計算機を道具として使いこなすために

— Frontier ガイド 1999 —

佐藤理史
敷田幹文

1999 年 3 月

はじめに

計算機が我々の生活を脅かしている。その傾向は、年々深まるばかりである。

道具は、そもそも、人間を助けるために、あるいは、より楽にするために創造される。しかし、筆者の経験では、多くの場合、道具による効果はそれとは反対の方向に働く。つまり、道具の発達は、我々により多くの仕事をもたらし、より忙しくするのである。

道具としての計算機は、その典型的な例である。例えば、修士論文を作成する状況を考えよう。ワードプロセッシングの発達によって、我々は多大な恩恵を受けているはずなのに、修士論文の作成作業の大変さは、手書きの時代から大して変わっていない。むしろ作業量は増えているようにも見受けられるのである。手書きの時代には、ディスクが潰れる心配も、LaTeX のエラーもなかったのである。

しかし、我々には、逆戻りする選択肢は与えられていない。となれば、計算機を手足のように道具として使いこなすようになる以外に道はないのである。本稿は、計算機初心者にとって、そのような道を歩む際の、道標を提供することをその目的としている。

本稿の前身は、92 年度に開講された情報科学研究科の計算機システムの講義資料である。この資料のうち、筆者の執筆しなかった部分をかなり書き改め、93 年度に一冊のテキストとしての体裁を整えた。このテキストは、情報科学研究科の計算機システムの講義のテキストとして使用された以外に、材料科学研究科の講義テキストとしても使用され、さらに、北陸先端科学技術大学院大学の滞在者に対しても、Frontier のガイドとして配布された。このような利用状況を鑑み、94 年度からタイトルを改めた。

本テキストの執筆・編集方針は以下の通りである。

1. 計算機初心者にとって、「とりあえずこうすればこうできるよ」ということがわかること。
2. 計算機初心者に、参考書が読めるようになるための基礎を提供すること。
3. 一番最初に覚えなければならないことを、最小限まで絞り込むこと。
4. 分厚いテキストなんて無意味。本文 100 ページ以内を目指す。
5. 参考書に十分書いてあることは、割愛してしまえ。

計算機の使用に関する筆者のポリシーのひとつは、

- 覚えなければならないことは、少なければ少ないほど良い

ということであり、もうひとつのポリシーは、

- できるだけ楽をしたい

というものである。この 2 つのポリシーは、しばしば衝突する。その妥協点を見つけることこそが、計算機に使いこなすことだと筆者は考える。

JAIST は、世界的に見ても有数の計算機環境を有していると言える。ここの環境を使いこなせれば、どこへ行ってももう恐くはない。筆者の願いは、すぐに計算機に慣れて、このテキストが早々とお払い箱になることである。

1994 年 3 月 著者しるす

95 年度版への補足

95 年度版では、新たに、

- ネットワークを使った情報検索
- ネットワークを使った情報発信

の章を追加した。

96 年度版への補足

96 年度版では、 Mule, X11R6, Netscape などの新しいソフトウェアを使用するようにした。また、これに伴い，標準設定ファイルを変更した.

98 年度版への補足

下平 博 助教授と守岡 知彦 君のご協力により、電子メールの読み書きに使用するソフトウェアを RMAIL から mh-e に変更した。

目 次

図 目 次

第 1 章

本テキストについて

1.1 目的・内容

本テキストの目的は、以下の通りである。

> JAIST の構成員として、日常的に必要な、計算機ユーザーとしての基本的な知識を習得することを可能とする。

つまり、読者には、計算機ユーザーとして、日常的に支障がないようなレベルに到達することを望むものである。

その要件としては、以下の 2 つがある。

1. 計算機システムに関して、基本的な知識を持っていること。
2. 計算機を道具として使いこなせること。これは、以下のことを意味する。
   - UNIX を使うことができる。
   - Editor を使って、ファイルの作成・修正を行なうことができる。
   - メール、ニュースの読み書きができる。
   - 文書処理システムを使って、文書を作成 (清書) できる。
   - 種々のインターネット上の情報収集・情報発信ができる。
   - 少なくとも 1 つ以上のプログラミング言語で簡単なプログラムができる。

このうち、本テキストでカバーする範囲は、以下の 3 つの項目である。

a. UNIX (ファイルシステム、シェル、プロセス、ウインドウシステム、プログラム開発環境、システム管理)

b. Editor (Emacs の基本と応用)

c. 文書処理システム (LaTeX の基本と応用)

本テキストに書かれている内容は、あくまでも、これらを学ぶための**道案内となるべき最低限の内容**であり、より進んだ内容は、**自分で修得すること**を要請する。

1.2 参考書

参考書としては、以下の本を用いる。

(U) 斎藤信男 編, 「ユーザーズ UNIX」, 岩波書店, 2,800 円.

(GE) 矢吹道郎 監修, 宮城史朗 著, 「はじめて使う GNU Emacs」, 啓学出版, 1,500 円.
Emacs について、やさしく書かれた本。これをマスターすれば、ほとんど事足りる。

(MU) 大木敦雄 著, 「入門 Mule」, アスキー出版局, 1,800 円.
Emacs のうち Mule 特有の部分についてはこちらを参照するとよい。

(RL) 野寺隆志 著, 「楽々 LaTeX」, 共立出版, 2,400 円.
LaTeX の入門書。比較的わかりやすい。

(C) B. カーニハン, D. リッチー 著, 石田晴久 訳, 「プログラミング言語 C 第 2 版」, 共立出版, 2,800 円. C の標準的な教科書。

本テキストは、上記の参考書の関係箇所を一読することを想定して書かれている。
さらに、最終的なマニュアルとして、以下の本を奨める。

(EM) R. Stallman 著, 竹内郁雄, 天海良治 訳, 「GNU Emacs マニュアル」, 共立出版, 2,500 円.
Emacs の作者が書いたマニュアル。古いバージョンについて書かれているので、現在のものと部分的に違いがあるが、 Emacs の入門を越えるにはこれがあると便利。

(MH) Jerry Peek 著, 倉骨 彰 訳, 「MH & xmh」, アスキー出版局, 5,800 円.
MH に関する専門書。各種コマンド使用法、カスタマイズ法、管理法が詳細に説明されている。 mh-e については、 Appendix H.2 に簡単な説明があるのみである。

(L) L. Lamport 著, E. Cooke, 倉沢良一 監訳, 「LaTeX」, アスキー出版局, 2,800 円.
LaTeX の作者が書いたマニュアル。最終的には、これが必要になることが多い。

本テキストがカバーする範囲をより詳細に記述したテキストに以下のものがある。

山口和紀 監修, 「The UNIX Super Text (上, 下)」, 技術評論社, (上)3,400 円, (下)3,700 円.

がある[1]。

[1] この本を読める人は、このテキストを読む必要はない。

第 2 章

入門

デスク上に置いてある計算機 (ワークステーション) の使い方の最も基本的なことについて述べる。

2.1 計算機の構成と取り扱いの注意

我々の使用する計算機 (Fujitsu S-4/IX, S-4/CL, S-4/5) は、以下のものから構成されている。

- 本体 (CPU、主記憶、ハードディスク)
- 表示装置 (ディスプレイ)
- 入力装置 (キーボード、マウス[1])

これは、標準的な構成であり、他の多くの計算機も同様な構成をとる。

計算機は、精密機械であるため、乱暴に取り扱うと故障の原因になる。また、キーボードは、必要以上に強く叩く人がいるが、これもキーボードを壊すことになりかねないので、奨められない。とにかく、丁寧に扱って欲しい。

本体の電源は、通常、入れっぱなしの状態とし、電源の on、 off は行なわない。これに対して、**表示装置 (ディスプレイ) の電源は、使用していない場合は、 off にすること**。これによって電気使用量をかなり減らすことができる[2]。

注意!! 計算機が何かおかしくなったからといって、電源を落してはいけない。最悪の場合、計算機を破壊する可能性がある。そのような状態に陥った場合には、教員等に連絡し、指示を乞うこと。また、計算機本体は、電源 on の状態で移動してはいけない。計算機の場所を移動したい場合も、教官等に相談すること。

つまり、通常、我々が触るところは、キーボード、マウス、ディスプレイの電源、輝度のつまみだけである。

[1] マウスには、金属盤のマウスパッドが必要な光学式マウスと、スポンジのマウスパッドが付属している機械式マウスがある。さらに、金属盤のマウスパッドには、格子模様の細かさの違う 2 種類があり、それぞれ交換できないので要注意である。

[2] 本学は国立大学であるので、その運営費は、税金によってまかなわれている。その点をいつも頭に入れておき、節約を心がけること。

2.2 キーボード配列とタッチタイピング

計算機に対する入力は、主に、キーボードを用いる。キーボードは、アルファベット順にならんでいるわけではないので、どこにどの文字があるのかを、覚える必要がある。

少なくとも、アルファベットに関しては、ブラインド (キーボードを見ない) で入力できるレベルに到達する必要がある。タッチタイピングはその標準的な方法であり、一度は練習してみて欲しい。

2.3 login と logout (U§3.1)

2.3.1 login

ログイン (login) とは、ユーザーが UNIX システムの利用を始める手続きである。ログイン名 (ユーザー ID) とパスワードを入力する。

```
SunOS UNIX (is17e0s00)

login: sato
Password:
Last login: Mon Mar 15 16:38:25 on console
SunOS Release 4.1.2 (GENERIC) #2: Wed Oct 23 10:52:58 PDT 1991
%
```

1 行目、空行の 2 行目、および、 3 行目の途中 (login:) までが、システムが表示しているメッセージである。このようなメッセージを表示している場合、システムは、ユーザーの login を受け付けることができる。まず、ユーザー ID (この場合は `sato`) を入力し、 return を入力する。すると、パスワードの入力を求めてくる。そこで、パスワードを入力し、最後に、 return を入力する。なお、入力したパスワードは、表示されないので、注意すること。

正しいパスワードが入力されると、システムは、上記のようなメッセージを出力し、ユーザーからの入力 (命令) 待ちの状態になる。

ユーザー ID を間違えたり、パスワードを間違えた場合は、システムに怒られる。

```
SunOS UNIX (is17e0s00)

login: sato
Password:
Login incorrect
login:
```

その場合は、もう一度、ユーザー ID から入力し直す。なお、ユーザー ID とパスワードの入力時には、後退 (バックスペース) による訂正が効かない。

2.3.2 セッション

上記の login によって、ユーザーは、 Unix システムを利用することができるようになる。ここで、 Unix システムの利用とは、具体的には、システムに対する要求 (コマンド) を入力し、実行させることができることを意味する。システム側から見ると、ユーザーが入力したコマンドを実行することにより、システムはユーザーの望むサービスを提供するということになる。このような Unix システムを利用できる状態をセッションと呼ぶ。セッションは、 login に始まり、 logout で終る。

システムは、ある特別な文字列 (入力促進記号 = プロンプト) を表示し、ユーザーの入力を促す。上記の例では、% がこれに相当する。これは、自由に変更できる[3]。

2.3.3 logout

セッションを終了させるコマンド。

2.4 コマンド (U§3.2 3.3)

システムに対する要求は、コマンドを入力することによって行なう。システムは、このコマンドを解釈し、それを実行する。これを行なうプログラムをシェルという。

コマンドは、一般に、以下のような書式を取る。

書式 コマンド名　オプションの並び　引数の並び

ここで、オプションとは、おおよそ、コマンドの命令の詳細を指定するものであり、引数は、おおよそ、コマンドを働かせる対象を指定するものである。以下にいくつかのコマンドの実行例を示す。

```
% w
  5:33pm  up 56 mins,  5 users,  load average: 0.07, 0.00, 0.00
User     tty       login@  idle   JCPU   PCPU  what
sato     console   4:38pm    55      3      3  /usr/bin/X11/xterm -n
Console -T
sato     ttyp0     4:38pm          1:26     49  w
sato     ttyp1     4:39pm    54                -csh
sato     ttyp2     4:39pm    27                -csh
sato     ttyp3     4:39pm    54                -csh
% date
Mon Mar 15 17:33:12 JST 1993
% cal
   March 1993
 S  M Tu  W Th  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

%
```

```
% cal 12 1960
   December 1960
 S  M Tu  W Th  F  S
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

%
```

コマンドの使い方等は、 man コマンドによって調べることができる。 man コマンドの使い方 **(U¶35)** は、以下のようにすれば調べることができる。

[3] 14章参照。

```
% man man
```

なお、表示が一画面で収まらない場合は、画面の最下行に以下のような表示がされる。

```
--More--(22%)
```

次のページに進みたい場合は、スペース、そこで打ち切る場合は、q を入力すればよい。詳しくは、more コマンドの使い方を見よ。

2.5 パスワード

2.5.1 パスワードの変更 (U¶37)

自分のパスワードは、 passwd コマンドによって、自由に変更できる。

```
% passwd
Changing NIS password for sato on jaist1.
Old password:
New password:
Retype new password:
NIS entry changed on jaist1
%
```

2.5.2 パスワードの重要性

パスワードは、 UNIX システムを利用できる自分の権利を他人から守る重要な防御壁であり、他人に知られないようにしなければならない。パスワードを知られると悪用される危険がある。

注意!! 学内の誰か一人でもパスワードの管理を怠り、外部からの侵入を許すと、学内の他のアカウントへの侵入も容易になり、**全員を危険にさらす**ことになる。一人一人が十分気を付けて欲しい。

2.5.3 よいパスワード

パスワードは、自分にとって覚えやすく、他人から推測できないような文字列にするのが望ましい。 6 文字以上[4]で、数字を含むものにすべきである。間違っても、自分の名前 (first name) や誕生日などにしてはいけない。

注意!! テンキー (キーボードの右にある数字のキー) を用いてパスワードを設定しないこと。数字の入力には、必ず、アルファベットのキーの上にある数字キーを用いること。

パスワード破りの標準的な方法は、辞書を用いる方法なので、通常の英単語や日本語の単語、固有名詞なども避けなければならない。侵入者は各国語の辞書や計算機用語辞書、人名辞書、日本のアイドル名辞書なども準備している。日本語のパスワードであれば海外からの侵入を防げると考えるのは浅はかである。頭を大文字にする、 'i' を '1' にする、逆順にするなどの簡単な操作でも容易に解析されてしまう。

[4] パスワードは 8 文字まで有効である。

よいパスワードを作るには、以下のような方法がある。

- 2 語を合成する.
- 途中の文字を削除する.
- 途中に無関係な文字や記号などを挿入する.

また、パスワードは適当な期間毎に変更することが望ましい。

2.6 標準設定ファイルのコピー

UNIX システムを便利に使うためには、各種の設定が必要である。標準的な設定をおこなうためのファイル (標準設定ファイル) を用意しているので、当分は、これを使用して欲しい。本テキストの説明は、この標準設定に基づいている。標準設定ファイルを使用するためには、以下のようにすればよい。 (これは、標準設定ファイルを各自が使えるようにコピーすることを行なっている。)

```
> cp /usr/local/lib/user.skel/.??* .
```

ここで行なっていることはとりあえずわからなくてよい。とにかく、上記をそのまま実行すること。 UNIX では、通常大文字と小文字は別の文字として扱われるので、上記の入力も全て小文字で行わなければいけない。

次に、一度 logout し、もう一度 login し直す。それによって、プロンプトが以下のように変わるはずである。

```
[sato@is17e0s00] 1 %
```

ここで、最初の`sato` がユーザ名、次の`is17e0s00` が利用している計算機の名前、最後がイベント番号 (入力したコマンドの順番) を表す。

2.7 パニック脱出法 — 困った時、人に聞く前に

変なキーを押してしまった等により、わからない状態に陥ってしまうことがあるかもしれない。このような場合は、以下のことを順に試されたい。

1. 表示が途中で止まり、入力を受け付けない。 (あるいは、プロンプトが出ない。)
   → control–q を入力する (control キーを押しながら、 q を押す)。
2. コマンドの実行が終了せず、いつまでたってもプロンプトが出ない。
   → control–c を入力する。
3. コマンドの実行が終了せず、いつまでたってもプロンプトが出ない。
   → control–z を入力する。
4. コマンドの実行が終了せず、いつまでたってもプロンプトが出ない。
   → control–d を入力する。
5. Emacs の中でコマンドの実行を中止したい。
   → control–g を入力する。

以上を試してみても正常な状態にならない場合は、先輩・教員等に助けを求めよ。くれぐれも、**安易に電源を落すなどの野蛮な行為に走らないこと。**

2.8 停電の場合

停電の場合は、計算機の電源を落す必要がある。これは、以下の手順で行なう。

1. login している場合は、 logout する。
2. shutdown というユーザー名で login する。
3. しばし待つ。 (約 1 分)
4. `ok`、または、>が表示されたならば、本体の電源を切る。
5. ディスプレイの電源を切る。
6. UPS(無停電装置) の電源を切る。

以上である。なお、停電が終了し、再度計算機を立ち上げる場合は、以下の手順で行なう。

1. UPS(無停電装置) の電源を入れる。
2. ディスプレイの電源を入れる。
3. 本体の電源を入れる。
4. しばし待つ。 (約 3 分)
5. login のプロンプトが出れば、立ち上げが完了である

2.9 演習問題

1. 自分の ID で login し、パスワードを変更せよ。
2. 標準設定用ファイルをコピーせよ。
3. 自分の誕生日のカレンダーを表示せよ。
   (ヒント: `cal` のマニュアルを見よ)
4. ⋆ パスワード (passwd) に関係するコマンドにはどのようなコマンドがあるか。
   (ヒント: `man` コマンドのオプションを調べよ)
5. `more` コマンドについて調べよ。
6. `finger` コマンドについて調べよ。
7. ⋆⋆ どのマシンに誰が login しているか調べる方法はあるか。

第 3 章

X ウィンドウ事始め

本章では、以降の章の準備のために、 X ウィンドウの起動と終了の方法について述べる。

3.1 X ウィンドウの起動

我々は、ほとんどの場合、 login した後、以下のコマンドを実行して X ウィンドウを起動する。

```
% xinit
```

しばらくすると、図 3.1のような画面になる。

なぜ、 X ウィンドウを起動するのか。それは、以下の理由による。

- X ウィンドウでは、日本語を表示できる。
- X ウィンドウでは、一つのディスプレイ装置の画面にいくつかの窓 (ウィンドウ) を設定することによって、あたかも別の入出力装置のように別々の作業を行なうことができる。
- X ウィンドウを使わないと利用できないソフトウエアがある。

ここでは、 X ウィンドウに関する詳しい説明はしない。現時点では、 X ウィンドウを起動することは、おまじないだと思っておいてほしい。なお、 X ウィンドウを起動した後は、どのウィンドウを使ってもよい。使いたいウィンドウにマウスでカーソルを持っていけば、そのウィンドウが使用できる。

3.2 X ウィンドウの終了

X ウィンドウの終了は、各ウィンドウをそれぞれ終了し、最後のコンソール・ウィンドウ (左上の小さなウィンドウ) を終了することによって行なう。各ウィンドウは、以下のコマンドを実行することによって終了する。

```
% exit
```

exit を実行すると、そのウィンドウが消滅する。最後にコンソール・ウィンドウを終了すると、ぐ

図 3.1: X ウィンドウ画面の例

ちゃぐちゃとメッセージ[1]を出力した後、 X ウィンドウを起動する前と同じ状態になる。セッションを終了する場合は、ここで、 logout すればよい。

3.3 演習問題

1. X ウィンドウを起動せよ。
2. ウィンドウの 1 つでコマンド (例えば、 cal) を実行せよ。
3. X ウィンドウを終了せよ。

[1] このメッセージは気にしなくてよい。

第 4 章

Emacs の基本的な使い方

我々の日常的な計算機の使い方は、

- 文章を作り、修正し、印刷する
- プログラムを作り、修正し、実行する
- メールやニュースを読み書きする

などであるが、ここで我々が作るものは、すべてファイルという形で電子的に保存する。 ファイルとは、言わば、電子の紙に相当するものである。

ファイルに関するほとんどの作業は、エディタと呼ばれるファイル作成・修正用のツールを用いて行なう。エディタは、実際の計算機の利用において最も使用頻度の高いツールであり、計算機を使いこなせるようになるためには、まずエディタに習熟することが必要である。ここでは、代表的なエディタの一つである Emacs の多国語化対応版である Mule (Multilingual Emacs)[1]の基本的な使用法について述べる[2]。この本では単に Emacs と呼ぶが、特に断らない限り Mule を意味する。なお、本章は、 **(GE§1)** の内容にほぼ対応しているが、この本の内容には現在の Mule と合わない部分もあるので注意が必要である。

4.1 Emacs の起動と終了

4.1.1 Emacs の起動 (GE§1.2)

Mule は、 mule コマンドによって起動する。

```
% mule &
```

行末の&については、ここでは説明しない。現時点では、おまじないだと思ってほしい。この他に、X ウィンドウのメニューから起動する方法もあるが、ここでは振れない。

4.1.2 Emacs の画面構成 (GE§1.3)

Emacs を起動すると、図 4.1のような画面が現れる。 Emacs の画面は、

[1]Mule の古い版として NEmacs がある。これは本学の環境でも一応使用可能であるが，古いソフトウェアであるため現在の環境には適していない。また、その他の Emacs として、 XEmacs がある。
[2]この他の代表的なエディタとしては、 vi がある。

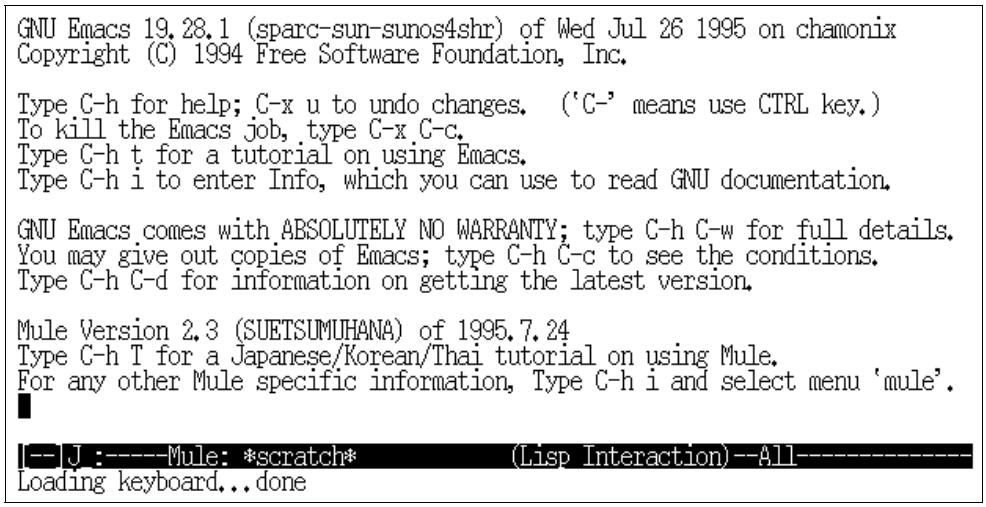


図 4.1: 起動直後の Emacs

- テキストウィンドウ —— 作成・修正するファイルを表示する。最下行には、そのウィンドウの状態を示すモード行がある。モード行は、白黒反転表示される。
- ミニバッファウィンドウ (最下行) —— コマンド等の入力を行なう

の 2 つのウィンドウから構成される。

4.1.3 Emacs の終了 (GE§1.5)

Emacs は、 C–x C–c によって終了する。
(まず、control を押しながら x を押し、次にcontrol を押しながら c を押す。)

4.2 編集の基本的なコマンド

エディタに関しては、以下のことを知っていれば、とにかくなんとか使うことができる。

- テキストの入力
- ポイント (カーソル) の移動
- テキストの消去
- ファイルの読み込みとセーブ
- ヘルプ
- 割り込み

4.2.1 テキストの入力 (GE§1.4)

起動の後、キーボード入力は、そのままテキスト入力となる。

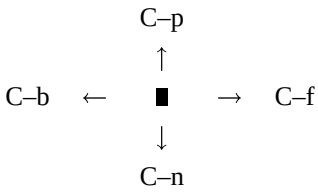
4.2.2 コマンド

通常のテキスト入力以外の入力は、コマンドと呼ばれ、 Control キーや Meta キーを使って入力する。例えば、ポイント (カーソル) の移動等もコマンドを使って行なう。

以下で、 C–a と表記した場合には「Control キーを押しながら a を押す」ことを意味し、 M–a と表記した場合には「Meta キーを押しながら a を押す」ことを意味する。 Sun のキーボードの場合 Meta キーは '◆' と表記されているが、他のキーボードでは 'Alt' と書かれていたり、キーが存在しないこともある。キーがない場合には ESC キーで代用できる[3]。

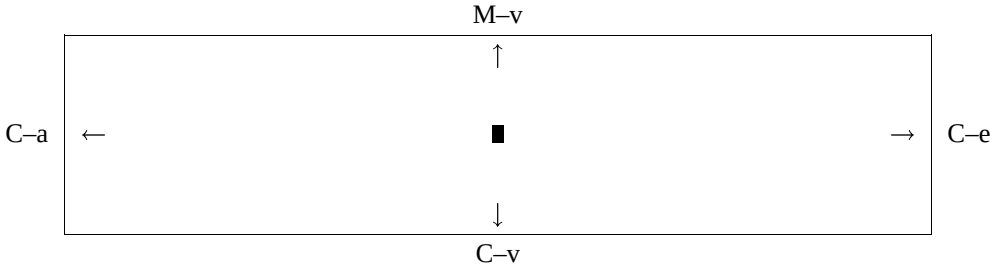
4.2.3 ポイント (カーソル) の移動 (GE§1.7)

4 方向のポイントの移動 (1 文字、 1 行単位) は、以下の通りである。



ここで、 f, b, n, p は、それぞれ、 forward, backward, next, previous の略である。

より大きな単位での移動は、以下の通りである。それぞれ、行の先頭への移動 (C–a)、行の末尾への移動 (C–e)、 1 画面スクロール (C–v)、逆スクロール (M–v) を表す。



この他に、テキストの一番先頭に移動する M–<とテキストの一番末尾に移動する M–>を良く使う。

4.2.4 テキストの削除 (GE§1.10)

テキストの削除には、以下の方法がある。

DEL ポイントのある場所のすぐ前の 1 文字を削除する。

C–d ポイントのある場所の 1 文字を削除する。

C–k ポイントのある場所からその行の最後までを削除する。

ポイントが行の先頭にある場合、 C–k を実行すると、その行の文字はすべて削除されるが、行自体は、なにも文字がない行として残る。そこで続けて C–k をすると、完全に行がなくなる。

[3] この場合、 M–a は「ESC キーを押してから a を押す」ことになる。

4.2.5 ファイルの読み込みとセーブ (GE§1.12, §2.3)

エディタを利用する場合には、新しいファイルを作成する場合とすでにあるファイルを修正する場合がある。

どちらの場合も、以下のコマンドでファイルを (テキスト) バッファに読み込みことができる。(テキストウィンドに表示することができ、編集の対象とすることができる。)

C–x C–f ファイルを読み込む。 C–x C–f を入力するとミニバッファに

```
Find file: ~/
```

という表示が現れ、ファイル名の入力を促す。ここでファイル名を入力する。ファイルが存在する場合は、そのファイルが読み込まれ、存在しない場合は、新しいファイルが作られる。

このとき、ファイルのコピーが読み込まれるので、次で述べるファイルのセーブをしない限り、オリジナルのファイルは修正されない。バッファとは、上で述べたファイルのコピーを保持するある特別の記憶領域である。

注意!! ファイル名を表現するためには、アルファベット、数字などを用いることができる。これに対して、**ひらがな、カタカナ、漢字などの日本語文字は、ファイル名として使用することはできない**。

作成したり修正したりしたファイルは、以下のコマンドでセーブすることができる。

C–x C–s ファイルをセーブする。

C–x C–w 名前を変えてセーブする。 C–x C–w を入力するとミニバッファに書き込み先ファイル名を求めてくる。これを指定すると、そのファイルに書き込まれる。

4.2.6 ヘルプ (GE§2.5)

Emacs のコマンド等は、ヘルプによって調べることができる。

C-h ヘルプ

ヘルプを実行すると、以下のようなメッセージがミニバッファに表示される。

```
h (Type ? for further options)
```

ここに書かれているように、?を入力すると、オプションが表示される。良く使うのは、k (C–x C–f のように特殊なキーを入力することによって実行できるコマンドの説明) やa (ある文字列を含むコマンドの一覧)、m (現在おかれている状態に関する機能・キー操作・設定変更法などの説明)、などである。ヘルプを実行すると、画面が 2 つに割れるが、 C–x 1 を入力すると、 1 つの画面に戻る。(詳しくは、 7章参照のこと。)

Emacs は自身の中に大量のドキュメントとそのいろいろな検索機構を持っており、ヘルプ機能を覚えるだけでほとんどの疑問は自分で容易に解決することができる。 H.2も参照して、できるだけ早く修得すること。

4.2.7 割り込み (GE§1.6)

Emacs で、何かわからない状態に陥った場合は、 C–g を押す (何回か押す)。ミニバッファに Quit と表示された場合、それは、正常な状態に戻ったことを意味する。

4.2.8 Undo (GE§1.8)

間違えて必要なものを削除してしまった場合などは、 Undo (変更の取消) が利用できる。

C–_ 1 回分の変更を取消す

C–x u 1 回分の変更を取消す

また、連続して複数回押すことにより、さらに前の状態に戻すこともできる。これは Emacs の強力な機能の一つなので十分活用して欲しい。

4.3 テキストの削除と移動

Emacs では、キルリング (kill ring) と呼ばれる特別の記憶領域を用いて、カットアンドペースト (切り貼り) を行なうことができる。カットアンドペーストとは、編集中のバッファ (ファイル) のある部分 (領域) を別のところ (領域) に移動したり、コピーしたりすることを言う。

4.3.1 マークとリージョン (GE§1.9)

カットアンドペーストを実行するには、まず、移動したりコピーしたりする領域を指定する必要がある。この領域を指定するために、マークとポイントを使う。マークとは、ある場所に文字通りマークをつけることを意味する。

C–SPC 現在のポイントにマークを付ける。

マークをつけた後にポイントを移動させたとき、マークの位置とポイントによってはさまれる領域をリージョンと呼ぶ。このリージョンがカットアンドペーストの対象となる。

4.3.2 カットアンドペースト (GE§1.11)

以下のコマンドを使うことによって、カットアンドペーストを行なうことができる。

C–w リージョンを消去し、キルリングに記憶する。

M–w リージョンのコピーをキルリングに記憶する。

C–y ポイントのある場所に、キルリングに記憶されている内容を挿入する。

4.4 探索と置換

バッファの中である特定の文字列を探したり (探索)、ある特定の文字列を別の文字列に置き換える (置換) コマンドは、比較的利用頻度が高い。

4.4.1 探索 (GE§1.13)

C–s 順方向 (前向き) のインクリメンタル探索。 C–s を入力すると、ミニバッファに探索すべき文字列のプロンプト (`I-search:`) が表示される。ここで、文字を入力すると、インクリメンタルに (一文字ずつ順に) 探索される。例えば、まず、`a` を入力すると、現在のポイント以降の、最も近くにある`a` という文字列までポイントが移動する。次に、`b` を入力すると、今度は、最も近くにある`ab` にポイントが移動する。このまま C–s をもう一度入力すると、次の`ab` にポイントが移動する。このインクリメンタル探索から抜けるのには、 RET を入力する。

C–r 逆方向 (後向き) のインクリメタル探索。

日本語の検索においては、検索文字列の入力にローマ字漢字変換が必要になるので、上記の検索コマンドによって検索文字列を入力できない。そのため、以下のコマンドが用意されている。

C–s C–k 順方向の (日本語の) 検索文字列の入力。

C–r C–k 逆方向の (日本語の) 検索文字列の入力。

これらのコマンドでは、検索文字列を入力し、リターンを入力すれば、後は、通常のインクリメンタル検索と同様である。

4.4.2 置換 (GE§1.14)

M–% 利用者に質問しながらの置換。 M–%を入力した後、ミニバッファで置換の対象になる (置換元) 文字列とそれを置き換える (置換先) 文字列を入力する。この指定が終ると、ポイントのあるところから、置換の対象となる文字列が探され、見つかったところで、そこを本当に置換するかどうか聞いてくる。 SPC を入力すれば置換され、 DEL を入力すれば置換されない。また、途中で RET を入力すれば置換を中断し、！でそれ以降の対象を一気に置換する。

4.5 演習問題

1. Emacs を使って、次のような内容の sample.txt というファイルを作成せよ。

```
   Emacs divides the screen into several areas, each of which contains
its own sorts of information.  The biggest area, of course, is the one
in which you usually see the text you are editing.
```

2. ポイント (カーソル) を動かすコマンドを実行せよ。

3. ポイントをテキストの先頭に移動させ、`of` を探索せよ。
   (ヒント: C–s を使う。)

4. ポイントをテキストの最後尾に移動させ、`of` を探索せよ。
   (ヒント: C–r を使う。)

5. sample.txt というファイルの`of` をすべて`fo` に置き換え、 sample2.txt というファイルにセーブせよ。
   (ヒント: M–%と C–x C–w を使う。)

6. `sample.txt` の内容が 10 回繰り返して存在する sample10.txt というファイルを作れ。
   (ヒント: カットアンドペーストを使う。)
7. `sample.txt` というファイルをバッファに読み込み、ポイントを先頭に持っていき、 M–f を実行せよ。何が起こるか？
8. M–b を実行せよ。何が起こるか？
9. C–h a を実行し、`Command apropos (regexp):`というミニバッファのプロンプトに対して`forward` を入力せよ。
10. 同様に、`backward` を入力せよ。
11. `sample.txt` において、`information.` の後ろにポイントを置いて、 C-x DEL を実行せよ。何が起こるか？

第 5 章

Egg: Emacs での日本語入力

本章では、 Emacs での日本語入力の方法として、 Egg を紹介する[1]。 Egg は、 Emacs の中で動く、かな漢字変換プログラムである。本章のより詳しい内容は、 **(GE§4.4)** に書かれている。一読すること。

注意!! 日本語を表示するためには、ウィンドウ・システムを使う必要がある。

5.1 jserver の起動

Egg は、 jserver と呼ばれるかな漢字変換プログラムと通信を行なうことによって、かな漢字変換を実行する。このため、 Egg を使用するためには、 Emacs を起動する前に、この jserver をあらかじめ起動しておく必要がある。 jserver を起動する方法は、以下のように、 jserver コマンドを実行すればよい。

```
% jserver
```

10 数行のメッセージを出力した後、プロンプトが出力されたら、 jserver の起動は完了である。なお、 jserver はログアウトしても動き続けるので、一旦起動したならば、以後起動する必要はない[2]。 jserver が動いているかどうかを確認するには、以下のコマンドを実行すればよい。

```
% ps -aux | grep wnn
```

このコマンドの実行により、 jserver を含む行が表示されれば、 jserver は既に起動されている。

5.2 Egg を始めて使う場合の注意

Egg は、個人用の辞書を作成し、そこに新しく登録された単語を格納したり、各種の学習ファイル (頻度ファイル) を格納したりする。 Egg を最初に起動した場合、それらに必要なものを作成するかどうかを以下のようにユーザーに聞いてくる。

1. 辞書ディレクトリの作成

[1] Emacs で日本語を入力する方法として、この他に、 SJ3 や SKK がある。
[2] 計算機を再起動した場合は、 jserver も再起動する必要がある。

```
directry "!/home/fs001/sato/Wnn" が無いよ。作る?(y or n)
```

2. 辞書ファイル等の設定

```
頻度ファイル "!/home/fs001/sato/Wnn/kihon.h" が無いよ。作る?(y or n)
```

これらの質問 (全部で約 10 回) には、すべて 'y' と答える。

これらに答えても「作成できません」と言われる場合には、`~/Wnn` を一旦削除してやり直すと解決することが多い。これを行うには以下のコマンドを実行する。

```
% rm -rf ~/Wnn
% mkdir ~/Wnn
% chmod og+w ~/Wnn
```

5.3 Egg の使い方 (GE§4.4)

5.3.1 入力モードの切替え

Egg では、入力モードを切替えることによって、アルファベットをそのまま入力したり、かな漢字変換を行なったりすることができる。

C- 入力モードを切替える。入力モードには、透過モード (アルファベットがそのまま入力される) とローマ字入力モードがある。また、ローマ字入力モードで変換操作後は確定するまで漢字変換モードになる。これらのモードは、モードラインの左端に表示される。[--] と表示される場合は透過モード、[あ] の場合はローマ字入力モード、[漢] の場合は漢字変換モードである。

かな漢字変換の対象となっている部分は、フェンス '|' によって囲まれる。そのため、ローマ字入力モードをフェンスモードとも呼ぶ。

5.3.2 フェンス内コマンド

フェンスの中では、以下のようなコマンドが利用できる。

SPC, C–n 変換、次候補

C–p 前候補

M–s 候補一覧

C–o 単語を伸ばす

C–i 単語を縮める

C–f フェンスの中の次の単語へ移動

C–b フエンスの中の前の単語へ移動

M–h 単語をひらがなに変換する

M–k 単語をカタカナに変換する

RET 確定

5.3.3 ローマ字の入力に関連して覚えておくと便利なこと

特殊な文字は以下のようにして入力することができる。

1. 'N' によって、いつでも「ん」を出すことができる。
2. 小さい仮名は、'x' ではじめる。例えば、'xi' は「ぃ」となる。
3. 全角の英字は、'Z' ではじめる。例えば、'Za' は。「ａ」となる。
4. 多くの特殊記号は、'z' ともう 1 文字で入力できる。例えば、'z/' で '・' が、'z.' で '…' が入力できる。
5. それ以外の特殊記号も名前で入力できる。例えば、'てん' を変換すると「・」が出てくる。

5.4 辞書登録

辞書登録は、以下のように行なう。

1. 登録したい単語を Emacs のリージョンで指定する。
2. `M-x toroku-region` を実行し、指示に従って、読み、品詞、保存先辞書を入力する。

保存先辞書名は通常 1 つだけなので表示される通りに指定すればよい。

5.5 演習問題

1. 適当な文章を使用して日本語入力の練習を行え。
2. 自分の名前を辞書登録せよ。

第 6 章

ファイルについてもう少し

6.1 ファイル (U§5.1, 5.3, 5.4)

計算機上で我々が作るものは、すべてファイルという形で電子的に保存する。ファイルとは、言わば、電子の紙に相当するものである。

ファイルを扱うためには、以下のことを行なう方法を知っておく必要がある。

1. ファイルの新規作成と修正
2. ファイルの一覧表の表示 — どんなファイルがあるか
3. ファイルの内容表示 — ファイルの内容を見る
4. ファイルの削除 — 不要になったファイルを捨てる

これらを行なうには、多くの方法がある。

1 を行なう標準的な方法は 4章で述べた、エディタを使う方法である。これに対して、 2,3,4 は、以下のような Unix コマンドを用いる方法が標準的である。

ファイルの一覧表表示	`ls`
ファイルの内容表示	`cat` *file*
ファイルの削除	`rm` *file*

以下に例を示す。

```
% ls
sample.txt    sample10.txt
% cat sample.txt
   Emacs divides therse, is the one
eral areas, each of which containsin which you usually see the text you
are editing.
% rm sample10.txt
% ls
sample.txt
%
```

6.2 ファイルの整理 — ディレクトリの利用

計算機を使うようになると、どんどんファイルが増えていく一方である。このため、うまく整理しておかないと、収拾がつかなくなる。

先に、ファイルとは、電子的な紙のようなものであると述べたが、丁度、何枚かの紙を封筒に入れて整理できるように、ファイルも封筒に相当するような**ディレクトリ**を利用することによって、うまく整理することができる。紙を封筒に入れ、そうした封筒のいくつかをまた封筒に入れ、といったことが紙に対して可能であるように、ファイルもディレクトリを利用して、複数段に渡って整理することができる。このようなファイルシステムを階層的ファイルシステムと呼ぶ。

いま、以下のような名前の 4 つのファイルがあるとしよう。

1. `letter1.txt` —— 手紙 1
2. `letter2.txt` —— 手紙 2
3. `main.c` —— メインプログラム (C)
4. `sub.c` —— サブプログラム (C)

このような場合、手紙は手紙でまとめ、プログラムはプログラムでまとめるのがわかりやすい。そこで、電子的な封筒に相当するディレクトリを 2 つ作って、それぞれのディレクトリの下に[1]それらのファイルを格納すればよい。以下に実行例を示す。

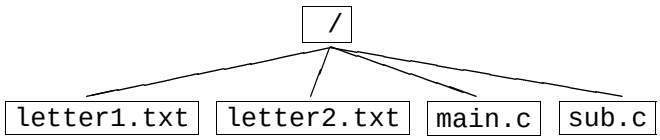
```
[sato@is17e0s00] 60 % ls
letter1.txt     letter2.txt     main.c          sub.c
[sato@is17e0s00] 61 % mkdir letter
[sato@is17e0s00] 62 % mkdir program
[sato@is17e0s00] 63 % ls
letter/         letter2.txt     program/
letter1.txt     main.c          sub.c
[sato@is17e0s00] 64 % mv letter1.txt letter/letter1.txt
[sato@is17e0s00] 65 % mv letter2.txt letter/letter2.txt
[sato@is17e0s00] 66 % ls
letter/         main.c          program/        sub.c
[sato@is17e0s00] 67 % cp main.c program/main.c
[sato@is17e0s00] 68 % cp sub.c program/sub.c
[sato@is17e0s00] 69 % ls
letter/         main.c          program/        sub.c
[sato@is17e0s00] 70 % rm *.c
[sato@is17e0s00] 71 % cd letter
[sato@is17e0s00] 72 % ls
letter1.txt     letter2.txt
[sato@is17e0s00] 73 % cd ../program
[sato@is17e0s00] 74 % ls
main.c  sub.c
[sato@is17e0s00] 75 %
```

上記の実行例で行なっていることは、以下の通りである。

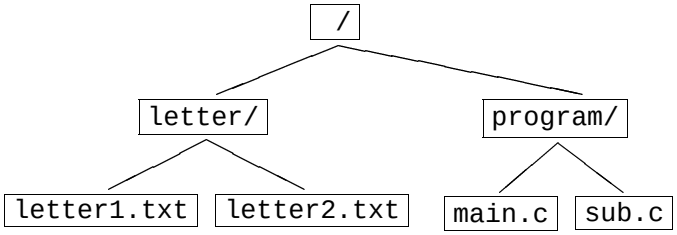
[1]封筒では「中に」となるが、ディレクトリでは「下に」という表現を普通用いる。

60 4 つのファイルがある。
61 `letter` というディレクトリを作る。
62 `program` というディレクトリを作る。
63 4 つのファイルと、2 つのディレクトリがある。ディレクトリは、最後に'/' がついた形で表示される。
64 `letter1.txt` を`letter` の下に移動した。
65 `letter2.txt` を`letter` の下に移動した。
66 移動したのでファイルは 2 つとなった。
67 `main.c` を`program` の下にコピーした。
68 `sub.c` を`program` の下にコピーした。
69 コピーなので、プログラムは存在したままである。
70 プログラムを消去した。ここで、'*' は、任意の文字列を意味する。つまり、'なんとか.c' というファイルを全部消去することを意味する。
71 `letter` の下へ移動した。
72 ここに、2 つの手紙が格納されている。
73 ディレクトリを一段上に移動し (親ディレクトリに移動し)、そこからさらに、`program` の下へ移動した。ここで、'..' は一段上への移動を意味する。
74 ここに、2 つのプログラムが格納されている。

つまり、最初の状態は、



であり、これを、以下のような状態に変更することを行なったわけである。



ディレクトリの指定に関して覚えておくことは、以下の通りである。

親ディレクトリ	`..`
子ディレクトリ	*directory*
複数段に渡って	*directory*/*directory*

これに伴い、ファイルは、ディレクトリを含めて指定することができる。

ディレクトリ直下のファイル	*file*
他のディレクトリのファイル	*directory*/*file*

ディレクトリに関して最低限必要なコマンドは、以下の通りである。

ディレクトリの作成	`mkdir` *directory*
ディレクトリの削除	`rmdir` *directory*
ディレクトリ間の移動	`cd` *directory*

また、ファイルの移動に関するコマンドは、以下の通りである。

ファイルのコピー	`cp` *file1 file2*
ファイルの移動	`mv` *file1 file2*

6.3 階層的ファイルシステム (U§5.2)

ここで、ディレクトリやファイルについてまとめておく。

1. Unix の全ファイルは、大きな木構造の形をしている。
2. ディレクトリは、その木構造の節点に対応する。
3. ファイルは、その木構造の葉に対応する。
4. その木構造の根をルートディレクトリと呼び、‘`/`’ で表す。
5. 各ユーザーには、あらかじめ、各自の専用ディレクトリが割り当てられている。これをホームディレクトリと呼ぶ。 JAIST では、通常、

   `/home/fsXXX/`*username*

   である。ユーザーは、自分のホームディレクトリ下を自由に使うことができる。
6. セッションの間、ユーザーは作業する場所としていつでも 1 つのディレクトリを持っている (1 つのディレクトリに**いる**)。このディレクトリをカレントディレクトリと呼ぶ。カレントディレクトリは、コマンド`pwd` で表示できる。カレントディレクトリは、‘`.`’ で表すことができる。
7. login した直後のカレントディレクトリは、ホームディレクトリとなる。
8. ディレクトリの移動は、コマンド`cd` を用いる。‘`..`’ は親ディレクトリを表す。子ディレクトリは名前で指定する。
9. ディレクトリは一度に複数段移動することができる。この場合、ディレクトリの名前を‘`/`’ でつなぐ。例えば、

   ```
   % cd ../..
   ```

   は、親の親のディレクトリに移動することになる。
10. ファイル (やディレクトリ) を指定する方法には 2 つの方法がある。 1 つは絶対パス指定であり、他方は相対パス指定である。
11. 絶対パス指定は、ルートディレクトリからの経路を指定する。例えば、

    `/home/fs001/sato/RMAIL`

のように。絶対パス指定では、最初が必ず ‘/’ で始まる。

12. 相対パス指定は、カレントディレクトリからの経路を指定する。例えば、カレントディレクトリが、`/home/fs001/sato` であれば、上記のファイルは、

    ```
    RMAIL
    ```

    となる。また、カレントディレクトリが、`/home/fs001/matuzawa` であれば、上記のファイルは、

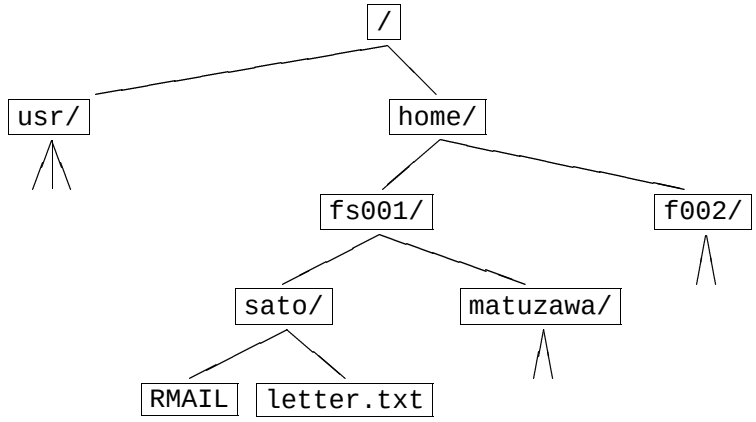
    ```
    ../sato/RMAIL
    ```

    となる。

13. この他に、ホームディレクトリを~で指定することができる。つまり、ユーザー sato は、現在のカレントディレクトリに関わらず。

    ```
    ~/RMAIL
    ```

    で上記のファイルを指定できる。



6.4 ファイルのモードとセキュリティ (U§5.5)

前節の説明より推察される通り、 unix では自分のホームディレクトリ下以外のファイルを見ることもできる。自分のファイルを他人に見られたくない場合にはどうしたらいいのであろうか。

このようなことを可能にするために、 unix の各ファイルは、オーナー情報と、保護モードという 2 つのものを持っている。これらを適切に指定することによって、ファイルを他人のアクセスから保護することができる。

オーナー情報とは、そのファイルの所有者が誰かということを示す情報である。通常、そのファイルを作成した人間が、そのファイルのオーナーとなる。オーナーは、コマンド`chown` によって変更できる。

保護モードとは、そのファイルに対するアクセス権の指定のことである。ファイルに対するアクセス権は、“所有者 (オーナー)”、“グループ内の人”[2]、“グループ外の人” それぞれに対して、読み出し (read), 書き込み (write), 実行 (execute) の 3 種類の権限を設定することができる。

[2]グループとは、ユーザーのまとまりのことをいう。ユーザーは、どこかのグループに属している。自分のグループを知るには、コマンド `groups` を使う。ファイルのグループ名を知るには コマンド `ls -lg` を使う。

すべてのファイルは、新規作成された時に、ある保護モードが設定される[3]。エディタ等で作成したファイルは、通常、所有者以外の人が書き込めないモードが設定される。ファイルのモードはコマンド `ls -l` によって知ることができる。(コマンド `ls -lg` を使えば、グループ名も含めて表示される。)

```
[sato@is17e0s00] 87 % ls -lg
total 4677
-rw-r--r--  1 sato     is         220201 Nov  2 11:11 a
-rw-r--r--  1 sato     is         220181 Nov  2 11:23 a2
-rwxr-xr--  1 sato     is            102 Nov  2 11:21 abc*
-rw-r--r--  1 sato     is         135744 Nov  2 10:12 bar
-rw-r--r--  1 sato     is           1093 Nov  2 11:51 etl
-rw-r-----  1 sato     is        4121770 Nov  2 09:45 fj.db.old
-rwxr-xr-x  1 sato     is            130 Nov  2 10:14 foo*
-rwxr-xr--  1 sato     is            495 Nov  2 11:21 getauther*
-rw-r--r--  1 sato     is          39368 Nov  2 10:02 head
drwxr-xr-x  2 sato     is            512 Feb 18 14:43 tmp/
[sato@is17e0s00] 88 %
```

既存のファイルのモードを変更するには コマンド `chmod` を使う。

`% chmod go-rw` *file1*	ファイル *file1* を他人が読み書きできないよう設定する。
`% chmod go+r` *file2*	ファイル *file2* を他人が読めるよう設定する。
`% chmod u-w` *file3*	ファイル *file3* を自分でも書けないよう設定する。
`% chmod go-rwx` *dir1*	ディレクトリ *dir1* の内容の読み書き および そのディレクトリへの移動 (cd) を他人に禁止する。

詳しくは、参考書 **(U§5.5)** を参照せよ。

6.5 ファイルの出力

最後に、ファイルのハードコピーをプリンタから出力する方法について述べる。これは、以下のコマンドによって実行できる。

`txt2ps` *file* `| lpr`

ここで、 `txt2ps` は、通常のファイルを PostScript と呼ばれる形式で記述されたファイルに変換するコマンドであり、次の `lpr` が、プリンタへの出力を行なうコマンドである。この lpr コマンドは、後述の dvi2ps 等いろいろなコマンドの出力をプリンタへ送ることができる。ハードコピーが出力されるプリンタは、通常、マシンがあるフロアのどちらかのプリンタである。なお、 `lpr -P`*printer* のように、明示的にプリンタ名を指定すると、指定したプリンタに出力することができる。

6.5.1 印刷ジョブの管理

lpr コマンドでプリンタへ送られるファイルは、一旦, 待ち行列 (キュー) に入り、先頭から順に印刷される。これは他人の印刷ジョブとの衝突を避けるための仕組みである。

印刷中、もしくは、印刷待ちのジョブは途中でキャンセルすることができる。現在のキューの一覧を見るには lpq コマンドを実行する。キューの中のジョブをキャンセルするには lprm コマンドを用いる。このとき、 lpq コマンドで表示された「ジョブ番号」を指定する。

[3] コマンド umask による設定によって決定される **(U§10.4 (3))**。

6.5.2 大きなファイルの印刷

通常の lpr コマンドの使い方では印刷できるファイルの大きさに制限がある。この大きさは、プリンタやそのサーバの設定によって異なるが、通常は 1MB から 10MB 程度である。この上限よりも大きなファイルを送ると上限で切られるため，正常に印刷できない．上限よりも大きなものをキューに送ってしまった場合には lprm コマンドでジョブを取り消した後で，前述の「…… | lpr」の替わりに以下の方法を行う。

```
% …… > file.ps
% lpr -s file.ps
(印刷が終了したら)
% rm file.ps
```

6.6 演習問題

1. ファイルの作成、削除を実際に実行せよ。
2. cp コマンドについて調べよ。
3. ディレクトリの作成例を実際に実行せよ。
4. 作成したディレクトリの削除を試みよ。
   (ヒント: システムが出力するメッセージを良く読むこと。)
5. pwd コマンドについて調べよ。
6. login した時のカレントディレクトリはどこか？
7. ファイルの保護モードを調べよ。
8. 他人に見られたくないファイルの保護モードを他人に見えないように変更せよ。
9. 自分がどのグループに所属しているかを調べよ。

第 7 章

Emacs のちょっと高度な使い方

7.1 マルチプル・バッファとマルチプル・ウィンドウ (GE§2.10)

Emacs では、一度に複数のファイルを編集することができる。それを可能にするのがマルチプル・バッファとマルチプル・ウィンドウの機能である。

7.1.1 マルチプル・バッファ

今、`main.c` というファイルを編集しているとしよう。ここで、`sub.c` というファイルの一部を変更しなければならないことに気がついたとしよう。その場合、どうしたらよいであろうか。勿論、`main.c` をセーブして、Emacs を終了して、また、Emacs を起動して、`sub.c` を読み込んで、その内容を修正するということは可能である。しかし、そのような手間をかけなくても、マルチプル・バッファの機能を用いれば、一度に複数のファイルを編集することができる。

すべきことは、読み込みたいファイル (ここでは、`sub.c`) を C–x C–f によって読み込むことだけである。Emacs は、自動的に 2 つめのファイルに対するバッファを生成し、その内容が画面に表示する。このバッファに対して、修正を行なえばいいのである。

もし、元の `main.c` というファイルを再び編集したくなった場合はどうすればよいのか。これをおこなうのが、表示するバッファの切替えである。

C-x b 表示するバッファを変更する

このコマンドを実行すると、表示するバッファの名前を聞いてくるので、これに答えると、画面は、そのバッファの表示に切り替わる。ここで、バッファの名前は、通常、ファイル名と同じである。

この他、バッファに対する操作コマンドには、以下のものがある。

C-x k バッファの削除

C-x C-b バッファの一覧表示

なお、カットアンドペーストは、バッファにまたがった操作が可能である。つまり、あるバッファ (ファイル) の一部を、別のバッファ (ファイル) に移動したり、コピーしたりすることが可能である。

7.1.2 マルチプル・ウィンドウの操作

Emacs では通常、画面全体を使って、一つのバッファを表示するが、画面を分割して複数のバッファの内容を同時に表示させることもできる。この機能は、他のバッファの内容を参照しながら、編集を行ないたい場合に便利である。

画面を上下 2 つに分けて 2 つのバッファを同時に表示するコマンドは、 C-x 2 である。このコマンドの実行した直後、 2 つのウィンドウは同じバッファを表示している。しかし、先の表示バッファの変更コマンド (C-x b) や、新規ファイルの入力コマンド (C-x C-f) によってウィンドウの表示を変更すれば、 2 つのウィンドウで異なるバッファを表示させることができる。

2 つのウィンドウのうち、カーソルがあるウィンドウに表示されているバッファが編集の対象となるバッファである。この、カーソルのあるバッファ (ウィンドウ) を**選択されている**バッファ (ウィンドウ) と呼ぶ。

以下に、ウィンドウ操作の代表的コマンドを示す。

C-x 2 ウィンドウを上下に 2 分割する。

C-x 3 ウィンドウを左右に 2 分割する。

C-x o 他のウィンドウを選択する。

C-x 0 選択されているウィンドウを消す。

C-x 1 選択されたウィンドウ以外の全てのウィンドウを消す。

7.1.3 バッファの一覧表示

バッファの一覧は、コマンド C-x C-b によって表示される。このコマンドを実行すると、画面が上下に分割され、下のウィンドウにバッファの一覧が表示される。表示例を以下に示す。

```
 MR Buffer          Size  Mode            File
 -- ------          ----  ----            ----
.*  emacs-2.tex    6924  TeX           /home/fs001/sato/lec/lg/93/emacs-2.tex
    .emacs          2796  Emacs-Lisp      /home/fs001/sato/.emacs
    main.tex        2693  LaTeX           /home/fs001/sato/lec/lg/93/main.tex
    *Help*          7467  Fundamental
    *scratch*       0     Lisp Interaction
 % RMAIL            102970 RMAIL          /home/fs001/sato/RMAIL
*  *Buffer List*   0     Buffer Menu
```

行の最初についている '*' の印は、そのバッファが最後にセーブした時から変更されていることを示している。'.' は、そのバッファが現在選択されていることを示している。

この一覧表示のバッファ (*Buffer List*) に **C-x o** などでカーソルを移動させ、所望のバッファ名の行にカーソルを移動して、そのバッファに対する色々な操作が可能である。そのバッファを表示するには、以下のコマンドが便利である。

f そのバッファを表示する。

7.2 ディレクトリ・エディタ (GE§2.4)

Emacs の中からファイル操作を行なう機能として、ディレクトリ・エディタ (dired) がある。ディレクトリ・エディタを使うと、ディレクトリ内のファイル名の一覧を見ながら、ファイルの読み込み、名前の変更、消去等の操作を行うことができる。

ディレクトリ・エディタは、コマンド C-x d で起動する。ミニバッファに、表示したいディレクトリ名をきいてくるので、それを入力する。ディレクトリ・エディタのウィンドウでは、カーソルのある行のファイルに対して、以下のコマンドが利用できる。

d ファイルに消去マークをつける。

u ファイルの消去マークをとり消す。

x 消去マークがついているファイルを本当に消去する。

~ Emacs のバックアップファイル (最後に ~ (チルダ) がついているファイル) にすべて消去マークをつける。

C ファイルをコピーする。

R ファイルの名前を変更する。

f ファイルをバッファに読み込む。

ディレクトリ `/usr/local/lib/user.skel` を表示させた場合のウィンドウ表示例を下に示す。

```
total 171
drwxrwsr-x  2 root          512 Mar 22 19:58 .
drwxrwsr-x 47 root         2048 Mar  9 12:34 ..
-rw-rw-r--  1 sato          558 May  8  1992 .cshrc
-rw-rw-r--  1 sato         1033 Apr 12  1992 .eggrc
-rw-rw-r--  1 sato         1553 May 11  1992 .emacs
-rw-rw-r--  1 sato          471 Apr 20  1992 .login
-rwxrwxr-x  1 sato          105 Apr 12  1992 .setinitials
-rw-rw-r--  1 sim            88 Apr 16  1992 .skk
-rw-rw-r--  1 sim        137644 Apr 23  1991 .skk-jisyo
-rw-rw-r--  1 sato         6592 Apr 14  1992 .twmrc
-rw-rw-r--  1 sato          718 Apr 12  1992 .xinitrc
```

これは便利! 変な名前のファイルを作ってしまい、 rm コマンドでファイルを消せない場合でも、dired を使うと消せる場合が多い。

7.3 演習問題

1. 前回作成した 3 つのファイル `sample.txt`, `sample2.txt`, `sample10.txt` をそれぞれバッファに読み込み (`C-x C-f` を使う)、バッファ表示の変更コマンドによって、表示が変更されることを確認せよ。 (`C-x b` を使う)
2. バッファ一覧を表示させ (`C-x C-b`)、現在表示中のものとは異なるバッファを表示させよ。
3. 一度 Emacs を終了させ (`C-x C-c`)、再起動した後、ディレクトリ・エディタ を使って、`sample.txt` をバッファに読み込め。 (`C-x d` および `f` を使用)

第 8 章

電子メールの使い方

8.1 電子メールとは

電子メール・システムとは、計算機ネットワークを利用した手紙配達システムである。通常の手紙は、紙を媒体とするが、電子メールの場合は、電子の紙、すなわち、ファイルを媒体として手紙配達を実現する。この電子メールを利用すると、電子的に作成した手紙を、紙に印刷する必要なく、そのまま相手方に電子的な手段で送ることができる。

電子メールは、 JAIST 内ならば、ほとんど瞬時に配達される。また、学外でも、国外でも、コンピューターネットワークの幹線に直結しているところならば、数秒以内に配達される。

電子メールに関して知っておかなければならないことは、以下の 4 つである。

1. 電子メールのアドレス
2. 電子メールの出し方
3. 電子メールの読み方
4. 電子メールの整理の仕方

8.2 電子メールアドレス

郵政省郵便では、手紙を届けたい相手の住所と名前を指定する。電子メールでこれに相当するものが、**電子メールアドレス**である。

我々の電子メールアドレスは、以下の通りである。

user`@jaist.ac.jp`

ここで、 *user* は各自のユーザー ID である。例えば、筆者のアドレスは、

```
sato@jaist.ac.jp
```

となる。ここで、`jaist.ac.jp` はドメイン名と呼ばれるもので、いわゆる住所に相当するものである。`jaist` は JAIST を、`ac` は大学を、`jp` は日本を表している。このアドレスは、全世界で通用するアドレスである。

なお、 JAIST 内では、

```
sato
```

のようにユーザー ID だけのアドレスも有効である。これは、「市内」に相当するようなものである。

8.3 MH (mh-e)

エディタに、 Mule(Emacs), vi, ed といった様々な種類があるように、電子メールを読んだり書いたりする方法にもいくつもの種類がある。ここでは、電子メールの処理ツール[1]として MH システム (**MH§1**) を取り上げる。 MH は UNIX 上でメールを操作する種々のコマンドの集合体である。 Mule から MH を利用するためのフロンドエンドプロセッサとして mh-e(**GE4.3**) がある。以下、 mh-e について説明する。

なお、 Emacs 標準のツールである RMAIL については付録 Gを参照せよ。

8.4 メールの送信

8.4.1 メールの作成と送信

メールを書いて送るには、以下の操作を行う。

M-x mh-smail　送信するメールを作成する。

mh-smail を実行すると、メールヘッダ (宛先や宛名) を作成するために、ミニバッファに問い合わせがいくつか表示されるので、 以下の手順で処理を進める。

1. `To:` の表示の後に、メールを送りたい相手のメールアドレスを書いて RET 。このとき、 `To:` とアドレスの間には空白が必要である。
2. `Cc:` の表示の後に、他にも同じメールを送信したい人がいれば、その人のメールアドレスを記入し、いなければ空欄のまま RET 。
3. `Subject:` の表示の後に、メールの題目を書いて RET 。

必要なヘッダ項目の入力が完了すると、図 8.1のように画面が上下 2 つに分割され、下がメール作成用の画面 (ウィンドウ) となる。メール画面において、“`--------`” の行の上部が、メールヘッダ、その下が、これから作成するメールの本文領域である。相手に送りたいメールの本文は、この本文領域に書き込んでいけばよい。

メールヘッダと本文との境界を示す“`--------`” の行は、これを消したり、書き換えたりしてはいけない。一方、ヘッダの項目 (`Subject:` など) の内容は必要があれば追加したり書き換えたりしても良い[2]

本文の作成が終了したら、最後に自分の名前を書き、メールヘッダの内容に誤りがないか確認後、以下のコマンドでメールを送信する。

C-c C-c　メールを送信する。

[1]正確には、電子メールユーザーエージェント (MUA) と呼ばれる

[2]電子メールの書式については国際的な取り決め (RFC) で規定されており、ヘッダの構成要素やヘッダや本文で使用できる文字等にはルールがある。

図 8.1: 送信メールの作成時の画面

もし、メールの作成を中止したい場合には、以下のコマンドを使う。

C-c C-q メールの作成を中止し、その内容を破棄する。

C-x k 作成中のバッファを消す。 (draft ファイルは存続する。)

8.4.2 複数の相手に同じメールを送る

メールヘッダの `To:` に、複数のメールアドレスを書くと、それぞれに同じメールが届く。このとき、各メールアドレスはコンマ (`,`) で区切る[3]。また、 `cc:` で始まる行にアドレスを書くと、そこにもメールが届く。 `cc` はカーボンコピーを意味する。

```
To: sim@jaist.ac.jp, mit@jaist.ac.jp
cc: morioka@jaist.ac.jp
Bcc: sim
```

基本的には、 `To:` には、そのメールを送信する相手、 `cc:`には、送信する相手ではないが、そのメールの内容を知らせておきたい相手、のように使い分ける。 `To:` や `cc:` は、メールの一部 (メールヘッダ) として相手に届くので、相手はそのメールが誰に送信されたのかを知ることができる。これに対して、 `Bcc:`は、ブラインド・カーボンコピーと呼ばれ、メールは送信されるが、その事実は他のメールの送信先には隠される。通常は、自分宛のカーボンコピー[4]はブラインド・カーボンコピーとする。

[3]必ずコンマ (`,`) を用いること。読点 (、) は使ってはいけない。

[4]`Bcc:` の代わりに `Fcc:` を使う方法もある。 `Fcc:` はフォルダコピーと呼ばれ、メールアドレスの代わりにフォルダ (8.5.4節) 名を指定する。

8.4.3 メール作成の注意

メールを使用する場合、以下の点に注意すること。

1. アドレスを良く確認し、間違えないようにすること。
2. 1 行の長さを 70byte (英字 70 文字、日本語 35 文字) 程度以下にすること。つまり、適当に改行すること。
3. あまり長いメールを送らないこと。 1 つのメールのサイズが 100Kbyte を越えるような場合は、分割すること。
4. サブジェクト (subject) で漢字を使用しないこと[5]。漢字はメールの本文のみで使用可能である。
5. 漢字コードは、 JIS コードを用いること。

8.5 メールを読む

メールを読む作業は、まず、メールの到着を知ることから始まる。メールの到着を知る方法には色々な方法がある。例えば、 login 時にメールが到着していれば、 You have new mail. 等のメッセージが出力される。また、ウィンドウ・システムを使っていれば、ディスプレイ右上のポストにメール到着の有無が表示される。 unix コマンド `from` は、到着したメールがどこから来たものかを教えてくれる。以下、 mh-e における mh-rmail による メールの読み方と、返事の書き方について説明する。

8.5.1 mh-rmail の起動

到着メールを読むには以下のコマンドを実行する。

M-x mh-rmail　メール表示コマンドの起動。

mh-rmail を実行すると到着メールがある場合には、ミニバッファ領域に “`inc +inbox...done`” と表示されて、到着メールの一覧表が表示される (図 8.2)。過去にメールを読んで以来、新たに到着したメールが何もない場合には、 “`No new mail`” と下に表示されるだけで、一覧表の画面には何も表示されない。

図 8.2において、各行の最初の欄の数字はメールに付けられた整理番号、次は メールの日付、発信人の氏名、 Subject:欄と本文先頭の一部が表示されている。

8.5.2 mh-rmail の終了

`mh-rmail` を終了するコマンドは以下の通りである。

q　`mh-rmail` を終了する。

このコマンドを実行すると、通常の Mule の画面に復帰する。

[5]MIME に対応している場合は漢字なども使用可能である。

図 8.2: 到着メールの一覧画面

8.5.3 mh-rmail のコマンド

メールの一覧表示画面で使える代表的なコマンドを以下に示す。

. テキストカーソル行のメールを表示する。

SPC 表示中のメールを次のページにスクロール。

DEL 表示中のメールを前のページにスクロール。

n 次の (消去印の付いていない) メッセージを表示する。

p 前の (消去印の付いていない) メッセージを表示する。

r テキストカーソル行のメールに返事を書く。 (8.5.5参照)

f テキストカーソル行のメールを他の人に転送する。 (8.5.5参照)

q `mh-rmail` を終了する (画面を閉じる)。

8.5.4 新着/過去のメールを読む

`mh-rmail` は、計算機システムが保有するシステムメールボックスの内容を表示しているわけではない。実は、システムメールボックスに届いたユーザ宛のメールを取り出して、ユーザのメールディレクトリ `~/Mail` の下にある一時保管用ディレクトリ `inbox` に収納して、そこでメールの表示等の処理を行っているのである。このとき、1 つのメールが 1 つのファイルとして収納され、1 から始まる一連番号がファイル名となっている。この保管用ディレクトリのことを、MH では **フ**

ォルダ と呼び、今現在、処理の対象となっているフォルダ (最初は inbox) を **カレントフォルダ** と呼んでいる。

mh-rmail を起動したときは、このとき フォルダ inbox に読み込んだメールの一覧しか表示されない。したがって、 mh-rmail の実行後に新たに到着したメールを読むためには、到着メールをフォルダに読み込まなくてはいけない。これには、コマンド i を実行する。また、過去にフォルダに読み込んだメールを再度読むためには、 M-r を実行する。

i 新たに到着したメールを フォルダ inbox に読み込む。

M-r カレントフォルダをスキャンし直して、メールの一覧表を表示する。

M-f カレントフォルダを指定のフォルダに変更する。 (8.6参照)

8.5.5 返事を書く

受け取ったメールに対して返事を書くには、メールの一覧画面上で該当するメールにテキストカーソルを合わせた状態で r と打つ。

r テキストカーソル行のメールに返事を書く。

このコマンドを実行すると、誰に対して返事を出すか、以下のような問合せがミニバッファに表示される。

```
Reply to whom:
```

発信人本人 (From: 欄のアドレス) にのみ返事を出す場合は単に RET、メールが配送された全ての人 (To: と cc: 欄に記載されているアドレス) にも返事を出す場合は、 all RET と答える[6]。すると、画面が上下に 2 分割され、一方にメール作成用の画面、もう一方に相手から届いたメールが表示される。メール作成用の画面では、相手のメールアドレス等の必要なメールヘッダの項目が自動的に記入されている [7]。例えば、 Subject: 欄には、相手のメールに対する返事 (reply) であることを明示するために用件の前に、 Re: が付加されている。本文の書き方、送信の方法は、 8.4.1 節と基本的に同じである。

注意!! 送信前に、 To: と cc: 欄が正しく記入されているか十分確認し、必要があれば編集すること。差出人にのみ返事するつもりが、 cc: 欄に他のアドレスも指定してあったため、そちらにもメールが送られてしまい、思わぬ恥をかくこともある。

相手のメールを自分のメールの中で引用したい場合には、以下のコマンドが使える。

C-c C-y 相手のメールの内容を自分のメールのテキストカーソルの位置にコピーする。

コピーした部分の行の先頭には引用を示すためのマーク > が自動的に挿入される。後は、適当に編集して使えば良い。その他、ファイル ~/.signature に自分の名前等を書いておくと、コマンド C-c C-s で、それを本文の最後に読み込むことができる。

[6] この他に、 to や cc

[7] 必要ならば、変更したり、追加したりしてよい。

図 8.3: 返事のメール作成時の画面

8.6 メールの整理

受け取ったメールをそのままにしておくと、メールの数が膨大になり収拾がつかなくなってくる。そのためメールの整理が必要になる。不要なものは消去したり、保存しておきたいものは用件毎にまとめて整理しておくとよい。

メールを消去するには、以下の 2 ステップが必要である。

d テキストカーソル行のメールに消去マークを付ける。

x マークの付いているメールに対して実際の処理を行う。(消去マークであれば、そのメールを消去する。)

複数のメールを消すときは、該当メール全てにコマンド d で消去マークを付け、最後にコマンド x を一回実行すればよい。コマンド x によって消去する前であれば、以下のコマンドで消去マークを取り消すことができる。

u テキストカーソル行の 1 行上のメールの消去マークを取り消す。

消去せずにとっておきたいメールは、別のフォルダに入れ直して整理しておくと良い。これには、コマンド o で移動マークを付け、コマンド x で移動を実行する。

o テキストカーソル上のメールを指定のフォルダに移すマークを付ける (^キーも同様)。

コマンド o を実行すると、ミニバッファに以下のような問い合わせが表示されるので、保存先のフォルダの名前を入力する。

```
Destination folder?  +
```

フォルダの実体はディレクトリなので、区切り記号 (/) を使った階層的なものも許される。フォルダを指定する際には、上の例のように名前の前に + 記号を付ける約束になっている。

ところで、 inbox 以外のフォルダに整理されたメールを見るには、コマンド `M-f` によってカレントフォルダを変更すれば良い (8.5.4参照)。

8.7 MH メールの管理

8.7.1 不要なファイルの削除

先の 8.6節においてメールを消去する方法について述べたが、この操作は、メールファイルを MH から見えなくしているだけで、ファイルそのものは名前が変更された状態で存在している。これには誤ってメールを消してしまっても復旧できるという良さがある半面、消去メールファイルを放置しておくと計算機のディスク資源を圧迫するという問題がある。

消去メールファイルの本当の意味での削除は、利用者が自分の責任で必要に応じて行うことになっている。最も単純な方法は、以下のように UNIX のファイル削除コマンド `rm` を用いることである。

```
% cd ~/Mail/inbox
% rm #*
```

1 行目で、フォルダ inbox に移動し、 2 行目で、ファイル名が # で始まる全てのファイルを削除している。これは、 MH で消去されたメールのファイルは、元のファイル名の先頭に# が付いたものに変更になっていることを利用している。

注意!! `rm #*` を間違って、 `rm # *` としてしまうと、必要なメールファイルも含めて全て削除してしまうので、十分に注意すること。安全を期するには、煩雑ではあるが、 `rm -i #*` とすべきであろう。

8.7.2 セキュリティ

フォルダ中のメールを他人に見られたり、操作されたりするのを防ぐためにフォルダの存在するトップディレクトリ `~/Mail` の保護モードを以下の unix コマンドで変更しておくと良い。

```
% chmod go-rwx ~/Mail
```

8.7.3 エイリアス

メールアドレスにエイリアス (別名) を登録しておくと、別名でメールアドレスを指定することができる。エイリアスの登録には、エイリアスファイル`~/Mail/Aliases` をエディタで作成する。そこに例えば以下のような内容を書いておく。

```
yuka:  yk1234@koko.soko.co.jp
i112:  matuzawa@jaist.ac.jp, sim@jaist.ac.jp
```

左の欄が登録する別名で、右の欄が対応する本当のアドレスである。別名の後には必ずコロン (:) を付ける。

次に、 MH の設定ファイル (プロファイル) `~/.mh_profile` に以下の行を追加しておく。

```
Aliasfile:  Aliases
```

8.7.4 ファイルを送るときの注意

メールで通常のメッセージ以外のもの、例えば、プログラム等を送るときは注意を要する。使うことのできる文字コードには国際的な取り決めがあるため、これ以外の文字コードを含むものは絶対に送ってはいけない。例えば、 LaTeX の dvi ファイル (12 章)、 C プログラムのコンパイル結果 (*xxx*`.o`, `a.out`) は、そのままの形で送ってはいけない。大雑把に言って、画面に表示できない文字コードを含むものは送っていけない。

また、通常のテキストファイルであっても、行の先頭にピリオド (`.`) があると、それ以降の行は無視されて相手に届かない場合がある。

8.7.5 メーリングリスト

電子メールは一人に対して送るだけでなく、一度に特定のグループに属する全員に送ることも可能である。このような仕組みをメーリングリストと呼ぶ。

学内で広く利用されているメーリングリストには次のようなものがあるが、この他にも各研究室や個人で管理するメーリングリストが多数存在する。

is-gakusei98 1998 年入学の情報科学研究科の学生全員

ms-gakusei98 1998 年入学の材料科学研究科の学生全員

ks-gakusei98 1998 年入学の知識科学研究科の学生全員

is-faculty 情報科学研究科・情報科学センターの教官

ms-faculty 材料科学研究科・新素材センターの教官

ks-faculty 知識科学研究科・知識科学教育研究センターの教官

all-students 学生全員

jaist-all JAIST の全構成員

これらのメーリングリストはいずれもメンバー数が多く、不用な情報を流すと大勢が迷惑するので、使用する際には十分注意しなければならない。また、一通のメールがメンバー全員宛に複製されるため、大きなメールを送るとメールサーバに大きな負担をかけることになる。特に、 jaist-all は教官・学生・事務職員などを全員含むものであるから、これらの全員に対して、緊急でかつ重要な情報を伝達する場合にのみ利用されるべきである。それ以外の場合には、ニュースの frontier.announce 等を利用するとよい。

注意!! 本学のアカウントを使用しなくなる場合、個人的にメンバー登録を行ったメーリングリストがあれば、それぞれの登録の抹消の手続きを自分で責任をもって行わなければならない。読まれないメールが大量に蓄積されるのは計算機資源の無駄使いである。

8.8 演習問題

1. Emacs から自分宛にメールを出す実験をせよ。
   - 宛先は ‘ユーザ名`@jaist.ac.jp`‘ とする。
   - `Subject:` 欄は、‘`test mail 1`‘ とする。
   - メールの本文は適当に書いて構わない (英文か日本語)。
2. Emacs から `mh-rmail` コマンドを利用して以下の実験をせよ。
   - mh-e を起動して、先に出したメールが届いているか確認せよ。 (`M-x mh-rmail` を使用)
   - そのメールに返事 (reply) を出せ。返事の内容は適当でよい。 (`r` を使用)
   - 今出したメールを mh-e のバッファに読み込んで、返事が届いているか確認せよ。 (`i` を使用) また、届いたメールのヘッダ部分がどうなっているか確認せよ。
   - そのメールを `mymail` という名前のフォルダに保存せよ。 (`o` を使用)
3. 自分以外の JAIST 内部の知人 2 人以上を宛先とするメールを書け。その際、ヘッダー部に、カーボン・コピー行 (`Cc:`) を追加し、そこに自分のアドレスを書いてメールを送ったときと、代わりにブラインド・カーボン・コピー行 (`BCC:`) に自分のアドレスを書いて送った場合の違いについて確認せよ。

第 9 章

ネットワーク・ニュース

9.1 ニュースとは

電子メールが基本的に 1 対 1 の通信 (パーソナル・コミュニケーション) であるのに対して、ネットワーク・ニュースは不特定多数間での通信手段 (マス・コミュニケーション) である。ネットワーク・ニュース (以下「ニュース」) は、多くの人たちが情報交換をしたり議論したりするための場を提供する。

ニュースに関して知っておくべきことは、以下の 2 つである。

1. ニュースの読み方
2. ニュースの出し方

ここでは、 Emacs を用いてニュースを読んだり書いたりする方法について述べる。

9.2 ニュースの読み方 (GE§4.5)

ニュースを読むためには、まず、 Emacs で GNUS を起動する。

```
M-x gnus
```

すると、オープニング画面のあと、ニュースグループのリストが現れる。

9.2.1 ニュースグループ

「ニュースグループ」とは、話題ごとに分割された、情報交換の「場」である。ニュースグループは階層構造を持っており、その一番上の階層によって記事の性格が多少異なる。代表的なものは、次のようになっている。

`frontier`	JAIST 専用 (基本的には学外へ流れない)
`hokuriku`	北陸地方の話題
`fj, jp` など	日本国内の話題
`comp, news, talk` など	特定の地域によらない話題 (通常は英語)

frontier で始まるニュースグループは JAIST 学内のローカル・ニュースであり、以下に示すようなニュースグループがある。

frontier.announce	行事予定、全学的アナウンス
frontier.campus	大学に関する一般的話題
frontier.circle	学内サークル活動の連絡や情報
frontier.classifieds	売ります／買います／あげます etc
frontier.general	誰もが読むべき最も一般的な話題
frontier.jokes	ジョーク
frontier.lecture	講義に関する連絡
frontier.library	図書館
frontier.living-in.avenir	アブニール
frontier.living-in.dormitory	学生寮
frontier.living-in.epoch21	エポック
frontier.living	生活情報
frontier.misc	その他の話題あれこれ
frontier.rec.cooking	料理
frontier.rec.sake	酒
frontier.rec.ski	スキー
frontier.rec	娯楽情報
frontier.sys.admin	計算機環境の管理者用
frontier.sys.announce	計算機環境のアナウンス
frontier.sys.mac	Macintosh に関する話題
frontier.sys.misc	計算機環境に関するあれこれ
frontier.sys.question	計算機環境に関する質問
frontier.sys.sun	Sun (SPARC) に関する話題
frontier.test	テスト用

9.2.2 ニュースグループの選択

GNUS が起動すると、ニュースグループ名の一覧表示の画面になる。各ニュースグループ名の左側の数字は、未読の記事の総数を示している。未読の記事が無い場合には、そのグループ名は表示されない[1]。

読みたいニュースグループの上にカーソルを持っていき、スペースを入力する[2]。

9.2.3 ニュース記事を読む

ニュースグループを選択すると、画面が 2 つにわかれる。上のカーソルのあるウィンドウは、ニュースの記事一つ一つに対応したサブジェクト (記事の題名) が表示される。下のウィンドウは、ニュースの記事が表示される。

ここでスペースを入力すると、下の記事のウィンドウがスクロールする。一つの記事の最後まで来ると、次のスペースで、自動的に次の記事に移動する。つまり、スペースを入力し続けることで、ニュースの記事を順番に読み進めていくことができる。

記事を順番に読むのではなく、読みたい記事から読む方法もある。そのためには、まず、読みたい記事のサブジェクトの上へカーソルを移動し、そこでスペースを入力すればよい。

はじめに選んだニュースグループの記事を全部読んでしまった状態でスペースを入力すると、次のニュースグループに進む。

なお、任意の時点で、‘q’ を入力することにより、記事を読むモードから、最初のニュースグループ選択の画面に戻ることができる。

[1] 全ニュースグループの一覧を表示させたいのであれば、‘L’ を入力する。
[2] ニュースグループ名を選択するには、この他に、‘j’ と入力した後、希望するニュース名を入力して指定する方法もある。

9.2.4 終了

ニュースグループ選択の画面で ‘q’ を入力すれば、本当に終了していいか聞いて来たのち、 GNUS を終了する。

9.3 ニュースを出す — 記事を投稿する

記事の投稿には、新しい話題を新規に自分が投稿する場合と、誰かの投稿した記事をもとに投稿 (フォロー) する場合がある。

- 新規に記事を投稿するには、カーソルがニュースグループの画面にいるときか、サブジェクトの画面にいるときに、 ‘a’ を入力する。また、通常の GNUS を起動していない場合は、 M–x gnus-post-news または M–x message-news[3]を実行すればよい。
- フォローアップするには、その記事を読んでいる時に、 ‘f’ を入力する。もし、その記事を自分の記事の中で参照したい場合には、 ‘F’ を入力すると、参照マーク ‘>>’ の後に記事が挿入される。

上記のコマンドを入力すると、以下のような問い合わせが表示されることがあるが[4]、その場合には ‘y’ と答える。

```
Are you sure you want to post to all of USENET? (y or n)
```

または、

```
Are you sure you want to followup to all of USENET? (y or n)
```

新規投稿では、投稿先のニュースグループ名、配布範囲 (distribution) の問い合わせが出るので、それに答える。

続いて、記事を書く画面になる。編集はメールを書く際と同じように、 Emacs の編集コマンドと殆ど同じものを使用することができる。

記事を書き終えたら、 ‘C-c C-c’ によって記事を投稿する。

9.4 記事投稿のルール

ニュース記事は広い範囲に渡って配布され、不特定多数の人が見るため、記事投稿においては、モラル等に十分注意する必要がある。まずは JAIST 内のニュースグループ (frontier.xxx) からスタートし、経験を積んでから外部のニュースグループにデビューすべきである。

以下に記事投稿において注意すべき点を示す。

- 投稿のテストには `frontier.test` を使用する。
- 記事の内容によって適切なニュースグループに投稿する。

[3] これは Emacs のバージョンによって異なる。
[4] Emacs や GNUS のバージョンによって異なる。

- 投稿者は自分の記事に責任を持つ。
  - 言葉使いは、投稿者の人格と品位をあらわす。
  - 他人を中傷したり、プライバシーを侵害する記事は投稿していけない。文章による表現は思わぬ誤解の原因になることがあるので、細心の注意を要する。
  - 著作権を侵害してはいけない。一般の文献の引用のみならず、他人のニュース記事やメールを無断で引用することは著作権の侵害にあたる。少なくとも出典を明確にすべきである。
- 営利目的に使用してはいけない。

第 10 章

シェル

10.1 シェルとは

シェルとは、 unix とユーザーを橋渡しするインタフェースである。シェルは、 login によって起動され、その後セッションの終了 (logout) まで、ユーザーの入力したコマンドを解釈し、実行することを繰り返し行なう。

シェルにはいくつかの種類があり、ユーザーは好みのシェルを用いることができる[1]。以下に、現在、 JAIST で使用可能な代表的シェルの一覧を示す。

プログラム名	通称	説明
sh	B shell (Bourne shell)	基本的シェル
csh	C shell	C 言語的なプログラムが書ける
tcsh	T C shell	csh の発展形
bash		Gnu シェル

JAIST のユーザー登録時には、 tcsh か csh が設定される。ここでは、両者に共通の基本的な機能について述べる。また、 tcsh の便利な機能についても触れる。

10.2 シェルの便利な機能

シェルの基本的な機能は、ユーザーの入力したコマンドを解釈して、それを実行することである。これ以外に、ユーザーのコマンド入力の労力を軽減する便利な機能がある。

10.2.1 コマンド行編集

tcsh では、コマンド行の編集に Emacs と同様の以下のような編集コマンドが使える。打ち間違えた時の訂正に便利である。

C–f, C–b, C–a, C–e, C–d, DEL, C–k, C–y

また、

C–p, C–n

[1] `man passwd` を参照せよ。

により、以前に入力したコマンドを現在のコマンド行にコピーすることができる。これを編集すれば、同じようなコマンドを入力するときに便利である。

10.2.2 ヒストリ機能 (U§6.8)

過去に実行したコマンドを覚えておいて、それを (修正して) 再実行できる。この機能を用いると、キー入力の手間を軽減できる。

`% !!`	直前に実行したコマンドを再実行する。
`% !mule`	mule という綴りで始まる過去のコマンドを再実行する。
`% history`	過去に実行したコマンドの一覧を表示する。
`% history` *n*	過去 *n* 個分のコマンド一覧を表示する。
`% !`*n*	ヒストリ番号 *n* のコマンドを実行する。

10.2.3 ファイル名の指定 (U§6.10)

ファイル名を指定する特殊文字として、`*`, `?`, `[ ]`, `{ }` 等が使える。複数のファイルを操作対象としたり、ファイルが存在するかどうかを調べる際に便利である。

`% ls -l *.c`	名前が `.c` で終わる全てのファイルの一覧を見る。
`% rm data?`	名前が ‘`data` + 任意の 1 文字’ であるファイルを削除する。
`% cat *.[pc]`	名前の最後が ‘.p’ か ‘.c’ であるファイルを表示する。
`% ls -l {sample,letter}.txt`	ファイル ‘sample.txt’ と ‘letter.txt’ の一覧を見る。

10.2.4 名前の完成と候補一覧

tcsh では、コマンドやファイル名の最初の一部を入力するだけで、コマンドやファイル名を自動的に完成させることができる。このような機能を、コンプリーション (completion) と呼ぶ。例えば、

```
% pass
```

と入力し、 tab を入力すると、`passwd` というコマンドの名前を完成してくれる。入力した文字列によって、コマンドの名前が一意に定まらない場合は、ビープが鳴らされる。この場合、 C–d によって、その候補一覧を表示させることができる。ファイル名に関しても同様である。

10.2.5 別名機能 (U§6.7)

コマンドを別の簡単な名前で登録して、その名前で実行できる。

`% alias ll ls -l`	`ls -l` を `ll` という別名で登録する。
`% ll`	`ls -l` を実行する。
`% alias`	別名登録の一覧を表示する。
`% unalias ll`	別名 `ll` の登録を削除する。
`% alias h history 20`	`history 20` を `h` として登録する。

この機能は、オプション等をいつも指定する場合などに便利である。例えば、標準設定では、ls は、 ls -F で実行するように alias が指定されている[2]。

[2] 14章参照。

10.2.6 ディレクトリ・スタック (U§6.9)

ディレクトリ移動の履歴を記憶しておいて、それを後から利用できる。複数のディレクトリにまたがって作業をする場合に便利である。

`% pushd` *dir*　カレント・ディレクトリをスタックに積んだ後、 *dir* に移動。
`% popd`　スタックをポップして、そのディレクトリに移動。
`% dirs`　スタック内容の表示。

10.2.7 特殊文字 (U§6.12)

シェルが解釈する文字のうち、以下のような記号は特別な意味を持つので注意が必要である。

`?, *, { }, [ ], ~, <, >, <<, >>, >&, >>&, $, %, !, |, ’, ", ‘, #, \`

もし、この特別な解釈を回避 (エスケープ) して本来の文字として使用したい場合には次のようなエスケープ文字を利用する。

`\`　続く 1 文字をエスケープする。
`’`　囲んだ文字列をエスケープする。
`"`　囲んだ文字列をエスケープする。

詳細は、参考書 **(U§6.12)** を見よ。

10.3 入出力の切替えとパイプライン (U§6.4, 6.5)

コマンドの出力をファイルに書き出したり、コマンドの入力をファイルから読み込んだりすることは、シェルの機能の 1 つである、入出力の切り換え (リダイレクション) を用いると簡単にできる。

`% ls -l >` *file1*　`ls -l` の表示をファイル *file1* に出力する (`>` は出力先変更記号)。
`% date >>` *file1*　`date` の表示をファイル *file1* に追加出力する。
`% wc <` *file1*　*file1* を `wc` コマンドの入力とする。

また、パイプ (‘|’) を用いると、あるコマンドの出力をそのまま別のコマンドの入力とすることができる。パイプは何段でも可能であり、上記のリダイレクションと組合わせて使うこともできる。

`% ls -l | wc`　`ls -l` の出力を `wc` コマンドの入力とする (`|` はパイプ記号)。
`% ls -l | grep tex | wc > result`

10.4 ジョブ管理 (U§6.6)

シェルでは、複数の仕事を同時に行なうことができる。それぞれの仕事 (コマンドの実行) は、ジョブと呼ばれる。ジョブには、以下の 2 種類がある。

フォアグラウンドジョブ コマンドを起動すると、その終了 (シェルのプロンプトが出る) まで次のコマンドを起動できない。

バックグラウンドジョブ コマンドを起動すると、その終了まで待たずに別のコマンドを起動できる。

いままでの説明は、すべてコマンドをフォアグラウンドジョブとして実行する場合であった。コマンドをバックグラウンドジョブとして実行する場合は、最後に ‘&’ をつけて入力する。バックグラウンドジョブは、実行にかなり時間がかかるコマンド等の実行に便利である。

`% cp file1 file2` cp をフォアグラウンドジョブとして実行する。
`% cp file1 file2 &` cp をバックグラウンドジョブとして実行する。

バックグラウンドジョブを走らせると、`[3] 1612` のようなメッセージが出力される。ここで、`[3]` はジョブ番号を表す。ジョブが終了すると、`[3] Done` といった終了メッセージが出力される。

```
% cp a b
% cp a b &
[3] 1612
%
[3]    Done                    cp a b
%
```

ジョブ関係のコマンドを以下に示す。フォアグラウンドジョブを一時中断して、バックグラウンドジョブとして再開したり、バックグラウンドジョブや一時中断しているジョブをフォアグラウンドジョブとして再開することなどが可能である。

C–z フォアグラウンドジョブを一時中断 (停止) する。
`% %%` カレントジョブをフォアグラウンドとして再開する。
`% %`*i* ジョブ番号 *i* のジョブをフォアグラウンドとして再開する。
`% %`*string* string で始まるジョブをフォアグラウンドとして再開する。
`% bg %`*i* ジョブ番号 *i* のジョブをバックグラウンドとして再開する。
`% jobs` ジョブのリストを表示する。
`% kill %`*i* ジョブ番号 *i* のジョブを強制終了させる。

Emacs を–nw で起動した場合には、 C–z で中断してシェルに戻り、シェルで色々な仕事をした後、`%%`や`%mule` などで Emacs に戻ることができる。

10.5 演習問題

1. シェルの便利な機能を確かめよ。
2. 入出力の切替えを利用して、コマンドの出力をファイルに書き出せ。
3. JAIST には、現在何人のユーザーが登録されているかを調べよ。
   (ヒント: `ypcat`, `wc`)

第 11 章

X ウィンドウ再び

本章では、 X ウィンドウの使い方について、説明する **(U§9.2)**。

11.1 標準設定の画面構成

xinit で X ウィンドウを立ち上げた場合、図 11.1に示すような画面となる。

画面は、以下のような 6 つのウィンドウと背景 (ルート・ウィンドウと呼ぶ) から構成されている。

左上	コンソール・ウィンドウ (xterm)	
その右	アイコン・マネージャー	アイコンがここに表示される。
その右	xbiff	メールの到着を知らせる。
右上	xclock	時計
左半分	kterm	作業用
右半分	kterm	作業用

通常の作業は、大きな 2 つのウィンドウで行なう。

11.2 マウスの基本操作

マウスをマウス・パッド[1]上で移動させると、マウス・カーソルが画面を移動する。このとき マウスが ウィンドウの外 (この黒い画面背景の部分を **ルート・ウィンドウ** という。) にあるとマウス・カーソルは ‘×’ のような形状をし、ウィンドウ内に入ると ‘I’ や ‘↖’ 等の別の形に変化することに注意せよ。これらの形状は、その時点でマウスによってどんな機能が使えるかを表している。

マウス・ボタンの操作方法には以下のようなものがある。

- ボタンを 1 回押してすぐ離す (クリック)
- ボタンをすばやく 2 回押してすぐ離す (ダブルクリック)
- ボタンを押したままマウスを移動させる (ドラッグ)

[1]Sun WS のマウスは光学マウスなので、必ずマウス・パッド上で操作する。そのときパッドは横長にしておくこと。

図 11.1: X ウィンドウ画面の例

11.3 ウィンドウの構成

ウィンドウには、色々な種類があるが、ここでは、端末ウィンドウ (kterm) を例にとってウィンドウの構成について説明する。

ウィンドウは、ウィンドウ本体と上部のタイトルバー、左端のスクロールバーから構成されている。タイトルバーには、左端と右端に 2 つのボタンがついている。それぞれ、**左タイトルボタン**、**右タイトル・ボタン**と呼ぶ。これらのボタンは、ウィンドウを操作する際に使う。

マウス・カーソルがウィンドウ内部に入ると、カーソルの形状が変化するとともに、ウィンドウの枠が黒くなったり、タイトル・バー の色が濃くなったりする。そのような状態となっているウィンドウを、**選択されているウィンドウ**と呼ぶ。

11.4 ウィンドウに対する操作

ウィンドウに対する操作には以下のようなものがある。

- ウィンドウの選択
- ウィンドウの移動 (Move)
- ウィンドウの拡大、縮小 (Resize)
- ウィンドウの上下関係の変更 (Raise/Lower/Circular)

- ウィンドウのアイコン化、逆アイコン化 (Iconify, Deiconify)
- カットアンドペースト

これらの操作はマウス・ボタン、タイトルバー (タイトルボタン)、キーボード等によって行う。どの動作を、どの方法で指定するかはユーザが自由に設定することができる。ここでは、 JAIST の標準設定に基づいて説明する。

11.4.1 選択

マウスカーソルをそのウィンドウの内部に移動させる。

11.4.2 移動

マウスカーソルをウィンドウのタイトルバーの上に置き、マウス左ボタンを押しながら所定の位置に移動させ、そこでボタンを離す。

11.4.3 拡大、縮小

マウスカーソルを右タイトルボタンの上に置き、マウス左ボタンを**押しながら**拡大、縮小させたい方向にマウスを移動させる。点線のウィンドウが表示されるので、所望のサイズになった時点でボタンを離す。

11.4.4 上下関係の変更

マウスカーソルをタイトルバーの上に置き、マウス左ボタンをクリックする (マウスのボタンを一回押しすぐに離す)。

11.4.5 アイコン化

アイコン・マネージャの中のアイコン化させたいウィンドウ名の欄をマウス左ボタンでクリックする。あるいは、マウスカーソルを左タイトルボタンの上に置き、そこでマウス左ボタンをクリックする。このとき、アイコンが表示される場合とそうでない場合がある。表示されない場合にはアイコン・マネージャ上の該当欄にアイコン化の印が表示される。

11.4.6 逆アイコン化

アイコン・マネージャの中の逆アイコン化させたいウィンドウ名の欄をマウス左ボタンでクリックする。もし、対象ウィンドウのアイコンが表示されていれば、それをマウス左ボタンでクリックしてもよい。

11.4.7 カットアンドペースト

コピーしたい箇所の先頭にマウス・カーソルを移動し、マウス左ボタンを押しながらマウスを移動しコピー領域の最後でボタンを離す。このとき指定された領域の表示が白黒反転する。次にコピー先にマウス・カーソルを移動し、そこでマウス中央ボタンをクリックするとペーストが実行される。

なお、以上の操作の多くは、ルート・ウィンドウ上でマウス左ボタンを押すとウィンドウ操作用のポップアップ・メニューが表示されるので、そこでも可能である。

11.5 新しいウィンドウの作成

起動時に作成されるウィンドウでは足りない場合は、いつでも新しいウィンドウを作ることができる。新しいウィンドウを作る方法はいくつかあるが、既存の端末ウィンドウ上でそれに対応するコマンドを起動すればよい。このとき、コマンドの後に '&' 記号を付けてコマンドをバックグラウンドジョブとして起動する。例えば、漢字表示可能な端末ウィンドウ 'kterm' を起動するには、

```
% kterm &
```

とすればよい[2]。

新しいウィンドウを作成すると、画面上に新たなウィンドウの枠だけが表示される。マウスによって希望の表示位置に移動させた後、マウス左ボタンをクリックするとウィンドウが完成する。

以下にウィンドウを作成するアプリケーションの例を示す。

xclock	時計表示
xeyes	マウスカーソルを向く '目' の表示
xload	WS の負荷モニター
xterm	シェル端末
kterm	シェル端末 (漢字対応)
xbiff	メール到着表示
xdvi	TEX dvi ファイルのプレビュア
xcalc	卓上 (画面上) 計算機
mule	Emacs (Mule)

最後の mule については、新しいウィンドウを作成して Emacs を起動する方法と、シェル端末ウィンドウをそのまま利用して Emacs を起動する方法の 2 種類がある。端末ウィンドウをそのまま使うには `mule -nw` とすればよいが、この方法では Emacs の機能が制限されてしまうので、通常は新しいウィンドウとして起動すべきである。

また、ルートウィンドウでマウス右ボタンをクリックすると新しいウィンドウを開くためのメニューが表示されるので、この中から選択すると新しい kterm や mule を開くことができる。このメニューはウィンドウ・マネージャの設定ファイル (.twmrc) に記述されているので、よく使うアプリケーションを自分で設定することもできる。詳しくは、ウィンドウ・マネージャ (twm) のマニュアルを参照せよ。

11.6 トラブル時の処理

X ウィンドウが異常終了したとき (正常に起動しなかったとき) などに、キー入力が正常に行えなくなることがある。このような時には以下の手順で他の計算機から問題の計算機に リモート・ログイン[3]して、`kbd_mode -a` と入力すると回復することがある。

[2] 他のホストのウィンドウを作成するには、`% xon host1 kterm` や `% xon host2 mule` とする。このときは `&` を付けないことに注意せよ。

[3] 別の WS からネットワークを通じて対象の WS にログインすること。

```
% rlogin hostname   (問題の WS に リモートログインする
% kbd_mode -a
% logout
```

11.7 演習問題

1. X ウィンドウ を起動後、マウスを移動させ、各ウィンドウを選択するとどのような変化が見られるか確認せよ。
2. 画面左側にある大きなウィンドウを右側のウィンドウの上に移動せよ。
3. 重なったウィンドウの下側のウィンドウを表側に変更せよ。
4. ルートウィンドウ上で、マウスのボタンを押し、 3 つのボタンそれぞれにどのような機能があるか調べよ。また、 SHIFT キー、 Control キー を押した状態においても調べよ。
5. 新しい端末ウィンドウ (kterm) を作成せよ。

第 12 章

LaTeX 事始め

12.1 LaTeX とは

文書をきれいな形に整形して印刷することを行なうプログラムを文書清書プログラムと呼ぶ。いわゆるワープロ (ワードプロセッサ) は、この一種である。

ここで紹介する LaTeX は、論文等の作成で最もよく使われる文書清書プログラムの 1 つである。例えば、この講義資料は、 LaTeX を使って作っている。また、今後、 JAIST の講義で提出するレポートは、おそらく LaTeX によって清書することが求められるであろう。

LaTeX を使うには、最低限、以下のことを知っておく必要がある。

- LaTeX で文書を作るときの色々な約束事 : LaTeX を用いる場合、文書ファイルには、通常の文章内容の中に、清書に関する色々な指定 (LaTeX コマンド) を書き込むことになる。その指定方法を知っておく必要がある。
- LaTeX の文書ファイルからどうやって印刷物を作るかを知っておく必要がある。

LaTeX は、定型的な文書を作るだけならば比較的簡単である。しかし、実は、奥が深い。

12.2 LaTeX 文書ファイルの作り方

LaTeX 文書ファイルを作るには、色々な約束事がある。しかし、とりあえずは、見本ファイルを作っておいて、それを使えば、多くの場合はそれでことが足りるであろう。

以下に見本ファイルを図 12.1に示す。 この見本の 17 行目のところから、文章を書けばよい。以下に注意すべきこと等を示す。

- ファイル名は、`xxx.tex` というように、`.tex` をつける。
- 文章は、適当に改行して入力すればよい。最終的な印刷物において、どこで改行するかは、すべて LaTeX がやってくれる。
- 段落をあらためたい場合は、空行を挿入すれば良い。
- 節を作りたい場合は、`\section` コマンドを使う。[1]

[1] LaTeX のコマンドは、 \ ではじまる。但し、この \ は、 ¥と表示されることもある。

```
\documentstyle{jarticle}
\setlength{\oddsidemargin}{0mm}
\setlength{\textwidth}{160mm}
\setlength{\topmargin}{-9mm}
\setlength{\headheight}{0mm}
\setlength{\textheight}{241mm}

\title{タイトル}
\author{佐藤理史 \\
北陸先端科学技術大学院大学 情報科学研究科 \\
(sato@jaist.ac.jp)}
\date{\today}

\begin{document}
\maketitle

% (ここに本文を書く)

\end{document}
```

図 12.1: LaTeX 文書ファイルの見本

- %はコメント記号である。

図 12.2に例を示す。

12.3 dvi ファイルの作成と印刷

LaTeX 文書ファイルを印刷するには、以下のような手順を踏む必要がある (図 12.3参照)。

1. `jlatex` コマンドを用いて、`xxx.tex` から`xxx.dvi` を作る。(なお、このとき、`xxx.log`、`xxx.aux` というファイルも同時に作られる。)

```
% jlatex xxx.tex
```

2. `xdvi` コマンドを用いて、出力イメージを画面で確認する。(このことをプレビューと呼ぶ。)

```
% xdvi xxx.dvi &
```

3. `dvi2ps` コマンドと`lpr` コマンドを用いて、`xxx.dvi` をプリンターから出力する。

```
% dvi2ps xxx.dvi | lpr
```

最後のステップは、`lpr -d xxx.dvi` でも、同じことができる[2]。以下に実行例を示す。(メッセージは各種設定によって異なる。) なお、プリントアウトは、マシンのある階のプリンターから出力される。

[2]但し、実はちょっとだけ違う。

```
\documentstyle[12pt]{jarticle}
\setlength{\oddsidemargin}{0mm}
\setlength{\textwidth}{160mm}
\setlength{\topmargin}{-9mm}
\setlength{\headheight}{0mm}
\setlength{\textheight}{241mm}

\title{サンプル\LaTeX 文書}
\author{佐藤理史 \\
北陸先端科学技術大学院大学 情報科学研究科 \\
(sato@jaist.ac.jp)}
\date{\today}

\begin{document}
\maketitle

\section{はじめに}

この文書は、\LaTeX のサンプル文書です。

\subsection{ファイルのありか}

このファイルは、
\begin{quote}
\verb+/usr/local/lecture/is/i2/sample/sample2.tex+
\end{quote}
にあります。

\section{おわりに}

これでおわります。

\section*{おまけ}

このようにすると、章の番号がつきません。

\subsection*{おまけのおまけ}

これも同じです。\footnote{とっても簡単。}

\end{document}
```

図 12.2: LaTeX 文書ファイルの例

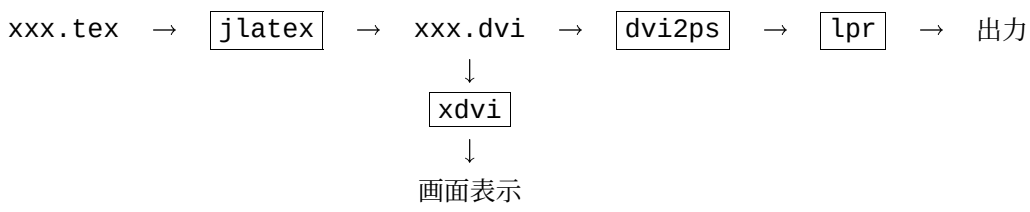


図 12.3: LaTeX 文書ファイルから出力まで

```
[is17e0s00:SS 65] % jlatex sample2
This is BigTeX, C Version 2.99 - j1.7e (no format preloaded)
(sample2.tex
LaTeX Version 2.09 <24 May 1989>
(/usr/local/lib/tex/macros/jarticle.sty
Document Style ‘jarticle’ <18 Dec 88>.
(/usr/local/lib/tex/macros/jart12.sty)) (sample2.aux) [1] (sample2.aux)
Output written on sample2.dvi (1 page, 1732 bytes).
Transcript written on sample2.log.
[is17e0s00:SS 66] % dvi2ps sample2 | lpr
 [/usr/local/lib/tex/dvi2ps/tex.ps][/usr/local/lib/tex/dvi2ps/ASCII-LW-fix.ps]
Font file for cmcsc10: magnification 1200 replaced by 1095
Font file for cmcsc10: magnification 2074 replaced by 1095

Prescanning .
Reading font info ................
[1]
[is17e0s00:SS 67] %
```

LaTeX 文書ファイルに何かエラーが含まれている場合、`jlatex` は、例えば以下のように?を表示して、入力待ちとなる。

```
[is17e0s00:SS 70] % jlatex sample2
This is BigTeX, C Version 2.99 - j1.7e (no format preloaded)
(sample2.tex
LaTeX Version 2.09 <24 May 1989>
(/usr/local/lib/tex/macros/jarticle.sty
Document Style ‘jarticle’ <18 Dec 88>.
(/usr/local/lib/tex/macros/jart12.sty)) (sample2.aux)
LaTeX error.  See LaTeX manual for explanation.
              Type  H <return>  for immediate help.
! \begin{quote} ended by \end{document}.
\@latexerr ...for immediate help.}\errmessage {#1}

\@checkend ...empa \@currenvir \else \@badend {#1}
                                                  \fi
\enddocument ->\@checkend {document}
                                       \clearpage \begingroup \if@filesw
\immed...

\end #1->\csname end#1\endcsname
                                    \@checkend {#1}\if@endpe \global \let
\@gte...
l.40 \end{document}

?
```

ここでの処置方法には、

1. **リターンのみを入力する** : とりあえず、処理を進ませる。
2. **x を入力し、リターンを入力する** : そこで処理を終了する。
3. **C-z を入力する** : ジョブを中断する。そのあと、`kill %%` とかして、そのジョブを殺す。

等がある。

12.4 論文等に必要な LaTeX コマンド (1)

以上で、とりあえず、なんとか普通の文章は出力できるようになったことだろう。ここでは、それ以外に、論文等を書くのに必要ないくつかの LaTeX コマンドについて紹介する。

12.4.1 特殊文字の出力 (RL§3.3.2)

LaTeX では、以下の 10 個の文字は特殊文字 (LaTeX のコマンドを記述するのに使われる文字) と解釈される。

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

これらの文字を出力するには、それぞれ、以下のようにすればよい。

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

以下のように出力される。

$ % & ~ _ ^ \ { }

12.4.2 数式 (RL§5)

数式は、`\( x+y=2/3 \)` のように`\( \)` ではさんで書けば、 $x+y=2/3$ のように出る。数式が短い場合、例えば、変数 1 個などの場合は、`$x$`のように`$`ではさんでもよい。また、
`\[ x \times y = \frac{2}{3} \]` のように、`\[ \]` ではさんで書けば、

$$x \times y = \frac{2}{3}$$

のように出る。とりあえず覚えておく必要があるのは、この`\( \)` と`\[ \]` である。これらではさまれた部分を数式モードと呼ぶ。

数式モードでは、色々な記号を出力することができる。例えば、上記の`\times` や`\frac` がその例である。しかし、それらをすべて覚えている人間は病的である。これらのコマンドは **(L¶44–46)** に書かれているということを覚えておき、後は適宜参照すればよい。

12.4.3 箇条書き (RL§4.2)

箇条書きは比較的よく使う。代表的なものとして、以下の 3 つがある。

itemize 単なる箇条書き

enumerate 番号付きの箇条書き

description ラベルつきの箇条書き

ちなみに、この例は、以下のように書かれている。 (description の例である。)

```
\begin{description}
\item[itemize] 単なる箇条書き
\item[enumerate] 番号付きの箇条書き
\item[description] ラベルつきの箇条書き
\end{description}
```

itemize を使うと

- 単なる箇条書き

- 番号付きの箇条書き
- ラベルつきの箇条書き

enumerate を使うと

1. 単なる箇条書き
2. 番号付きの箇条書き
3. ラベルつきの箇条書き

のようになる。それぞれ、以下のように指定すればよい。

```
\begin{itemize}
\item 単なる箇条書き
\item 番号付きの箇条書き
\item ラベルつきの箇条書き
\end{itemize}

\begin{enumerate}
\item 単なる箇条書き
\item 番号付きの箇条書き
\item ラベルつきの箇条書き
\end{enumerate}
```

12.5 演習問題

1. サンプルファイルをコピーし、 jlatex をかけて出力せよ。
2. サンプルファイルを適当に変更し、 jlatex をかけて出力せよ。

第 13 章

LaTeX の続き

今回は、論文を書くために必要な LaTeX コマンドの続きと、その他の補足的説明を行なう。

13.1 論文等に必要な LaTeX コマンド (2)

論文等を書くのに必要な LaTeX コマンドの続きを紹介する。

13.1.1 文字の種類の変更 (RL§4.7)

文字の大きさは、例えば、`{\large 少し大きく}`のように指定すると、{\large 少し大きく}なる。逆に、`{\small 少し小さく}`のように指定すると{\small 少し小さく}なる。大きさの変更には、以下のコマンドがある。

```
\tiny \scriptsize \footnotesize \small \normalsize \large \Large
\LARGE \huge \Huge
```

書体の変更も同様である。例えば、`{\bf bf ゴシック}`は、{\bf bf ゴシック}に、`{\it it}`は{\it it}となる。

```
\rm \em \bf \it など
```

文字の種類を変更する場合は、サイズ、書体の順で指定する。サイズを変更すると、書体は、roman に戻ってしまう。

13.1.2 表形式 (RL§6.3)

tabular を用いると、表形式が簡単に作れる。

担当教員	講義室	(1–2 回の講義場所)
松澤	I 講 1	4F
佐藤	I 講 2	3F
下平	I 講 3	5F

これを出力するためには、以下のように記述する。なお、このように、最初と終りをそれぞれ `\begin`、`\end` で指定するものを**環境**と呼ぶ。

```
\begin{tabular}{|l|l|c|} \hline
担当教員 & 講義室 & (1--2回の講義場所) \\ \hline
松澤 & I講1 & 4F   \\
佐藤 & I講2 & 3F \\
下平 & I講3 & 5F   \\ \hline
\end{tabular}
```

tabular 環境の引数で、1 行にいくつのカラムを作るか、カラム間に線を引くかなどを指定する。例えば、この例では、`{|l|l|c|}`となっているが、これは、左寄せ (l)、左寄せ (l)、センタリング (c) の 3 つのカラムを作り、その間と両端に線 (|) を引くことを意味している。

tabular の本体では、表の内容について記述する。各カラムは、&で区切り、各行は、\\で区切る。\\の後に、`\hline`(横線) を指定すると、横線が引かれる。

この他に、 tabular でよく使うものに、部分的に横線を引く`\cline` や、いくつかのカラムをまとめて 1 つのカラムとする`\multicolumn` などがある。以下に例を示す。

教官	部屋	
佐藤	9F	左ウイング　左
西野		左ウイング　右

```
\begin{center}
\begin{tabular}{|c|ccc|} \hline
教官 & \multicolumn{3}{c|}{部屋} \\ \hline
佐藤 & 9F & 左ウイング & 左 \\ \cline{1-1} \cline{3-4}
西野 &    & 左ウイング & 右 \\ \hline
\end{tabular}
\end{center}
```

なお、ここで、 center 環境 (`\begin{center} \end{center}`) は、センタリングの指定である。

13.1.3 計算機アウトプット (RL§4.5)

レポートなどでは、計算機の出力等をそのまま文書の一部として載せたいことがあるだろう。その時は、 verbatim 環境を用いればよい。

```
[is17e0s00:SS 77] % date
Fri Mar 19 13:00:48 JST 1993
[is17e0s00:SS 78] % time
24.720u 7.630s 1:03:41.3e3 0.8% 0+874k 191+161io 362pf+0w
[is17e0s00:SS 79] %
```

これを出力するには、以下のように記述する。

```
\begin{verbatim}
[is17e0s00:SS 77] % date
Fri Mar 19 13:00:48 JST 1993
[is17e0s00:SS 78] % time
24.720u 7.630s 1:03:41.33 0.8% 0874k 191+161io 362pf+0w+
[is17e0s00:SS 79] %
\end{verbatim}
```

このように、 verbatim 環境の中では、入力がそのままタイプライタフォントで出力される。(つまり、通常の改行等の処理が行なわれず、入力のままのイメージで出力される。)

また、`\verb` コマンドを用いると、段落の一部をタイプした通りに出力できる。例えば、

```
\verb+abcdefg%$#@&+
```

は、

```
abcdefg%$#@&
```

と出力される。

なお、この資料で計算機出力を表示するために使っているのは、非標準の特別なコマンドである[1]。

13.1.4 図表 (RL§6.4, 7.3)

図表は、本文とは別に作られ、最終的な出力では、適当な位置に挿入されるものである[2]。これらは、それぞれ、 table 環境、 figure 環境を用いて記述する。

table 環境の記述例を示そう。

```
\begin{table}
\caption{担当教官一覧}
\label{table:lecturers}
\begin{center}
\begin{tabular}{|l|l|c|} \hline
担当教員 & 講義室 & (1--2 回の講義場所) \\ \hline
松澤 & I 講 1 & 4F   \\
佐藤 & I 講 2 & 3F \\
下平 & I 講 3 & 5F   \\ \hline
\end{tabular}
\end{center}
\end{table}
```

このような記述により、表が作成され、本文の適当な位置に挿入される。

本文中で、表を参照する場合は、表`\ref{table:lecturers}` とすればよい。表 13.1のように通し番号が出力される。なお、この例の`table:lecturers` は、ラベルと呼ばれ、ユーザーが自由に設定することができる。

figure 環境についてもほとんど同様である。但し、図の場合は、キャプションは図の下につけるのが一般的である。以下に例を示す。

[1] 通常の verbatim 環境は、行間が空き過ぎるので気に入らないからである。
[2] どこに図表が挿入されるかは、あるアルゴリズムによって決定される。詳細は、 **(LC 8.1)** 参照のこと。

表 13.1: 担当教官一覧

担当教員	講義室	(1–2 回の講義場所)
松澤	I 講 1	4F
佐藤	I 講 2	3F
下平	I 講 3	5F

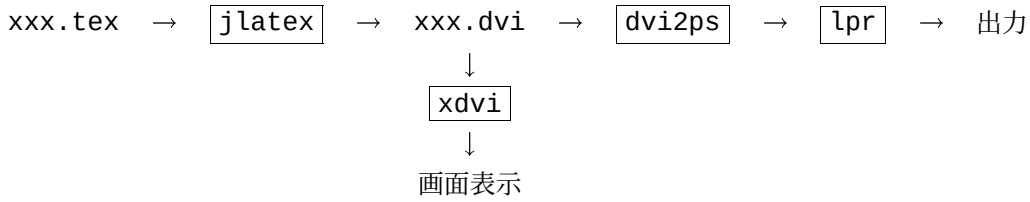


図 13.1: LaTeX 文書ファイルから出力まで

```
\begin{figure}
\begin{center}
\begin{tabular}{ccccccccccc}
\verb+xxx.tex+ & $\rightarrow$ & \fbox{\verb+jlatex+} & $\rightarrow$ &
\verb+xxx.dvi+ & $\rightarrow$ & \fbox{\verb+dvi2ps+} & $\rightarrow$ &
\fbox{\verb+lpr+} & $\rightarrow$ &
出力 \\
& & & & $\downarrow$ & & & & & & \\
& & & & \fbox{\verb+xdvi+} & & & & & & \\
& & & & $\downarrow$ \\
& & & & 画面表示 \\
\end{tabular}
\end{center}
\caption{\LaTeX 文書ファイルから出力まで}
\label{fig:tejun2}
\end{figure}
```

なお、ラベルをつけた場合、`jlatex` を 2 回かける必要がある。(`jlatex` を一回実行すると、ラベルと番号の対応関係が決定され、その情報が `.aux` ファイルに書かれる。二回目に実行した時、その情報を用いて、`\ref` で参照した場所に番号が埋められる。)

13.1.5 参考文献 (RL§9.1)

論文の最後には、参考文献を示すのが一般的である。参考文献の記述には、 thebibliography 環境を用いる。

```
\begin{thebibliography}{99}

\bibitem{LaTeX}
Leslie Lamport,
\newblock 文書処理システム\LaTeX,
\newblock Cooke, 倉沢 (監訳),
\newblock アスキー出版局,
\newblock 1990.

\end{thebibliography}
```

また、参照には、`\cite{LaTeX}`のように`\cite` コマンドを用いると、[1] のように出力される。

13.2 ドキュメントスタイル (RL§3.1.2, 8.1)

以上述べてきたことは、論文の標準的なスタイル (書式) で利用できるコマンドである。

実は、LaTeX には、論文以外に、色々な文書を作るための標準的なスタイルがあらかじめ用意されている。以下にそれらを示す。

jarticle, j-article	日本語の論文
jreport, j-report	日本語のレポート (論文よりは、すこし長いもの)
jbook, j-book	日本語の本
article	英語の論文
report	英語のレポート
book	英語の本

作ろうとしている文書において、どのスタイルを用いるかは、文書ファイルの先頭の行で宣言する。これをドキュメントスタイルと呼ぶ。

```
\documentstyle{jarticle}
```

ドキュメントスタイルには、オプションをつけることができる。例えば、

```
\documentstyle[12pt]{jarticle}
```

のように記述すると、 12pt の大きさの文字で文書が出力される。なお、指定しなかった場合は、10pt が選ばれる。

この他にも、例えば、二段組 (twocolumn) のようなオプションがある。

13.3 長い LaTeX ソースファイルの分割

比較的短いドキュメントを作成するならば、その文書を 1 つの LaTeX ファイルとして作成すればよい。しかし、例えば、本テキストのように長いドキュメントを作成する場合は、ファイルが大きくなって、扱いづらくなる。このような場合、`\input` コマンドを用いることによって、LaTeX ファイルを分割することができる。例えば、

```
\input{intro}
```

のように、記述すると、`\input{intro}`の部分を、 `intro.tex` の内容で置き換えてくれる。

13.4 環境変数 TEXINPUTS

ドキュメントスタイルでスタイルやオプションが指定された場合、 LaTeX は、環境変数 TEXINPUTS で指定されたディレクトリの下のスタイルファイル (例えば、 jarticle.sty) を探しに行く。すなわち、環境変数 TEXINPUTS は、スタイルファイル (や LaTeX 文書ファイル) のありかを指定するための変数である。 JAIST では以下のように設定することを推奨する。 (ここでは、 2 行に渡って表示されているが、実際には、 1 行である。) 標準設定を使用している場合は、この設定をする必要はない。

```
% setenv  TEXINPUTS ".:/usr/local/lib/tex/jaist.macros:/usr/local/lib/tex/
local.macros:/usr/local/lib/tex/macros"
```

注意!! TEXINPUTS には、カレントディレクトリ (.) を必ず含めること。そうしないと、カレントディレクトリの LaTeX 文書ファイルを読んでくれなくなる。

13.5 マクロ (RL§10.1)

LaTeX では、新しいコマンドや環境をユーザーが自由に作ることができる。例えば、

```
\newcommand{\caution}[1]{%
\begin{description}
\item[注意!!] #1
\end{description}}
```

のように`\caution` コマンドを定義すると、

```
\caution{この節はわからなくても心配する必要はありません。}
```

によって、

注意!! この節はわからなくても心配する必要はありません。

と出力される。

本テキストでは、いくつかのマクロを定義して使用している。どのように定義されているかを見たい人は、`/usr/local/doc/jaist/frontnet/macro.tex` を見よ。

13.6 図の挿入

LaTeX は文章や数式の清書のための強力な機能を備えているが、図を描くための機能は充分ではない。それでも図を入れたいことは多いので、以下のような方法が用いられる。

- 紙で切り貼りする

- tabular でがんばる
- picture 環境を使う
- epsf スタイルを使う

最後の方法が、最もきれいになるので、ぜひこの方法をマスターしたい。これは、他のアプリケーションで作成した図や画像を PostScript でファイルに保存しておくことで可能になる。 xfig や tgif などのアプリケーションで図を書き、それを PostScript のファイルとしてセーブしてもよいし、PostScript に変換できればほとんどの図や画像が挿入できるので、 Macintosh を利用して図を描くことも可能である。なお，作図ツールで漢字を書くときの重要な注意や、画像をイメージスキャナで取り込んで利用するための方法は付録 H.3で述べる。

以下の手順で epsf スタイルが利用できる。

1. LaTeX ファイルの先頭の`\documentstyle` の行に、`epsf` を追加する。
2. 文書ファイル中の挿入したい位置に、`\epsfile{....}`を挿入する。

   `\epsfile{....}`の `{ }` 中には、次の例のような情報をコンマで区切って列挙する。ただし、通常 `width` と `height` はどちらか一方を指定する。

	例	意味
ファイル名の指定	`file=fig1.ps`	fig1.ps を挿入する
幅の指定	`width=10cm`	幅が 10cm になるよう拡大縮小する
高さの指定	`height=3in`	高さが 3 インチになるよう拡大縮小する

以下に epsf スタイルの使用例を示す。通常はこの例のように figure 環境の中で使うことが多いであろう。

```
\documentstyle[a4j,epsf]{jarticle}
\begin{document}
図\ref{fig1}に Macintosh で描いた図を示します.
\begin{figure}
  \epsfile{file=macfig.ps,width=5in}
  \caption{Mac の図}
  \label{fig1}
\end{figure}
\end{document}
```

13.7 その他の情報

- LaTeX は、実は、 TeX と呼ばれる文書処理システムのマクロ集として実現されている。エラーメッセージが意味不明であるのは、このためである。
- 日本語化された LaTeX(jlatex) は、実は 2 つある。我々が使っているのは、アスキー版の jlatex である。この他に、 NTT 版の jlatex がある。この 2 つでは、日本語の扱いが異なる。
- `/usr/local/lib/tex/macros` には、アスキー版の標準スタイルファイルが置かれている。
- `/usr/local/lib/tex/local.macros` には、非標準のスタイルファイルが置かれている。

- `/usr/local/lib/tex/jaist.macros` には、 JAIST 標準のスタイルファイルが置かれている。
- 参考文献を作るには、先に述べた方法以外に、文献データベースと BIBTEX を使う方法がある。詳しくは、 **(RL§9.2)** を参照せよ。
- `dvi2ps` のオプションについては、 man を参照のこと。よく使うオプションとしては、
  1. `-f x`　(x ページから後ろのみを印刷)
  2. `-t x`　(x ページから前のみを印刷)

  がある。これを組み合わせることによって、指定のページのみを印刷することができる。
- `dvi2ps xxx.dvi | psnup -2 | lpr` とすれば、 A4 用紙 1 頁に、 2 頁分を縮小して印刷することができる。

13.8 演習問題

1. ⋆ 「計算機システム」または「材料機能概論 (情報処理)」の講義資料の LATEX 文書ファイルの中身を解読せよ。 (わかる部分だけでよい。)

第 14 章

快適な個人環境への長い道のり

本稿では、いままで、 JAIST の標準的な設定に沿って説明してきた。ここでは、標準的な設定を行なっているファイルについて述べ、それらを変更して、自分用の環境を作る方法 (カスタマイズの方法) について述べる。

14.1 環境設定用ファイル

環境設定用ファイルは、通常、 '.'(ドット) で始まる名前が付けられる。このことから、別名、ドットファイルと呼ばれることもある。

ドットファイルは、コマンド `ls` では表示されない。ドットファイルも表示させるためには、オプション`-A` を指定する必要がある。

```
[sato@is17e0s00] 98 % ls
[sato@is17e0s00] 99 % ls -A
.Xresources     .login          .newsrc         .xinitrc
.cshrc          .logout         .skk            .xsession
.emacs          .mnews_setup    .twmrc
[sato@is17e0s00] 100 %
```

JAIST 標準設定は、 JAIST 標準設定用ファイルによって行なわれている。これらのファイルは、

```
/usr/local/lib/user.skel/
```

にある。具体的には、以下のファイルである。

`.cshrc`	シェルに関する設定を行なう
`.login`	login 時の設定を行なう
`.logout`	logout 時の設定を行なう
`.xinitrc`	X ウィンドウの起動時の設定を行なう
`.Xresources`	X ウィンドウの各アプリケーションの設定を行なう
`.twmrc`	ウィンドウ・マネージャーの設定を行なう
`.emacs`	Emacs の設定を行なう

これらのファイルは、それぞれの起動時に自動的に読み込まれ、必要な設定を行なう。

なお、以下のファイルの表示では、 1 行に収まらないところを適宜改行した。そのためこのままではうまく動かない。最終的には、実際のファイルを参照してほしい。

14.2 .cshrc

シェルの設定を行なうファイルが、.cshrc である。このファイルには、シェルコマンドを書くことができる。ここでは、 umask の設定や、 path の設定、プロンプトの設定、別名の設定などを行なっている。

```
#
# .cshrc sample
#
```

コメントである。シェルのコメント記号は、#であり、各行において、#以降の部分がコメントとして解釈される。

```
umask 022
```

umask を設定している。この設定によって、新規に作成されるファイルは、本人以外は書き込めない設定となる。

```
limit coredumpsize 0
```

core ファイルを作らないことを指定している。 core ファイルとは、プログラムが異常終了した場合に自動的に作成されるファイルである。プログラム開発などで core を作る必要がある場合は、この行をコメントアウト (先頭に#をつける) すればよい。

```
set path=(~/bin \
        /usr/local/X11R6/bin /usr/bin/X11 /usr/local/bin \
        /usr/ucb /bin /usr/bin /usr/lang/bin)
```

path を設定している。ここで path とは、コマンドとして実行できるファイルを探しにいくディレクトリのことである。 unix では、ほとんどすべてのコマンドが、その実体である「ファイル」として存在する。例えば、`cat` というコマンドの実体は、`/bin/cat` というファイルである。各コマンドの実体を調べるためのコマンドとして、`which` や`where` がある。

```
setenv EDITOR 'mule -nw'
setenv JSERVER `hostname`
setenv TEXINPUTS ".:/usr/local/lib/tex/jaist.macros:
                  /usr/local/lib/tex/local.macros:/usr/local/lib/tex/macros"
setenv MANPATH "/usr/local/X11R6/man:/usr/man:/usr/lang/man:/usr/local/man"
```

環境変数 (**U§6.2**) を設定している。 setenv は、環境変数を設定するコマンドである。この環境変数は、色々なプログラムで参照される。例えば、環境変数 TEXINPUTS は jlatex によって参照される変数で、 jlatex が入力ファイルを探しにいくディレクトリである。

```
if ($?prompt == 0) exit
set history=100
set prompt="[$user@HOST] \! % "
set notify filec
```

プロンプト等を設定している。

```
alias l  ls -lA
```

別名の設定。ここでは、 'l' で ls を -l と -A のオプション付きで実行するように指定している。

14.3 .login

.login は、 login 時に実行されるファイルである。ほとんどがおまじないである。

```
#
# .login sample
#
```

コメントである。.login も、.cshrc と同じように、シェルコマンドで記述する。

```
stty dec new cr0
tset -I -Q
```

ターミナル属性設定などのおまじないである。

14.4 .xinitrc

.xinitrc は、 xinit によって実行されるファイルである。 X ウィンドウの起動画面に何を表示するかは、このファイルによって指定する。このファイルも、シェルコマンドで記述する。

```
#!/bin/sh
```

コメントである。がおまじないの意味をもっている。

```
xrdb $HOME/.Xresources
```

各種アプリケーションの設定ファイルである ~/.Xresources を読み込むよう指示している。

```
twm &
```

ウィンドウ・マネージャーの起動。

```
xclock &
xbiff &
kterm -name left &
kterm -name right &
```

各種ウィンドウの起動。ここで、 2 つの kterm は、画面の左側用と右側用にするため、次に述べる.Xresources ファイルに記述されたそれぞれの設定を使うよう指示している。

```
case $DISPLAY in
:0*|unix:0*)
        exec xterm -C -name console ;;
*)
        exec xterm -name login ;;
esac
```

コンソールウィンドウの起動。状況によってオプションが異なるため、おまじないが使われている

が、 2 つの xterm のうちの一方が実行される。各行の最後に ‘&’ がないことに注意せよ。このウィンドウを exit すると、ウィンドウシステムが終了する。

14.5 .Xresources

.Xresources には、 X ウィンドウの各アプリケーションの表示などに関する設定が書かれている。このファイルは X ウィンドウの起動時に読み込まれた後は参照されない。そのため、内容を変更した場合には、次のように xrdb コマンドで読み直しを指示しなければいけない。

```
% xrdb ~/.Xresources
```

標準設定ファイルの中ではいろいろな設定を行っているので、ここでは一部のみを説明する。

```
!!!!    .Xresources --- X11 Resource File
```

コメントである。このファイルでは、行頭に!があるとコメントになる。また、この他に C 言語と同様の /* */ も使用できる。

```
#define Fixed(size) -*-fixed-medium-r-normal--size-*
#define AFont(size) Fixed(size)-iso8859-1
#define RFont(size) Fixed(size)-jisx0201.1976-0
#define KFont(size) Fixed(size)-jisx0208.1983-0

#efine FontSize 16

#define AFONT AFont(FontSize)
#define RFONT RFont(FontSize)
#define KFONT KFont(FontSize)
#define FONTSET Fixed(FontSize)
```

フォントに関する定数や関数の定義を行っている。このように、 C 言語のプリプロセッサの cpp と同様の定数や関数の宣言、条件分岐などが記述できる。

```
XTerm*scrollBar:                on
KTerm*scrollBar:                on
XTerm*vt100*font:               AFONT
KTerm*vt100*fontList:           FONTSET
```

xterm と kterm のスクロールバーを有効にし、フォントも設定する。

```
console*vt100*geometry:         80x6+0+0
console*title:                  Console
console*iconName: Console
left*vt100*geometry:            80x46+0-0
left*title:                     Left
left*iconName: Left
right*vt100*geometry:           58x36-0-0
right*title:                    Right
right*iconName: Right
```

最初に起動される xterm と kterm の表示位置とタイトルの設定を行う。

```
XClock*geometry:        100x100-0+0
XBiff*geometry:         100x100-105+0
XBiff*Mailbox.update:   60
```

これらは、時計 (xclock) とメールチェッカ (xbiff) の表示位置などの設定である。

行頭の文字はアプリケーションの名前でその後ろが属性名であるが、各アプリケーションがどのような名前でどのような属性を持っているかは、それぞれのマニュアルや、 xwininfo, editres などのツールを用いて調べる。

14.6 .emacs

.emacs は、 Emacs の起動時に実行されるファイルである。このファイルは、いままで説明してきたファイルと違って、 Emacs Lisp と呼ばれるプログラミング言語によって記述される。

このファイルでは色々な設定を行なっている。その中心となるのは、 setq による設定変更用変数の書き換えである。 Emacs では、 Emacs Lisp によって書かれた便利なライブラリプログラムがたくさんある。例えば、 dired などもその一つである。

JAIST では、オリジナルのライブラリプログラムもいくつかある。例えば、 jncd は、その 1 つであり、オンラインの辞書検索システムである。また、 libserv は、図書館の図書検索システムである。これらのシステムを Emacs の中から使えるようにするため、それぞれの設定を行なっている。

このファイルについても一部のみを説明する。

```
;;;
;;; sample .emacs
;;;
```

コメントである。 Lisp のコメント記号は、‘;’ である。

```
(setq load-path (cons (expand-file-name "~/lib/emacs") load-path))
```

ライブラリプログラムを探しにいくディレクトリに、個人のディレクトリを追加している。

```
(setq-default fill-column 68)
(setq next-line-add-newlines nil)
;;(setq transient-mark-mode t)
;;(setq highlight-nonselected-windows nil)
;;(setq search-highlight t)
;;(setq query-replace-highlight t)
```

ここで、いくつかの変数の値を書き換えて、全体的な設定を変更している。 Emacs には、振る舞いを細かく制御するためのこのような変数が大量にある。

例えば、 1 行目はテキストの改行位置を 68 桁目にし、 2 行目はファイルの末尾に不用意に空行が入らなくするものである。また，コメントになっている後半の 4 行を有効にすると、 X ウィンドウ上でのリージョンや検索・置換などを、カラーを用いて見やすく表示することができる。

```
(global-set-key "\C-x\C-y" 'compile)
(global-set-key "\C-xV" 'set-variable)
```

ここでは、いくつかのキーの設定を変更している。 2 行目で C-x V に割り付けているコマンドは、Emacs を利用している途中で設定変更用の変数の値を対話的に変更するためのものである。

```
(setq wnn-host-name (or (getenv "JSERVER") (system-name)))
(setq jserver-list (list (getenv "JSERVER") (system-name)))
;;(setq enable-double-n-syntax t)
```

ここでは、 EGG に関する設定をしている。 2 行目のコメントを外して有効にすると、ローマ字入力で ‘nn’ が ‘ん’ になる。

```
(setq c-indent-level 4)
(setq c-argdecl-indent 0)
(setq c-brace-offset 0)
(setq c-label-offset -4)
(setq c-continued-statement-offset 4)
```

ここでは、 C プログラムを書く場合の清書の仕方を変更している。

```
(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

ここでは、テキストの編集では自動的に改行を調整するように設定している。

```
;;(setq mail-self-blind t)
(setq mail-yank-prefix "> ")
(setq mail-host-address "jaist.ac.jp")
;;(setq rmail-file-name "~/Mail/RMAIL")
;;(setq rmail-secondary-file-directory "~/Mail/")
;;(setq rmail-secondary-file-regexp "")
```

最初の行の先頭のコメント記号 (‘;’) を外すと、 RMAIL を用いてメールを出すときに自動的に自分に Bcc を送るようになる。次の行は、メールやニュースで他の記事の引用を行うときに挿入される文字列を指定している。

また、後半のコメントになっている 3 行は、 RMAIL のファイルを `~/Mail` の下に置く場合の設定である。

```
(setq gnus-nntp-server (or (getenv "NNTPSERVER") "jaist-news"))
(setq gnus-local-domain "jaist.ac.jp")
(setq gnus-local-organization
      "Japan Adv. Inst. of Sci. and Tech., Ishikawa, Japan")
(setq gnus-default-article-saver 'gnus-Subject-save-in-rmail)
(setq gnus-auto-select-first nil)
(setq gnus-use-generic-from t)
(setq gnus-use-generic-path t)
```

GNUS 用の設定を行なっている。

```
(autoload 'eiwa-word "jncd" "English Japanese dictionary" t)
(autoload 'eiwa-current-word "jncd" "English Japanese dictionary" t)
(autoload 'readers-word "jncd" "English Japanese dictionary" t)
(autoload 'readers-current-word "jncd" "English Japanese dictionary" t)
(autoload 'waei-word "jncd" "Japanese English dictionary" t)
(autoload 'kojien-word "jncd" "Koujien" t)
```

オンライン辞書を使えるようにしている。

14.7 カスタマイズの方法

自分用に色々な環境を設定するには、上記のファイルを適切に変更すればよい。といっても、それには、かなりの専門知識が必要になる。

JAIST では、新しいソフトウエアなどがインストールされた (使えるような状態になった) 場合、frontier.sys.announce に必要な設定が紹介される。まずは、それをそのまま実行すればよい。

その次のステップは、類推である。例えば、新しい別名を登録したい場合は、同じようなことをしているところに、同じように書けば、おそらくうまくいくであろう。それを繰り返したり、マニュアルを読んだりしていくうちに . . . 徐々に慣れてくることを期待する。

第 15 章

定型作業をやっつける — プログラム事始め

15.1 定型作業をやっつける

計算機を使って色々な作業を行なっているうちに、同じようなことを繰り返し何度も行なわなければならない場面にブチ当たる。3 回ならば別に何とも思わないかもしれない。しかし、それが 10 回、 100 回となると、やってられなくなる。そのような場合には、ちょっとしたプログラムを書いて、プログラムにやらせると楽ができる。

と言いつつも、王道は、

- プログラムをできるだけ書かないで済ませる

ことである。 unix や Emacs の機能をうまく使うと、かなりのことは、ほとんどプログラムを書かないで済ませることができる。

15.2 Emacs のキーボードマクロを使う

Emacs のキーボードマクロの機能を使うと、テキスト編集の定型作業を簡単に行なうことができる。

今、以下のようなデータファイル (教官の電子メールのアドレスと名前のデータ) があったしよう。

```
sato 佐藤
matuzawa 松澤
sim 下平
shinoda 篠田
```

これを、以下のような LaTeX の表に整形したいとしよう。

sato	佐藤
matuzawa	松澤
sim	下平
shinoda	篠田

この表を出力するためには、

```
\begin{center}
\begin{tabular}{ll}
sato & 佐藤 \\
matuzawa & 松澤 \\
sim & 下平 \\
shinoda & 篠田
\end{tabular}
\end{center}
```

のようにすればよい。確かに、 4 人分なら大したことはない。しかし、これが 40 人分だったならほとんどの人はやる気がしないだろう。

ところが、キーボードマクロを使うと、オチャノコサイサイである。 1 回分の作業を実際に行なって、あとは、それを繰り返せというだけである。上記の例では、まず、各行に注目する。各行のアドレスと名前の間に ‘&’ を書き込み、行の最後に ‘\\’ を書き込めばいいのである。行の先頭にポイント (カーソル) がある場合、これは、以下のように実現できる。

1. M–f で 1 単語分前方に移動。
2. SPC を入力。
3. ‘&’ を入力。
4. C–e で行末へ移動。
5. SPC を入力。
6. ‘\\’ を入力。
7. C–n で次の行へ移動。
8. C–a で行の先頭へ移動。

キーボードマクロは、これを行なう前に、まず、以下のコマンドでキーボードマクロの定義のスタートを宣言する。

C–x (キーボードマクロの定義をスタートする。

続いて、上記のコマンド列を実行して、 1 行分を実際に書き換える。そして、

C–x) キーボードマクロの定義を終了する。

によって、キーボードマクロを定義する。定義したキーボードマクロは、以下のコマンドで実行できる。

C–x e キーボードマクロを実行する。

さらに、繰り返し実行したい場合は、以下のようにすればよい。

C–u *i* **C–x e** キーボードマクロを *i* 回繰り返して実行する。

15.3 シェルスクリプトを使う

シェルコマンドを用いた定型作業には、シェルスクリプト (**U§8**) を使うことできる。シェルスクリプトとは、実行したいシェルコマンドを順にファイルに書いたものである。そのファイルは、あたかも 1 つのコマンドのように実行することができる。以下に簡単な例を示す。

```
% cat foo
#!/bin/sh
# sample shell script
date
who
% ls -l foo
-rw-r--r--  1 sato           41 Mar 24 13:47 foo
% chmod u+x foo
% ls -l foo
-rwxr--r--  1 sato           41 Mar 24 13:47 foo*
% foo
Wed Mar 24 13:49:46 JST 1993
sato     console Mar 24 11:53
sato     ttyp0   Mar 24 11:51   (asagi)
sato     ttyp1   Mar 24 11:53   (unix:0.0)
sato     ttyp2   Mar 24 11:53   (unix:0.0)
sato     ttyp3   Mar 24 11:53   (unix:0.0)
yonezaki ttyp5   Mar 23 20:13   (yn-quadra700.cs.)
%
```

シェルスクリプトを記述したファイルを実行するには、まず、ファイルの保護モードを実行可能に設定する必要がある。これを設定すると、そのスクリプトは、コマンドと全く同じように実行することができる。

シェルスクリプトには、単に順番にコマンドを実行するだけではなく、もっとプログラムに近い色々なことを書くことができる。

例えば、今、あるディレクトリの下に、漢字コードが SJIS のファイルが 30 個あったとしよう。そのファイルをすべて EUC に変換する問題を考えよう。それぞれのファイルに対しては、以下のように、漢字コード変換プログラムを使えば良い。

```
% ack -S -e < a.sjis > a.euc
```

これを、全部のファイルに対して実行するシェル・スクリプトは以下のようになる。

```
#!/bin/csh
# *.sjis -> *.euc
foreach file (*.sjis)
ack -S -e < $file > $file:r.euc
end
```

`foreach` コマンドによって、`*.sjis` にマッチするファイル名を変数`file` にとりだし、そのファイルに対して、漢字コード変換を実行している。`$file:r.euc` は、例えば、`a.sjis` から`a.euc` という名前を作り出すことを意味している (**U§8.2**)。

15.4 AWK, Perl を使う

Emacs のキーボードマクロや、シェルスクリプト (これは一種のプログラムであるが) で事が済まない時、ついに、重い腰を上げて、プログラムを書こうかということになる。日常的に必要なちょっとしたことならば、ほとんどの場合、プログラミング言語 Perl[2] や AWK[3] で事が済むことが多

い。これらの言語は、使用頻度が非常に高く、かつ、かなり強力である。覚えておいて損はない。

15.5 C を使う

日常的な定型作業のために、プログラミング言語 C を使うことは、(少なくとも筆者にとっては) ほとんどない。なぜならば、ほとんどの場合、もっと簡単に済ませる方法があるからである。

本当にエンドユーザーとして、単に計算機を道具として使うだけならば、C を学ぶ必要はおそらくない。しかし、もう少しアドバンストなユーザーを目指すならば[1]、C は身に付けるべきプログラミング言語の優先順位ナンバーワンである。なぜならば、ほとんどあらゆる計算機で C を使うことができるとともに、UNIX では、最終的に C がわからないと困ることが多いからである。

本屋に行くと、山のように C の参考書が置いてあるが、正統派は、カーニハン・リッチーの教科書 **(C)** を学ぶべきであると筆者は考える。この本の 1 章を読み、そこに出てくるプログラムを全部打ち込み、動かしてみることが最良の学習法である。

なお、C コンパイラとしては、`cc` 以外に、`gcc` が利用できる。後者の使用を奨める。

15.6 母国語を作る

色々なプログラミング言語を学ぶよりも、まず、一つの言語の習得に集中すべきであると筆者は考える。その言語が、あなたにとってのプログラミング言語の母国語となるように。

[1] 少なくとも、情報科学研究科の学生は、こうあって欲しい。

第 16 章

計算機の舞台裏

いままでは、主に、ユーザーの立場に立って、計算機の使い方について述べてきた。本章では、今まで述べてきた機能を実現している計算機の舞台裏について学ぶ。

16.1 どんなファイルがあるのだろう

unix のファイルシステムが 1 つの大きな木構造を形成していることはすでに 6章で述べた。普段、我々が使っているのは、自分のホームディレクトリ下の部分である。それ以外にどのようなファイルがあるのだろうか。

いま、ルートディレクトリに移動し、どのようなファイルがあるか調べてみよう。

```
% cd /
% ls
bin@          export/       lib@          mnt2/         sbin/         usr/
boot          ftp/          local/        net/          sys@          var/
dev/          home/         lost+found/   pcfs/         tmp/          vmunix*
etc/          kadb*         mnt/          pub@          tmp_mnt/
%
```

見たこともないようなファイルやディレクトリがいくつもあることがわかる。ここで、`vmunix` というファイルがあるが、これこそが、 unix の本体である。

このファイル以外のファイルの多くも、 unix になくてはならないファイルである。これらのファイルは、おおよそ、以下のディレクトリに整理されて、格納されている。

`/bin, /usr/bin`	一般コマンド
`/lib, /usr/lib`	コンパイラ、各種ライブラリ
`/etc, /usr/etc`	システム管理用コマンド
`/dev`	入出力装置 (特殊ファイル)
`/var/spool`	各種スプール
`/usr/local`	ローカルソフト関係
`/usr/man`	man ファイル
`/usr/include`	include ファイル
`/tmp_mnt`	マウントディレクトリ (automount 用)
`/home`	一般ユーザ用ディレクトリ

我々が計算機を便利に使えるのは、これらの多くのファイルのおかげなのである。

次に、 df というコマンドを実行してみよう。このコマンドは、ファイルシステムが実際にどのようなディスクで構成されているかを表示するコマンドである。

```
% df
Filesystem           kbytes    used   avail capacity  Mounted on
/dev/sd0a             15487    3833   10106    27%    /
/dev/sd0g             43247   37367    1556    96%    /usr
lss-is19:/var/spool/rwho
                     203687  178366    4953    97%    /var/spool/rwho
fs0-e4:/export/share/sunos.4.1.2
                    1276877 1108388   40801    96%    /tmp_mnt/usr/share
fs0-e4:/home/fs001  1276877  666330  482859    58%    /tmp_mnt/home/fs001
fs0-e4:/export/exec/sun4c.sunos.4.1.1/local
                    1276877 1108388   40801    96%    /tmp_mnt/usr/local
fs0-e4:/home/fs003  1276877  378592  770597    33%    /tmp_mnt/home/fs003
mex:/var/spool/mail   30991   11646   16245    42%    /tmp_mnt/var/spool/mail
%
```

このうち、上の 2 つは、ローカル・マシンのディスクであり、他の行は、他のマシンのディスク (記憶装置) を借用している。これは、 NFS (Network File System) という機能によって実現されている。 NFS は、ネットワークでつながっている他のマシンのディスクを、あたかも自分のマシンのディスクのように使うことができるようにするしくみである。

この機能を用いることによって、 JAIST のほとんどのマシンで、同じコマンドが使えることを可能にしたり、どのマシンでも自由に自分のホームディレクトリ下のファイルを変更できることを可能にしている。

16.2 舞台裏では何が行なわれているのだろう

ファイルだけが、計算機を便利に使えるようにしている主役ではない。もう一方の主役は、プロセス (**U**§**7**) と呼ばれるものである。我々ユーザーが計算機を使っている舞台裏では、色々なプロセス (プログラム) が動いており、それが、計算機を便利に使うことを可能にしているのである。

現在動いているプロセスの一覧は、以下のように見ることができる。

```
USER     PID %CPU %MEM   SZ  RSS TT STAT START TIME COMMAND
sato     232 15.4  1.6  224  476 p3 R    15:14   0:00 ps -aux
root       1  0.0  0.0   52    0 ?  IW   14:20   0:00 /sbin/init -
root       2  0.0  0.0    0    0 ?  D    14:20   0:00 pagedaemon
root      73  0.0  0.6   56  184 ?  S    14:21   0:01 in.routed
root      54  0.0  0.0   68    0 ?  IW   14:20   0:00 portmap
sato     156  0.0  0.0   40    0 co IW   14:22   0:00 xinit -ar1 500
bin       59  0.0  0.0   36    0 ?  IW   14:20   0:00 ypbind
sato     142  0.0  0.0  196    0 co IW   14:21   0:00 -tcsh (tcsh)
root      61  0.0  0.0   40    0 ?  IW   14:20   0:00 keyserv
root     109  0.0  1.4  120  412 ?  S <  14:21   0:00 /usr/etc/xntpd -c /etc/n
root      76  0.0  0.0   16    0 ?  I    14:21   0:00  (biod)
root      77  0.0  0.0   16    0 ?  I    14:21   0:00  (biod)
root      78  0.0  0.0   16    0 ?  I    14:21   0:00  (biod)
root      79  0.0  0.0   16    0 ?  I    14:21   0:00  (biod)
root      90  0.0  0.0   60    0 ?  IW   14:21   0:00 syslogd
root     100  0.0  0.0   88    0 ?  IW   14:21   0:00 /usr/lib/sendmail.bin -b
root     121  0.0  0.7  128  204 ?  S    14:21   0:07 automount -D ARCH=sun4 -
root     105  0.0  0.0   84    0 ?  IW   14:21   0:00 rpc.lockd
root     104  0.0  0.0   40    0 ?  IW   14:21   0:00 snmpd
root     106  0.0  0.0   52    0 ?  IW   14:21   0:00 rpc.statd
root     124  0.0  0.0   12    8 ?  S    14:21   0:10 update
root     134  0.0  0.0   48    0 ?  IW   14:21   0:01 inetd
root     127  0.0  0.0   56    0 ?  IW   14:21   0:00 cron
root     132  0.0  0.2   40   64 ?  S    14:21   0:00 ewhod lss-is19
root     137  0.0  0.0   52    0 ?  IW   14:21   0:00 /usr/lib/lpd
sato     157  0.0  7.0 1252 2128 co S    14:22   0:02 X :0
sato     158  0.0  0.0   64    0 co IW   14:22   0:00 /bin/csh -f /home/fs001/
sato     160  0.0  0.0  292    0 co IW   14:22   0:00 /usr/bin/X11/twm
sato     171  0.0  0.0  200    0 p1 IW   14:22   0:00 -csh (tcsh)
sato     161  0.0  3.3  192  996 co S    14:22   0:00 /usr/bin/X11/xclock -upd
sato     162  0.0  3.6  240 1084 co S    14:22   0:00 /usr/bin/X11/xload -upda
sato     163  0.0  0.3  200  104 co I    14:22   0:01 /usr/bin/X11/xbiff -geom
root     164  0.0  0.0  392    0 co IW   14:22   0:00 /usr/bin/X11/kterm -geom
root     165  0.0  0.0  380    0 co IW   14:22   0:00 /usr/bin/X11/kterm -geom
root     166  0.0  0.0  260    0 co IW   14:22   0:00 /usr/bin/X11/xterm -n Co
sato     173  0.0  0.0  208    0 p0 IW   14:22   0:00 -csh (tcsh)
sato     172  0.0  0.0  192    0 p2 IW   14:22   0:00 -csh (tcsh)
wnn      213  0.0  3.4 2084 1036 ?  I    14:30   0:05 jserver
root     195  0.0  0.1   24   28 ?  S    14:23   0:06 in.rlogind
sato     196  0.0  1.5  196  440 p3 S    14:23   0:00 -tcsh (tcsh)
sato     229  0.0  5.1 1164 1536 p3 T    15:00   0:14 nemacs -nw
root       0  0.0  0.0    0    0 ?  D    14:20   0:01 swapper
```

ここで、 USER がroot となっているプロセスが、舞台裏で動いているプロセスである。また、sato となっているのは、実際にユーザーsato が実行しているプログラム (プロセス) である。

プロセスに関するコマンドを覚えておくと、トラブルが生じた際に便利である。

`% ps`	プロセスの一覧表示 (自分のプロセスのみ)
`% ps -aux`	すべてのプロセスの表示
`% kill` *proc*	*proc* 番号のプロセスを殺す
`% kill -HUP` *proc*	*proc* 番号のプロセスに HUP 信号を送る

16.3 なぜ、我々は計算機を使えるのだろう

我々が計算機を使えるのは、その計算機にユーザーとして登録をしているからである。通常、ユーザー登録は、/etc/passwd というファイルに必要な情報記述することによって行なわれる。しかし、我々の計算機の/etc/passwd を見ても、そこには、我々の名前はない。

```
% cat /etc/passwd
root:eiYMoGWTtLOS6:0:1:Operator:/:/bin/csh
nobody:*:65534:65534::/:
daemon:*:1:1::/:
sys:*:2:2::/:/bin/csh
bin:*:3:3::/bin:
uucp:*:4:8::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/bin/csh
ingres:*:7:7::/usr/ingres:/bin/csh
audit:*:9:9::/etc/security/audit:/bin/csh
sync::1:1::/:/bin/sync
sysdiag:*:0:1:Old System
Diagnostic:/usr/diag/sysdiag:/usr/diag/sysdiag/sysdiag
sundiag:*:0:1:System
Diagnostic:/usr/diag/sundiag:/usr/diag/sundiag/sundiag
+:
%
```

JAIST では、 NIS (Network Information Service) という機能を用いて、ユーザーを管理している。NIS は、ユーザー情報、ホスト情報等の共有、一括管理を行なう仕組みであり、これを利用することによって、個々のマシンにユーザー登録しなくとも、そのマシンにログインすることができるようになる。

我々のユーザー情報は以下のようにして見ることができる。

```
% ypcat passwd | grep sato
akagi:d3AkxJ2N/rXu2:10049:1000:Masato Akagi,IS-18,1236,:/home/fs002/akagi:/bin/tcsh
msato:v9tKXFqKml0pg:10103:1000:Masayuki Sato,,,:/home/fs011/msato:/bin/tcsh
sato:EaBwmG4w43J..:10036:1000:Satoshi Sato,IS-19,1221,:/home/fs001/sato:/bin/tcsh
shimizu:70HbqVsC/U20g:10009:1003:Shimizu Masato:/home/fs2/shimizu:/bin/csh
furuta:VtNqY.dsysGag:10144:1000:Masatoshi Furuta,,,:/home/fs014/furuta:/bin/tcsh
suzuki:oRJ.G4O84NsEI:10042:1000:Masato Suzuki,IS-16,1252,:/home/fs000/suzuki:/bin/tcsh
ton:70cGVqCs/2U0g:10010:1003:Shimizu Masato:/home/fs2/ton:/bin/csh
%
```

注意!! ユーザ情報は**絶対に**学外へ持ち出してはいけない。持ち出す人に悪意がなかったとしても、転送作業中に第三者が内容を盗み取って解析し、侵入を試みることは充分考えられる。学外からの login 中に表示させるだけでも同じことが起こる。

第 17 章

ネットワーク環境

- 計算機はネットワークによって結合され、相互利用が可能である。
- JAIST の計算機ネットワークはどうなっているのか。
- JAIST 内では、どの計算機をつかっても同じユーザー環境が得られるのはなぜか。

17.1 計算機ネットワーク早わかり

計算機ネットワークとは 複数の計算機 (ホスト) を接続し、相互通信を可能にするもの。

実現手段

- 物理的な電線 — ethernet(同軸ケーブル)
- アドレス — (Mac address), IP address
- 通信規約 (プロトコル) — TCP/IP

代表的な機能

1. 遠隔端末 (リモートログイン)
2. ファイル転送
3. 遠隔ジョブ実行
4. 電子メール

17.2 ネットワーク関連コマンド (U§11)

17.2.1 IP address を知る

IP address は、/etc/hosts に書かれるか、あるいは、 NIS(後述) によって管理されている。 JAIST では後者である。

前者の場合は、

```
% grep host /etc/hosts
```

後者の場合は、

```
% ypmatch host hosts
```

で *host* の IP address を調べることができる[1]。

```
% ypmatch is17e0s00 hosts
150.65.242.3    meteor is17e0s00
%
```

17.2.2 遠隔端末 (リモートログイン)

遠隔端末とは、現在利用している端末を、ネットワークを介して他の計算機の端末として利用する方法である。“リモートログインする”とも言う。

この機能を用いると、例えば、自分の机から JAIST 内の他の計算機や、他大学の計算機にログインすることなどが可能になる。

JAIST 内で今後よく使われると思われるのは、コンソールやキーボードがハングした (まったく入力等を受け付けなくなった) 場合である。このような場合は、他の計算機からリモートログインし、おかしくなった計算機を元に戻すことが可能である (場合が多い)。

リモートログインには、次の 2 種類のコマンドがある。

1. `rlogin` *host*
2. `telnet` *host*

これらの違いは、実際に実行して調べられたい。なお、 telnet の場合は、ホスト名以外に、 IP address で対象ホストを指定することができる。

17.2.3 ファイル転送

ファイル転送とは、異なる計算機間で、ファイルをコピーすることである。

しかし、 JAIST 内では NFS(後述) のおかげで、このファイル転送を行なう必要は通常生じない。JAIST 外の計算機を利用するようになると、ファイル転送の必要が生じるだろう。

ファイル転送には、次の 2 種類のコマンドがある。

1. `rcp` *host1:file1 host2:file2*
2. `ftp` *host*

後者の ftp では、上記のコマンドで、まずリモート計算機に login し、 ftp コマンドによってファイルの転送等を行なう。詳しくは、マニュアルを見よ。

[1] ホスト名 (*host*) は、コマンド`hostname` で調べることができる。自分が日常的に使っているマシンのホスト名は覚えておくこと。

17.2.4 遠隔ジョブ

遠隔ジョブ (リモートジョブ) とは、リモート計算機上でジョブを実行することである。 JAIST 内では、通常、使うことはないであろう。

rsh コマンドによって実行できる。

`rsh` *host command*

なお、リモート計算機に対するアクセス権がなければ、遠隔ジョブは実行できない。詳しくは、**(U§11.2)** を見よ。

17.3 FRONTNET と FRONTIER

JAIST のネットワークは、 FRONTNET と名付けられている。 FRONTNET 上に構築される情報環境を FRONTIER と呼ぶ。

17.3.1 概要と基本理念

FRONTIER の概要と基本理念 (お役所作文バージョン) を以下に示す。

概要

先端的科学技術分野における第一級の教育研究成果が期待される本学においては、高度な教育研究活動を支援するための学内情報の基本的な設備として、北陸先端科学技術大学院大学情報環境システム FRONT Information EnviRonment （以下 FRONTIER と称する）の構築を行なっている。 FRONTIER では、既存の技術を用いながらも開放インタフェースを持った最高水準のシステム構成機器を用いて高度な水平・垂直分散システムを構成することにより、効率的なシステムを構成することが必要となっている。このため、北陸先端科学技術大学院大学統合情報ネットワークシステム（以下 FRONTNET と称する）を基盤としてその上に情報環境を築いていくものである。

基本理念

北陸先端科学技術大学院大学では、その建学の目的から、先端的分野において第一級の教育研究成果を上げることが期待されているが、それを支える一つの条件が先端的教育研究設備およびそれらの教育研究活動を支援する高度な情報環境である。情報環境とは、情報の生成・発生・蓄積・利用など、情報と関わる全ての局面を支援する統合的システムのことで、個々の研究者、学生、職員はこのシステムの上でその機能を利用しながらそれぞれの作業を行なうことになる。情報およびその処理技術は、全ての科学技術の基盤になっており、本学においても情報環境を先端的とすることが教育研究の効率的な推進上必要である。本学は、校舎等の施設・設備の整備にあたっては基本理念を「FRONT 計画」と称し、学術研究の進展と社会の要請等の変化に柔軟に対応できるよう、当初からインテリジェント・キャンパスを創造し、計画的に構築しようというものであるが、学内情報処理においては、これを FRONTNET を基盤としてその上に構築される FRONTIER によって実現しようとするものである。これらの要件の元に、FRONTIER では統合的な情報環境を構築することを目標とする。

17.3.2 FRONTNET の構成

FRONTNET は大きく

1. 基幹ネットワーク — 建屋間をつなぐ
2. 建屋内ネットワーク — 各フロアをつなぐ
3. フロア内ネットワーク

から成っている (図 17.1)。以下の説明および図 17.1は 1998 年度の状態であり、 1999 年からは GigabitEthernet を用いた新しいバックボーンネットワークが稼働する予定である。

基幹ネットワーク

- HIPPI バックボーン (800Mbps)

  超高速 HIPPI スイッチを用いて，建屋内 FDDI ネットワーク間をつなぐ。現在は、学内の主な通信にこちらのネットワークを用いている。

- ウルトラネット (500Mbps)

  情報科学センター (U1000) と各研究棟をつなぐ。これは、 HIPPI/FDDI のネットワークの構築前に利用されていたもので、現在は補助的な利用と障害時のバックアップに用いられている。

建屋内ネットワーク

- FDDI/FastEther ネットワーク

  超高速スイッチングルータとファイルサーバや各フロアの高速スイッチをつなぐ。

- ウルトラネット (125Mbps)

  各研究棟 (U250) とルータ (rNN) をつなぐ。

フロア内ネットワークとワークステーション

- 情報科学研究棟と知識科学研究棟では、各フロアに高速スイッチがあり、建屋内ネットワークとフロア内ネットワークをつないでいる。
- 材料科学研究棟では、各棟に高速スイッチがあり、建屋内ネットワークとフロア内ネットワークをつないでいる。
- 各フロアに ethernet(10/100Mbps)2 系統
- lss-isXX, lss2-isXX, lss3-isXX, lss4-isXX — 2 系統の ethernet の両方に接続し、各種サービスを行う。
- sXX — 個人用ワークステーション (SPARCstation)
- mXX — 個人用ワークステーション (Macintosh)

情報科学センター

- jaist1(nis) — NIS, Name Server
- mail, pop — メールサーバ
- jaist-news — ニュースサーバ
- www — WWW サーバ
- proxy — WWW 用 proxy サーバ
- fs00, ..., fs03, fs1, ... — ファイルサーバ
- cray-j90 — Cray J90 (小規模計算サーバー)
- isc-onyx — PowerONYX (小規模計算サーバー)
- cray-t3e, ncube3 — Cray T3E, nCUBE (超並列計算機)
- vxe — 富士通 VX-E (計算サーバー)
- fep — 外部からアクセスできる機械
- WIDE 東京 NOC、 WIDE 京都 NOC、 SINET 金沢 NOC 等と専用線で接続

これらの計算機は変更されることがある。最新情報や詳しい使い方などは情報科学センターのホームページ (http://www.jaist.ac.jp/iscenter/) を参照すること。

17.4 FRONTIER のユーザー環境

JAIST では、以下の NIS と NFS を用いることによって、ユーザーは、 (ほとんど) 全てのマシンにおいて、同一ユーザー環境が保たれるようにしている。

17.4.1 NIS

Network Information Service の略。ユーザー情報、ホスト情報等の共有、一括管理を行なう仕組み。これにより、個々のマシンにユーザー登録しなくとも、そのマシンにログインすることができるようになる。

17.4.2 NFS

Network File System の略。ネットワーク上の計算機のディスクをあたかも自分のマシンのディスクであるかのように扱う仕組み。これを用いて、

1. ユーザーのホームディレクトリは、ファイルサーバー (のディスク) 上に作り、そのディスクを全てのマシンで同じパス名で参照できるようにマウントすることによって、どのマシンでもホームディレクトリを読み書きできるようになる。
2. `/usr/local` などのディレクトリに、共用ディスクをマウントすることによって、これらのローカルファイルを全てのマシンで参照できるようになる。

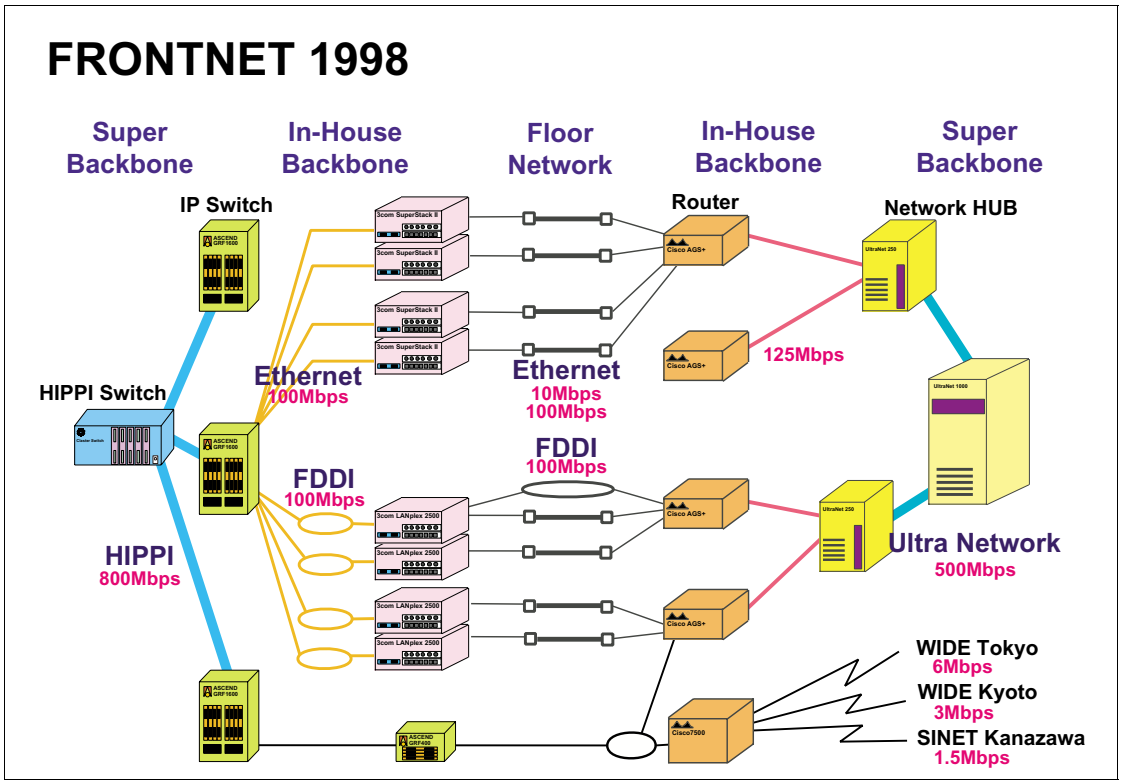


図 17.1: FRONTNET 構成図 (1998 年度)

第 18 章

ネットワークを使った情報検索

1994 年の大きな出来事の一つは、インターネット、マルチメディア、情報スーパーハイウエーといった情報関連の話題が、一般のマスコミに大きく取り上げられたことである。ここでは、その一翼を担った WWW (World-Wide Web) に対するアクセスの方法について述べる。

18.1 World-Wide Web

World-Wide Web が何であるかを説明することはそれほど単純ではない。教科書的な言い方をするならば、インターネットにおける情報サービスのひとつ、あるいは、情報検索システムということになるのだろう。ここでは、そのような定義のことは忘れ、まずは、使い方をマスターする。

18.2 Netscape の起動と終了

我々の計算機環境において、 WWW にアクセスする一つの方法は、 Netscape と呼ばれるプログラムを利用する方法である。 Netscape は、 WWW のクライアントプログラム (ユーザーズエージェント) の一つである。

Netscape は Macintosh や Windows 上でも同様のものがあるが、ここでは X ウィンドウ上の Netscape について説明する。

18.2.1 Netscape の起動

X 上で、コマンド netscape によって起動する。

```
% netscape &
```

しばらくした後、新しいウインドウが現れ、図 18.1 のような画面となる[1]。

中央の大きな表示画面には、ある指定されたドキュメント (ここでは、情報の単位のことをドキュメント、あるいは、ページと呼ぶ) が表示される。未設定の場合には Netscape のホームページが表示されるが、設定によっては JAIST Home Page が表示される。

[1] この章では、 Netscape Version 3 を用いて説明しているが、 Netscape の最新版は Version 4 になっており、表示や機能が大きく変わっている。最新版を利用する際には注意せよ。

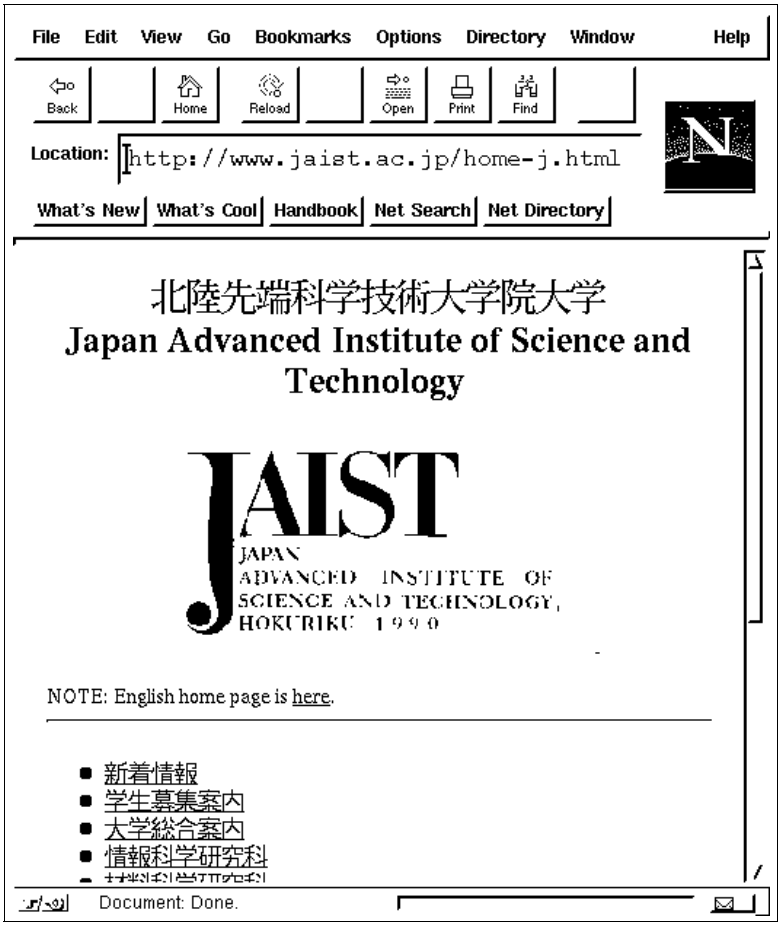


図 18.1: Netscape の起動画面

指定されたドキュメントが画面上に収まり切らない場合は、右端のスクロールバー、あるいは、下のスクロールバーによって、表示されている部分が示され、これらのバーをマウスで動かすことによって、隠れている部分を表示させることができる。なお、スクロールバーの上下、あるいは、左右にある三角印をマウスでクリックすることによっても、表示する部分を移動することができる。

各種設定は 'Options' メニューの中で行える。日本語が正しく表示されない場合には、 'General Preferences' の Fonts を設定して、 'Language Encoding' を 'Japanese (Auto-Detect)' にするとよい。また、設定変更後は 'Save Options' を行う。

18.2.2 Netscape の終了

左上の 'File' をクリックするとプルダウンメニューが現われるので、そのメニューから一番下の Exit を選ぶと Netscape が終了する。

18.3 ハイパーテキスト

Netscape の中央部に表示されるドキュメントのところどころに、下線が表示されているのに気が付くだろう。その下線は、その部分に関連したページが存在することを示しており、その部分をクリックすることによって、そのページに移動することができる。

概念的に見るならば、テキスト (ドキュメント) は、一次元の文字の並びとみなすことができる。ここで、「一次元」という言葉から明らかなように、テキストには、はじまりから終わりへの明確な順序が存在する。

先に述べた、「関連するページ」は、この一次元的順序を逸脱するもので、テキストに複数の読み方（読む順序）を与えるものである。すなわち、読者は、そのテキストを、一次元的な順序で読むことができるだけでなく、テキストに埋め込まれた関連ページへのポインタ (これをリンクと呼ぶ) を自由にたどりながら、自分の読みたい順序でテキストを読むことができる。このようなテキストのことをハイパーテキストと呼ぶ。 WWW は、このハイパーテキストを基本とする情報検索システムである。

18.4 URL

概念的に、ハイパーテキストは、テキストと、その中に埋め込まれた他のテキストへのリンクから構成される。これを実現するために、あるテキスト (ドキュメント) を一意に指し示すことができる名前付けシステムが必要になる。

WWW において、この名前付けシステムに相当するものが、 URL (Uniform Resource Locator) と呼ばれるものである。例えば、 Netscape を起動した際に表示されるドキュメントは、

```
http://www.jaist.ac.jp/index-jp.html
```

という名前が付けられている。この URL は、一般に、

method`://`*hostname*`/`*path*

という形式で表わされる[2]。

[2]ここで、 *method* は、通信プロトコルを指定するもので、通常は、 http (HyperText Transfer Protocol) が用いられるが、他のプロトコル（例えば、 ftp）なども可能である。また、 *hostname* は、ホストマシンの名前である。

WWW によってサービスされる情報は、巨大なハイパーテキストであり、どこからでも好きなところ（ドキュメント）から、その巨大なハイパーテキストを読むことができる。所望のドキュメントを Netscape で表示させるためには、 File メニューの中の、 'Open Location' を選択し、 URL を入力すればよい。

WWW によってサービスされる情報が、巨大な 1 つのハイパーテキストだからと言って、すべてのドキュメントが同じ待ち時間でアクセスできるわけではない。あるドキュメントは、 JAIST 内にあり、また、別のドキュメントは、アメリカにあったりする。見かけ上、これらのドキュメントは同じように扱われるが、そのドキュメントを表示するためには、そのドキュメントを存在する場所から表示するマシンまでコピーしてくる必要があり、その転送時間には、大きなばらつきがある。そのことは、きちんと頭の中に入れておく必要がある。

なお、あるドキュメントを表示させようとしたが、なかなか応答が返ってこない場合は、 'Stop' ボタンをクリックすることによって、そのドキュメントに対するアクセスを中止することができる。

18.5 テキスト以外のメディアの表示と外部プログラム

WWW がこれ程「はやった」原因の一つは、 WWW は、テキストだけでなく、画像や音声といったメディアを扱うことができるからであろう。

テキスト以外のメディアと取り扱い方は、それぞれのクライアントプログラムによって異なっている。 Netscape では、テキスト以外に、例えば GIF 形式や JPEG 形式で格納された画像データを画面に表示することができる。

それ以外の画像データ、音声データ、動画データなどは、それらの表示・再生用の外部プログラムが自動的に呼ばれ、それらのプログラムによって表示・再生されることになる。また，標準では対応していないデータ形式でも，対応する外部プログラムをユーザが明示的に設定することが可能である.

18.6 ネットサーフィン

これで、 Netscape の最低限の使い方はおしまいである。あとは、実際に Netscape を使って WWW にアクセスし、それを通して、色々な機能の使い方をマスターされたい。とりあえず、 JAIST Home Page からたどれるページを探訪しながら、操作に慣れ、次に、各種サイトにアクセスしてみるのがよいだろう。

表 18.1に、よく使ういくつかの機能をあげておく。この中の印刷機能は、 6.5節で説明した lpr コマンドを用いるため、印刷する内容の大きさに注意しなければならない。画像が含まれている場合には予想以上に大きいことがあるので、そのような場合には一旦ファイルに保存して、端末ウィンドウから lpr -s コマンドを実行し、印刷終了後にそのファイルを削除する。詳しくは 6.5.2を参照せよ。

この章の参考文献

1. 高田敏宏. インターネット上の情報の集め方. bit, Vol. 27, No. 2, pp4-12, 1995.

表 18.1: Netscape の便利な機能

名称	操作	機能内容
Back	Back ボタン	一つ前に見たドキュメントに戻る
Forward	Forward ボタン	Back の逆
Bookmark	Bookmarks メニュー	見たドキュメントの場所をメモしておく機能
Document Source	View メニュー	表示しているドキュメントのソース (HTML 形式) を表示する
Save As	File メニュー	表示しているドキュメントをローカルマシンにセーブする。
Print	Print ボタン	ドキュメントの印刷

第 19 章

ネットワークを使った情報発信 — html 入門 —

前章では、 WWW にアクセスする方法について学んだ。 WWW で公開されている情報は、インターネットに参加している誰かが作ったものであり、我々も、同じように情報提供者として参加することができる。ここでは、その方法について述べる。

19.1 HTML 事初め

ここでは、まず、自分のホームページを作成することを通して、 WWW を通して情報発信する方法について説明する。

19.1.1 ホームページの作成

とりあえず、以下の手順で、自分用のホームページを作成してみよう。

1. 自分のホームディレクトリに、`public_html` というサブディレクトリを作成し、他人から読めるようにする。(`chmod ugo+rx public_html`)
2. ホームページの見本`/usr/local/lib/user.skel/sample.html`を`public_html` の下に、`index-j.html` という名前でコピーする。
3. `index-j.html` の内容を適当に変更する。

以上である。

JAIST の設定では、各自のホームディレリ下の`public_html` というサブディレクトリ下のファイルは、 URL では、`http://www.jaist.ac.jp/~user/`というパス名が与えられる。例えば、

```
~sato/public_html/home-j.html
```

は、

```
http://www.jaist.ac.jp/~sato/home-j.html
```

という URL で参照できる。

19.2 HTML の道具立て

ここでは、 WWW のドキュメントを記述する記法である HTML(HyperText Markup Language) について簡単に説明する。 HTML は、 SGML(Standard Generalized Markup Language) に従っており、テキストの構造化に関する情報を、タグ (エレメント) という形でテキストに埋め込む。

19.2.1 ヘッダ部とボディ部

HTML ドキュメントは、ヘッダ部とボディ部からなる。すなわち、以下のような形式となる。

```
<HTML>
<HEAD>
ここがヘッダ部
</HEAD>
<BODY>
ここがボディ部
</BODY>
</HTML>
```

この例よりわかるように、 HTML のタグは、<*Tag*>、あるいは、</*Tag*>と書かれる。多くのタグは、はじめと終わりのペアで用いられるが、いくつかのタグは、単独で用いられる。

ヘッダ部

ヘッダ部は、 HTML ブラウザ (例えば、 Netscape) に対する指示を指定する部分である。記述されるものは、タイトル (<TITLE> ... </TITLE>) などである。

ボディ部

ボディ部は、その HTML ドキュメントの本体である。ボディ部でよく使われるタグを表 19.1に示す。

19.2.2 アンカー

他のドキュメントの参照、あるいは、同一ドキュメントの参照、および、そのための名前付けは、全てアンカーと呼ばれるタグによって行う。

- 他のドキュメントの参照。
  <A HREF="*URL*">...</A>
- 同一ドキュメント内の違う部分の参照。
  <A HREF="*identifier*">...</A>
- ドキュメント内の名前付け。
  <A NAME="*identifier*">...</A>

アンカーの記述で用いられる URL の書き方には、いくつかの (省略) 記法がある。主なものを以下に示す。

表 19.1: ボディ部でよく使われるタグ

<H1>...</H1>	大きい見出し。 LaTeX の section に相当。
<H2>...</H2>	次に大きい見出し。 H6 まである。
<HR>	横線。
<P>	段落の終わり。
 	強制改行
<PRE>...</PRE>	あらかじめフォーマットされたテキスト。改行、空白等をそのまま表示する。 LaTeX の verbatim 環境に相当する。
<UL>...</UL>	番号なしリスト。 LaTeX の itemize 環境に相当する。
<OL>...</OL>	番号付きリスト。 LaTeX の enumerate 環境に相当する。
<LI>	リストの要素。 LaTeX の\item に相当する。
<DL>...</DL>	見出し付きリスト。 LaTeX の description 環境に相当する。見出しは、<DT>の後ろに記述し、その見出しに対する説明は、<DD>の後ろに記述する。
<ADDRESS>...</ADDRESS>	アドレス (著者情報) の表示。

1. 相対パス記法　そのドキュメントからの相対パスで指定する。
   `<A HREF="home.html">`

2. 絶対パス記法　そのサイトの絶対パスで指定する。
   `<A HREF="/user/home.hmtl">`

3. ドキュメント内の場所指定付き記法　あるドキュメントのある部分を指定する。
   `<A HREF="/user/home.hmtl#place">`

19.2.3 画像の埋め込み

HTML では、<IMG SRC="*filename*" ALT="*Comment*"> のような形式で、テキストに画像を埋め込むことができる。ここで、 SRC は、画像ファイルを示すものであり、 ALT は、その画像をクライアントプログラムが表示できない場合に表示するテキストを指定する。先に述べたように、Netscape では、 GIF および JPEG 形式の画像ファイルがインラインで表示できる。

なお、画像ファイルは、一般に大きなサイズとなりやすいので、画像のサイズを小さくして、適切なファイルサイズとすべきである。

19.3 その他の情報

WWW で公開されている情報の HTML ソースファイルは、 Netscape の View メニューの ‘Document Source’ によって見ることができる。これによって、どのようなソースを書けば、どのような表示ができるかを簡単に学ぶことができるであろう。

19.4 情報発信のルール

WWW で発信する情報は不特定多数の人が見るため、作成するときにはモラル等に十分注意する必要がある。

自分の作成したページの内容に責任を持ち、他人を中傷したり、**プライバシーや人権**を侵害することのないように気を付ける。また、引用を行う場合には著作権に気を付ける必要がある。特に、画像や音楽は**著作権**や**肖像権**などを侵害する可能性が高いので充分注意すること。

プライバシーの問題などを解決する方法の一つに**アクセス制御**がある。 WWW では通常、作成したページは世界中から誰でも参照可能であるが、アクセス制御を行うと、特定のページだけを特定のユーザに限り参照を許可することができる。

例えば、 JAIST 内でのお知らせや、学外から参照されるとセキュリティ上の問題が生じる情報は多い。そのような場合には、学内の機械のみから参照可能なように設定する。

アクセス権の設定方法は入門の域を越えるためここでは詳しく述べないが、入門を卒業したらぜひ修得して欲しい機能である。

この章の参考文献

1. 高田敏宏. インターネット上の情報の作り方. bit, Vol. 27, No. 3, pp4-10, 1995.

2. 吉村伸. NCSA httpd の設定と HTML 入門. UNIX Magazine, Vol. 9, No. 10, pp52–60, 1994.

3. 吉村伸. HTML 入門 (2). UNIX Magazine, Vol. 9, No. 11, pp43–50, 1994.

付録 A

UNIX の便利なコマンド

A.1 シェル操作環境

echo	引数のエコー	`echo $path`
printenv	環境変数の一覧表示	
setenv	環境変数の設定	
cd	カレント・ディレクトリの変更	
pwd	カレント・ディレクトリ名の表示	
basename	パス名からディレクトリ名の除去	`basename $home`
dirname	パス名からディレクトリ名の抽出	
line	1 行の読み取り	
expr	引数の式としての評価	`set a = ‘expr $a + 3‘`
test	条件評価コマンド	
sleep	実行の一時中断	

A.2 ファイル操作・管理

cat	ファイルの連結と操作	`cat` *file...*
head	ファイル先頭部の表示	`head [`*-num*`]` *file...*
tail	ファイル後尾の表示	`tail [`*-num*`]` *file...*
more	ファイル内容の画面表示	
less	ファイル内容の画面表示	
pr	ファイルの (書式付き) 出力	`pr -w60 prog1.c`
od	ファイル内容のダンプ	`od -x file1`
lpr	プリンタへの出力依頼 (*1)	
cp	ファイルのコピー	
mv	ファイルの移動	`mv` *file1 file2*
rm	ファイルの削除	
ln	リンク	
mkdir	ディレクトリの作成	

rmdir	ディレクトリの削除	
ls	ファイル一覧の表示	
file	ファイル・タイプの表示	
cmp	二つのファイルの比較	
diff	二つのファイルの差分比較	
awk	パタン操作および処理言語	
grep	ファイルのパタン検索	`grep 'the' *`
sort	ファイル内容のソート	`sort` *file*
sed	ストリーム・エディタ	
tr	文字の変換	
wc	行数、語数、文字数のカウント	
chmod	ファイル・モードの変更	
dd	ファイル変換とコピー	
find	ファイルの探索	
compress	ファイルの圧縮	
uncompress	圧縮ファイルの復元	
zcat	圧縮ファイルの表示	
split	ファイルの分割	
tar	テープ・ファイル・アーカイバ	

A.3 プロラミング環境

ed	テキスト (行) エディタ
ex	テキスト・エディタ
emacs	画面エディタ
vi	画面エディタ
cc, gcc	C コンパイラ
dbx	ソース・レベル・デバッガ
f77	Fortran コンパイラ
make	プログラムの自動保守、更新、再生
sccs	ソース・コード・バージョン管理

A.4 ユーザ・システム状況

finger	ユーザ情報の表示
ps	プロセス状態の報告
df	ディスク空き容量の報告
du	ディスク使用状況の報告
w	ログインユーザ情報の報告
who	ログインユーザの表示

lpq	プリンタジョブ列の表示

A.5 その他の基本コマンド

banner	ポスタ (花文字) の表示	
cal	カレンダの表示	
date	日付の表示	
dc	卓上計算機	
stty	端末のオプション設定	
tty	端末名の表示	
time	コマンド実行時間の表示	
ftp	会話型リモートファイル転送	
hostname	ホスト名の表示	
rcp	リモート・ファイル・コピー	
rsh	リモート・シェル	
write	他ユーザへのメッセージの書き込み	
talk	他ユーザとの端末上での会話	
man	オンライン・マニュアル	
whatis	コマンドの簡単な説明	`whatis man`
whereis	指定コマンドのパス名の表示	

A.6 その他のアプリケーション

xfig	作図、 PostScript で出力すると LaTeX 文書等に挿入できる
tgif	作図、 PostScript で出力すると LaTeX 文書等に挿入できる
xv	画像ファイルの表示、フォーマット変換
acroread	PDF ファイルの表示

(*1) lpr は現在、ポストスクリプト・ファイル か dvi ファイルしか出力できない。

付録 B

日本語関連

B.1 漢字コード

UNIX では、以下のような 3 種類の漢字コードが使用される。ディスプレイやエディタの漢字設定において、正しいコードが設定されていないと、正しく漢字が表示されない。

名称	説明
JIS	JIS 規格コード。メールやニュースで用いられる。
EUC	拡張 UNIX コード。新しい JIS 規格に適合している。
SJIS	Shift-JIS コード。パソコンで主に使用される。

どの漢字コードを使用するかは計算機およびその上のコマンドによって得手、不得手があるので一概には決められないが、最近の UNIX 計算機では EUC 用のコマンドが多い。

ただし、メールやニュースにおいては JIS コードを使用する取り決めになっている。

どの漢字コードを使用するかは、 Emacs のコマンドや起動ファイル `~/.emacs` 等で変更することができる。

B.2 漢字コード変換

B.2.1 UNIX でコマンドを用いる場合

漢字コード変換プログラムとして、 ack や nkf がある。

- `ack -j <` *file1* `>` *file2*
  file1 を JIS に変換して *file2* に書き出す。

- `ack -e -z <` *file1* `>` *file2*
  file1 を EUC に変換して *file2* に書き出す。入力に半角カタカナがあれば全角カタカナに変換する。

- `ack -E -s <` *file1* `>` *file2*
  file1 を SJIS に変換して *file2* に書き出す。入力ファイルが EUC であることを明示的に指定する。

B.2.2 Mule を用いる場合

Mule はいろいろな漢字コードを扱えるだけでなく、 UNIX, DOS, Macintosh の改行コードも扱える。 Mule では、これらのコードに “coding system” という独自の名前を付けており、これを指定してファイルを読み書きすることで、漢字コード変換と改行コード変換が一度にできる。これを行うには、通常の C–x C–f, C–x C–w の代わりに C–u C–x C–f, C–u C–x C–w というコマンドを使う。

また、変換という考え方をしなくても、例えば DOS のフロッピーを dosmount して (E.3 を参照)、その上のファイルを直接編集することももちろん可能である。

コード変換によく使われる coding system の名前は次のようになる。

*euc-japan*unix	UNIX 上の EUC コード
*junet*unix	UNIX 上の JIS コード
*sjis*dos	DOS のコード
*sjis*mac	Macintosh のコード
noconv	無変換 (バイナリファイルが編集可能)

B.2.3 Macintosh を用いる場合

JCONV-DD を用いると漢字コードと改行コードをある程度自動判断して変換してくれる。

付録 C

C プログラミングに関するいくつかのこと

- どうしたら、プログラミング言語を簡単にマスターできるか。
- どうしたら、良いプログラムが書けるようになるか。

C.1 プログラミング言語早わかり

1. ほとんどのプログラミング言語には共通な要素 (概念) がある。
2. あるプログラミング言語をマスターするということは、それらの要素をその言語ではどういうように書き表すかということを知ることである。

その要素とは、...

C.1.1 プログラム単位

1. ある (1 つの) プログラムは、通常、いくつかの部分 (プログラム単位) から構成される。
2. 典型的なプログラム単位は、関数 (function) の定義や手続き (procedure) の定義である。
3. (この他に、大域変数の定義などがある。)
4. つまり、 1 つのプログラムは、いくつかの関数や手続きから成る。
5. ある決められた方法で、関数や手続きを定義する。

C では

1. C には、関数しかない。
2. C プログラムは、 1 つの main 関数といくつかの関数から成る。
3. 関数は以下のように定義する。

type function-name (*parameter-list*)
{
body
}

例

```
int main()
{
    char name[32];
    int  r[3];
    float f;

    while (scanf("%s %d %d %d", name, &r[0], &r[1], &r[2]) != EOF) {
        f = final(r);
        printf("%s %c (%3.1f) %d %d %d\n",
                name, grade(f), f, r[0], r[1], r[2]);
   }

}

float final(int r[])
{
    return((r[0]+r[1]+2.0*r[2])/4.0);
}
```

C.1.2 文

1. 関数や手続きの本体 (body) は、複数の文からなる。
2. 1 つの文とは、 1 つの実行単位 (多くの場合、最小単位) である。
3. 文には、宣言文と実行文がある。
4. 文の書き方には約束がある。
5. 複数の文をまとめてひとつの文相当にする方法がある。

C では

1. ほとんどの文は、; で終る。
2. 多くの文は、式の後ろに; をつけたものである。

   expression ;

3. ここで、式とは、代入や関数呼び出し、宣言などである。
4. {, } で囲うことによって、複数の文を一つの文相当にすることができる。

例

```
float f;
f = final(r);
printf("%s %c (%3.1f) %d %d %d\n",
        name, grade(f), f, r[0], r[1], r[2]);
{ i = 1 ; j = 2 ; }
```

C.1.3 データ型と変数

1. プログラムが扱うデータには、いくつかの種類がある。これをデータ型 (type) と呼ぶ。
2. 典型的なデータ型には、整数 (integer)、浮動小数 (float)、文字 (character) などがある。
3. プログラミング言語は、変数を持つ。変数は、データを格納する箱である。
4. 多くのプログラミング言語では、変数を使う場合、どのような変数を使うか先に宣言する。
5. 変数には、型付き変数 (例えば、整数を入れる変数というように、入れるデータの型を限定するもの) と型なし変数がある。

C では

1. データ型として、整数 (integer)、浮動小数 (float)、文字 (character) などがある。
2. (この他に、それぞれのデータ型へのポインタがある。)
3. 変数は、型付変数であり、宣言する必要がある。
4. 以下のように変数を宣言する。

   type variable-name ;

例

```
int i;
char x;
float f, g;
```

C.1.4 制御構造

1. プログラムは、連接、反復、分岐の 3 つの構造から成る。
2. それぞれを記述する方法がある。

C では

1. 連接 — 文を順に並べて書く

   sentence1
   sentence2
   sentence3

2. 反復 — while, do–while, for

   `while` (*condition*) *sentence*
   `do` *sentence* `while` (*condition*)
   `for` (*expression1* ; *expression2* ; *expression3*) *sentence*

3. 選択 — if-then, if-then-else, (switch)

   `if` (*condition*) *sentence*
   `if` (*condition*) *sentence* `else` *sentence*

例

```
get_output_id(char value[])
{
    int i;

    for (i = 0 ; i < OSIZE ; i++)
         if (!strcmp(ODB[i],value)) return(i);

    ODB[OSIZE] = copy(value);
    return(OSIZE++);
}
```

C.1.5 配列

1. 同じ型の変数を他数作る方法として、配列がある。
2. 配列とは、配列名と添字 (番号) によってアクセスできる変数である。配列の要素 (配列名と添字によってアクセスできる 1 つの変数) は、通常の変数となんら変わるとことがない。
3. 1 次元配列以外に、多次元配列が用意されている場合もある。
4. 配列は使う前に宣言する必要がある。
5. 多くのプログラミング言語では、コンパイル時に配列の大きさが決まっていなければならないという制約がある。

C では

1. 配列は以下のようにして宣言する。

   array-name`[`*constant-expression*`] ;`

2. 多次元配列の場合は、以下のようにして宣言する。

   array-name[*constant-expression1*][*constant-expression2*] ;

例

```
char name[32];
int  r[3];
int  a[30][40];
```

C.1.6 構造体

1. いくつかのデータをひとまとまりのデータとして扱う方法に構造体がある。

2. 構造体は、いくつかのメンバから構成される。1 つのメンバが 1 つのデータに相当する。

3. 構造体を用いることによって、新しいデータ型を作ることができる。つまり、構造体の定義とは、新しいデータ型の定義である。

4. この定義により、新しいデータ型は、既存のデータ型とまったく同様に用いることができるようになる。

5. ある決められた方法で、構造体のメンバにアクセスすることができる。

6. 旧式の言語には、構造体を持たないものがある。また、新式の言語では、オブジェクトという概念に格上げされているものもある。

C では

1. 構造体は以下のようにして宣言する。ここで、本体とは、いくつかの宣言文である。

   ```
   struct structure-name {
   body
   };
   ```

2. 構造体のメンバには、以下の記法でアクセスできる。

   structure-name.*member*

例

```
struct point {
    int x;
    int y;
};

struct point makepoint(int x, int y)
{
    struct point temp;

    temp.x = x;
    temp.y = y;
    return temp;
};
```

C.1.7 入出力

1. プログラミング言語は、データの入出力の方法を持つ。

2. データの入出力には、バイト単位の入出力とフォーマット入出力が通常用意される。

3. データの入出力には、キーボード・ディスプレイに対する入出力の他に、ファイルに対する入出力が通常用意される。

C では

1. C 自体には、入出力がない。その代わりに、標準ライブラリ stdio が用意されている。

2. バイト単位の入出力には、 getc, fgetc, putc, fputc、フォーマット入出力には、 printf, fprintf, scanf, fscanf などがある。

3. プログラム内では、ファイル (とデバイス) は、ストリームとして扱われる。

4. 標準入力 (キーボード入力) ストリーム stdin、標準出力 (ディスプレイ出力) ストリーム stdout が用意されている。

5. (この他に、標準エラー出力 stderr が用意されている。)

6. fopen によって、ファイルをオープンし、ストリームを作る。

7. fclose によってストリーム (ファイル) をクローズする。

8. 入出力先は、ストリームを指定することによって指定する。

例

```
main()
{
    int c, nl;

    nl = 0;
    while ((c = getc()) != EOF)
        if (c == '\n') ++nl;
    printf("%d\n", nl);
}
```

```
void read_bgh_index()
{
    FILE *fp;

    fp = fopen("bgh_index", "r") ;

    sort_files = 0;
    while ( fscanf(fp, "%s", sort_file[sort_files])  != EOF &&
            fscanf(fp, "%s", index_word[sort_files]) != EOF )
    sort_files ++ ;
    fclose(fp) ;
}
```

C.2 良いプログラム・悪いプログラム

C.2.1 良いプログラムとは

- わかりやすい (読みやすい)
- シンプルでコンパクト (無駄がない)
- バグがない (少ない)
- 効率がよい

C.2.2 プログラム書法

- Kernighan と Plauger の名著、日本語訳は共立出版 [4]
- 77 個の規則集
- 少し古い (Fortran IV, PL/I) が、おそらく、最良の参考書

C.3 まとめ

- プログラミングは、誰も教えてくれない。
- しかし、「正しいプログラミング道」はある。
- 師匠について、修行する必要がある。

- プログラムに対する批評的な目を養おう。
- 動くからといって、満足してはいけない。
- 自分のスタイルを確立しよう。
- 美しいプログラムは芸術である。

C.4 演習問題

1. いままでに自分の書いたプログラムを検討し、改善すべきところを改善せよ。

付録 D

プログラム開発環境

D.1 プログラムの分割

大きなプログラムを一つのファイルで作成していると、いろいろな問題が生じてくる。そこで、プログラムをいくつかのファイルに分けて作成する方法がしばしば用いられる[1]。分割の利点を挙げると以下のようである。

- プログラムの理解、編集が容易になる。
- プログラムの生産性が向上する。
  - コンパイル時間の短縮
  - デバッグが容易
  - プログラムの拡張が容易
  - 多人数でプログラムを分担作成するとき、作業が容易
  - プログラム・モジュールを繰返し再利用可能

D.1.1 分割方針

- 汎用性のある関数や手続きは分割する。良く利用するものは、まとめて保存しておく (ライブラリ化する) と良い。
- 分割後、共通に使用される変数 (広域変数)、型 (structure の型等)、マクロ (#で始まる行) は、別の **ヘッダファイル** で定義し、個々のファイルから参照するようにする[2]。ヘッダファイルの拡張子には通常 ".h" が用いられる。
- 適度な分割を心がける。細かするぎる分割はかえって弊害をもたらす。

具体例として、バブルソート・プログラムを利用する。ここでは、プログラムを以下のような 4 つのファイルに分割した[3]

[1] プログラミング言語によっては、標準 Pascal のように分割化ができないものもある。
[2] C では、`#include` 文を使用する。
[3] この分割例は、あまり良い例ではない。

bsort.h	ヘッダファイル
bsort.c	バブルソート関数 (b_sort(), compare(), swap())
bsort-main.c	main プログラム (main())
mylib.c	変換関数ライブラリ (get_num())

以下に、各ファイルの内容を示す。

bsort.h

```
#define RECORD_WIDTH 112
#define MAX_RECORD  112

int cmp;        /* 比較回数 */
int mov;        /* 移動回数 */

/* 1 行のデータを格納する構造 */
struct record {
    int num;                    /* 学籍番号 */
    char data[RECORD_WIDTH];    /* 行データ */
};
```

bsort-main.c

```
#include <stdio.h>
#include "bsort.h"

main()
{
    struct record e1[MAX_RECORD+1];     /* 全データ */
    int recnum;                         /* レコード数 */
    int i;

    /* 標準入力からのデータの読み込み */
    i = 0;
    while (fgets(e1[i].data, RECORD_WIDTH, stdin) != NULL) {
        if (i >= MAX_RECORD) {
            fprintf(stderr,"Error: Too many records.\n");
            exit(1);
        }
        e1[i].num = get_num(e1[i].data);
        i++;
     }
    recnum = i;

    /* ソートの実行 */
    cmp = mov = 0;
    b_sort(e1, recnum);

    /* 結果の出力 */
    for (i = 0 ; i < recnum ; i++)
         printf("%s", e1[i].data);
    printf("Compare Num. is %d\n", cmp);
    printf("Moving Num. is %d\n", mov);

}
```

bsort.c

```
#include "bsort.h"

/* バブルソート */
b_sort(struct record e1[], int recnum)
{
    int i, j;

    for (i = 1 ; i < recnum ; i++)
         for (j = recnum-1 ; j >= i ; j--)
              if (compare(e1[j-1].num, e1[j].num)) swap(&(e1[j-1]), &(e1[j]));
}

/* 比較述語 */
compare(int x, int y)
{
    cmp++;
    return (x > y);
}

/* 要素の交換 */
swap(struct record *r1, struct record *r2)
{
    struct record tmp;

    tmp = *r1;
    *r1 = *r2;
    *r2 = tmp;

    mov = mov+3;
}
```

mylib.c

```
/* 学籍番号の切りだし */
get_num(char s[])
{
    int x;

    sscanf(s, "%d", &x);
    return (x);
}
```

D.1.2 分割コンパイル

上記のソースプログラムから、実行形式 bsort-main を生成するには、全てのソースファイルをコンパイルコマンドの引数として与える。

```
% gcc bsort-main.c bsort.c mylib.c -o bsort-main
```

D.1.3 分割化の弊害

プログラムの分割化によって以下のような問題が新たに生じる.

1. コンパイルが複雑化、煩雑化
2. 共通に使用する変数、マクロの管理が複雑化
3. ファイル数の増加によって、ファイル管理が複雑化

それぞれの問題に対する対処法には、以下のようなものがある。

1. 次に示す UNIX の make コマンドを使用する。
2. 共通部分をヘッダファイルにまとめ、それを各ファイルから参照する。
3. ディレクトリの利用。 make コマンドの利用。バージョン管理コマンド (SCCS や RCS) の利用。

D.2 make コマンド

UNIX の make コマンド [5] は、分割されたプログラムを効率良くコンパイルして目的のプログラムを自動作成するためのコマンドである[4]。

たとえば、あるヘッダファイルを修正した場合、それを参照しているプログラムファイル全てを再コンパイルする必要がある。しかし、どのファイルがそのヘッダファイルを参照しているか調べてまわるのは大変である。 make コマンドでは、個々のファイルの依存関係を `Makefile` という名前のファイルに一度定義しておけば、

```
% make targetfile
```

のように、目的のファイル名 (*targetfile*) を指定するだけであとは自動的に必要なコンパイルを行ってくれる[5]。

ファイルの依存関係を記述する `Makefile` はソースプログラムのあるディレクトリで作成しておく。例えば、 `prog.c` をコンパイルして実行形式 `prog` を作成するのであれば、 `Makefile` に

```
prog:    prog.c
```

と記述しておいて、

```
% make prog
```

とすれば良い。すると make コマンドの中から自動的に以下のようなコンパイルコマンドが起動され prog が作成される。

```
cc  -c prog.c -o prog.o
cc prog.o -o prog
```

[4]実は、 make コマンドは ” プログラムジェネレータ ” と呼ばれ、単なる自動コンパイル・コマンド以上の機能を有する。

[5]make コマンドには、いくつかの種類があり、それぞれ仕様が若干異なる。 Sun の make コマンドは BSD Unix あるいは ATT/System V の make コマンドの拡張版になっている。さらに、優れた機能を有するものに GNU の gmake コマンドがある。

一般に Makefile の記述は以下のような書式をとる[6]。

ターゲット 1:　　依存関係 1
　　[コマンド 1]
ターゲット 2:　　依存関係 2
　　[コマンド 2]
　　⋮
ターゲット n:　　依存関係 n
　　[コマンド n]

- **ターゲット** とは、作成したい対象 (通常はファイル) であり、**依存関係** とは、ターゲットを作成する上で、必要とされるファイル (**依存ファイル**) のリストである。
- 依存ファイルは他の行でターゲットであっても構わない。
- 依存ファイルが別の箇所でターゲットとして宣言されていれば、 make は、まず、そちらから解析作業を始める。
- ターゲットよりも、依存ファイルの更新日付が新しければ、ターゲットを再作成する必要があるので、 make は続くコマンド行を (あれば) シェルコマンドとして実行する。
- コマンド行が存在しない場合や、依存ファイルが存在しない場合は、 make は内部規則によってそれを作成・実行することを試みる。

代表的な内部規則には次のようなものがある[7]。

- *file*.o は *file*.c から C のコンパイルコマンドによって作成される。
- *target* は *target*.o からリンクコマンドによって作成される。

先のバブル・ソートの分割コンパイルの場合、 `Makefile` は以下のように書ける。

```
CC=gcc
CFLAGS=-g
#
bsort-main: bsort-main.o bsort.o mylib.o
     $(CC) bsort-main.o bsort.o mylib.o -o mylib.o
bsort-main.o: bsort-main.c bsort.h
bsort.o: bsort.c bsort.h
```

1 行目では、コンパイルコマンドとして、既定値 (default) の 'cc' の代わりに'gcc' を使用するための設定を行っている (D.2.1 参照)。 2 行目では、プログラムのデバック時にデバッガ (debugger) を使用できるように、コンパイルオプションとして'-g' を指定している。 3 行目 (#で始まる行) はコメント行である。 4 行目から 6 行目までが、 bsort-main を作成するための依存関係を記述している。この例では、コマンド行等が省略されているが、以下のよう明示的に書くこともできる。

[6] ここで、[ ] は、省略可能を意味する。
[7] この規則から分かるように、先の prog.c から prog を作成する場合は、実は Makefile には何も書かなくても prog.c から prog を自動生成してくれる。

```
bsort-main: bsort-main.o bsort.o mylib.o
     $(CC) bsort-main.o bsort.o mylib.o -o bsort-main
bsort-main.o:  bsort-main.c bsort.h
     $(CC) -c $(CFLAGS) bsort-main.c
bsort.o:       bsort.c bsort.h
     $(CC) -c $(CFLAGS) bsort.c
mylib.o:       mylib.c
     $(CC) -c $(CFLAGS) mylib.c
```

実行形式ファイル `bsort-main` を作成するには、単に

```
% make bsort-main
```

と入力するだけで良い。注目すべきことは、 make は `bsort-main` を作成するのに必要最小限のコンパイル、リンクを行うという点である。例えば、 bsort-main を一回作成した後、 mylib.c を変更したとすると、以下の作業のみを行い、不必要なコンパイルは行わない。

```
cc -g mylib.c -o mylib.o
cc bsort-main.o bsort.o mylib.o -o bsort
```

D.2.1 マクロ

make ではシェルにおけるシェル変数と似たような **マクロ** が使用できる。マクロの定義は Makefile で、

マクロ名 = 文字列

のように指定する。マクロを参照するには、$(マクロ名) のように記述する。 make コマンドで特別に使用している主要なマクロ名を以下にあげる。

`CC`	C コンパイルコマンド
`CFLAGS`	C コンパイルのオプション
`LDFLAGS`	リンカのオプション

この他、シェルの環境変数も make のマクロとして参照することができる。

D.2.2 より進んだ使い方

先の Makefile の例をより実用的なものに書き変えた例を以下に示す。

```
CC      = gcc
CFLAGS  = -g

OBJS    = bsort-main.o bsort.o mylib.o
PROG    = bsort-main
DESTDIR = $(HOME)/bin

all:    $(PROG)

bsort-main:     $(OBJS)
$(CC) $(OBJS) -o bsort-main
bsort-main.o:   bsort-main.c bsort.h
bsort.o:        bsort.c bsort.h

install:        $(PROG)
        strip $(PROG)
        cp $(PROG) $(DESTDIR)/$(PROG)

clean:  $(PROG) $(OBJS)
rm $(PROG) $(OBJS)
```

この例では、新にターゲットとして、 `all`, `install`, `clean` が宣言されている。 `clean` は、元のプログラムファイルのみ残して、オブジェクトと実行形式を消すためのターゲット名である。

```
% make clean
```

とすると、

```
rm bsort-main bsort-main.o bsort.o mylib.o
```

が実行される。引数に何も与えないと、 make は Makefile 中の 一番最初のターゲット (この例では `all`) が指定されたと仮定して処理を開始する。

なお、 make に `-n` オプションを指定すると、コマンドの本当の実行は行わすに、実行手順のみが表示されるので、動作の確認に便利である。

D.3 Emacs からのコンパイル

プログラムのコンパイルには、シェルのレベルから行う方法の他に、 Emacs の中から、

```
M-x compile
```

によって、行う方法もある。このとき、ミニバッファに `make -k` のような表示が出るので、作成したいターゲットファイル名を入力する。コンパイルが開始すると、画面が分割され、コンパイルのエラーメッセージ用のウィンドウ・バッファが作成・表示される。

コンパイル・エラーが表示されたら、`C-x ‘` (コマンド名: next-error) によってプログラム中のエラー箇所にカーソル (ポイント) を移動させることができる。エラー箇所の修正後、次のエラー箇所に移動するには、再び `C-x ‘` を入力すればよい。

プログラムの修正が終了したら、再び、 compile を実行すればよい。この `compile` コマンドはしばしば、使用するので、`~/.emacs` にキーとして定義しておくと便利である。例えば、次の設定では `C-x C-y` によって compile コマンドが起動される。

```
(global-set-key "\C-x\C-y" 'compile)
```

D.4 デバッガによるデバッグ

プログラムのデバッグに、デバッガ (debugger) と呼ばれるデバッグ専用のコマンドを用いることができる。これは、プログラム中に printf 文などを埋め込んでデバッグする方法に比べて、より柔軟性のある高度なデバッグ環境を提供する。その特徴を列挙する。

- プログラムが異常終了した箇所を知ったり、その時の状況をモニタ (変数の値、関数呼び出しの履歴 等の表示) ができる。
- デバッガからプログラムの実行を制御できる。すなわち、指定の行 (break point) で実行を一旦停止させ、状況をモニタし、再度、実行を継続させることができる。また、プログラムを行単位に逐次実行させたり、変数の値が変化した時点で一旦停止させることもできる。
- 変数の値を一時的に変更することができる。

デバッガには、 OS や 言語等によって様々なものがある。 Unix 標準のデバッガとしては `dbx` がある[8]。 GNU の gdb はより強力なデバッギング環境を提供する。 gdb は Emacs のコマンドとしても用意されており、そこでは、プログラム表示用のウィンドウとデバッガ用ウィンドウを連動させてより使い易い環境が提供される。 gdb コマンドの詳細については、 Emacs の Info コマンドで知ることができる。

D.5 演習問題

1. 自分のディレクトリにあるソースプログラムファイルを分割化し、 `Makefile` に依存関係を書くことによって、 make コマンドによる自動コンパイルを実験せよ。

[8] System V 系 Unix では `sdb`。

付録 E

ディスク管理とバックアップ

E.1 ディスク管理

磁気ディスクは 1 台あたり 1-2GB 程度の有限の容量しかもたない。それを複数のユーザが共有している。ディスクの空容量が少なくなると、新規ファイルの作成・保存ができなくなったり、計算機の正常な動作が阻害される。そのため、日頃から各ユーザは次の点を心掛けておくべきである。

- ディスクの空容量を日頃から知っておく (⇒ E.1.1 節)
- 不必要なファイルを作成しない。できてしまったときは、速やかに削除する。
  例: `a.out`、 `core`
- 長期間使用しないファイルはフロッピディスクや磁気テープに保存して、ディスク上のファイルは消しておく (⇒ E.3 節)
- 大きなファイルは圧縮した形で保存しておく (⇒ E.2 節)

E.1.1 ディスク容量のチェック

ディスクの空容量を知るには、 `df` コマンドを用いる。引数に何も与えないと、マウントされている全ディスク装置の情報が表示される。ディレクトリ名を指定すると、そのディレクトリが存在するディスク装置に関する情報が表示される。

例

```
% df .
Filesystem            kbytes    used   avail capacity  Mounted on
fs0-e2:/home/fs005   1276877  168085  981104    15%    /tmp_mnt/home/fs005
%
```

ここで、左端の `Filesystem` 欄は、磁気ディスク装置を意味し、右端の `Mounted on` 欄は、その装置がマウントされているディレクトリ名を意味している。 `kbytes` 欄はディスクの全容量 (Kbyte)、 `used` 欄は現在の使用量 (Kbyte)、 `avail` 欄は現在の空容量 (Kbyte)、 `capacity` 欄は使用割合 (%) を それぞれ示している[1]。

[1]通常、空容量、使用割合は、ディスクの全容量と使用量から算出される値と一致しない。これは、安全係数をかけているためである。

ディスクの正常な空容量は、そのディスクの使用目的によって異なる。システムコマンドの保存用ディスクで、容量の増加が今後見込まれないものであれば、空容量は殆どなくても大丈夫であろう。しかし、到着メールを一時保存しておくスプールディレクトリ`/var/spool/mail`や、コンパイラやエディタの作業ファイルを置いておく `/var/tmp` などは、変化が大きいので十分な容量が必要である (少なくとも、数 MB)。ユーザのディレクトリについては、さらに大きな空容量が必要である。

E.1.2 ファイル使用量のチェック

個々のファイルの使用量を知るには、 `ls` コマンドを用いればよい。

```
% ls -l filename
```

ディレクトリ単位の使用量を知るには、 `du` コマンドを使う。

```
% du directory
```

引数を省略するとカレントディレクトリ (.) が指定されたと見なす。

例

```
% du
229     ./doc/i2
53      ./doc/etc
341     .
%
```

この例では、カレントディレクトリの下にさらにディレクトリ `doc/i2`、 `doc/etc` があるので、それに関する使用量が表示された後、最後に全体の使用量が表示されている。

E.1.3 修了後のファイル

JAIST を修了した人の個人ファイルは，アカウントと共に一定期間保持される．しかし，修了と入れ替わりに新入生が来るわけであるから，不必要なファイルを残しておくことは好ましくない．JAIST を離れる前に，**不必要なファイルを削除**して，ディスクの使用量をできるだけ減らすこと．なお，必要なファイルの**バックアップも自分自身で**行わなければならない．

E.2 ファイルの圧縮

UNIX にはファイルの内容を符合化してファイル容量を小さくする (圧縮) プログラム、および、元に戻す復元プログラムがある。最もよく使用されるものは、 `compress` (圧縮用)、 `uncompress` (復元用) コマンドである[2]。 compress は、

```
% compress filename
```

のようにして使用する。このとき、”`-v`” オプションを付けると圧縮率 (最低で 0 %) が表示される。圧縮ファイルの名前は *filename*.Z のように拡張子 “`.Z`” が付き、元のファイルは自動的に消去される。復元コマンド uncompress も同様に、

[2] この他に、非標準ではあるが圧縮率が高い `gzip` と `gunzip` もある

```
% uncompress filename
```

とする。このとき引数のファイル名には拡張子 “.Z” があっても無くてもよい。

uncompress することなしに、圧縮ファイルの内容を見るには zcat コマンドを使用する。

なお、ファイルの圧縮率はファイルの内容によって異なる。一般に、文章ファイルやソースプログラム・ファイル等のいわゆるアスキー・ファイルは圧縮率 (60%〜70%程度) が高く、コンパイル結果等のいわゆるバイナリ・ファイルは低い (0%〜40%程度)[3]。

E.3 ファイルのバックアップ

計算機で作成したファイルが、自分のミスや計算機のトラブルによってある日、突然消えてしまって大騒ぎになることがある。様々な原因が考えられる。

- rm コマンド等で、誤って消してしまう。
- エディタの操作ミスでファイル内容を書き変えてしまう。
- コマンドの出力先を既存のファイルに指定して、その内容を破壊してしまう。
- 計算機の磁気ディスクが故障する[4]。

いずれの原因においても、**UNIX では一度消えてしまったファイルの実体を復元することは不可能である。**

このような被害を最小限に抑えるために、大事なファイルは日頃から次のような措置をとっておくとよい。

1. ファイルの書き込み許可フラグを外しておく。 (chmod -w の利用)
2. コピー、すなわち、**バックアップ** をとっておく

最初の、 1 は操作ミスによるファイル内容の破壊を防ぐための消極的な方法である。次の 2 はより積極的な措置であり、その方法には以下のように、いくつかのレベルがある。

1. 同じディレクトリ内にファイル名を換えてバックアップファイルを作成する
2. 別のディレクトリにバックアップをとる
3. 別の装置にバックアップをとる

1 番目は、手軽な方法であり過渡的なバックアップとしては有効である。しかし、ファイル数が多いと混乱をきたすので、 2 番目の方法のように別のディレクトリにとる方法もある。さらに、 3 番目の方法は常用のディスク装置が故障した場合にも有効であり効果的である。以下、この 3 番目の方法について説明する。

共用の計算機では、計算機の管理者が磁気テープ等にバックアップを取っていてくれる場合もあるが、多数のユーザのファイルを頻繁にバックアップするのは困難であるため、たとえバックアップがあったとしても、最新のファイルを回復できるとは限らない。そこで、各自がバックアップをとっておくことが必要になってくる。

バックアップの際には、以下の点に注意する。

[3]実行形式ファイルのサイズを少なくするには、他に、 strip コマンドが有効である。 (ただし、 strip してしまうと、 gdb、 sdb 等のデバッガが使用できなくなる。

[4]このことを俗に、「**クラッシュ**」するという。ディスクのクラッシュは意外と頻繁に、そして前回の教訓を**忘れたころに起こる** (ものによっては 1、 2 年に一度)。

- 不要なファイルのバックアップは行わない (バックアップ・ファイルの容量を少なく抑えるため)
- バックアップの操作には細心の注意を払う。操作を誤ると、元のファイルを破壊してしまう恐れがある。

別の装置にバックアップする際のメディアとしては以下のようなものがある。

メディア	容量	マウントの可否
固定磁気ディスク	-	可
フロッピディスク (2DD)	**0.7MB 程度**	可
フロッピディスク (2HD)	**1.4MB 程度**	可
磁気テープ (1/2、 1/4 インチ)	150MB 程度	不可
磁気テープ (DAT)	5GB 程度	不可
磁気テープ (8mm)	10GB 程度	不可
光磁気ディスク	120MB 程度	場合により可

固定磁気ディスクは、通常、ユーザが使用しているものである。同じディスク装置をバックアップ用にすることは、共用資源の無駄使いであるとともに、バックアップの意味があまりないので勧められない。磁気テープは各 LSS で利用可能である。ここでは、各 Sun WS に装備されているフロッピディスクによるバックアップ法を説明する。

E.3.1 バックアップ方法 1

1. フロッピの準備
   3.5 インチ 2HD フロッピディスクを準備する。

2. フロッピの初期化
   購入したフロッピディスクを初めて使用する際には、以下のようにディスクの**物理フォーマット**を行っておく必要がある。

   (a) 新しいフロッピディスクを Sun 本体右側面のフロッピドライバに挿入する。このとき、ディスクの表を上にし、表面の矢印の書いてある向きにディスクを挿入する。

   (b) フォーマット・コマンド `fdformat` を以下のように起動する[5]。

   ```
   % fdformat
   ```

   この時、

   ```
   Press return to start formatting floppy.
   ```

   のようなメッセージが出るが、リターンキーを入力する。

   注意!! 物理フォーマットを行うと、それまでの内容は破壊されてしまう。

3. ホームディレクトリに移動[6]

[5] 他に、`fdmount h` でも可。2DD フロッピを使用するときは、代わりに `fdformat -l` あるいは、`fdmount f` を実行する。
[6] 必ずしも、ホームディレクトリである必要はない。

```
% cd home
%
```

4. バックアップ開始
   バックアップコマンドとして、 tar を使用する。フロッピに新規 (初めて) バックアップを作成するには

   `tar cvf /dev/fd0` *filename*`...`

   の形式で指定する。ここで、 *filename* はカレントディレクトリからの相対パス名を指定し、複数指定可能である。

   例 1 カレントディレクトリ以下、全てのバックアップをとるとき。

   ```
   % tar cvf /dev/fd0 .
   ```

   例 2 ディレクトリ `doc/i2` 以下の全ての LaTeX ファイルのバックアップをとるとき。

   ```
   % tar cvf /dev/fd0 doc/i2/*.tex
   ```

   上記は、新規バックアップフロッピの作成法であったが、既に作成済のバックアップフロッピに、**追加書き込み**する場合には、 tar の 第 1 引数 `cvf` の `c` を `r` に換えて `rvf` とする。即ち、

   `tar rvf /dev/fd0` *filename*`...`

   さらに、前回のバックアップ以降に変更されたファイルのみ保存 (**update**) するのであれば、第 1 引数として `uvf` を指定する。

   `tar uvf /dev/fd0` *filename*`...`

   `rvf`, `uvf` いずれにおいても、過去のバックアップの内容に新規分が追加書き込みされる。

5. バックアップ終了
   バックアップが終了したら、 eject コマンドでフロッピを WS から取り出す。

   ```
   % eject
   ```

   取り出したフロッピには、ラベルを張って、内容が分かるような見出し等を書いて保存する。

ここで示した tar コマンドでは、複数のフロッピにまたがってひとつの大きなバックアップファイルを作成する機能 (マルチボリューム) がない。すなわち、フロッピの最大容量 (2HD ディスクで で約 1.4MB) を越えるファイルのバックアップはできない。これができるコマンドとして bar がある。

E.3.2 バックアップ内容の確認

バックアップディスクの内容を確認するには、 tar コマンドで以下のように指定する。

```
% tar tf /dev/fd0
```

第 1 引数として、 `tf` に `v` を加えて `tvf` とすると、個々のファイルの日付、サイズ等の細かい情報が表示される。

E.3.3 リストア方法 1

バックアップ・ファイルの内容を元に戻してファイルを復元することを**リストア** (restore) と呼ぶ。 tar コマンドによるバックアップ・ファイルから、必要なファイルをリストアする際にも、やはり tar コマンドを用いる。以下に手順を示す。

1. フロッピを WS に装着する
2. バックアップをとった時と同じディレクトリに移動する。

   例

   ```
   % cd home
   ```

3. 以下のような形式でリストアする。

   ```
   tar xvf /dev/fd0 filename...
   ```

   ここで、 *filename* は、リストア対象のファイル名である。ディレクトリ名を指定すると、そのディレクトリ以下の全てがリストアの対象となる。 *filename* が一つも指定されないと全内容をリストアする。

   例 1 内容全部をリストアする場合。

   ```
   % tar xvf /dev/fd0
   ```

   例 2 ファイル "`./doc/i2/report1.tex` をリストアする場合[7]。

   ```
   % tar xvf /dev/fd0 ./doc/i2/report1.tex
   ```

4. バックアップが終了したら、 eject コマンドでフロッピを取り出す。

E.3.4 バックアップ方法 2

フロッピディスクをファイルシステムとしてマウントし、元のファイルやディレクトリと同じイメージでバックアップを取っておくことができる。この場合のバックアップディスクの最大容量は、ファイルシステムを作成しない場合に比べて 1〜2 割程度減少する。

1. フロッピの準備
   物理フォーマット済みのフロッピ (E.3.1参照) を準備し、 WS に装着する。
2. フロッピの論理初期化
   ファイルシステムと使用するには、物理フォーマットの後にさらにファイルシステムを構成するために以下の論理フォーマットを行う。

   ```
   % fdmount i
   ```

   このとき、以下のようなメッセージが表示されるが、単にリターン・キーを入力する。

[7] リストアのファイル名指定で、シェルのワイルドカード (*) 等は使用できない。どうしてか？

```
Press return to start formatting floppy.
```

3. ディスクのマウント
   マウントは以下のように行う。

   ```
   % fdmount m
   ```

   すると、何行か診断メッセージが表示されるが、最終的に、

   ```
   fdmount: 'fd0c' mounted on '/floppy'.
   ```

   のような表示が出て、ディレクトリ `/floppy` がファイルシステムとして使用可能となる。
   フロッピの空容量を知るには、

   ```
   % df /floppy
   ```

   とすれば良い。

4. ホームディレクトリに移動
   (あるいは適当なディレクトリに移動)

5. バックアップ開始
   コピーには以下のようないくつかの方法がある。ここで、 *dest-directory* は `/floppy` で始まるディレクトリ名である

   形式 1 `cp` コマンドでファイルをディレクトリにコピーする場合。

   % cp -pr *filename*... *dest-directory*

   フロッピ上にディレクトリが既に存在していなくてはいけない。
   例: doc/i2 のバックアップを取る。

   ```
   % cp -pr doc/i2 /floppy/doc/
   ```

   形式 2 `tar` コマンドで、ファイルやディレクトリをコピーする場合。(以下の書式は 1 行である)

   % (chdir *src-directory*; tar cf - *filename* ...) | % (chdir *dest-directory*; tar xvf -)

6. バックアップ終了
   以下の手順でフロッピを取り外す。

   (a) マウントの取り消し

   ```
   % fdmount u
   ```

   (b) フロッピの回収

   ```
   % eject
   ```

E.3.5 リストア法 2

バックアップ法 2 で作成したフロッピは、ファイルシステムとして使用できるのでリストアは簡単である。

1. フロッピを WS に装着
2. フロッピを WS マウントする

   ```
   % fdmount m
   ```

   これによって、/floppy にフロッピがマウントされる。
3. バックアップ開始
   /floppy ディレクトリ以下から必要なファイルを見つけ、それをコピーすればよい。
4. バックアップが終了したら、ディスクをアンマウントし、 WS から回収する。

   ```
   % fdmount u
   % eject
   ```

E.3.6 その他のバックアップ法

tar, cp コマンドの他にも、 find, cpio 等のバックアップ用のコマンドがある。要求に応じて選択すべきである。詳しくはマニュアルを参照されたい。

その他、パソコン の MSDOS 形式のファイルも dosmount コマンド によってマウントすることができる。このコマンドの使い方は fdmount と同様である。また、/usr/local/bin/mtools にあるコマンドは MSDOS フロッピの様々なアクセスを提供している。いずれのフロッピーも IBM 型の MSDOS フォーマット (720KB または 1.44MB) を対象としている。 PC98 では 2DD ディスクを 720KB でフォーマットした場合、使用することができる。詳しくは、マニュアルを参照されたい。

付録 F

システム管理

F.1 システム管理者の仕事

UNIX のシステムの管理を行うための特別アカウント (ログイン名) として、“root” がある。一般ユーザは、他人のファイルを消したり、その内容を変更することはできないが、 root になると、全ての保護モードの制約から開放されて作業を行ことができる。このような特権を持つ root を、**スーパーユーザ** (super user) と呼ぶこともある。

root は他人のファイルの内容を覗いたり、システムのファイルを変更したりする強力な権限を持つため、 root に成ることのできる人、すなわち、 root のパスワードを知っている人は極少数の計算機管理者に限定することが重要である。

root の仕事には以下のようなものがある。

- システムのインストール
- 日常のシステム管理
  - 新規ユーザの登録
  - システムの立ち上げ、停止
  - ディスク使用状況、プロセス状態監視、ネットワーク環境の監視
  - 不法侵入、不法アクセス等の監視
  - ファイルのバックアップ
- ソフトウェアのインストール
- ハードウェアの追加、障害の監視
- (root のパスワードの秘密保持)

F.2 ユーザ登録

新規ユーザの登録は以下の手順で行う

- データベース (パスワードファイル) への登録

- ホームディレクトリの作成
- 初期設定ファイル (`.login`、 `.cshrc` 等) の準備

UNIX の各ユーザは、/etc/passwd というデータベースファイル (**パスワードファイル**) によって管理されている。ここに新規ユーザの情報を追加する。パスワードファイル中の各行が、ユーザの個人情報を示しており、行はコロン (:) で区切られた 7 つの欄 (フィールド) から構成されている。例えば、こんな感じである。

```
totoro:7L7Huxbf5amKL:761:1000:Satsuki Kusakabe,5F,6543,:/home/fs021/totoro:/bin/tcsh
```

各フィールドの意味は、左から順に、

> ログイン名、パスワード (暗号化されたもの)、 UID、 GID、 GCOS フィールド、ホームディレクトリ名、ログイン・シェル名

となっている。 UID (User ID) は、ログイン名に対応する整数値で、計算機内部の処理はログイン名の代わりに、この値が利用されている[1]。 GID (Group ID) は、ユーザが属するグループ名に対応する整数値である[2]。 GCOS フィールドは、ユーザのフルネーム等を書いておく欄で、電子メールの`From:` 行等に差し出し人の名前を自動的に挿入する機能等に使用される。

なお、 JAIST のような多数の WS を有するネットワークでは、個々の WS で パスワードファイル を持つことは好ましくなく、 1 台の WS で集中管理する方式が使用されている。この管理システムを NIS (Network Information Service) と呼んでいる。 NIS 使用時のパスワード・ファイルは特別のコマンド ypcat で参照する[3]。

```
% ypcat passwd
```

なお、 WS に現在ログイン中のユーザを知るには、 `who` コマンドを使用する。また、これまでの ログインの履歴を知るには `last` コマンドが使える。

F.3 システムの停止

長期間 WS を使用しない場合、あるいは、システムの設定を変更するときなどに、システム管理者は、システムを停止 (shutdown) させることがある。システムの停止には、必ず、専用のコマンド shutdown を使用する。さもないと、 UNIX の ファイルシステムが破壊されることがあるので、この点は特に大切である。

shutdown の手順

1. who, ps コマンド等で、他にユーザや、ユーザのプロセスが存在しないか確認する。 (存在する場合は、停止してよいか確認する。)
2. WS の コンソール で root になる。

[1] root の UID は 0 である
[2] 自分の属するグループ名を知るには、コマンド groups が使える。また、グループ名は /etc/group に登録されている。
[3] 厳密には、/etc/passwd ファイル中の '+' で始まる行が NIS の対象になっている。

3. 以下の要領で shutdown コマンドを起動する。

```
% cd /
% /etc/shutdown -h now
```

コマンドが起動されると、いくつかのメッセージの表示後コンソールに

```
ok
```

の表示が出る。これは、 UNIX システムが停止し、**モニタ** と呼ばれる特別のレベルにあることを示すプロンプトである。

4. (電源を切る必要があれば、この段階で本体等の電源を切る。)

なお、 JAIST では、ユーザー名 shutdown で login することにより、システムを停止することができる。

F.4 システムの立ち上げ

停止していた WS を起動するには、基本的に、電源を入れるだけでよい。以下に立ち上げの流れを示す。

1. 装置 (本体、ディスプレイ等) の電源を入れる。
2. 本体の電源が入ると、自動的に立ち上げ作業が以下の順に始まる[4]。
   (a) メモリのチェック
   (b) 周辺装置の確認
   (c) ファイルシステムのチェック
   (d) 各種デーモンの起動
   (e) `login:` メッセージの表示 (立ち上げ完了)

F.5 プロセス管理

UNIX のコマンドは内部で プロセス という単位で実行、管理されている。プロセスの中には、ユーザが発行したコマンドもあれば、システムのものもある。

一つの WS の処理能力は有限であるから、プロセスの数が多くなると、 WS の処理速度が低下する。非常にメモリを使用するプロセスがあっても同様である。バックグラウンドジョブの機能があるからといって、無闇やたらと同時に多数のコマンドを実行させることはかえって、順に実行する場合よりも遅くなることも有り得る。

使用中の WS のプロセスの状態を監視するには、 `ps` コマンド (詳細は、 5.2 節参照) の他に、`sps` コマンドが使える。 sps は プロセスの親子関係を整理して表示する。

システム管理者は以下の項目に日頃から注意を払っておくべきである。

- プロセス数。いつもより異常に多くないか?

[4]shutdown コマンドによってモニタレベルにいるときは、プロンプトに対して、"`boot`" と打つと起動 (boot) が始まる。

- プロセスのメモリ使用量。
- プロセスの CPU 占有率
- 不法なプロセス、変な名前のプロセスはないか?

F.6 不法アクセスの監視

- ログイン・ユーザ名、起動プロセスの確認。
- /tmp, /var/tmp ディレクトリの確認。変なファイルがないか?
- ログイン記録の監視。
  last コマンドによって、過去のログイン履歴を見ることができる。
- コンソール・ウィンドウの監視
  root ログインなどの重要なイベントは画面に表示される。
- システム・ログ・ファイルの確認
  重要なイベントは /var/adm/messages, /var/log/syslog 等のファイルに記録される。

補足

WS の利用者は以下の点に注意して欲しい。

- 帰宅時など長時間席を離れる場合は、ディスプレイの電源を切っておく。 (**注: WS 本体の電源は切ってはいけない**)
- 席を離れる際は、他人にいたずらされないように、ログアウトしておくか、 xlock コマンドによって、端末をロックしておく。 root 権限で作業しているときは、特に注意すること。

付録 G

RMAIL の使い方

電子メールをやり取りするためのアプリケーションは色々あるが、ここではEmacs 標準の RMAIL を用いる方法を述べる。

G.1 電子メールの送信 (GE§4.1)

G.1.1 メールの作成と送信

メールの作成には、以下のコマンドを用いる。

C–x m 送信するメールを作成する。

C–x m を実行すると、図 G.1のような画面が現れる。以下の手順でメールを作成し、送信する。

1. `To:` の後ろに、メールを送りたい相手のアドレスを書く。
2. `Subject:`の後ろに、メールの題目を書く。ここでは、日本語（漢字コード) を使ってはいけない。
3. `--text follows this line--`の行の下の行から、メールの本文を書く。この表示は消してはいけない。この行は実際に送られるメールには含まれない。
4. 最後までメールを書いた後、自分の名前を書き、以下のコマンドでメールを送信する。

C–c C–c メールを送信する。

G.1.2 複数の相手に同じメールを送る

`To:`のところに、複数のアドレスを書くと、それぞれに同じメールが届く。また、`Cc:`で始まる行を作り、そこにアドレスを書くと、そこにもメールが届く。 Cc は、カーボンコピーを意味する。

```
To: ssato@kuee.kyoto-u.ac.jp, ssato@atr-ln.atr.co.jp
Cc: sato-lab
BCC: sato
```

基本的には、`To:`は、そのメールを送信する相手、`Cc:`は、送信する相手ではないが、そのメール

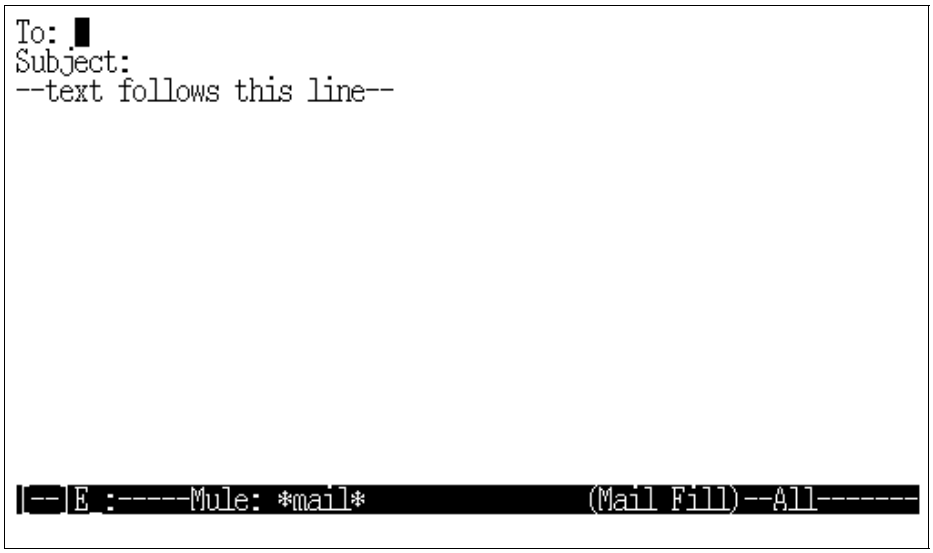


図 G.1: 送信するメッセージの作成画面

の内容を知らせておきたい相手、のように使い分ける。`To:`や`Cc:`は、メールの一部として相手に届くので、相手は、そのメールが誰に送信されたのかを知ることができる。これに対して、`BCC:`は、ブラインド・カーボンコピーと呼ばれ、メールは送信されるが、その事実は、他のメールの送信先には隠される[1]。通常は、自分宛のカーボンコピーはブラインド・カーボンコピーとする。

G.1.3 メール作成の注意

メールを使用する場合、以下の点に注意すること。

1. アドレスを良く確認し、間違えないようにすること。
2. 1 行の長さを 70byte (英字 70 文字、日本語 35 文字) 程度以下にすること。つまり、適当に改行すること。
3. あまり長いメールを送らないこと。 1 つのメールのサイズが 100Kbyte を越えるような場合は、分割すること。
4. サブジェクト (subject) で漢字を使用しないこと[2]。漢字はメールの本文のみで使用可能である。
5. 漢字コードは、 JIS コードを用いること。

G.2 Rmail (GE§4.2)

電子メール読んだり、整理したりする方法には、いくつもの方法がある。ここでは、主に、 Emacs の rmail という機能を使った方法について述べる。他に、 mh-e (**GE§4.3**) がある。

[1] これがブラインドという意味である。
[2] MIME に対応している場合は漢字なども使用可能である。

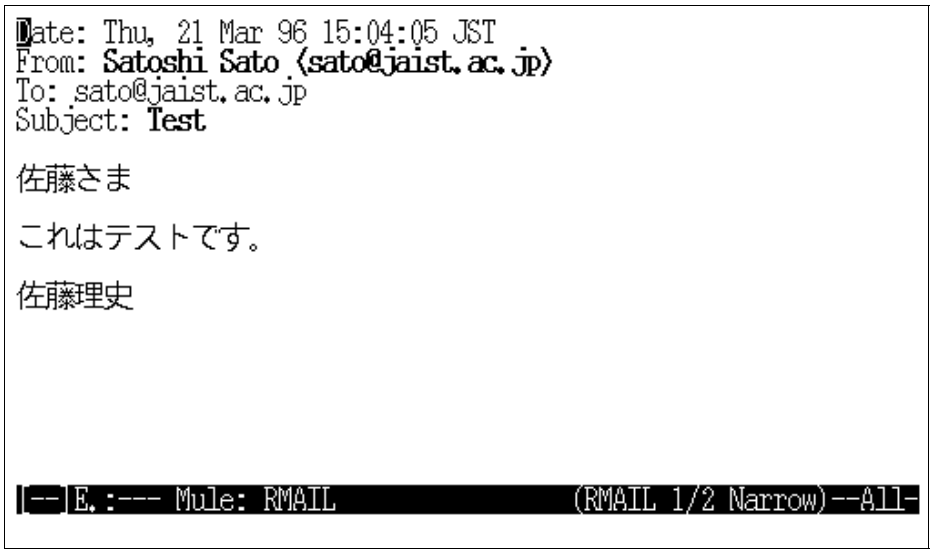


図 G.2: rmail の起動直後の画面

G.2.1 メールの到着

メールを読む作業は、まず、メールの到着を知ることから始まる。メールの到着を知る方法には、色々な方法がある。例えば、 login 時にメールが到着していれば、`You have new mail.` 等のメッセージが出力される。また、ウィンドウ・システムを使っていれば、右上のポストにメールの有無が表示される。 unix コマンド`from` は、到着したメールがどこから来たものかを教えてくれる。

G.2.2 rmail の起動

到着したメールを読むには、まず、以下のコマンドで rmail を起動する。

M–x rmail rmail を起動する。

起動後の画面を図 G.2に示す。画面には、新しく到着したメールが表示される。 rmail では、メールに関する色々な処理ができる。

なお、この図において、上から 5 行を、メールのヘッダーと呼ぶことがある。ここには、送信元、日付け、返事の宛先、宛先、主題 (Subject) などが書かれる。

G.2.3 rmail の終了

rmail を終了するコマンドは、以下の通りである。

q rmail を終了する。

rmail を終了すると、通常の Emacs の状態に戻る。

G.2.4 メールを読む — rmail のコマンド (1)

rmail の基本的な機能の一つは、到着したメールを見ることである。
rmail は、起動した後、まだ読んでいないメールのうち、最も前に到着したものを表示する。
メールが一画面に収まり切らない場合は、以下のコマンドが便利である。

SPC 順方向スクロール

DEL 逆方法スクロール

勿論、通常の、 Emacs のポイントを移動するコマンドもそのまま使える。
rmail では、到着したメールは到着順に並べられる。より前に到着したメール (前のメール) や、より最近到着したメール (次のメール) へは、以下のコマンドで移動できる。

n 次のメールへ移動

p 前のメールへ移動

現在見ているメールが何番目のメールかという情報は、モード行に表示される。例えば、図 G.2の

```
(RMAIL 1/2 Narrow)
```

は、現在 2 通 (分母) のメールがこのファイル中にあり、 1 通目 (分子) のメールが表示されていることを示している。
また、以下のコマンドによって、メールのサマリだけを表示させることもできる。

h サマリの表示

このコマンドによって表示されるサマリ画面においても、メールに関する色々な処理ができる。必要ならば各自で調べよ。
なお、 rmail を起動した後に到着したメールは、自動的には、 rmail には読み込まれない。明示的に、以下のコマンドを実行する必要がある。

g 新しく到着したメールを読み込む。

G.2.5 返事を書く — rmail のコマンド (2)

到着したメールに対して返事を書くには、そのメールを表示させた状態で、以下のコマンドを実行する。

r 返事を書く

このコマンドを用いて返事を作成している画面を図 G.3に示す。このコマンドを実行すると、画面が 2 つに分割されて、下半分がメール作成用の画面となる。この場合、相手のアドレスやサブジェクトは自動的に埋められる[3]ので非常に便利である。つまり、本文だけを書いて、 C–c C–c で送信すればよい。

注意!! 相手からのメールが別の人に Cc されていた場合は、 r による返事の作成では、 Cc されていた人にも返事が Cc されるように設定される。もし、それが不都合な場合は、自分で明示的に `Cc:` の行を編集する必要がある。

[3]必要ならば、変更したり、追加したりしてよい。

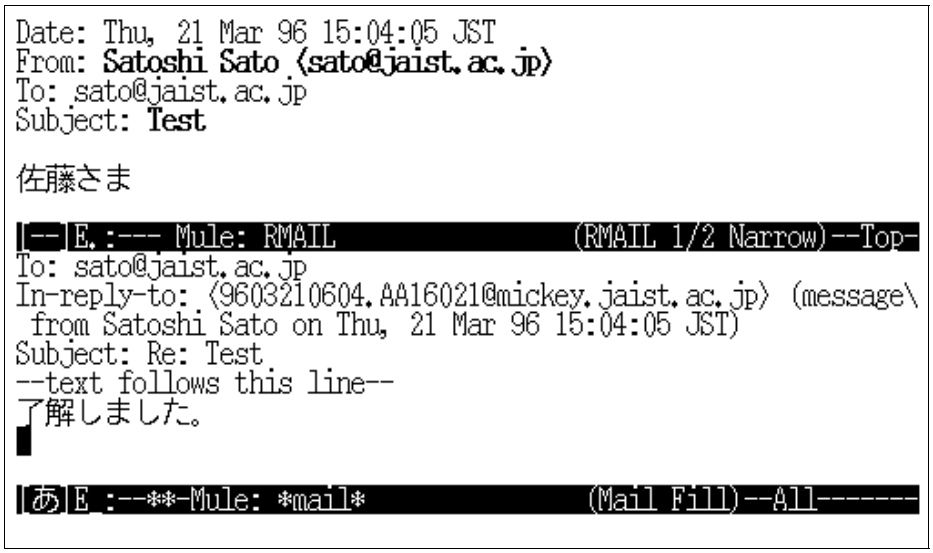


図 G.3: 返事の作成画面

返事を書く場合に、相手からの手紙の内容を参照したい場合がある。これには、以下のコマンドが便利である。

C–c C–y 元のメールの内容を返事のメールにコピーする

後は、適当に編集して使えば良い。

到着したメールに返事を書くのではなく、そのメールをそのまま別の誰かに送りたい (forward したい) 場合には、以下のコマンドを実行する。

f メールをそのまま forward する

メールの内容が自動的にコピーされるので、 forward したい相手先のアドレスだけを入力して送信 (C–c C–c) すればよい。

rmail を起動後は、以下のコマンドでメール送信用の画面にすることができる。

m 送信するメールを作成する。

これは、 C–x m と全く同じである。勿論、 C–x m も使える。

G.2.6 メールを整理する

どんどんメールが到着すると、メールはたまる一方で収拾がつかなくなってくる。そのためメールの整理が必要になる。

不要になったメールは消せば良い。これには、まず、消去マークをつけ、次に、実際に消去するという 2 ステップが必要である。

d 現在表示されているメールに消去マークをつける。

x 消去マークのついたメールを実際に消去する。

実際に消去してしまう前ならば、以下のコマンドで、消去マークをとり消すことができる。

u 現在表示されているメール、あるいは、その前のメールの消去マークをとり消す。

必要ではなくなったが、一応とっておきたいメールは、いくつかのファイルに整理して保存しておけば良い。それを説明する前に、多少の予備知識が必要である。

1. rmail は、`~/RMAIL` というファイルを利用する。到着したメールは、明示的に消去しないかぎり、このファイルに保存される。

2. `~/RMAIL` は、 rmail 形式と呼ばれる形式のファイルで、複数のメールを 1 つのファイルに格納している。

3. rmail は、複数の rmail 形式のファイルを扱うことができる。

4. `~/RMAIL` は、 rmail 起動時に読み込まれる標準ファイルである。

5. 他の rmail 形式のファイルは、明示的な指定によって読み込まれる。

つまり、 rmail には、標準的なファイル (`~/RMAIL`) が用意されており、通常はそれを使う。不要となったメールは、そのファイルから別のファイルに移して管理すればよいのである。もし、別のファイルに入れたメールを見る必要が生じたならば、そのファイルを明示的に読み込めばよい。これらを行なうために、以下のコマンドが用意されている。

o 現在表示されているメールを、 rmail 形式でファイルにセーブ (追加) する。

i rmail 形式のファイルを、バッファに読み込む。

なお、コマンド o によって、他のファイルに書き出しても、そのメールは、明示的に消さない限り、消去されない。

この他に、普通のファイルにメールの内容を書き出したい場合は、以下のコマンドが便利である。

C–o 現在表示されているメールを、 unix mail 形式でファイルにセーブする。

保存しておくメールを全て`~/RMAIL` に入れておくと、頻繁に変化するために誤操作などで大事な**メールが失われる可能性が高い**。メール用のディレクトリを作り、 o コマンドを用いて、差出人や目的別に分類するとよい。

G.2.7 その他

上記の説明の通り、 rmail では、メールを保存するために、`~/RMAIL` というファイルと、そのバックアップ用ファイル`~/RMAIL~`というファイルを使用する。メールを他人から見られたくない場合は、これらのファイルの保護モードを、以下のように設定しておく必要がある。

```
% chmod go-rwx ~/RMAIL ~/RMAIL~
```

G.2.8 メーリングリスト

電子メールは一人に対して送るだけでなく、一度に特定のグループに属する全員に送ることも可能である。このような仕組みをメーリングリストと呼ぶ。

学内で広く利用されているメーリングリストには次のようなものがあるが、この他にも各研究室や個人で管理するメーリングリストが多数存在する。

is-gakusei98 1998 年入学の情報科学研究科の学生全員

ms-gakusei98 1998 年入学の材料科学研究科の学生全員

ks-gakusei98 1998 年入学の知識科学研究科の学生全員

is-faculty 情報科学研究科・情報科学センターの教官

ms-faculty 材料科学研究科・新素材センターの教官

ks-faculty 知識科学研究科・知識科学教育研究センターの教官

all-students 学生全員

jaist-all JAIST の全構成員

これらのメーリングリストはいずれもメンバー数が多く、不用な情報を流すと大勢が迷惑するので、使用する際には十分注意しなければならない。また、一通のメールがメンバー全員宛に複製されるため、大きなメールを送るとメールサーバに大きな負担をかけることになる。特に、 jaist-all は教官・学生・事務職員などを全員含むものであるから、これらの全員に対して、緊急でかつ重要な情報を伝達する場合にのみ利用されるべきである。それ以外の場合には、ニュースの frontier.announce 等を利用するとよい。

注意!! 本学のアカウントを使用しなくなる場合、個人的にメンバー登録を行ったメーリングリストがあれば、それぞれの登録の抹消の手続きを自分で責任をもって行わなければならない。読まれないメールが大量に蓄積されるのは計算機資源の無駄使いである。

G.3 演習問題

1. Emacs から自分宛にメールを出す実験をせよ。 (`C-x m` を使用)
   - 宛先は ‘ユーザ名`@jaist.ac.jp`‘ とする。
   - `Subject:` 欄は、‘`test mail 1`‘ とする。
   - メールの本文は適当に書いて構わない (英文か日本語)。
2. Emacs から `Rmail` コマンドを利用して以下の実験をせよ。
   - Rmail を起動して、先に出したメールが届いているか確認せよ。 (`M-x rmail` を使用)
   - そのメールに返事 (reply) を出せ。返事の内容は適当でよい。 (`r` を使用)
   - 今出したメールを Rmail のバッファに読み込んで、返事が届いているか確認せよ。 (`g` を使用) また、届いたメールのヘッダ部分がどうなっているか確認せよ。

- `~/Mail` というディレクトリを作り、そのメールを `~/Mail/mymail` という名前で保存せよ。(`o` を使用、ディレクトリの保護モードに気を付けよ)

3. 自分以外の JAIST 内部の知人 2 人以上を宛先とするメールを書け。その際、ヘッダー部に、カーボン・コピー行 (`Cc:`) を追加し、そこに自分のアドレスを書いてメールを送ったときと、代わりにブラインド・カーボン・コピー行 (`BCC:`) に自分のアドレスを書いて送った場合の違いについて確認せよ。

付録 H

人に聞く前に

計算機に関することは、基本的に、全て約束事であるからして、知らなくてもそれを恥じる必要は全くない。知らないことは、聞けばいいのである。しかし、多くの場合、色々なことを知っている人は、忙しい場合が多い。それ故、本章では、多くの人がつまずきそうな点に関して、まとめて述べておく。

H.1 UNIX 関連

- **こんなことをするコマンドはないのかしら?**
  - 関連するキーワードがわかるならば、`man -k` *keyword* で調べる。
  - 参考書 (**U**) を見る。

H.2 Emacs 関連

Emacs は自身の中に大量のドキュメントとそのいろいろな検索機構を持っており、ほとんどの疑問は自分で容易に解決することができる。

- **特定のコマンドの機能を知りたい**
  - キー操作から C-h k で調べる。
  - コマンド名から C-h k で調べる。
  - キーワードから C-h a で調べる。
- **設定変更用変数の機能を知りたい**
  - 変数名から C-h v で調べる。
  - C-h m で現在のモードに関する主な変数を調べる。
  - キーワードから M-x apropos で調べる。
- **ライブラリの機能を知りたい**
  - C-h m で主な機能や設定変更法などを調べる。
  - M-x info で詳しく調べる。

H.3 LaTeX 関連

- **図を書く**
  - ブロック図などは、 tabular で書ける場合がかなりある。
  - picture 環境を使う。
  - xfig や tgif, idraw などのアプリケーションで図を書き、それを PostScript のファイルとしてセーブして、 epsf.sty を使って LaTeX に取り込む。
  - 清書して、切り貼りする。
- **作図時の注意**

  UNIX 上の作図ツールでは、漢字に関していくつか注意が必要である。これは X ウィンドウと PostScript で漢字の扱い方が異なることに起因している[1]。

  この注意を怠ると、画面上では何も問題がないのに、後で印刷時に悩むことになる。これは出力機器によっても変わるので、描いた直後に印刷できても、半年後にも印刷できる保証はないし、図が多い場合はどの図が原因かもわからず悩むことになるので、充分に注意されたい。

  xfig 漢字は使えない。
  図はできる限り英語で書こう!!

  jxfig 漢字も使えるが、古いもので、あまりよくない。
  1 つの文字列中に、英数記号と漢字を混在させるのはよくない。英数記号の文字列には通常のフォントを、漢字の文字列には日本語フォントを指定し、複合フォントはできる限り利用しない。間違ったフォントを指定していても描画時にエラーは出ないが、印刷時に問題が起こることもある。

  tgif 漢字も使える。
  1 つの文字列中に、英数記号と漢字を混在させない。英数記号の文字列のフォントの指定では、**必ず**日本語フォントを**空欄**にする。フォントの指定が間違っていても描画時にエラーは出ないが、印刷時に問題がよく起こる。

  要点は、英数字と漢字で文字列を分けることと、それぞれのフォントの指定を間違えないことである。また、複合フォントや Windows の 'P' が付くフォントを利用するのはよくない。

  このことをよく把握したメーカーのアプリケーションでは、英語フォントと日本語フォントを組み合わせて、 1 つの文字列中に混在させても自動的にそれぞれのフォントに切り替えてくれる機能を持つものがある。
- **図を取り込む**
  1. スキャナが接続された Macintosh で図をスキャンする。 EPSON のスキャナであれば、EpScan Mac という付属のアプリケーションが利用できる。
  2. 必要な部分の切り出しや整形を Macintosh 上で行う、
  3. TIFF, GIF, JPEG など UNIX 側で処理できる形式で保存する。
  4. UNIX 側へファイルを転送する。

[1] 最近、巷ではびこっている Windows でも同様の問題があり、さらに複雑である。

5. ファイル形式に応じて、 xv, xpr, tifftopnm & pnmtops などのツールを用いて PostScript に変換する。
6. epsf.sty を使って LaTeX に取り込む。

この手順は一例であり、整形や PostScript への変換は UNIX と Macintosh のどちらで行ってもよい。

- **グラフを書く**
  - xgraph, gnuplot などのグラフ作成ソフトを使って、グラフを作り、それを PostScript のファイルとしてセーブして、 epsf.sty を使って LaTeX に取り込む。

H.4 ファイル転送とコード変換

H.4.1 Macintosh のファイル

UNIX と Macintosh の間のファイル転送を行うには Mac 側で操作するが、主な方法は次の通りである。

1. Fetch

   一番手軽に利用できる方法であろう。予め設定する必要は何もなく、 Macintosh で Fetch というアプリケーションを起動する。 Open して、自分の機械のホスト名とパスワードを入力すると接続されるので、その画面の中で自由にファイルを選択して Get / Put できる。なお、最後に Close することを忘れないように気を付けること。

   Macintosh 側から Get する場合には、ファイルのタイプはある程度自動判別され、テキストファイルであれば改行コードの変換もされる。しかし、漢字コードの変換はされないので、予め UNIX 側で行っておくか、変換を行う Macintosh のアプリケーションを用意しなければならない[2]。

   Macintosh 側で Put する場合には、いくつかのファイルの変換方法を明示的に指定する。テキストファイルであれば “Text” を選び、 TIFF などのバイナリファイルの場合には “Raw Data” を選ぶのが一般的であろう。

2. AppleShare

   各講座の LSS には Macintosh と情報をやり取りするためのソフトウェアがインストールされている。そのため、 UNIX 側で設定しておくと、 Macintosh で UNIX 上のディスクを自由に読み書きすることが可能である。

   初めて使う前に次のような設定を行う。途中の *name*は好きな文字列であるが、この文字列が Macintosh 上に表示されるので、共有 Macintosh を利用することを考えるなら、自分の login 名などがよいであろう。

```
% mkdir ~/Mac
% echo '~/Mac   name' > ~/.AppleVolumes
```

[2]付録 B を参照

これを利用する場合には、 Macintosh 上で “セレクタ” を開き、 AppleShare で LSS を選択する。 “OK” すると、次にユーザ認証の画面が現れるので、自分の login 名とパスワードを入力する。認証が正しく行われると、ボリューム選択画面が現れるので、自分が利用する名前のものを選択する。ただし、セキュリティ上の問題があるので、**ここで右端のチェックボックスをマークしてはいけない!!**

以上の手順を行うとデスクトップ上にボリュームが現れるので、その後は、それが Macintosh のディスクであるかのように自由に読み書きできる。ただし、最後に **“ごみ箱” へ捨てることを忘れないように**十分気を付けること。

注意!! ディスクの使用量に十分注意すること。 (E.1.2を参照)

3. DOS フォーマットフロッピー

   DOS フォーマットのフロッピーディスクは UNIX でも Macintosh でも利用可能である。これを利用すればフロッピーディスクを介したファイル転送も可能である。

H.4.2 DOS のファイル

DOS のファイルを UNIX 側とやり取りするには、 E.3 で述べたように、 UNIX 側で dosmount や mtools を用いて DOS のフロッピーを読み書きするのが一般的である。

Macintosh でも現在の System では標準で DOS フォーマットフロッピーをサポートしているので、この機能を利用していれば DOS のフロッピーを単に挿入するだけで自由に読み書きできる。

いずれの場合でも、漢字コードと改行コードの変換は別に行なう必要がある。

なお、 Windows NT や Windows95 が動いている機械で、ネットワークに接続され、 TCP/IP が利用できるならば、 Windows 側で ftp することも可能である。

H.5 プリンター関連

- **出力するプリンターを変更する**
  - `lpr -P`*printer-name* でプリンターを明示的に指定して出力する。出力できるプリンター (の名前) は、`/etc/printcap` を見ればわかる。
  - `setenv PRINTER` *printer-name* で default printer を切替る。

H.6 ニュースに質問を投稿する

frontier.sys.question は、計算機および frontnet に関する質問投稿用のニュースグループである。各種質問は、このニュースグループに投稿すればよい。

参考文献

[1] Leslie Lamport: 文書処理システム LaTeX, Cooke, 倉沢 (監訳), アスキー出版局, 1990.

[2] Wall, L. and Schwartz, R. L., (近藤嘉雪 訳), Perl プログラミング, ソフトウエアバンク株式会社, 1993.

[3] Aho, A., Kernighan, B. and Weinberger, P: *The AWK Programming Language*, Addison-Wesley Publishing Company, 1989. (邦訳は、トッパンより出版されている)

[4] Kernighan and Plauger: プログラム書法, 共立出版.

[5] スティーブ・タルボット, (矢吹道郎 監修, 菊池 彰 訳): UNIX ユーティリティ ライブラリ “make”, 啓学出版, 1990.