファイルをコピーしてきて、英語の文章のところに「WAT」「(of) “of” of? of! of*」などの文字を打ち込んで整形出力し、カーニングの処理がされていることを確認しなさい。また、「fire」「flower」「differ」などの単語を打ち込んで整形出力し、リガチャが処理されていることを確認しなさい。その他、詰め合わせや配置の調整など、整形処理において工夫されていることを探してみなさい。これらを通じて気がついたことがあれば報告しなさい。

[9] これらの指定を div 以外の要素と組み合わせることも、もちろん可能です。ただ、div と組み合わせることが多いので、ここでまとめて紹介しました。

1-2. 普段自分が使っているワープロソフトにおいて、上と同様のものを (半角ないし英語モードで) 打ち込み、カーニングやリガチャが処理されているかどうか調べてみなさい (フォントによって処理が変わることがあるので注意。また、整形処理における工夫も探してみなさい。これらを通じて気がついたことがあれば報告しなさい。

1-3. latex のサンプルファイルの内容 (文章、数式、表など) を 50%以上、自分固有の内容に書き換えて整形しなさい (数式や表や箇条書きは内容や個数は変更してもよいが、1 つ以上残すこと)。また、冒頭の documentclass を次のように切り替えて観察しなさい。

```
\documentclass[a4j]{jarticle}
\documentclass[12pt,a4j]{jarticle}
\documentclass[a4j,twocolumn]{jarticle}
```

これらを通じて気がついたことがあれば報告しなさい。

1-4. 普段自分が使っているワープロソフトで、1-3 で作成したもの「そっくりさん」を作ってみなさい (文字サイズや段組みの設定はどれか 1 つを選ぶ)。できる範囲でよい。これらを通じて気がついたことがあれば報告しなさい。

1-5. HTML で作成したページ (既に作ってあるものでも、今回新たに作ったものでもよい) に、CSS によるスタイル指定を入れて見なさい。どのような指定を入れるかは任意とします。これらを通じて気がついたことがあれば報告しなさい。

1-6. HTML で作成したページに、箇条書き、表、リンク、画像から 2 つ以上を入れてみなさい。表については、「特定の箱について CSS で指定して色を変える」こともやってみなさい。これらを通じて気がついたことがあれば報告しなさい。